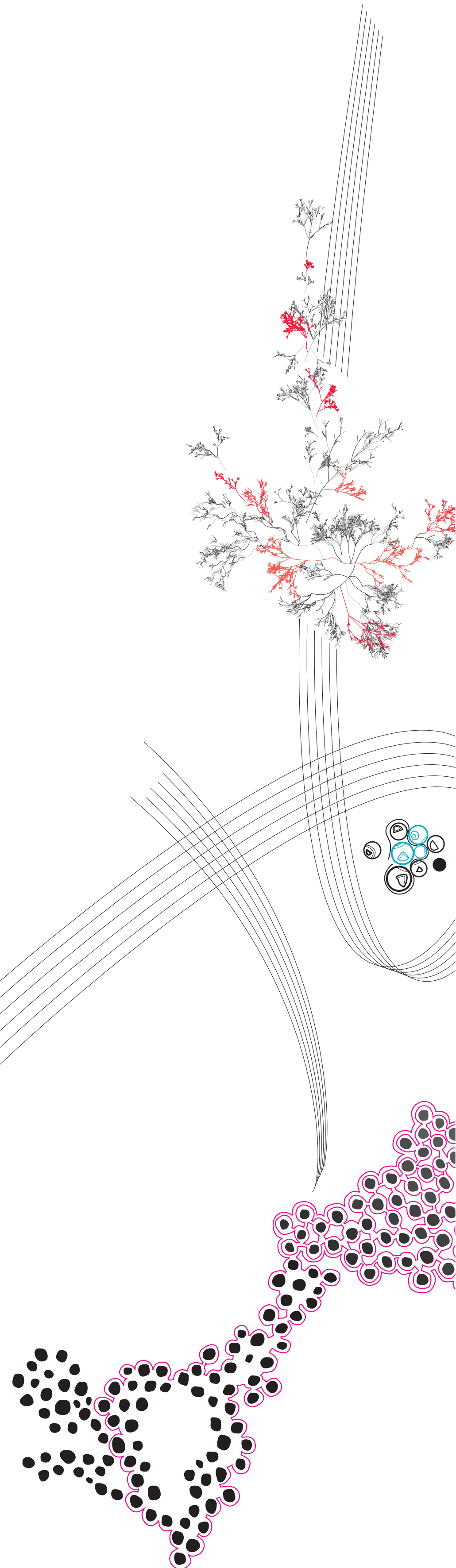


Master Thesis
Industrial Engineering and Management

Developing a plate nesting algorithm for a steel processing company

Britt van Oosten
Student Number: 2809214

First Supervisor: Dr. A. Trivella (Alessio)
Second Supervisor: Dr. B. Alves Beirigo (Breno)
Company Supervisor: L.S. Stoverink (Luc) (Product Owner)

22 August, 2024
Publication Date: 30 August 2024

Department of IEM
Faculty of Behavioural,
Management and Social Sciences,
University of Twente

UNIVERSITY OF TWENTE.

Preface

Welcome to my master's thesis titled "Developing a plate nesting algorithm for a steel processing company". This report represents the final project of my studies at the University of Twente, where I have been enrolled in the Master's program in Industrial Engineering & Management since September 2022. My work on this thesis spanned from February to August 2024. Prior to this, I completed a Bachelor's degree in Mathematics at Utrecht University and a pre-master course in *OR Models* in 2021 as preparation for my master with a specialization in Production & Logistics Management.

In this thesis, I tackled the two-dimensional irregular bin packing problem, a subject I had no knowledge of. The choice to focus on this topic stems from my desire to integrate my mathematical background with practical industrial applications. This project not only allowed me to apply mathematical theories to real-world problems but also pushed me to improve my programming skills, an area I have always found challenging. Additionally, this thesis provided valuable experience in documentation, communication, and understanding of what it is like to work within a company. The combination of theoretical and applied mathematics, coupled with the opportunity to integrate my interest in problem-solving and my hobby of tackling complex puzzles with professional work, made this topic an ideal fit.

I would like to thank Dr. Alessio Trivella for his excellent guidance and support throughout the process. His feedback and brainstorming sessions maximized my learning opportunities. I would also like to thank Dr. Breno Alves Beirigo for his constructive feedback and ideas, which helped to improve my thesis. Furthermore, I want to thank my company supervisor, Luc Stoverink, whose willingness to assist, brainstorm about certain ideas, and collaborative approach made my time at the company both enjoyable and educational.

Finally, I want to express my gratitude to my family, my boyfriend, and my friends for being there for me and helping me get through the rougher parts of this thesis process. To you, the reader, I hope you find this work engaging and insightful.

Britt van Oosten
Enschede, August 2024

Management Summary

Company C is a family-owned business that has specialized in the construction of steel structures and steel processing machines since 1970. Over the past fifty years, it has become a leading machine builder in the industry, focusing on designing and developing Computer Numerical Controlled (CNC) machines and software solutions tailored to the steel construction and manufacturing sector. This focus aligns with company C's goal of being a total supplier, offering comprehensive solutions that encompass both hardware and software. To support this vision, company C established Department D, dedicated to creating software that seamlessly integrates design and production processes.

In today's competitive manufacturing environment, industries strive to enhance productivity while reducing costs and lead times. For company C's CNC metal-cutting machines, which cut 2D parts from metal sheets, a key challenge is optimizing the layout of diverse-shaped, mostly irregular 2D parts on available metal sheets to maximize material utilization and minimize waste. This optimization process, known as 2D nesting, requires an algorithm/software that can determine the optimal layout for cutting orders on metal sheets. While some existing algorithms show promise in nesting complex parts, none fully meet the company's specific objectives, restrictions and requirements, leading to inefficiencies and a reliance on manual interventions.
Additionally, the company aims to reduce its dependence on third-party nesting software, which often lacks the flexibility to be customized according to their own or their client's needs. This has created a need for the company to develop its own nesting algorithm, enabling the creation of proprietary nesting software that can, in the future, fully incorporate all necessary restrictions and requirements. However, the company's reliance on external solutions has created a knowledge gap in plate nesting, particularly in developing its own nesting algorithm.
As a result, the following research objective has been defined:

"Research and develop a foundational plate nesting algorithm that generates nesting layouts, optimizes item placements on metal sheets, and minimizes unusable scrap, all within a reasonable time frame while incorporating the company's requirements."

Before developing the plate nesting algorithm, it is essential to understand the general nesting process and identify specific restrictions and requirements that the company wants to incorporate. Additionally, the theory behind the nesting problem itself must be identified and understood. The primary restrictions for the algorithm include part rotation, product spacing, and an efficient crop line to separate the used portion of the sheet from the unused portion. The company also has several other desires, such as incorporating lead-ins and lead-outs, bevel cutting, and zoning of parts on the sheets. However, these features are not essential for the basic functioning of a nesting algorithm and thus left out of scope.

We have developed a plate nesting algorithm designed to automatically generate efficient nesting layouts for cutting operations. The algorithm addresses the 2D Irregular Bin Packing Problem by determining an optimal arrangement of irregularly shaped parts on identical rectangular sheets, aiming to minimize the number of sheets used and reduce material waste. The solution employs a three-phase heuristic algorithm that strategically places and orients each part, according to some allowed fixed rotation angle set, to maximize sheet utilization and distinguishes between solutions that use the same number of sheets. It adheres to constraints such as part orientation, sheet

dimensions, and non-overlapping requirements while also allowing for optional product spacing. The core objective is to achieve the most efficient nesting layout, utilizing the minimum number of sheets to lower material costs and improve operational efficiency.

In the first phase, parts are placed on sheets using the Bottom Left Fill (BLF) heuristic, which prioritizes positioning items at the bottom-leftmost available positions on the sheet without overlapping the sheet's boundaries or other placed items. This allows for the quick generation of initial feasible layouts. Multiple iterations of the BLF are performed using different assignment orderings. The first iteration always follows an assignment order that prioritizes placing larger items first, allowing smaller items to fill the gaps. Subsequent orderings are randomized based on a probability function to escape local optima and explore significantly different packing layouts, considering that the BLF heuristic is quite sensitive to initial assignment ordering.
In the second phase, layouts that use the fewest whole sheets are selected, and an overall utilization efficiency metric, known as F-values, is computed using the following formulas:

$$U_j = \frac{\sum_{j_m=1}^{n^j} s_{j_m}}{L \times W} \quad \text{and} \quad F = \frac{\sum_{j=1}^{N} U_j^2}{N},$$

where U_j denotes the utilization rate of the j_{th} sheet, and s_{j_m} the area of the m_{th} piece within the j_{th} sheet. Layouts with the highest utilization rates are chosen as potential optimal solutions. This phase ensures that only the most efficient found configurations are considered for further refinement.
The third and final phase focuses on refining these selected layouts to minimize scrap. It explores alternative placements and reconfigures parts to enhance space utilization for the least utilized sheets. The K-value, which denotes the fractional number of bins after packing, is used to determine the most efficient layout. The K-value is calculated using the formula:

$$K = N - 1 + P^*.$$

Here P^* is the percentage of utilization corresponding to the least utilized bin after it has been vertically ánd horizontally partitioned from the unused portion of the sheet, and N is the number of used sheets. The layout that achieves the lowest K-value is ultimately chosen as the best layout since the highest remnant sheet area is then obtained for the least utilized bin. This remnant sheet could in practice then be used for future nesting orders.

The algorithm's performance is evaluated using three well-known sets of irregular-shape instances from the literature: the jigsaw puzzle instances JP1 and JP2, where all synthetically created items can fit together perfectly, as well as the irregular strip packing instances that contain more realistic, industrially shaped items. These results are then compared to the state-of-the-art findings by Martinez-Sykora et al. (2017), Zhang et al. (2022), and Wang et al. (2022).

In the JP1 and JP2 instances, our algorithm showed less efficiency than the other methods, especially when dealing with larger average piece sizes, with utilization gaps reaching up to 40% in certain cases. However, performance improved significantly with smaller average piece sizes, reducing the gaps to 17% in JP2 and 6.6% in JP1. Despite these improvements, the methods by Zhang et al. (2022) and Martinez-Sykora et al. (2017) outperformed ours, indicating a need to improve assignment strategies and rotation options.
In the Nest-SB category, our algorithm struggled with complex instances like swim and shirts, which featured intricate shapes and numerous vertices. However, it performed well on simpler instances like albano, dighe2, and fu, with average utilization gaps of 7.3% compared to Zhang et al.'s LocalSearch and 3% compared to Martinez-Sykora et al.'s strategies. These results suggest that while our algorithm excels with simpler shapes, it would benefit from improved rotation and assignment strategies for more complex geometries.
In the Nest-MB category, the algorithm's performance on instances like dighe2, trousers, and albano was hindered by suboptimal assignment orderings. This was highlighted by significant improvements observed in the poly instances, resulting largely due to effective assignment orderings like SCH and PBP, underscoring the importance of proper assignment ordering when working

with relatively smaller sheet dimensions. Although we achieved slightly better results for the mao instance compared to Martinez-Sykora et al. (2017), the LocalSearch by Zhang et al. (2022) still demonstrated superior overall performance.
In the Nest-LB category, our algorithm generally performed well, achieving a total average utilization gap of just 7.2%. We matched Zhang et al.'s results for poly1a and outperformed Martinez-Sykora et al.'s strategies by 1.3%, with an even greater improvement of 2.4% for mao. These results support the idea that, for larger sheets, packing strategies can outweigh the importance of assignment strategies.

To effectively utilize the plate nesting algorithm, we recommend company C:

- Implement assignment strategies like PBP and SCH, which prioritize placing larger items first. This approach, used in our initial iteration(s), has consistently resulted in better nesting layouts, particularly for smaller sheets.
- Integrate local search techniques, such as those proposed by Zhang et al. (2022) and Wang et al. (2022). These techniques refine the initial assignment by strategically swapping and relocating items between sheets, typically shifting items from less utilized sheets to higher utilized sheets. By adopting this method, overall utilization efficiency can be improved and can lead to more efficient nesting solutions.
- Incorporate more rotation flexibility, as suggested by Wang et al. (2022), to improve the arrangement of parts on the sheets, especially for larger or more complex shapes with a lower rectangularity factor.
- Use the additionally equidistant buffer method for larger product spacing distances. The original equidistant method can encounter issues when the spacing exceeds the notches of the item, leading to incorrect product spacings. The buffer method addresses this by simplifying the item's shape and reducing the notches, ensuring accurate product spacing.

While the developed nesting algorithm demonstrates promising performance, it is important to acknowledge certain limitations in the research. First, the current implementation, coded in Python, requires significant time to deliver solutions- often tens of minutes to over an hour- whereas state-of-the-art algorithms achieve results in seconds or minutes. Transitioning to a faster programming language like C++, which is the preferred language for implementation within company C, could potentially reduce our computation times. Second, the existing code may not be optimized for performance. Enhancing the code structure could further improve execution speed. Third, the iterative nature of the repacking algorithm can lead to long computation times, especially with complex instances. Implementing parallel processing or exploring alternative strategies like simulated annealing might accelerate convergence to optimal packing arrangements.

Apart from computation time efficiency, some iterations involving complex shapes with interlocking concavities or jigsaw-like features showed overlaps, which can be attributed to limitations in the current No Fit Polygon (NFP) generation method. To address this, adopting the Start Points method from Burke et al. (2006) could improve accuracy by considering all feasible touching positions of the moving polygon and avoiding overlaps through iterative checks. Additionally, the algorithm's current handling of piece rotation affects performance, particularly in cases where free rotation could improve packing efficiency. Enhancing rotation capabilities could help reduce material waste. Furthermore, the current approach's focus on repacking only the least utilized sheet to improve remnant sheet area may not always yield optimal results. Expanding the repacking strategy to include multiple sheets could increase overall packing efficiency and better utilize remnant sheets. Lastly, the algorithm assumes uniform sheet sizes, which is unrealistic in industrial contexts where varying sheet sizes and remnants are common. Incorporating support for different sheet sizes would better reflect industrial practices and improve nesting flexibility.

The developed algorithm provides a strong foundational tool for company C to build upon, It includes key features such as the option for product spacing between items and the ability to select rotations from a predefined set. It is well-suited to the company's typical use of large sheets,

often several meters in size, which often involves nesting smaller items alongside occasional larger ones. Its strong performance in the Nest-LB category, and in instances where there is a balance between larger and smaller items, suggests its potential effectiveness in company C's operational environment.

Additionally, the irregular strip packing instances closely resemble the items that company C and its clients cut from metal sheets. For example, the jakobs1 instance is particularly relevant for steel construction tasks, while other instances like albano, trousers, han, mao, swim, shirts, shapes, and jakobs correspond well to scenarios in (mechanical) engineering. Given our average utilization gap of just 7.2% for the Nest-LB instances compared to state-of-the-art methods, the algorithm could serve as a robust starting point for the company, offering ample opportunity for further refinement and customization.

Contents

1 Introduction **1**
1.1 Introduction to company C . . . 1
1.2 Problem identification . . . 1
1.3 Motivation and objective . . . 3
1.4 Research scope . . . 4
1.5 Research questions . . . 4
1.5.1 Current situation . . . 5
1.5.2 Literature research . . . 5
1.5.3 Solution design . . . 6
1.5.4 Experiment design . . . 6
1.5.5 Analysis of the results . . . 6
1.5.6 Implementation plan . . . 7
1.6 Deliverables . . . 7
1.7 Conclusion . . . 7

2 Current situation **8**
2.1 Nesting process overview . . . 8
2.1.1 CAD software . . . 8
2.1.2 CAM software . . . 8
2.1.3 G-code . . . 8
2.1.4 Post-processor . . . 8
2.1.5 General steps in nesting process . . . 9
2.2 Nesting process of company C's machines . . . 10
2.2.1 Workings of company C's flatbed and pass-through machine . . . 10
2.2.2 Explanation of nesting process on company C's flatbed machine . . . 11
2.3 Key considerations for nesting on CNC machines . . . 14
2.3.1 Rotation . . . 14
2.3.2 Product spacing . . . 14
2.3.3 Crop line . . . 17
2.3.4 Restricting parts to specific zones of the sheets . . . 17
2.3.5 Possible additional considerations . . . 18
2.4 Some existing nesting algorithms and software . . . 18
2.4.1 Software used by company C . . . 18
2.4.2 Overview of other nesting software . . . 20
2.5 Conclusion . . . 23

3 Literature Research **26**
3.1 How is the plate nesting problem known in literature? . . . 26
3.2 Geometry overview . . . 27
3.2.1 Pixel/Raster method . . . 28
3.2.2 Direct trigonometry . . . 28
3.2.3 No Fit Polygons . . . 29
3.2.4 Phi-function . . . 30
3.3 Selection heuristics for offline BPP . . . 30

3.4 Placement heuristics . . . 31
3.5 Solution methods/algorithms . . . 32
3.5.1 Exact methods . . . 32
3.5.2 Heuristics . . . 33
3.5.3 Meta-heuristics . . . 33
3.6 Method selection . . . 34
3.7 Research gap . . . 35
3.8 Conclusion . . . 36

4 Solution design **37**
4.1 Formal problem statement . . . 37
4.2 The developed algorithm . . . 39
4.2.1 Data input phase . . . 39
4.2.2 Assignment strategy . . . 41
4.2.3 Packing strategy . . . 42
4.2.4 Optimization phase . . . 44
4.3 Conclusion . . . 46

5 Experiment design **47**
5.1 Data instances . . . 47
5.1.1 Jigsaw puzzle instances (JP1 and JP2) . . . 47
5.1.2 Irregular strip packing instances . . . 47
5.2 Parameterization . . . 48
5.2.1 Iterations for initial packing solutions . . . 48
5.2.2 Permutations in the repacking phase . . . 49
5.2.3 Rotation handling . . . 49
5.3 Experiment execution . . . 49
5.4 Conclusion . . . 50

6 Analysis of the results **51**
6.1 Results of the JP1 instances . . . 51
6.2 Results of the JP2 instances . . . 53
6.3 Results of the strip packing instances . . . 54
6.3.1 Results of the Nest-SB instances . . . 54
6.3.2 Results of the Nest-MB instances . . . 56
6.3.3 Results of the Nest-LB instances . . . 57
6.4 Analysis of computation times . . . 58
6.5 Overlapping issues . . . 59
6.6 Offset generation results and handling complex polygons . . . 60
6.7 Conclusion . . . 61

7 Implementation, Conclusions, and Future directions **63**
7.1 Implementation . . . 63
7.2 Conclusion and recommendations . . . 63
7.3 Discussion . . . 65
7.4 Further research . . . 66

Appendices **72**
A Problem cluster . . . 72
B Overview of nesting software . . . 73
B.1 SVGnest . . . 73
B.2 Deepnest . . . 74
B.3 Nest&Cut . . . 76
B.4 Inventor nesting . . . 77
C Benchmark instances explained . . . 78
C.1 JP1 benchmark test instances . . . 78
C.2 JP2 benchmark test instances . . . 79

C.3 Irregular strip packing benchmark test instances 82
D Pseudo-code of the equidistant method 83
E Pseudo-code of the simplified equidistant method 84
F Pseudo-code of the randomisation and rotation for the assignment order 85
G Pseudo-code of the utilisation efficiency metric 87
H Pseudo-codes for the repacking strategy 88
I Pseudo-code of the K-value 92
J Nesting results for the JP1 instances 93
K Nesting results for the JP2 instances 106
L Nesting results for the Nest-SB instances 117
M Nesting results for the Nest-MB instances 131
N Nesting results for the Nest-LB instances 140
O Nesting results of product spaced items 147

List of Figures

1.1 Possible layout to minimize scrap of the squared metal sheet (SVGnest, 2024a). . . 2
1.2 Minimization of software program. 3

2.1 CAD to cut (HyperTherm, 2024). 9
2.2 Illustrations comparing a flatbed machine (left) and a pass-through machine (right). 11
2.3 Nesting process of flatbed machine. 12
2.4 Inside and outside spacing area of an item, depicted in grey, with the orange area representing space for additional part-in-part nesting. 14
2.5 Nesting with common cutting: items sharing common borders (Jetcam, 2024). . . 15
2.6 Examples of common lead-ins and lead-outs. 16
2.7 Comparison between pre-pierce on the edge and in the center, showing two available options for positioning the pierce point. 16
2.8 Representation of a bevel cut on the right side of the quadrilateral, showcasing an angled edge. 17
2.9 Various crop line options illustrated, including vertical, dynamic, and horizontal configurations, facilitating efficient material usage. 17
2.10 Illustration of tabbing on a 5 mm thick sheet. 18
2.11 Comparison between nesting layouts generated by the software and manually adjusted layout. 19
2.12 Illustration of the DXF (Drawing Exchange Format) files of the two examples, used in the future for comparison between nesting software. 20
2.13 Nesting layout of both examples after 5 minutes using company C's software, along with associated statistics. 21
2.14 The Deepnest output for the drawbot example employing the bounding box optimization, showcasing the nesting layout generated by the software. 22
2.15 Nesting layout of both examples after 5 minutes using Nest&Cut software, along with associated statistics. 23
2.16 The Inventor Nesting output for the drawbot example, obtained after a five-minute run with the gaps set to 0.01 mm. 25

3.1 $0-1$ Representation for irregular pieces. 28
3.2 NFP method representation for convex polygons. 29
3.3 Example of phi-function. 30
3.4 Placement heuristics. 31
3.5 Two solution layouts featuring identical items arranged on the same sheet. 36

4.1 The reference point r_{P_b} of piece P_b is defined as the bottom-left corner of the piece when rotated to a specific angle (Wang et al., 2022). 37
4.2 Flowchart of the developed algorithm for 2D irregular bin packing. 39
4.3 Illustration of equidistant offset, $A_1B_1C_1D_1E_1$, of a polygon, $ABCDE$, taken from Wang et al. (2022). 40
4.4 Impact of different decay rates on packing efficiency. 42

5.1 Sensitivity analysis of F-values to iteration counts: balancing solution quality and computation time. 48

6.1 Comparison of Wang et al.'s results (LS-PBP with and without slit) to other literature, including LocalSearch-WLFD from Zhang et al (2022), and LS2-PBP from Martinez-Sykora et al. (2017). The bold values denote the best solutions (Wang et al., 2022). . . . 52
6.2 Results for Nest-MB instances from Wang et al. (2022). . . . 57
6.3 The start point generation process (E. K. Burke et al., 2007). . . . 59
6.4 NFP for shapes with interlocking concavities, jigsaw pieces, and holes. . . . 60
6.5 Offset examples for complex shapes using the original offset method. . . . 61
6.6 Offset generation using the buffer method for the various shapes. . . . 61

7.1 Repacking multiple sheets for better nesting solutions. . . . 66
2 Problem cluster displaying the interconnected causes, core problems, and consequences stemming from the action problem. . . . 72
3 NFP and IFP in SVGnest. . . . 73
4 Configuration parameters in SVGnest, from (SVGnest, 2024b), highlighting adjustable settings for nesting optimization. . . . 74
5 SVGnest output of the demo from (SVGnest, 2024b), after 5 minutes, showcasing the nesting layout generated by the software. . . . 74
6 The different optimization types in Deepnest, providing users with various strategies for nesting optimization. . . . 75
7 Several parts haven't been nested in Deepnest, indicating that one sheet isn't sufficient for the nesting process. . . . 76
8 The cutting order of the second example, resulting from Nest&Cut, after running the nest for 5 minutes. . . . 76
9 Example where not all items are nested on the available sheets in Nest&Cut. . . . 77
10 In Inventor Nesting you can make manual adjustments to each item in the nest properties tab. . . . 78
11 Example of irregular generated instance (Terashima-Marín et al., 2010). . . . 78
12 Description of the convex problem instances (López-Camacho et al., 2013). . . . 79
13 Irregular characteristics of the JP1 benchmark instances. . . . 79
14 The solution of two JP1 instances of different types, where top solution is instance of class L and bottom solution is instance of class O (Martinez-Sykora et al., 2017). 80
15 Comparison of concavity degree and convex hull of non-convex piece. . . . 81
16 Characteristics of the JP2 instances. . . . 81
17 Irregular strip packing instances (Martinez-Sykora et al., 2017). . . . 82
18 Trigonometry applied to the ABCDE piece. . . . 83
19 Nesting solution layout of type A of JP1. . . . 93
20 Nesting solution layout of type B of JP1. . . . 94
21 Nesting solution layout of type C of JP1. . . . 95
22 Nesting solution layout of type D of JP1. . . . 95
23 Nesting solution layout of type E of JP1. . . . 96
24 Nesting solution layout of type F of JP1. . . . 96
25 Nesting solution layout of type G of JP1. . . . 97
26 Nesting solution layout of type H of JP1. . . . 98
27 Nesting solution layout of type I of JP1. . . . 99
28 Nesting solution layout of type J of JP1. . . . 99
29 Nesting solution layout of type K of JP1. . . . 100
30 Nesting solution layout of type L of JP1. . . . 100
31 Nesting solution layout of type M of JP1. . . . 101
32 Nesting solution layout of type N of JP1. . . . 101
33 Nesting solution layout of type O of JP1. . . . 102
34 Nesting solution layout of type P of JP1. . . . 103
35 Nesting solution layout of type Q of JP1. . . . 104
36 Nesting solution layout of type R of JP1. . . . 105
37 Nesting solution layout of type A of JP2. . . . 106
38 Nesting solution layout of type B of JP2. . . . 107

39 Nesting solution layout of type C of JP2. . . . 108
40 Nesting solution layout of type F of JP2. . . . 108
41 Nesting solution layout of type H of JP2. . . . 109
42 Nesting solution layout of type L of JP2. . . . 109
43 Nesting solution layout of type M of JP2. . . . 110
44 Nesting solution layout of type O of JP2. . . . 111
45 Nesting solution layout of type S of JP2. . . . 111
46 Nesting solution layout of type T of JP2. . . . 112
47 Nesting solution layout of type U of JP2. . . . 113
48 Nesting solution layout of type V of JP2. . . . 113
49 Nesting solution layout of type W of JP2. . . . 114
50 Nesting solution layout of type X of JP2. . . . 114
51 Nesting solution layout of type Y of JP2. . . . 115
52 Nesting solution layout of type Z of JP2. . . . 116
53 Nesting solution layout of albano of the Nest-SB instances. . . . 117
54 Nesting solution layout of trousers of the Nest-SB instances. . . . 118
55 Nesting solution layout of shapes0 of the Nest-SB instances. . . . 119
56 Nesting solution layout of shapes1 of the Nest-SB instances. . . . 120
57 Nesting solution layout of shirts of the Nest-SB instances. . . . 121
58 Nesting solution layout of dighe2 of the Nest-SB instances. . . . 122
59 Nesting solution layout of dighe1 of the Nest-SB instances. . . . 122
60 Nesting solution layout of fu of the Nest-SB instances. . . . 123
61 Nesting solution layout of han of the Nest-SB instances. . . . 123
62 Nesting solution layout of jakobs1 of the Nest-SB instances. . . . 124
63 Nesting solution layout of jakobs2 of the Nest-SB instances. . . . 125
64 Nesting solution layout of mao of the Nest-SB instances. . . . 125
65 Nesting solution layout of poly1a of the Nest-SB instances. . . . 126
66 Nesting solution layout of poly2b of the Nest-SB instances. . . . 126
67 Nesting solution layout of poly3b of the Nest-SB instances. . . . 127
68 Nesting solution layout of poly4b of the Nest-SB instance. . . . 128
69 Nesting solution layout of poly5b of the Nest-SB instances. . . . 129
70 Nesting solution layout of swim of the Nest-SB instances. . . . 130
71 Nesting solution layout of albano of the Nest-MB instances. . . . 131
72 Nesting solution layout of trousers of the Nest-MB instances. . . . 132
73 Nesting solution layout of shapes0 of the Nest-MB instances. . . . 132
74 Nesting solution layout of shapes1 of the Nest-MB instances. . . . 133
75 Nesting solution layout of shirts of the Nest-MB instances. . . . 134
76 Nesting solution layout of dighe2 of the Nest-MB instances. . . . 134
77 Nesting solution layout of dighe1 of the Nest-MB instances. . . . 135
78 Nesting solution layout of fu of the Nest-MB instances. . . . 135
79 Nesting solution layout of han of the Nest-MB instances. . . . 135
80 Nesting solution layout of jakobs1 of the Nest-MB instances. . . . 136
81 Nesting solution layout of jakobs2 of the Nest-MB instances. . . . 136
82 Nesting solution layout of mao of the Nest-MB instances. . . . 137
83 Nesting solution layout of poly1a of the Nest-MB instances. . . . 137
84 Nesting solution layout of poly2b of the Nest-MB instances. . . . 137
85 Nesting solution layout of poly3b of the Nest-MB instances. . . . 138
86 Nesting solution layout of poly4b of the Nest-MB instances. . . . 138
87 Nesting solution layout of poly5b of the Nest-MB instances. . . . 139
88 Nesting solution layout of swim of the Nest-MB instances. . . . 139
89 Nesting solution layout of albano of the Nest-LB instances. . . . 140
90 Nesting solution layout of trousers of the Nest-LB instances. . . . 140
91 Nesting solution layout of shapes0 of the Nest-LB instances. . . . 141
92 Nesting solution layout of shapes1 of the Nest-LB instances. . . . 141
93 Nesting solution layout of shirts of the Nest-LB instances. . . . 142
94 Nesting solution layout of dighe2 of the Nest-LB instances. . . . 142

95 Nesting solution layout of dighe1 of the Nest-LB instances. 142
96 Nesting solution layout of fu of the Nest-LB instances. 143
97 Nesting solution layout of han of the Nest-LB instances. 143
98 Nesting solution layout of jakobs1 of the Nest-LB instances. 143
99 Nesting solution layout of jakobs2 of the Nest-LB instances. 143
100 Nesting solution layout of mao of the Nest-LB instances. 144
101 Nesting solution layout of poly1a of the Nest-LB instances. 144
102 Nesting solution layout of poly2b of the Nest-LB instances. 144
103 Nesting solution layout of poly3b of the Nest-LB instances. 144
104 Nesting solution layout of poly4b of the Nest-LB instances. 145
105 Nesting solution layout of poly5b of the Nest-LB instances. 145
106 Nesting solution layout of swim of the Nest-LB instances. 146
107 Nesting solution layout of albano of the Nest-LB instances with product spacing. . 147
108 Nesting solution layout of trousers of the Nest-LB instances with product spacing. 147
109 Nesting solution layout of shapes0 of the Nest-LB instances with product spacing. 148
110 Nesting solution layout of shapes1 of the Nest-LB instances with product spacing. 149
111 Nesting solution layout of shirts of the Nest-LB instances with product spacing. . . 150
112 Nesting solution layout of dighe2 of the Nest-LB instances with product spacing. . 150
113 Nesting solution layout of dighe1 of the Nest-LB instances with product spacing. . 151
114 Nesting solution layout of fu of the Nest-LB instances with product spacing. 152
115 Nesting solution layout of han of the Nest-LB instances with product spacing. . . . 152
116 Nesting solution layout of jakobs1 of the Nest-LB instances with product spacing. . 153
117 Nesting solution layout of jakobs2 of the Nest-LB instances with product spacing. . 154
118 Nesting solution layout of mao of the Nest-LB instances with product spacing. . . 154
119 Nesting solution layout of poly1a of the Nest-LB instances with product spacing. . 155
120 Nesting solution layout of poly2b of the Nest-LB instances with product spacing. . 156
121 Nesting solution layout of poly3b of the Nest-LB instances with product spacing. . 157
122 Nesting solution layout of poly4b of the Nest-LB instances with product spacing. . 158
123 Nesting solution layout of swim of the Nest-LB instances with product spacing. . . 159

List of Tables

2.1 Comparison of Nesting Software Features 24

3.1 Comparison between the 2DIBPP solution methods. 35

6.1 Results for JP1 instances. 52
6.2 Results for JP2 instances. 54
6.3 Results for nesting instances with small bins. 55
6.4 Results for nesting instances with medium bins. 56
6.5 Results for nesting instances with large bins. 58

1 Description of the JP2 benchmark test instances (López-Camacho et al., 2014). . . 80

Glossary

BF:	Best Fit
BFD:	Best Fit Decreasing
BL:	Bottom-Left
BLF:	Bottom Left Fill
BPGD:	Bin Packing with Greedy Decisions
CA:	Constructive Approach
CAA:	Constructive Approach (Minimum Area)
CAD:	Computer-Aided Design
CAM:	Computer-Aided Manufacturing
CNC:	Computer Numerical Control
CO:	Combinatorial Optimization
C&P:	Cutting and Packing
DCH:	Direct Construction Heuristic
DJD:	Djang and Finch
DXF:	Drawing Exchange Format
FF:	First Fit
FFD:	First Fit Decreasing
FFI:	First Fit Increasing
GA:	Genetic Algorithm
GLS:	Guided Local Search
GRASP:	Greedy Randomized Adaptive Search Procedure
ILS:	Iterated Local Search
IFP:	Inner Fit Polygon
IFR:	Inner Fit Rectangle
LP:	Linear programming
MIP:	Mixed-integer (linear) programming
ML:	Minimum Length
MU:	Maximum Utilization
NC:	Numerical Control
NF:	Next Fit
NFP:	No Fit Polygon
NFPAssistant:	A program that pre-computes all No Fit Polygon configurations and stores them in a list for quick access during packing
NKF:	Next-K-Fit
PBP:	Partial Bin Packing
PSO:	Particle Swarm Optimization
SA:	Simulated Annealing
SBSBPP:	Single Bin Size Bin Packing Problem
SCH:	Simple Construction Heuristic
SVG:	Scalable Vector Graphics
TPS:	Two Phases Strategy
TS:	Tabu Search
VNS:	Variable Neighbourhood Search
WF:	Worst Fit
2DICSP:	Two-Dimensional Irregular Cutting Stock Problem
2DIBPP:	Two-Dimensional Irregular Bin Packing Problem
2DISPP:	Two-Dimensional Irregular Strip Packing Problem

List of Symbols

Symbol	Description
$B = \{b_j \| j = 1, \cdots, N\}$	Set of sheets used for packing
d_1	Slit distance between any two pieces
d_2	Slit distance between a piece and the sheet's edge
F	Utilization efficiency metric
$h_{ab}(\mathcal{P}_j, V_j, R_j, b_j)$	Overlap between two pieces
K	Fractional number of bins after packing
L	Length of the sheet
m_d	Maximum length or width across all pieces in their initial orientation
N	Total number of sheets used for packing
n^j	Quantity of pieces packed into the j_{th} sheet
p_0	Point of a piece P_i
$PD(P_a, P_b)$	Penetration depth between two items
$PD(b_j, P_b)$	Penetration depth between an item and the sheet's edges
P_i	Piece i to be packed
$\mathcal{P} = \{P_i \| i = 1, \cdots, n\}$	Set of pieces to be packed
P^*	Percentage of utilization corresponding to the least utilized sheet after it has been vertically and horizontally partitioned
r_{P_i}	Reference point of piece i
$R_{\mathcal{P}} = \{r_{P_1}, r_{P_2}, \cdots, r_{P_n}\}$	Reference point set of all pieces to be packed
s_i	Area of piece i
U_j	Utilization rate of the j_{th} sheet
v	Penetration vector
v_t	Translation vector
ϑ_i	Set of allowable rotation angles
θ_i	Rotation angle of piece i
W	Width/height of the sheet
$k_a(\mathcal{P}_j, V_j, R_j, b_j)$	Overlap between a piece and the sheet's edges
$\oplus$	Translational operator

Chapter 1: Introduction

This research will develop a plate nesting algorithm at a steel processing company, henceforth referred to as company C. This algorithm will aim to efficiently cut parts from metal sheets. The chapter will start with a small introduction to the company in section 1.1. Section 1.2 describes the problem and in section 1.3 the motivation and objective of the research will be given. After this, section 1.4 will discuss the scope of the research, and in section 1.5 the research questions and methodology used to answer the research questions will be stated. Finally, section 1.6 offers the deliverables, and in section 1.7, a small summary is given.

1.1 Introduction to company C

Company C is a family business that has been active since 1970 in the construction of steel structures and steel processing machines. Over the past fifty years, the company has evolved into a prominent machine builder for steel processing machines. Their core activities include building machines, developing software, and providing high-quality service. Company C, until recently, was comprised of two divisions, each excelling in its unique strengths and specialties. For over fifty years, one of those divisions has been dedicated to designing and developing Computer Numerical Controlled (CNC) machines and solutions tailored to the steel construction and manufacturing industry. The other division, which was sold last year, focused on designing, producing, and delivering high-quality steel projects.
The goal of the company is to be a total supplier, which means they not only want to deliver machines but also provide software solutions. These software solutions are engineered at department D of company C and were established in 1990. The department is designed to disrupt and innovate the market by minimizing the need for user interaction in creating a fully seamless data chain, enabling information to effortlessly transition from model (CAD) to machine (CAM). Department D is aiming to revolutionize the steel processing industry through its innovative, full cloud-based software solution. With the founding of Department D, data can now be used to flow from model to machine and can integrate seamlessly with other important systems. The assignment of this thesis will be performed at department D.

1.2 Problem identification

Company C focuses on the design and development of high-precision sheet metal working machines, with which companies in the manufacturing industry can produce complete parts. With processes such as marking, drilling, 3D plasma bevel cutting, and oxyfuel cutting, the CNC sheet metal working machines include numerous functions to optimize the workflow and reduce costs per part. Within company C's portfolio, two primary CNC types of machines can be useful for this thesis, the flatbed machine and the pass-through machine. Chapter 2 will go into more detail to explain how these types of machines work and what they are used for. For now it is sufficient to know that these types of machines are used to cut items on metal sheets.

As mentioned before, the goal of company C is to be a total supplier, wanting to not only deliver the machines but also provide software for these machines. This of course is also the goal for the flatbed and pass-through machines. In the current competitive manufacturing environment, industries are striving to enhance productivity while simultaneously reducing costs and lead times. For the flatbed and pass-through machines, that cut 2D parts from metal sheets, a key challenge is to optimize the layout of diverse-shaped 2D parts on available sheets to maximize material utilization and minimize waste, all within a reasonable time frame[1]. For this, software is needed that,

[1]This reasonable time frame specifically relates to the duration required for devising a layout solution, rather than the time needed for cutting all items on the machines.

before cutting, determines the optimal layout for an order of items that need to be cut on metal sheets. This process of arranging cutting patterns strategically to, for example, reduce the amount of raw material wastage is called nesting. In this case, we are talking about 2D nesting, since the nesting is done on metal sheets.
For explanation purposes, consider a 2D metal sheet, and let's assume this metal sheet is square. From the metal sheet we now want to cut the letters from the following sentence: 'Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit'. The goal of nesting is to fit and cut all these letters into the square, utilizing the least amount of material. In cases where a single square proves insufficient, the aim then is also to minimize the overall number of squares employed. A possible layout that minimizes the scrap percentage of the squared metal sheet can be found in Figure 1.1 (SVGnest, 2024a).

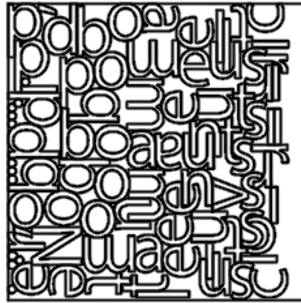


FIGURE 1.1: Possible layout to minimize scrap of the squared metal sheet (SVGnest, 2024a).

If all the items were rectangular, the arrangement task would be simpler, since we are then dealing with a 2D knapsack problem. However, in reality, including the scenarios faced by company C, there is a multitude of irregular parts requiring nesting. From Martinez-Martinez et al. (2021) and Xie et al. (2007) it follows that this challenge involves intricate geometry, adherence to constraints like non-overlapping and containment, and computationally intense processes. As the 2D knapsack problem is already classified as NP-hard, this problem with irregular parts is also classified as NP-hard, making it impractical and costly to seek an optimal solution due to the vast number of potential solutions. While nesting processes have been automated and numerous tools developed, many industries still rely on manual methods. Material wastage depends on the operator's expertise, leading to potential inefficiencies and higher costs. Despite some algorithms showing promise in nesting complex parts, none meet company C's restrictions and requirements, highlighting a lack of practical industry algorithms hindering effective automatic nesting.

To give an example of such an algorithm that does not meet company C's restrictions and requirements, let's look at the following software system. This software in which a nesting algorithm is incorporated can only be applied to flatbed machines to have higher material optimization and cutting efficiency. The explanation for this can be found in Chapter 2.
The software system, that can be implemented on flatbed machines, aims to reduce scrap percentages by minimizing the horizontal width of each item to be cut in comparison to the metal sheet. This strategy is visually represented in Figure 1.2. Looking at the red beams, the software places them diagonally from the bottom left corner, ensuring that the rightmost corner of the beam is in close proximity to the left side of the metal sheet. This aligns with the goal of minimizing the horizontal width of each cutting item. However, this approach can lead to the creation of unusable space, as evident from the closed-in triangle in Figure 1.2. This triangular area remains unused and is too small to serve as a remnant piece in future operations. Consequently, while pursuing objectives such as those facilitated by this software system, there is a risk of suboptimal utilization of the metal sheet.
This existing nesting algorithm, among others, although capable, faces limitations that hinder the full potential of a flatbed machine and yield sub-optimal results in certain situations.[2] Some examples include an inadequate objective function, long running times to compute a solution, and a

[2]These limitations will be discussed in Chapter 2.

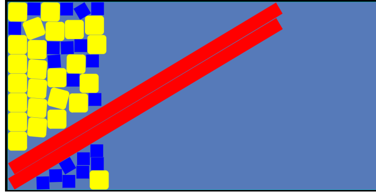


FIGURE 1.2: Minimization of software program.

lack of customization or adaptability in the program/algorithm.
This is not strange, since the algorithm developers do not know the machines and processes of company C. This prompts many of company C's customers to resort to manual nesting or make manual adjustments after automated nesting, using some software. To address this, Department D aims to develop a solution that streamlines the process without requiring manual interventions. The new algorithm's objectives include minimizing scrap percentages, considering practical constraints like reusable scrap plates, and delivering solutions within a reasonable time frame.
All this gives us the following action problem:

"There is a lack of an existing plate nesting strategy for the flatbed machine that results in effective solutions for real-world use cases, including the required restrictions from company C."

1.3 Motivation and objective

The preceding section outlined the action problem and its underlying causes. Appendix A shows the problem cluster, illustrating the progression from the action problem to core issues and their consequences. The core problems identified are as follows:

- The existing algorithm's objective function does not align with the desired goals of company C.
- Existing algorithms cannot accommodate specific restrictions imposed by company C and their customers.
- Existing algorithms are too slow/exhibit sluggish performance.

These core problems yield the following consequence:

- If company C fails to address/react to these concerns by providing nesting software, another company might seize the opportunity, potentially leading to a loss of business and/or dissatisfaction among company C's customers.

This core consequence defines the problem statement as follows:

"Company C faces the risk of losing business and/or experiencing customer dissatisfaction if it does not respond to concerns regarding the lack of nesting software for machine X. Failure to provide such software may allow competitors to exploit the opportunity, potentially resulting in a loss of market share and discontent among company C's customer base."

To enhance the current nesting process of machine X, research and proposals for a nesting algorithm are necessary. This algorithm will be a start for the company to develop its own nesting algorithm that can generate nesting solutions that minimize scrap and also the number of metal sheets used while requiring minimal manual interventions of work preparators. The solution (nesting) should, while generating the nesting, also adhere to company standards for the steel industry. Additionally, it should accommodate company restrictions[3], including industrial requirements and time constraints. The end-goal of the company is to minimize manual interventions and achieve optimal positioning and cutting of items, thereby reducing costs within a reasonable time frame. The goal of this thesis is to provide the foundations for this end-goal by developing an initial

[3]See Chapter 2.

working nesting algorithm, that the company can build upon. Hence, the research objective is defined as follows:

"Research and develop a foundational plate nesting algorithm that generates nesting layouts, optimizes item placements on metal sheets, and minimizes unusable scrap, all within a reasonable time frame while incorporating the company's requirements."

1.4 Research scope

This research focuses on the development of an efficient algorithm tailored to meet the requirements of company C for the flatbed machine. Specifically, it targets cutting on a fixed metal sheet rather than on a pass-through sheet. The following considerations outline the scope of research:

- **Predefined items for cutting**; It is assumed that the items to be cut are known in advance, following some planning program used by company C or its customers.
- **Plate sequence planning;** The sequence in which sheets are to be used for cutting is known. For example, prioritizing cutting items on remnant sheets before using full sheets.
- **Distribution of items**; The known items to be cut can be distributed across all available metal sheets.
- **Use of whole metal sheets**; The algorithm will operate under the assumption that only whole metal sheets are used, thus without the use of remnant sheets.

The primary goal is to develop an algorithm capable of efficiently arranging items on metal sheets to minimize unusable scrap within a reasonable time frame while adhering to company C's restrictions and requirements.
To achieve this goal, the research will involve:

- **Determining a reasonable time frame**; We need to investigate existing plate nesting algorithms to establish what can constitute a reasonable time frame for efficient nesting.
- **Understanding company requirements and restrictions**; Interviews need to be conducted with company C to ascertain their specific requirements and restrictions for the algorithm.
- **Analyzing existing algorithms**; We need to investigate the limitations of existing plate nesting algorithms and identify which configuration options are most commonly utilized in such algorithms/software.

It is important to note that the implementation plan, including the execution of mobilizing it on the flatbed machine and performing practical tests, is beyond the scope of this research. However, while developing the algorithm the needed CAM restrictions will be taken into account. Additionally, it is important to mention that, most probably, no better-working nesting algorithm will be found, compared to other existing nesting algorithms/software. The goal of this thesis is to provide a nesting algorithm that can serve as a foundation for a better working algorithm in the future.

In summary, the research will focus exclusively on the flatbed machine, with predetermined items for cutting, and will acquire knowledge through investigating existing plate nesting algorithms, determining a reasonable time frame, and aligning with company C's restrictions/objectives. The end goal is to develop an algorithm that serves as a foundation for future algorithm development and that optimally nests items on metal sheets, minimizing scrap material and the number of sheets used, within the specified constraints.

1.5 Research questions

The research objective in section 1.3 leads to the following main research question:

"How can a foundational plate nesting algorithm be developed to optimize item placement on metal sheets, minimize unusable scrap and the number of sheets used, while meeting the company's requirements within a reasonable computational time frame?"

The main research question leads to the formulation of several sub-questions, organized into six phases: current situation, literature research, solution design, experiment design, analysis of results, and implementation plan. Each phase begins with the presentation of the research question followed by the sub-questions and research design.

1.5.1 Current situation

Question 1. *What is the process flow of nesting and how is it currently executed?*

1.1 What are the general steps involved in a nesting process on CNC machines?

1.2 How does the current nesting process of company C's flatbed machine work?

1.3 What are the typical requirements and constraints for nesting on CNC machines?

1.4 What nesting algorithms or software solutions, if any, are commonly utilized in the metal sheet industry?

1.5 What specific requirements, constraints, configuration parameters, and objectives are associated with those nesting algorithms or software?

1.6 How do existing plate nesting algorithms perform in terms of efficiency and usability?

1.7 What are the common challenges encountered in plate nesting within manufacturing industries?

Before initiating improvements to the current state, it is essential to comprehend the existing conditions thoroughly. Examining various nesting software and algorithms can also inspire ideas for configuration parameters in our own nesting algorithm. This is addressed by the first research question. Chapter 2 provides an overview of the current nesting process, along with existing nesting algorithms and software.

1.5.2 Literature research

Question 2. *Which methods suggested in the literature are most applicable to solving the company's problem?*

2.1 How is the nesting problem of company C known in the literature?

2.2 What nesting methods have been proposed in the literature?

2.3 Which nesting method is best suited to address the company's problem?

2.4 What research gap exists between the selected nesting method and the problem faced by company C?

2.5 How can the research gap between the chosen nesting method and the problem of company C be addressed effectively?

After gaining insights into the current state, we delve into existing literature to identify relevant methods applicable to company C's plate nesting problem. With this information, we can conclude if we can use an existing method, or a mix of existing methods, to solve the nesting problem or if we need to develop a new one. Chapter 3 describes the literature research that applies to company C's case.

1.5.3 Solution design

Question 3. *What should be the design of the algorithm?*

3.1 What input does the algorithm need; what properties does it have?

3.2 How can the data be initialized?

3.3 What is the primary objective of the algorithm?

3.4 Which techniques or heuristics will be employed to optimize item placement and minimize scrap during the nesting process?

3.5 What is the structural outline of the algorithm?

3.6 How will the specific requirements and constraints of company C be incorporated into the algorithm development process?

3.7 What is the output of the algorithm; what properties should it certainly have (strict requirements)?

3.8 How can the best layout of the items on the metal sheets be determined?

Chapter 4 describes the plate nesting algorithm, detailing its problem-solving approach, functionality, and data requirements.

1.5.4 Experiment design

Question 4. *What does the experimental design look like?*

4.1 How can we prove that the developed algorithm works?

4.2 How can the test instances be created?

4.3 What criteria will be used to evaluate the efficiency and effectiveness of the developed algorithm?

Chapter 5 describes the experimental design implemented to assess the performance of the plate nesting algorithm.

1.5.5 Analysis of the results

Question 5. *How does the developed algorithm perform?*

5.1 What are the outcomes observed from implementing the developed plate nesting algorithm?

5.2 How do the experimental results compare with those obtained from existing solutions?

5.3 What insights can be derived from analyzing the results of the developed algorithm and its comparison with other solutions?

5.4 What is the overall quality of the solutions generated by the algorithm?

5.5 What are the key strengths and limitations observed in the performance of the developed algorithm?

Chapter 6 delves into the analysis of the experimental results obtained from the plate nesting algorithm, providing insights into its performance and effectiveness.

1.5.6 Implementation plan

Question 6. *What are the steps for implementing the plate nesting algorithm in practice?*

The final chapter, Chapter 7, will answer question 6. It furthermore will summarize the research findings and provide recommendations and conclusions. Additionally, limitations and potential areas for future research are discussed.

1.6 Deliverables

This project aims to research and develop a plate nesting algorithm capable of automatically generating nesting proposals for a given list of items to be nested on certain metal sheets. Additionally, this thesis will be a deliverable, which includes an implementation plan outlining how to implement the algorithm.

1.7 Conclusion

In conclusion, Chapter 1 has provided an overview of the research context and objectives. Company C's background and goals were outlined, emphasizing the need for an efficient plate nesting algorithm to optimize the utilization of metal sheets in their manufacturing processes. The problem statement highlighted the current challenges faced by company C, including limitations in existing nesting algorithms and the potential risks of market competition. Motivated by these concerns, the research objective and scope were defined, focusing on developing a tailored algorithm for machine X while considering specific requirements and constraints. Subsequently, the main research question and sub-questions were formulated, and structured into six phases to guide the research process. Lastly, the deliverables of the project were outlined, emphasizing the development of the plate nesting algorithm and an implementation plan to support company C's objectives.

Chapter 2: Current situation

This chapter describes the current situation at company C, by answering the first research question: *"What is the process flow of nesting and how is it currently executed?"*. In section 2.1, we describe the general steps involved in the nesting process. Section 2.2 describes the nesting process at company C's flatbed machine. Section 2.3 describes key considerations for nesting on CNC machines, such as the flatbed machine. Section 2.4 describes some nesting algorithms and software that, following engineers' and software developers' opinions, are currently among the best on the market. The chapter will finish with a small conclusion/summary.

2.1 Nesting process overview

In this section, we will outline the contemporary nesting process within the steel manufacturing sector. Before delving into the specifics, let's clarify some key terms closely associated with this nesting process.

2.1.1 CAD software

CAD (Computer-Aided Design) software facilitates the digital development and design of parts, allowing users to create precise digital blueprints of components with specific dimensions and features. This digital representation serves as a virtual prototype, easing the transition from design to physical manufacturing (Tormach, 2024). This is where CAM software comes into play.

2.1.2 CAM software

CAM (Computer-Aided Manufacturing) software bridges the gap between digital design and physical production by converting CAD drawings into machine-readable instructions for manufacturing processes. It imports CAD designs and translates them into code language, known as G-code, instructing CNC machines on how to produce components (Tormach, 2024).
CAM systems simulate tool movements, enabling early assessment of the required tool movements and fixture utilization to ensure producibility. This proactive approach prevents the design of components that cannot be feasibly manufactured (Cadix, 2024). Essentially, CAM software acts as an intermediary in the transition from digital design to physical production.

2.1.3 G-code

G-code serves as the language that directs CNC machines during the manufacturing process. Generated by CAM software, G-code comprises a series of commands and coordinates that dictate machine movements, tool changes, and other operational parameters. For instance, within a nesting scenario, the G-code might specify drilling operation 10 at coordinates (x, y), or cutting a line from position (x, y) to (x', y').
It serves as the link between digital design and physical fabrication, guiding CNC machines in executing precise manufacturing tasks (Tormach, 2024).

2.1.4 Post-processor

A post-processor serves a function analogous to a printer driver. Tailored to specific CNC machines, a post-processor translates the output from CAM software into instructions compatible with the unique characteristics of each machine. Much like printer drivers must match the specifications of individual printers, post-processors ensure seamless communication between CAM software and CNC machines, optimizing manufacturing efficiency and accuracy (Jetcam, 2024).

2.1.5 General steps in nesting process

The nesting process in steel manufacturing involves several iterative steps, each playing a crucial role in the overall workflow. Traditionally, nesting systems were considered numerical control (NC) programming tools, wherein 2D models of components generated in CAD software were utilized to generate G-code for driving CNC machines. However, modern advancements offer desktop-based nesting software with enhanced features, such as automated import of orders and items, plate inventory management, and automated nesting and NC program generation (ESAB, 2024).

The process begins with identifying items for fabrication, along with the material thickness of the sheets, and the dimensions of the items. Precise CAD drawings of these items and metal sheets are then created, serving as blueprints for subsequent manufacturing steps, see Figure 2.1.

FIGURE 2.1: CAD to cut (HyperTherm, 2024).

These CAD drawings are imported into CAM software. In CAM, sophisticated algorithms can be used to generate tool paths tailored to specific manufacturing requirements. Automatic nesting features use these algorithms to quickly rotate and position the parts, optimizing plate plate utilization and minimizing scrap (HyperTherm, 2024). However, manual nesting is still common in many companies due to unmet requirements by existing algorithms and software.

Once the pattern is created, CAM software generates a cutting order and NC code, which the CNC control system interprets and converts into electrical signals to control the cutting process. This marks the beginning of the automated cutting process. Further processing of the NC file involves providing instructions to the cutting system and various machine components before cutting items (HyperTherm, 2024). These instructions include selecting appropriate cutting tools and devising effective work-holding strategies to secure the components during machining.
In summary, the NC code determines the execution of features and operations on items, such as creating holes through drilling, and the sequence in which the operations will be executed on the machine. Additionally, the remnant piece or crop line can be determined within the NC code or manually.

Once the tool path is generated, CAM software generates the G-code necessary to operate the CNC machine for the specific part. Many nesting software programs can manage multiple machines, leading to a generic (NC) process. Post-processors, customized for each machine, translate the NC-code to G-code. G-code provides instructions understood by the machine, directing its actions and displaying the items to be cut on the machine's dashboard.
The conversion from NC to G-code is automated, following predefined rules set by software developers. Tool path generation itself can be manual or automatic through software. Any desired modifications to the G-code requires changes to the NC-code.

In summary, CAM software manages various phases, from process planning and tool and raw material procurement to quality control checks after part machining. Tools are controlled using coordinates (Cadix, 2024).

With the G-code prepared, the CNC machine is configured accordingly, including loading the metal sheet(s) and setting cutting parameters. The machine then executes the cutting process based on the instructions provided in the G-code.
A nesting program thus considers both raw material optimization (producing the desired quantity with minimal waste) and the operational aspects (determining which tools are needed and when). It oversees the entire production process and adjusts accordingly.

Pre-NC

In addition to the standard NC process, there exists a concept known as Pre-NC. Imagine a scenario where a company manufactures identical parts regularly. It can become tedious to repeatedly specify details like drilling instead of cutting for each hole. Pre-NC provides a solution by enabling the storage of such instructions during the CAD design phase. This involves embedding specific directives within the part itself before generating the overall NC-code (tool path). Instead of creating a separate NC-code for each machine, Pre-NC generates customized NC codes for individual items. These Pre-NC instructions seamlessly integrate into the NC-code when nesting the items.

2.2 Nesting process of company C's machines

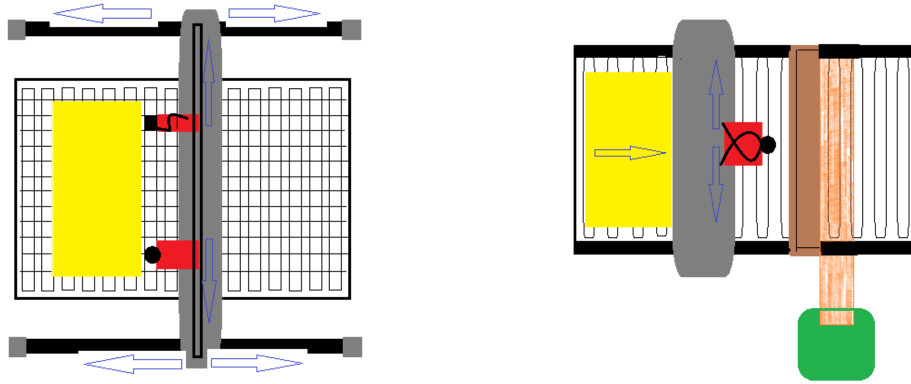
In this section, we will delve into the current nesting process utilized on the flatbed machine at company C. To provide context, we will first outline the operations of both the flatbed machine and the pass-through machine, explaining why our focus for future nesting algorithm development will be on the flatbed machine.

2.2.1 Workings of company C's flatbed and pass-through machine

In Chapter 1, we introduced two types of machines used for nesting items at company C: the flatbed machine and the pass-through machine, as illustrated in Figure 2.2. Both are CNC sheet metal working machines capable of processes like 3D plasma bevel cutting, oxyfuel cutting, drilling, and marking. This subsection delves into their workings and applications.

Let's start with the primary type for this thesis, the flatbed machines, depicted in Figure 2.2a. These machines feature a stationary metal sheet fixed on a raster, while the cutting, marking, and drilling components move along both the x and y axes. Equipped with plasma cutting technology, they can automate processes like carbide drilling, 3D bevel cutting, and contour milling. Additionally, they offer a dashboard interface for real-time progress updates. While company C's software enables unmanned operations on these machines, manual or magnet-assisted removal of cut items from the metal sheet is still required.
The second type, pass-through machines, shown in Figure 2.2b, operate differently. Here, the metal sheet moves exclusively in the x-direction, while the cutting and drilling apparatus moves solely in the y-direction. Equipped with both plasma and oxyfuel-cutting technologies, pass-through machines streamline part handling by transporting cut items on a folding table to a conveyor belt leading to a collection bin. This process, driven by the sheet's x-directional movement, thus eliminates the need for manual removal of cut items from the metal sheet.
The automated process of the pass-through machines is particularly advantageous for small-part production scenarios. However, they are less suitable for larger components that cannot pass through the folding table, necessitating manual removal and reducing automation efficiency. Conversely, flatbed machines are efficient for cutting larger items, as small items may fall through the raster, requiring manual retrieval.
Another disadvantage of pass-through machines is reduced cutting accuracy due to the metal sheet's movement during the cutting process, unlike flatbed machines where the metal sheet remains stationary.
Furthermore, as is now known, the cut items on a pass-through machine fall through the folding table into a collection bin. This requires precise nesting [1] to facilitate product release with a single

[1]The items must, for example, be positioned neatly in columns.



(A) Illustration of the flatbed machine. The yellow rectangle denotes the metal sheet secured on the grid. The machine, featuring a drill and laser depicted by the red boxes with black symbols, can move along both the x and y axes, as indicated by the blue arrows.

(B) Illustration of the pass-through machine. The yellow rectangle represents the metal sheet, advancing along the rolling bench in the x-direction. The machine, depicted by the red box with black dot, can only move in the y-direction. The folding table is depicted in brown and the conveyor belt beneath the rolling bench and bin are the orange and green figures, respectively.

FIGURE 2.2: Illustrations comparing a flatbed machine (left) and a pass-through machine (right).

flap movement. Failure to adhere to this requirement results in loose products, halting the machine's progress. Additionally, at company C, mostly rectangular products are cut on pass-through machines due to the folding table constraints, whereas these limitations do not apply to flatbed machines.

In conclusion, while pass-through machines offer a more continuous process for small items, they come with extra restrictions and nesting requirements. Therefore, this thesis will focus solely on possibly implementing the new nesting algorithm on flatbed machines.

2.2.2 Explanation of nesting process on company C's flatbed machine

The general steps of nesting, as explained in section 2.1.5, can also be executed on company C's CNC flatbed machine. Here, we will provide an overview of the nesting process specifically tailored to company C's CNC flatbed machine. This section will cover the steps involved, from item and sheet selection to the generation of NC code and execution of machine operations, as depicted in Figure 2.3. By illustrating the interplay between software and machine functionalities, this subsection aims to elucidate how digital designs translate into physical components.

Item and sheet selection

The process begins with the identification of items scheduled for cutting within a defined time frame, typically encompassing all items slated for completion within a three-week window. The selection process considers factors such as material thickness and dimensions, ensuring alignment with production goals, see Figure 2.3. Items from various orders may be nested together if their thicknesses match and space permits. A list of available sheets, specifying the types and dimensions required to fulfill the orders, is complied accordingly.

CAD drawings

The digital geometry of selected items and sheets can be created using CAD software or nesting software with integrated CAD capabilities. Once CAD drawings are generated, the blueprints are

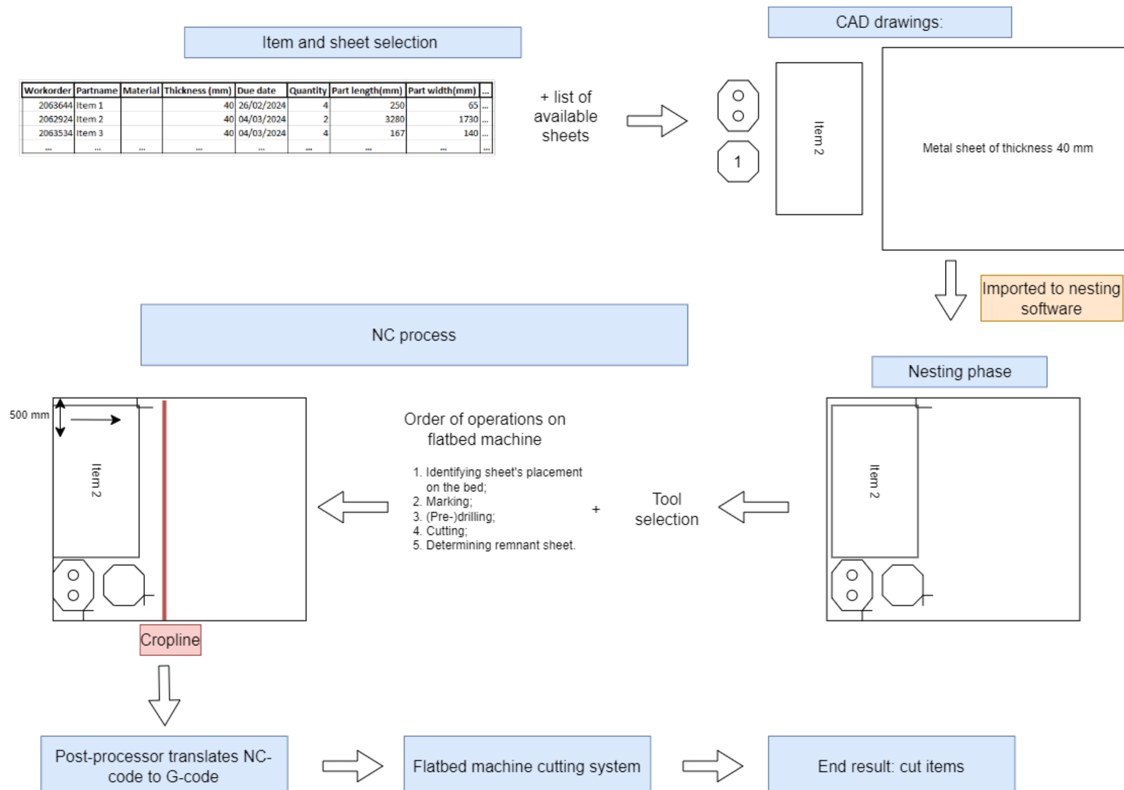
imported into the CAD/CAM nesting software, which serves as the platform for arranging items on metal sheets.

Nesting

Company C utilizes CAD/CAM nesting software equipped with a built-in algorithm for automated or manual nesting. The software provides functions for configuring task parameters, manually nesting parts, and modifying the nesting layout. Users can specify material specifications, part clearance, edge distance, and nesting strategy to optimize nesting efficiency.
A part clearance value is used to specify the distance between items on the nest. The edge distance is used to specify the border around the sheet in which no items are to be nested. The nesting strategy of this software is to nest items vertically starting in the nest reference position. Such a position is chosen by the user of the software and can for example be to start in the left top corner of the metal sheet. The objective of the algorithm is to minimize the horizontal width the same way as was presented in section 1.2.

The algorithm is a time-based nesting engine that continuously nests and re-nests parts until the yield cannot be improved within the set time interval, often guided by the due date of the order.[2] Users can choose to rotate items during nesting to find the best layout. Manual nesting and modification options are also available for fine-tuning the automated process. Moreover, a "Pre-nesting" function allows manual grouping of parts for cohesive nesting. For instance, multiple parts can be pre-nested to form an ‘L’-shape, ensuring they are nested accordingly on the sheet.

While the software offers automated nesting, it also offers extensive manual interference options. This is because manual adjustments are often required to achieve better nesting layouts.



Workorder	Partname	Material	Thickness (mm)	Due date	Quantity	Part length(mm)	Part width(mm)	...
2063644	Item 1		40	26/02/2024	4	250	65	...
2062924	Item 2		40	04/03/2024	2	3280	1730	...
2063534	Item 3		40	04/03/2024	4	167	140	...
...	...	...	...	...	...	...	...	...

FIGURE 2.3: Nesting process of flatbed machine.

[2]Initially, items with the earliest due date are nested, followed by nesting the remaining items in order of their due dates.

NC process

After finalizing the nesting layout, the software generates NC-code, providing a tool path for machine operations. Users can opt for automatic or manual NC generation. The software can assign tools for nesting, specify cutting methods and generate crop lines for remnant sheets. Additionally, users can manually create NC paths and specify part contours for cutting.

When selecting automatic NC generation, the software employs various strategies to generate an NC path for the nested items. Parameters for setting up cutting orders, preferences, and NC path methods are available. For instance, users can select cutting order strategies, where a specified width serves as the path zone width, zigzagging from left to right across the sheet to perform the operations (see Figure 2.3). Additionally, users can designate starting point of the NC path and determine the sequence of processes for nested items. A choice can be made to first complete all processes for an individual item before moving on to the next item or complete a process for the entire sheet before moving on to the next process.
The software can also automatically assign plasma cutting for items with a drill diameter and provide beveled edges to items.

The final step involves determining the crop line on the metal sheets, which dictate the remnant. Users also have the option to manually create the NC path and determine the sequence of part contours to be cut.

Usually the following typical sequence of operations is performed on the flatbed machine:

1. Identification of the sheet's placement of the machine's bed;
2. Marking of the tool path, indicating the placement of all items to be cut;
3. Pre-drilling and drilling of all work lines;
4. Plasma cutting;
5. Cutting a crop line if sufficient remnant sheet remains.

The identification of the sheet's placement involves fixing the metal sheet onto the machine's bed using laser welding at multiple points to ensure stability during cutting. Subsequently, the machine establishes the reference point, typically at the lower right corner of the metal sheet, and traces the sheet's bottom to determine its angle on the bed.
Once all the parameters are set, the tool path is marked on the plate using a plasma torch, often involving pre-drilling for thick sheets to aid in laser penetration. During this marking process, the plasma torch creates light incisions without fully penetrating the metal sheet.
Following marking, drilling of all components is performed before cutting the items using plasma technology. The sequence concludes with the optional cutting of a crop line to separate the remaining pieces from the main sheet.
While users have the option to manually determine which parts require cutting or drilling and the sequence of these operations, the outlined sequence represents the commonly adopted approach for the flatbed machine.

G-code & machine intelligence

The generated NC-code is translated into G-code by a post-processor, enabling machine execution. G-code contains instructions for machine operations, dictating tasks such as identifying sheet's placement, marking, drilling, and cutting. While G-code dictates operation sequences and which operations are performed with which process, machine intelligence software governs operational variables such as cutting speeds, angles, and tool exchanges. This integration ensures efficient execution of the nesting process, translating digital designs into physical components with accuracy and reliability.

2.3 Key considerations for nesting on CNC machines

In addition to optimizing cutting patterns, nesting software must account for various constraints and features related to materials and machining technology. These considerations include limitations where machining cannot occur due to material clamping and requirements for minimal offset between parts in specific cutting methods like plasma cutting. In essence, the nesting algorithm must address complex geometries while adhering to no-overlapping and containment constraints. Alongside determining good part placement, it is essential to factor in quality considerations and CAM restrictions during nesting. Quality improvements may involve rotating parts as needed, while CAM restrictions include elements like lead-ins/-outs, pre-piercing, and bevels.

This section will discuss some requirements and constraints associated with nesting on CNC machines, that aligning with the requests of company C. It will delve into considerations beyond mere part placement, emphasizing their significance in achieving the best layouts feasible with CNC machines.

2.3.1 Rotation

Rotating parts during nesting is critical to achieving the most efficient placement and layout. The ability to freely rotate parts or adhere to specified angles provides flexibility in arranging parts optimally. Software typically offers options for rotation, such as manual rotation or rotation according to specified angles. Ideally, the nesting algorithm itself should rotate parts incrementally to find the best arrangement.

Control over rotation angles significantly impacts quality and layout optimization during nesting. Effective rotation ensures both quality and the attainment of the best possible layout, thereby contributing to efficient CNC machining processes.

2.3.2 Product spacing

This feature ensures that parts are nested within a specified clearance distance from each other and the sheet edge, preventing overlaps and enabling efficient nesting. Key aspects related to product spacing include:

- **Part clearance**: The distance between parts on the sheet, essential for spacing and avoiding overlaps during nesting.
- **Edge distance**: Specifies the distance from the sheet's edge, where parts should not be nested.

The product spacing area, depicted in Figure 2.4, defines where parts can be placed, incorporating considerations for lead-ins/-outs, pre-piercing, and bevels. This section will explain what these terms mean. The product spacing area is crucial for translating nesting from CAD to CAM, ensuring no overlap of items during the nesting phase.

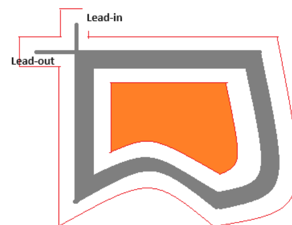


FIGURE 2.4: Inside and outside spacing area of an item, depicted in grey, with the orange area representing space for additional part-in-part nesting.

Besides nesting items side by side, considering the product spacing of each item allows for part-in-part nesting, further optimizing material usage by minimizing scrap. Both external and internal spacing areas need consideration when implementing part-in-part nesting, as illustrated in Figure 2.4, where the orange area denotes space for another item, including its product spacing, to be nested efficiently.

The product spacing varies depending on the thickness of the metal sheet, necessitating adjusted spacing settings to accommodate material properties. This adjustment accounts for variations in kerf width resulting from cutting processes like plasma cutting or laser cutting. Kerf represents the width of material removed during the cutting process (University, 2024). As the thickness of the metal sheet increases, the width of the kerf widens, influenced by factors such as cutting current, torch height, speed, and gas settings, particularly evident in plasma cutting processes.
Consequently, to ensure optimal spacing between parts and sheet margins, product spacing settings must be adapted accordingly. The machine automatically factors in the kerf width into the clearance distance, simplifying the nesting algorithm and streamlining the overall process.

While the product spacing is crucial for traditional nesting methods, common cutting, depicted in Figure 2.5, offers an alternative approach. Common cutting involves parts sharing common edges, allowing a single cut to separate them rather than leaving material between them (known as the 'skeleton').

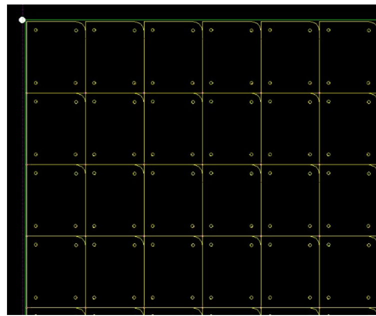


FIGURE 2.5: Nesting with common cutting: items sharing common borders (Jetcam, 2024).

Common-line cutting proves particularly beneficial for rectangular parts in sheet metal, reducing cutting time and waste. However, its applicability depends on factors such as edge quality requirements, process stability, and software capabilities. This is because one side of the cut may be less favourable due to the cutting angle and swirling motion of the plasma, potentially impacting edge quality. Therefore, careful consideration and implementation are necessary, as not all nests may be suitable for common-line cutting (Jetcam, 2024). This is the reason it will not be one of the main requirements for the algorithm in this thesis.

Lead-in and lead-out

In addition to determining the optimal item placement strategy, nesting algorithms must also identify suitable locations for lead-in and lead-out points for each item. Both lead-in and lead-outs should be adjustable and rotatable to accommodate various cutting scenarios.
Lead-in and -out operations play a crucial role in achieving precise cutting operations. These tool path operations establish entry and exit points for the cutting tool, ensuring clean cuts without unwanted artifacts such as tear-out, chipping, or burning in the final piece. An illustration of lead-in/-out points can be observed in Figure 2.4.

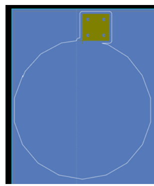
There are several types of lead-ins, including straight-in-straight-out and arc-in-arc-out. Most software platforms allow users to manually incorporate lead-ins during part creation, utilizing geometry functions to draw and define them accurately. Figure 2.6 showcases examples of common lead-ins/-outs. The type and size of lead-in and lead-out points can significantly influence cut quality.

FIGURE 2.6: Examples of common lead-ins and lead-outs.

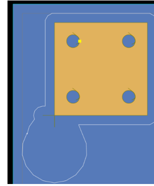
Pre-pierce

Pre-piercing, also known as pre-pierce, is an operation step in the cutting process aimed at preserving the cutting tool head and enhancing cutting efficiency. When applying pre-piercing in plate nesting, it involves creating holes or openings at specified points on the material before the actual cutting process commences. The pre-pierced points serve as starting points for subsequent cutting operations.
By pre-piercing the material, the cutting tool head (such as a plasma torch) encounters less resistance when initiating the cutting process. This reduces wear and tear on the tool head, prolonging its lifespan and maintaining cutting quality over time. Pre-piercing also expedites the cutting process by providing predefined entry points for the cutting tool. This eliminates the need for the tool to pierce through the material from scratch at each cutting location, resulting in faster overall cutting times.

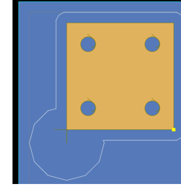
Some software packages offer options for configuring pre-pierce parameters, allowing users to customize settings according to their specific needs.
Each component can undergo pre-piercing, a process conducted separately from the lead-in/-out, necessitating consideration of additional clearance. The pre-pierce diameter thus contributes to the product spacing calculation, as illustrated in Figure 2.7a, which depicts an example of a pre-pierce with an unrealistic diameter of 500 millimeters.



(A) Illustration of a pre-pierce operation with an exaggerated diameter of 500 millimeters.



(B) Illustration of an "on the edge" pre-pierce.



(C) Illustration of an "in the center" pre-pierce.

FIGURE 2.7: Comparison between pre-pierce on the edge and in the center, showing two available options for positioning the pierce point.

During pre-piercing, users must specify the desired diameter and position. Currently, for company C, there are two available positions: "On the edge", aligning the pierce position with the lead-in on the part's edge, or "In the center", aligning it with the lead-in at the center. Figure 2.7 illustrates both options, with the company typically opting for the "On the edge" option.

Bevel

A bevel is a sloped or angled edge cut into a material, typically applied to the edge of an item to create a sloped surface rather than a straight edge. Beveling is commonly employed in industries like metal fabrication, welding, and construction, where precise edge preparation is necessary for welding, brazing, or joining operations. Beveling serves various purposes, including improving weld joint quality, enhancing the aesthetics of the finished product, and facilitating component fit-up during assembly.

The angle of the bevel can vary depending on specific application requirements such as material thickness, welding process, and desired joint configuration. In essence, the bevel acts as an

extra feature that accounts for the cutting angle of the laser or other cutting tool. For an illustration, refer to Figure 2.8, where the right side of the quadrilateral demonstrates a bevel.

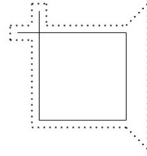


FIGURE 2.8: Representation of a bevel cut on the right side of the quadrilateral, showcasing an angled edge.

At company C, several items typically require bevels, highlighting the importance of considering this factor when arranging the nesting layout.

2.3.3 Crop line

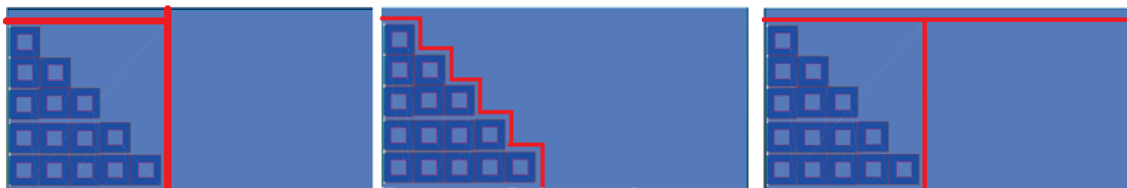
As discussed in section 2.1, the final step of the NC process involves determining the crop line on the metal sheet. The crop line concept in nesting refers to defining the unused section of the sheet leftover after the nesting process, also known as a remnant. By defining the crop line, nesting algorithms can efficiently arrange parts within the specified area, maximizing the number of parts cut from the sheet while minimizing scrap material.

Additionally, the crop line aids in creating remnants from the leftover material, reducing material costs and enhancing overall manufacturing efficiency. Operators have the flexibility to create remnants electronically within the software by cropping the sheet or manually entering the dimensions of irregular sheets found in the manufacturing facility. This allows leftover material to be reused for other jobs. The crop line can also be cut manually after the entire nesting and cutting process has been completed on the machine.

There are several crop line types, as illustrated in Figure 2.9:

- Vertical crop line at the edge of the last part nested.
- Dynamic (or L-shape) crop line along the edges of the nested parts.
- Horizontal crop line across the length of the sheet, resulting in two remnants from the remaining material (vertical and horizontal crop lines).

The company at the moment mainly uses vertical crop line, but they would like to go more to an L-shape crop line.



(A) Illustration of a vertical crop line. (B) Illustration of a dynamic crop line. (C) Illustration of a horizontal crop line.

FIGURE 2.9: Various crop line options illustrated, including vertical, dynamic, and horizontal configurations, facilitating efficient material usage.

2.3.4 Restricting parts to specific zones of the sheets

For production purposes, certain components occasionally require placement within specific areas of the sheet. This might include machine constraints related to tool limitation, or for facilitating

unloading systems, particularly applicable to the flatbed machine, where small parts might slip through the raster, making it more convenient for operators to retrieve them from the machine borders rather than the center. For these purposes you would want a nesting algorithm that enables the restriction of nested part positions to designated zones on the sheet. These zones could vary in shape, potentially consisting of multiple connected components. Various parts can be confined to different zones, and these zones may partially overlap.

2.3.5 Possible additional considerations

Tabbing

When dealing with very thin plates, typically measuring between five to six millimeters in thickness, cutting thin strips poses challenges due to their susceptibility to warping under heat. These plates lack the required rigidity and thickness to withstand the thermal stresses induced during cutting. To address this issue, tabbing can be employed.

Tabbing involves leaving small uncut sections, or tabs, along the edges of the parts or between neighbouring parts. These tabs serve to keep the parts connected to the main sheet, providing support during the cutting process, see Figure 2.10.

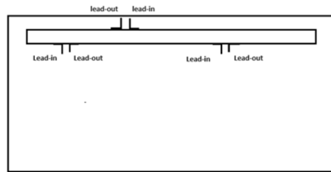


FIGURE 2.10: Illustration of tabbing on a 5 mm thick sheet.

Once the cutting operation is complete, the tabs can be easily removed by breaking or cutting them away, leaving the individual parts intact and ready for use. By employing tabbing, delicate or small parts can remain securely attached to the sheet until they are required, thereby minimizing the risk of damage or loss during handling.

Priority on parts

Priorities can be assigned on part to establish a production sequence. Parts with higher priorities will be manufactured ahead of those with lower priorities. During nesting, algorithms then have to prioritize placing parts with the highest priority onto the current sheet first. If space runs out, lower priority parts can be filled in, with higher priority parts that do not fit potentially nested on subsequent sheet. While it can always be considered, priorities may reduce yield compared to more flexible algorithms.

2.4 Some existing nesting algorithms and software

In section 2.1.5, the importance of nesting algorithms and software in CAM for generating tailored tool paths was highlighted. However, as discussed in Chapter 1.1, the existing plate nesting algorithms and software often fall short of accommodating the specific requirements of company C while maintaining efficiency and speed.
This section aims to address this by describing some of the existing algorithms and software considered among the best, according to some engineers and software developers (Andy, 2021; Eziil, 2024; Watts, 2024).

2.4.1 Software used by company C

Firstly, let's explore the nesting software employed by company C, hereafter referred to as the company software. While a general overview of its functionalities was provided in section 2.2, we

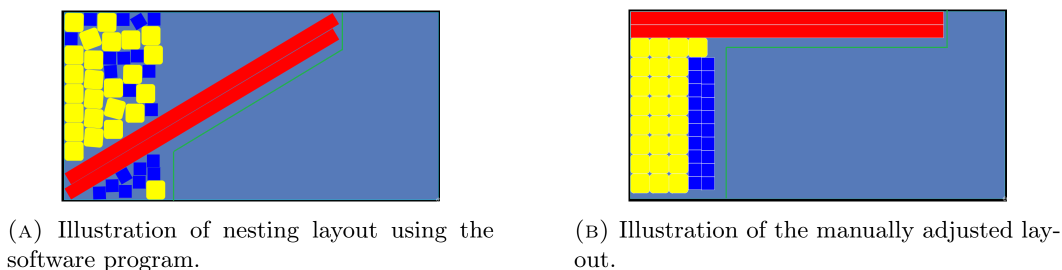
will now delve deeper into its capabilities.

The company software is tailored for the efficient nesting of sheet metal and plate materials. Here is a concise summary of its key features:

- **Nesting optimization**: The software optimizes part arrangement on sheets, considering factors such as material type, size, and cutting parameters to enhance efficiency and minimize scrap. It allows for rotation or mirroring of parts to find the best fit within available space.
- **Integration with CNC machines**: Seamless integration with CNC machines ensures smooth communication between the nesting software and cutting equipment.
- **Material utilization**: It minimizes material waste by placing parts on sheets, considering part geometry and any remnants from prior cuts.
- **Automation**: Streamlining the nesting process, the software automates tasks like tool-path generation, reducing manual intervention and improving efficiency.
- **Compatibility**: Compatible with various cutting technologies (plasma, laser, water jet, oxyfuel), making it adaptable for different manufacturing processes.
- **Reporting and Analytics**: The software offers reporting and analytics tools for tracking material usage, machine efficiency, and other key metrics.

The software employs algorithms to analyze part geometries, material properties, and cutting requirements to generate efficient nesting layouts. Features like automatic nesting, remnant management, and common-line cutting further enhance production efficiency. Users can customize parameters like preferred cutting direction, part priority, and cutting tolerances. And it seamlessly integrates with various CAD/CAM systems and cutting machines, ensuring compatibility across different workflows, maintaining high accuracy, and productivity levels.

As previously mentioned, the software's primary goal is to reduce scrap percentages by minimizing the horizontal width relative to the metal sheet. While this optimization method initially seems beneficial for minimizing scrap, it can lead to a significant issue. By solely focusing on minimizing the horizontal width, the software may inadvertently create substantial unusable space, potentially rendering a significant portion of the scrap plate virtually unusable.
A direct comparison between Figures 2.11a and 2.11b illustrates this concern, particularly when examining the resulting reusable scrap plates. In both figures, a crop line denoted by the green line separates the nested plate from the scrap plate. However, the manually adjusted layout, where the red beams are positioned horizontally against the top side of the metal sheet, results in a considerably larger amount of remnant sheet compared to the software approach. The closed-in triangle formed by the red beams, along with the yellow and blue items, leads to unusable scrap plate material.
Therefore, solely minimizing the horizontal width does not necessarily minimize scrap. It is crucial to consider the crop line when forming the nesting layout.



(A) Illustration of nesting layout using the software program.

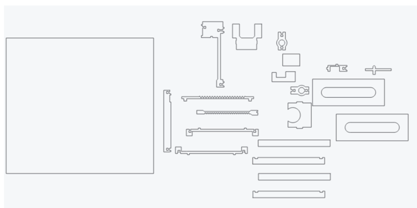
(B) Illustration of the manually adjusted layout.

FIGURE 2.11: Comparison between nesting layouts generated by the software and manually adjusted layout.

Additionally, the software has other drawbacks. It lacks customization according to customer

needs, and even if customization were feasible, company C lacks the expertise and knowledge to modify existing algorithms accordingly. As a response, Department D aims to establish in-house knowledge of nesting algorithms to create tailored algorithms adaptable to customer preferences.
Moreover, the software has a time-related limitation in its pursuit of a 'good' solution, with the stop criterion for finding no better solutions set at 120 seconds, potentially leading to prolonged computation times and increased costs. The time interval can be reduced, but this often results in poorer nesting layouts compared to using the full 120 seconds. Additionally, the minimum number of sheets for a task need to be manually specified by the user, in order for the nesting software to find the absolute minimum number of sheets needed for the nesting. The number can be manually increased if there still are too many parts to nest on the sheets, but this is not initially done by the software.
Furthermore, the presumed constructive heuristic nature of the algorithm poses a limitation. Initially grouping identical items as a starting solution often proves inadequate due to the irregular shapes of the items requiring cutting. Consequently, there is an emphasis on improving the initial solution generated, aiming for a more effective constructive heuristic algorithm.[3].

Having discussed the software's strengths and weaknesses, let's evaluate its performance compared to other nesting software, discussed later in this section, using the same examples for a clear comparison. The first example involves the drawbot, with items and a sheet depicted in Figure 2.12a, using a part and sheet border gap of 0.01 millimeters during nesting. The second example, depicted in Figure 2.12b, utilizes a sheet measuring 6000 by 3000 millimeters for nesting the items, with a part and sheet border gap of 10 millimeters applied during nesting.



(A) Illustration of the items and sheet for nesting the drawbot example. Here the sheet is the left most rectangular object.

(B) Illustration of the items for nesting the second example.

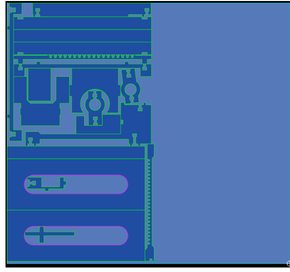
FIGURE 2.12: Illustration of the DXF (Drawing Exchange Format) files of the two examples, used in the future for comparison between nesting software.

As some nesting software impose a built-in time limit of five minutes to generate a nesting layout, we will use these five minutes as a benchmark for all nesting software discussed in this section. Following this benchmark, the result of the drawbot example after running the company nesting software for five minutes is depicted in Figure 2.13a. Additionally, accompanying statistics are provided by the software to offer a comprehensive understanding of the output. Similarly, the result of the second example, along with its statistics, after running the company nesting software for five minutes is depicted in Figures 2.13c and 2.13d.

2.4.2 Overview of other nesting software

This section provides a short description of several nesting software tools used for optimizing material usage in various industries. Here we delve into the functionalities of SVGnest, Deepnest, Nest&Cut, and Inventor Nesting, comparing their performance against each other and the company software. More detailed information of every nesting software can be found in Appendix B. Table 2.1 also provides a comparison of the most important features, strengths, limitations, and performance of each nesting software tool.

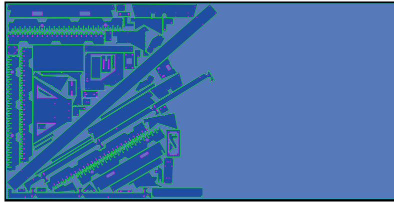
[3] As mentioned earlier, it is not known how the nesting algorithm in the software works, since it is part of a commercial software. So based on the limited information and most common heuristics, we, for now, assume that finding a better initial solution should result in a faster found and maybe even better-found solution (Hansen & Mladenović, 2006).



(A) Company software output of the drawbot example after 5 minutes.

Sheet Name	
Material	S355JR
Thickness	30
Length	346.492
Width	319.206
No. of Repeats	1
Stack Quantity	1
Scrap %	24.7254
Plate Used %	52.0467
Length Used	180.3376
Width Used	319.206

(B) Accompanying statistics.



(C) Company software output after 5 minutes.

Sheet Name	006-5069_Fake_Sheet
Material	S355JR
Thickness	10
Length	6000
Width	3000
No. of Repeats	1
Stack Quantity	1
Scrap %	26.8807
Plate Used %	46.2668
Length Used	3244.8222
Width Used	2990

(D) Accompanying statistics.

FIGURE 2.13: Nesting layout of both examples after 5 minutes using company C's software, along with associated statistics.

SVGnest

SVGnest is a browser-based vector nesting tool that offers a free and open-source alternative for resolving nesting challenges. It utilizes a genetic algorithm for global optimization, allowing it to competently address nesting problems, including arbitrary containers and concave edge cases. Notably, SVGnest supports part-in-part functionality, allowing parts to be positioned within the voids of other parts. The nesting strategy comprises two fundamental aspects: placement strategy and optimization strategy. For part placement, SVGnest utilizes the concept of "No Fit Polygon" (NFP) and "Inner Fit Polygon" (IFP) to determine feasible part placements. In terms of optimization, SVGNest adopts the "First-Fit-Decreasing" heuristic, prioritizing larger parts during placement and refining the nesting layout iteratively using a genetic algorithm. Additionally, SVGnest offers some configuration parameters which are described in Appendix B.1.

Despite its strengths, SVGnest has notable drawbacks. It exclusively supports SVG (Scalable Vector Graphics) file formats, which may limit compatibility with other CAD programs and file formats such as DXF (Drawing Exchange Format). Additionally, it lacks advanced configuration options, preventing users from specifying thickness dimensions or preventing individual items from rotating. The software may also encounter overlapping issues, particularly when using the "Explore concave areas" configuration, and performance degradation when adding space between parts. Furthermore, users cannot adjust individual part clearance, which can be crucial for preventing warping due to heat, and the software may not provide clear feedback when dealing with oversized items or failing to generate a nesting layout.

One notable feature of SVGnest is its ability to automatically minimize the number of sheets required to efficiently nest all items without the need to specify a minimum number of sheets beforehand. It also depicts the material utilization while nesting. However, it struggled to find nesting layouts for the examples, possibly due to the complexity of part-in-part requirements and specific part gap configurations.

Deepnest

Deepnest is another open-source nesting software known for its simplicity and optimization capabilities. Developed by the team behind SVGnest, it offers a wide range of configuration options,

including space between parts, curve tolerance, part rotations, and optimization types. For more information, see Appendix B.2. Its user-friendly interface simplifies the nesting process into three straightforward steps. Deepnest supports both SVG and DXF file formats, making it compatible with various design software.

However, Deepnest has limitations such as the lack of support for importing multiple files simultaneously and the need for users to specify the number of sheets beforehand. It may struggle with larger item quantities or specific part gap configurations. Despite these drawbacks, Deepnest provides an effective solution for optimizing material usage in various nesting projects.

In the drawbot example, Deepnest achieved a material utilization of approximately 54.8%, slightly higher than the company software. Since Deepnest does not give any statistics of the nesting, we have computed this material utilization ourselves, by estimating that the overall length used is approximately 235.96 mm and the width used is approximately 256.93 mm, see Figure 2.14. However, it encountered difficulties running the second example, possible due to the complexity of the part gap configuration.

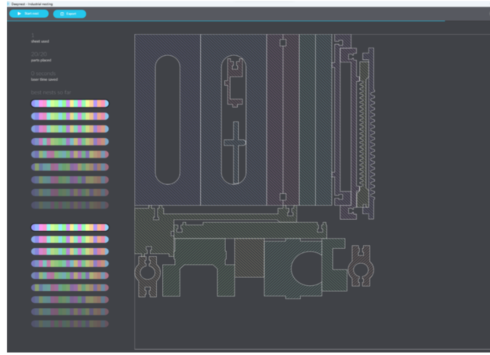


FIGURE 2.14: The Deepnest output for the drawbot example employing the bounding box optimization, showcasing the nesting layout generated by the software.

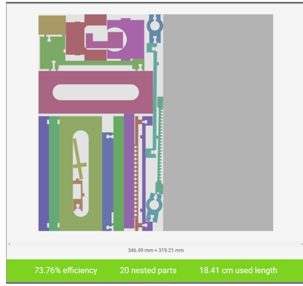
Nest&Cut

Nest&Cut, developed by Alma, is a sophisticated web-based application designed for optimizing material usage by arranging shapes on a sheet to minimize waste. It offers subscription-based access to high-performance automatic nesting functions for various complex 2D shapes. With a user-friendly interface, Nest&Cut simplifies the process of initiating automatic nesting in the cloud and delivers optimized nesting layouts ready for use in cutting software or numerical control machines.

Nest&Cut features advanced functions for cleaning DXF or DWG geometries, CAM support, multi-format nesting, and recognition of various CNC and laser machines for exporting optimized NC files. It also provides tools for accurately estimating material consumption and associated costs.

While Nest&Cut offers advantages such as the ability to select multiple sheet sizes and prioritize sheets, it has limitations such as requiring users to specify the number of sheets that can be used. However, it does reveal which items have not been nested if sufficient sheets are available, see Appendix B.3.

In comparing Nest&Cut to the company software, it appears that Nest&Cut's output for the drawbot example, see Figure 2.15a, is marginally inferior due to its higher space utilization. For the second example, see Figure 2.15c, although Nest&Cut initially seems to produce a favorable outcome with a larger rectangular remnant sheet measuring 21857 by 3000 millimeters, the company software's item placement could potentially yield a larger remnant sheet without necessitating it to be rectangular. Additionally, the sheet border gap of 10 millimeters is not consistently maintained in Nest&Cut's output.



(A) The Nest&Cut output for the drawbot example after five-minutes.

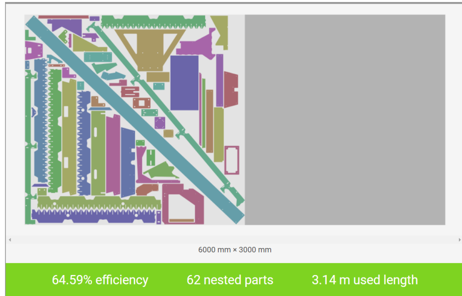
NESTING LAYOUT INFORMATION

Sheet name	draw-bot2#1
Sheet dimensions	346 mm x 319 mm
Sheet multiplicity	1
Cutting length / marking length	7.347 m / 0.000 mm
Number of: nested parts / external contours / internal contours	20 / 20 / 2
Nesting efficiency	73.76%
Nesting: length / surface	184.050 mm / 58749.72 mm²
Total parts surface	43331.76 mm²

GENERATED OFFCUT

Offcut dimensions	162 mm x 319 mm

(B) Accompanying statistics.



(C) The Nest&Cut output for the second example after five minutes.

NESTING LAYOUT INFORMATION

Sheet dimensions	6000 mm x 3000 mm
Sheet multiplicity	1
Cutting length / marking length	149.157 m / 0.000 mm
Number of: nested parts / external contours / internal contours	62 / 62 / 266
Nesting efficiency	64.59%
Nesting: length / surface	3.143 m / 9.43 m²
Total parts surface	6.09 m²

GENERATED OFFCUT

Report n°1312190 - 28 feb. 2024 Page 4 / 19 https://app.nestandcut.com

Offcut dimensions	2857 mm x 3000 mm

(D) Accompanying statistics.

FIGURE 2.15: Nesting layout of both examples after 5 minutes using Nest&Cut software, along with associated statistics.

Inventor Nesting

Inventor Nesting, a component of Autodesk's Inventor software suite, seamlessly integrates with Autodesk Inventor to optimize material usage and streamline sheet metal fabrication. It employs algorithms to automatically organize parts on sheets, minimizing waste and maximizing efficiency. Key features include integration with Inventor, consideration of material type and cutting parameters, customization options, and reporting and analysis tools.

Compared to other nesting software, Inventor Nesting offers the advantage of manual adjustments to each individual item, providing greater control over the nesting process. It also supports the selection of multiple sheet sizes and offers automatic adjustment of the number of sheets that can be used.

In evaluating its performance, Inventor Nesting provides detailed reports that include cutting order information. This report includes the cutting order, similar to Nest&Cut. However, it only references the names of the items, requiring users to recall their appearance.
The nesting solution for the drawbot example, see Figure 2.16, appears inferior to other software, except for SVGnest. Unfortunately, difficulties were encountered in generating a nesting layout for the second example, possibly due to specific part clearance requirements of 10 mm.
Overall, Inventor Nesting took considerable time to provide an initial nesting layout solution and encountered difficulty recognizing certain DXF files that posed no problem for other software.

2.5 Conclusion

In the first three sections, we explored the plate nesting procedures, focusing on company C's flatbed CNC machine and the essential factors for optimizing CNC nesting. We began with an overview of the nesting process, highlighting the fundamental principles that guide efficient material utilization through nesting algorithms.

Table 2.1: Comparison of Nesting Software Features

Feature	SVGnest	Deepnest	Nest&Cut	Inventor Nesting	Company Software
Pricing	Free and open-source	Free and open-source	Subscription-based	Part of Autodesk's Inventor suite	Proprietary
File Formats Supported	SVG	SVG, DXF	DXF	Various CAD formats	Various CAD formats
Part-in-Part Functionality	Yes	Yes	Yes	Yes	Yes
Part Rotation	Yes	Yes	Yes	Yes	Yes
Optimization Strategy	Genetic algorithm, FFD	Iterative algorithm	Unknown	Unknown	Unknown
Remnant objective	Minimize horizontal width	Minimize horizontal, vertical, or both	Minimize horizontal width	Minimize horizontal, vertical, or both	Minimize horizontal width
Configuration Options	Part spacing, curve tolerance, GA settings	Part spacing, curve tolerance, GA settings	Sheet sizes, part & sheet border gaps, GA settings, cutting paths	GA settings, fixing items from rotation	Part priority, part spacing, cutting paths
Compatibility	Limited to SVG format	SVG, DXF	DXF	Various CAD software, cutting technologies	Various CAD software, cutting technologies
Advanced Features	Automatic sheet minimization, depict material utilization	Automatic merging of part edges, time & material optimization balance	Automatic cleaning of geometries, utilization remnant sheets, prioritization sheets	Integration Autodesk Inventor, automatic sheet selection, reporting & analysis tool	Reporting & analysis tool, CAM support, mirroring
Strengths	Free and open-source, part-in-part functionality	User-friendly interface, manual adjustments	Advanced nesting functions, CAM support	Integration with Inventor, multiple sheet sizes	Advanced nesting functions, manual nesting
Weaknesses	Limited file format compatibility, no advanced configuration options, overlapping issues concave areas	Limited file format compatibility, struggles with nesting large quantities, no automatic sheet addition	Lack of clear error messages, limited compatibility, no material utilization depicted	Difficulty recognizing certain file types, initial layout generation time	Limited file format compatibility, proprietary

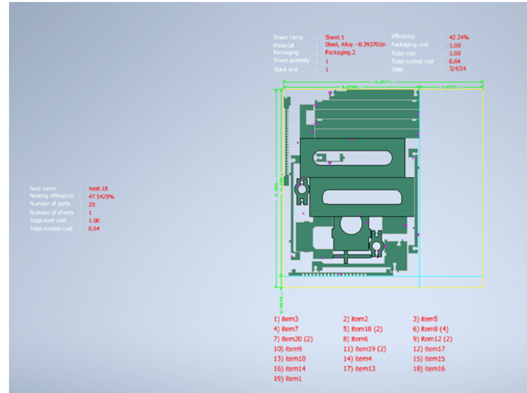


FIGURE 2.16: The Inventor Nesting output for the drawbot example, obtained after a five-minute run with the gaps set to 0.01 mm.

Next, we conducted an in-depth analysis of company C's existing nesting process, examining the software and stategies used to optimize material usage on their flatbed CNC machine. While company C's process showcased sophisticated algorithms, it still faces limitations, prompting the consideration of developing an in-house algorithm. Key considerations for nesting on CNC machines were identified, emphasizing material properties, part geometry, and production constraints.

We further evaluated various alternative nesting algorithms and software solutions, assessing their strengths and weaknesses. The suitability of each solution depends on specific requirements and priorities. Company C's proprietary software demonstrated strengths in seamlessly integrating with CNC machines but struggles with limitations in remnant utilization and objective function, customization, computational time, and the quality of initial solutions. Open-source alternatives like SVGnest and Deepnest leveraged GAs and heuristic strategies for cost-effective nesting solutions, but lack the ability to handle specific requirements, such as part-in-part nesting, complex geometries, and faster computation times. Commercial solutions like Nest&Cut and Inventor Nesting offered sophisticated capabilities but faced limitations in recognizing file formats, accommodating part clearance requirements, and optimizing remnant sheet utilization. These assessments underscored the importance of customization, compatibility, computational efficiency, and nesting optimization strategies.

Section 2.4 was introduced to delve deeper into the current challenges and configuration parameters within existing nesting algorithms and software. This section was important to identify common pitfalls, such as the inefficiencies and extended computation times associated with part-in-part nesting, highlighting which options should be excluded from our scope. The limitations of current algorithms to fix item rotation and customize essential features to meet company C's specific requirements further emphasize why these existing solutions are not viable for the company.
Given the need for a foundational algorithm, it is important to limit the introduction of new configuration parameters and to exclude badly working and computationally expensive features. This approach ensures that the algorithm remains focused, efficient, and aligned with company C's specific operational needs, or a future option to include these needs. Developing an in-house algorithm will allow for the necessary customization, addressing the shortcomings of existing solutions and optimizing material utilization, computational efficiency, and overall nesting performance.

Chapter 3: Literature Research

This chapter reviews the literature by answering the second research question: "*Which methods suggested in the literature are most applicable to solving the company's problem?*". The chapter starts by explaining how the nesting problem of company C is known in the literature. Section 3.2 describes some geometric tools that can be used to generate candidate placement positions for the items on the sheets. Sections 3.3 and 3.4 describe some common selection and placement heuristics used for the piece allocation and piece packing phase respectively. Section 3.5 describes solution methods for the packing problem proposed in the literature, after which we choose the most suitable method for our plate nesting problem. Section 3.7 describes the research gap between the chosen solution method and the plate nesting problem and how to close this gap. The chapter concludes with a summary.

3.1 How is the plate nesting problem known in literature?

The plate nesting problem, also known as an irregular packing problem in literature, is a subset of Cutting and Packing (C&P) problems. These problems involve determining which items to produce from which larger items, like bins, sheets, or stocks while considering the specific geometry of the smaller items to be cut. Solutions to C&P problems must fulfill both quantitative and geometrical criteria, ensuring optimal utilization of both small and large pieces while preventing overlap between the small items themselves and the small items with the large item (Wäscher et al., 2007). Irregular packing problems find applications across various industries, including garment manufacturing (Hu et al., 2020), sheet metal cutting (Wang et al., 2022), furniture making, shoe manufacturing, and shipbuilding (Xu et al., 2016).

This problem can be categorized based on spatial dimensions and application types, including one-dimensional, two-dimensional, three-dimensional, and n-dimensional packing problems (Wäscher et al., 2007). In the context of this thesis, the focus is solely on the two-dimensional aspect, as only two dimensions of the items are relevant due to the consistent third dimension shared by all items placed on the same large item (sheet), a characteristic also observed in the plate nesting problem of company C.
The packing problems can further be categorized based on the regularity of shapes into orthogonal problems, where all pieces are regular, and irregular problems[1], where one or more pieces are non-rectangular, including shapes like polygons, convex or not. Depending on the application, pieces may be rotated freely or constrained to specific angles. Objectives often revolve around minimizing cutting area or maximizing piece value. In practice, problems may involve both regular and irregular items assigned to larger bins or sheets to minimize material or space waste (Alvarez-Valdes et al., 2013). Irregular problems, being more complex, combine combinatorial hardness with the computational difficulty of geometric non-overlap constraints, making them harder to solve compared to regular ones (Leao et al., 2020).

Even in the case of regular (rectangular) shapes, 2D nesting problems are NP-hard, implying that most solution approaches rely on heuristic methods as achieving optimality is often unfeasible for large-scale instances (Li & Milenkovic, 1995).
Several studies, such as Toledo et al. (2013) and Jones (2014), have successfully tackled the 2D irregular packing problem optimally, albeit for small instances[2], often requiring lengthy computation times, typically focusing on the so-called 2D strip packing problem.

[1] We define a piece to be irregular if it requires a minimum of three parameters to identify it. For example, a circle needs just a single parameter, the radius, and a rectangle needs two parameters, its length, and width (Guo et al., 2022).

[2] In these studies, the optimal solution was achieved for instances with a maximum number of pieces in the twenties.

In the context of two-dimensional packing problems, the irregularity of shapes prompts further classification into scenarios where the sheet has either infinite or fixed length. For instance, the 2D strip packing problem (2DISPP) involves packing pieces within a strip stock sheet of infinite length and a fixed width to minimize waste. Conversely, 2D bin packing problems (2DIBPP) and 2D cutting stock problems (2DICSP) aim to minimize the number of stock sheets required to pack all items within sheets of fixed length and width. Although significant attention has been given to the 2DISPP, particularly in clothing industries, where fabric rolls serve as stock with infinite length (Cai et al., 2023), the 2DIBPP and 2DICSP are comparatively less studied. These problems, classified by Wäscher et al. (2007) as input minimization problems, entail arranging all (irregularly) shaped pieces into rectangular bins while maximizing bin utilization.

For all different problem types, it is essential, during the packing process, to adhere to two key constraints (Wang et al., 2022): (1) the pieces do not overlap, and (2) the pieces cannot exceed the contour of the sheet(s). There are typically two operational modes for the packing problem: online and offline. Our research focuses on the offline mode, where the shapes of all items to be packed are predetermined. Decisions regarding placement and orientation are made sequentially based on this information. This aligns with our company's plate nesting problem in contrast to online mode, where the dimensions of the next item's shape to be packed only become known once the current item has been packed (Xu et al., 2016).

For our particular case, where all items need to be cut, the set of large items must be sufficient to accommodate them. Following Wäscher et al (2007), this means we are in the input minimization class. The basic type of problems for offline packing of 2D (irregular) items are:

- **Open Dimension Problem**; Involves accommodating all small items within large objects where the extension in at least one dimension of the large objects can vary. The 2D strip packing problem falls under the open dimension problems.
- **Cutting Stock Problem**; Requires completely allocating a weakly heterogeneous assortment of small items to a selection of large objects with fixed extensions in all dimensions. The large objects can consist of identical objects, but it could also be a weakly or strongly heterogeneous assortment.
- **Bin Packing Problem**; Involves completely assigning a strongly heterogeneous assortment of small items to a set of identical or heterogeneous large objects, aiming to minimize the number or total size of the necessary large objects.

Given the company's need to handle various small items and fixed sheets, our plate nesting problem can be categorized as a 2DIBPP, particularly a two-dimensional single bin size bin packing problem (SBSBPP) with irregularly shaped pieces, since we assume identical sheets in one nesting. Notable characteristics of our problem include irregular shapes with concavities, permission for continuous rotation of pieces due to homogeneous material, and the typical requirement of multiple stock sheets to satisfy demand sets.

3.2 Geometry overview

Geometric complexity presents a significant challenge in nesting problems, particularly in determining whether pieces overlap, touch, or remain separate on a sheet. This necessitates sophisticated computational tools to analyze piece placements on sheets. Various geometric approaches exist, ranging from simple to complex, each influencing model types, solution accuracy, implementation time, and computational outcomes (Leao et al., 2020). The geometric tools are mainly utilized during method searches and pre-processing phases to manipulate computational and mathematical representations of pieces, boards, and solution layouts.

3.2.1 Pixel/Raster method

The pixel/raster method divides the continuous stock sheet into discrete units called pixels or squares, with each assigned a value indicating its occupancy states by a piece. This reduces the geometric complexity into a grid matrix. Proposed coding schemes vary, often tailored to suit specific placement algorithm leveraging this geometric information (Bennell & Oliveira, 2008).
For example, Oliveira and Ferreira (1993) introduced a binary scheme, where a value of 1 represents the existence of a piece, while 0 denotes empty space, see Figure 3.1. A value higher than 1 indicates overlap of pieces. This straightforward approach allows for easy representation of piece placement by adding the piece matrix to the layout matrix, with each cell value indicating the number of pieces occupying that position.

(A) The raster representation. (B) The pixel representation.

FIGURE 3.1: $0-1$ Representation for irregular pieces.

Alternatively, Babu and Babu (2001) proposed a coding scheme where 0 represents the inner part of pieces, and adjacent pixels are assigned increasing numbers from right to left. This scheme has benefits when using a bottom-left placement heuristic, based on movement over the layout, as it allows multiple cells to be skipped at once.
Raster methods offer simplicity and versatility, handling both convex and non-convex polygons. They facilitate straightforward piece movements and contact resolution by counting cells in the desired direction. However, they require substantial memory resources and struggle to accurately represent pieces with non-orthogonal edges. Increasing grid size for better accuracy exacerbates memory usage and extends processing times for feasibility checks (Bennell & Oliveira, 2008).

3.2.2 Direct trigonometry

Unlike raster methods, direct trigonometry uses polygons directly, with information proportional to the number of vertices rather than the size of the pieces or layout. However, assessing feasibility or placement quality with direct trigonometry requires additional evaluation methods.
One such method involves employing direct trigonometry tests for line intersection and point inclusion, which are more computationally complex compared to raster methods. While raster methods have quadratic time complexity for feasibility checks, direct trigonometry exhibits exponential complexity based on the number of edges in the polygons (Bennell & Oliveira, 2008).

A comprehensive approach for evaluating overlap between polygons involves hierarchical tests, starting with bounding box checks. These checks determine if the bounding boxes of polygons or their edges overlap before proceeding to more detailed analyses. Tests for edge intersection and vertex inclusion further refine the evaluation process (Bennell & Oliveira, 2008).

The D-function, denoted as D_{ABP} (see equation 3.1), plays a crucial role in determining the relative position of points (represented by P) with respect to oriented edges (represented by AB), aiding in identifying edge intersections. Additionally, the use of bounding boxes significantly reduces computational intensity by minimizing unnecessary calculations (Bennell & Oliveira, 2008).

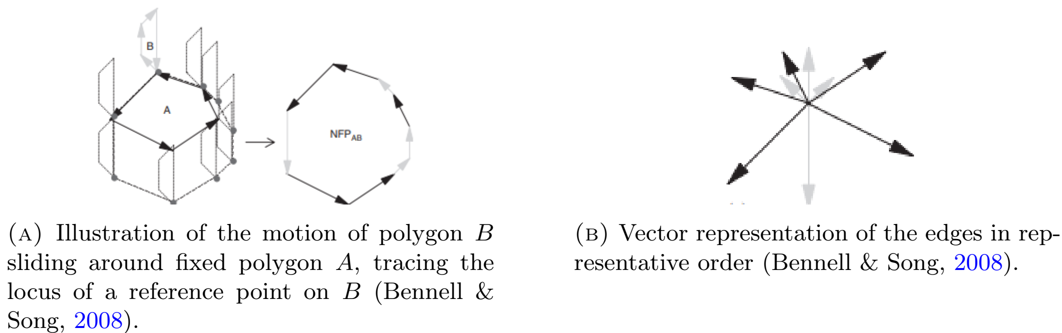
$$D_{ADP} = ((X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P)), \tag{3.1}$$

Although direct trigonometry provides precise representations and accurate overlap assessments, its computational intensity, especially with floating-point calculations, poses challenges. Consequently, iterative search heuristics may not be optimal for direct trigonometry methods, as recalculations are required for each polygon placement change. However, constructive algorithms can effectively

utilize direct trigonometry to address geometric nesting problems by sequentially analyzing pieces (Bennell & Oliveira, 2008).

3.2.3 No Fit Polygons

The No Fit Polygon (NFP) method is a cornerstone tool in irregular shape cutting and packing problems. Initially introduced by Adamowics and Albano (1976), the concept of NFPs revolves around studying the relative positions of two polygons to prevent overlap while ensuring they remain in a touching position. Mahadevan (1984) contributed by presenting an algorithm for building NFPs, while Cunninghame-Green (1989) proposed an approach to generate NFP_{AB} by tracing one polygon, A, around another, B.



(A) Illustration of the motion of polygon B sliding around fixed polygon A, tracing the locus of a reference point on B (Bennell & Song, 2008).

(B) Vector representation of the edges in representative order (Bennell & Song, 2008).

FIGURE 3.2: NFP method representation for convex polygons.

Given two polygons, A and B, the NFP is created by tracing one shape around the boundary of another. Polygon A remains fixed in position while polygon B moves around its edges, ensuring they touch but never intersect. In order to create the NFP_{AB} object, a reference point from B, usually the bottom-left vertex, is chosen which will be traced as B moves around A. The reference point must maintain the relative position with respect to polygon B as this is required when using the NFP to test for overlap. Usually the origin of A is at $(0, 0)$, but it can also be placed at an arbitrary position (x, y). In this case, the position of B's reference point must first be transposed by $(-x, -y)$ before testing its relative position with NFP_{AB} (Bennell & Oliveira, 2008).
To determine overlap, NFP_{AB} and B's reference point are used. If the reference point is inside NFP_{AB}, it overlaps with polygon A. If it is on the boundary, they touch, and if it is outside of NFP_{AB}, they are separate (E. K. Burke et al., 2007).

The process of calculating the NFP for two polygons, differs significantly depending on whether the shapes are convex or non-convex. For convex polygons, the method introduced by Cunninghame-Green (1989) is straightforward. It involves converting the edges of both polygons into vectors, aligning them to a common origin, and then combining them in a specific order to form NFP_{AB}, see Figure 3.2. However, dealing with non-convex polygons requires additional considerations, as it fails to preserve the edge order. Three main approaches are commonly used: the orbiting algorithm by Mahadevan (1984), decomposition into star-shaped or convex polygons by Li and Milenkovic (1995), and Minkowski sums, employed by various authors (Bennell & Song, 2008; Bennell et al., 2001; Dean et al., 2006; Ghosh, 1991; Milenkovic et al., 1991).
The sliding or orbiting method, mimics the movement of one polygon sliding around another. It starts with the highest point of the sliding polygon touching the lowest point of the fixed polygon. NFP vertices are determined by point-edge combinations sliding against each other counterclockwise. However, this method cannot detect feasible positions for one polygon inside possible holes of the other.

Another approach involves decomposing non-convex polygons into smaller convex polygons. Subpieces overlapping indicate polygon overlap, but assembling the final NFP can be challenging due to the required heuristic efficiency.
Considered the most elegant, Minkowski sums involve adding all vector points in one polygon with

those in the other. This method requires opposite orientations of the polygons' vectors. Ghosh (1991) introduced the slope diagram for this purpose. While effective for non-convex polygons, complexity increases with more concavities. An alternative proposed by Bennell et al. (2001) replaces concavities with dummy edges to simplify the calculations.

The NFP method offers computational efficiency, particularly in pre-processing phases, with an algorithmic complexity of $O(n)$, where n is the edge count of the NFP (Bennell & Oliveira, 2008). More efficient that direct trigonometry and its application extending beyond the cutting and packing problems to include stock sheet optimization, as demonstrated by the inner-fit polygon (IFP) concept, implementing the NFP tool can be challenging.

3.2.4 Phi-function

The ϕ-function, introduced by Stoyan et al. (2002,2004), serves to describe all possible relative positions of two polygons, extending beyond the NFP concept. While primarily explored by Stoyan et al., its broader adoption is hindered by the lack of an universal algorithm for arbitrary shapes (Bennell & Oliveira, 2008).
The ϕ-function yields a value indicating the interaction between two objects, where positivity signifies separation, negativity denotes overlap, and equality to zero indicates touching without overlapping. Normalising the ϕ-function produces the Euclidean distance between the objects.

(A) Representation of all touching positions of two circles.

(B) Plot of the phi-function for two circles.

FIGURE 3.3: Example of phi-function.

Consider two circles, see Figure 3.3a, an an illustrative example. If circle 1 is anchored at the origin, the equation $\sqrt{x^2 + y^2} = r_1 + r_2$, describes all the coordinate positions of circle 2 such that their boundaries touch without overlapping. This equation defines the ϕ-function for two circles. Generalising this for circles positioned at arbitrary coordinates yields the expression: $\Phi(x_1, y_1; x_2, y_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} - (r_1 + r_2)$
The ϕ-function offers insights beyond NFPs, aiding geometric analyses in nesting optimization problems. However, the derivation of ϕ-functions for complex shapes relies heavily on trigonometric principles and manual calculation, posing a barrier to widespread implementation.

3.3 Selection heuristics for offline BPP

The 2DIBPP is often decomposed into two sub-problems: piece allocation and piece packing. In the piece allocation stage, pieces are assigned to bins according to some selection heuristic. This section explores various selection heuristics commonly employed in this stage (Coffman et al., 2013; López-Camacho et al., 2013).

- **Next Fit (NF) and Next-K-Fit (NKF)**; NF operates by packing each subsequent item into the bin containing the last packed item, and opening a new bin if necessary. NKF, a variant of NF, keeps the last k bins open and selects the first available bin where the item fits.

- **First Fit (FF)**; FF packs items into the first bin where they fit, opening new bins as necessary. Unlike NF and NKF, FF considers all partially filled bins as potential candidates for the next piece. Additional variants of FF include First Fit Decreasing, which sorts items by decreasing area before packing, and First Fit Increasing, which sorts items by increasing area before packing.
- **Best Fit (BF)**; BF prioritizes open bins based on increasing free area. It places each item in the first bin where it fits, aiming to minimize waste. Best Fit Decreasing is a variant of BF, that sorts the items by decreasing area before assigning them.
- **Worst Fit (WF)**; WF prioritizes open bins based on decreasing free area, thus placing each item in the first bin that has the largest available room. Almost Worst-Fit, a variant of WF, packs items in the second emptiest bin available.
- $\frac{1}{3}$ **Djang and Finch (DJD)**; DJD is a heuristic that prioritizes placing items in a bin, starting with the largest item until the bin reaches a threshold of at least $\frac{1}{3}$ fullness. It then attempts to find combinations of items, one, two, or three, that fill the bin completely or partially, incrementally increasing allowed waste until a suitable combination is found. If any combination fails, it opens a new bin. DJD requires a placement heuristic, just like other selection heuristics, to determine the precise positioning of items within the bin.
  There are also variants that use thresholds of $\frac{1}{2}$ or $\frac{1}{4}$.
- **Partial Bin Packing (PBP)**; PBP focuses on assigning pieces to a single bin, aiming to avoid excessive reassignment and the tendency of greedy algorithms to pack small pieces early (Martinez-Sykora et al., 2017). It employs a knapsack formulation to maximize the value of the bin by selecting and packing larger pieces first.
- **Direct Construction Heuristic (DCH)**; DCH, in contrast to PBP, dynamically allocates pieces during packing, arranging them by area and sequentially placing them into open bins, employing specific packing algorithms. This process continues until all pieces are accommodated (Wang et al., 2022).

3.4 Placement heuristics

Once a piece and bin are selected, the placement heuristic evaluates the candidate's position. It determines how the piece is finally positioned within the bin. This step is computationally intensive due to the need for repeated geometric algorithms in every attempt (López-Camacho et al., 2013).

(A) Bottom-left heuristic from López-Camacho et al. (2013).

(B) Positions considered in the Constructive Approach (López-Camacho et al., 2013).

FIGURE 3.4: Placement heuristics.

- **Bottom-Left (BL)**; BL, a widely used placement heuristic, starts by positioning the piece at the top-right corner of the bin and then slides it down and left until further movement is impossible. If the final position does not overlap with the bin's boundaries, the piece is placed here. BL does not allow pieces to skip around each other, and its performance depends heavily on the initial ordering of the pieces. Despite this, BL's advantage lies in its simplicity and speed. BL is most often used to obtain an initial position for each piece, where the candidate positions of the pieces are obtained using some geometrical tool, usually NFP.

- **Minimum Length (ML)** ML selects the position that minimizes the length from the origin of the bin to the right-most x-coordinate of the last placed piece (Abeysooriya et al., 2018). The piece is oriented to minimize the right-most x-coordinate.
- **Maximum Uilization (MU)** MU selects the position that provides the maximum area utilization in the earliest bin (Abeysooriya et al., 2018). It computes the utilization rate based on the area of the convex hull of already placed polygons and the area of the newly placed polygon.
- **Constructive Approach (CA)**; CA begins by placing the first piece at the bottom-left of the bin, serving as the starting point. For each placed piece, alternative positions, using the maximum and minimum coordinates of the placed piece, are computed and stored. These positions are typically determined based on the dimensions of the pieces and the remaining space in the bin, with overlapping positions discarded. Among the remaining positions, the bottom-left one is chosen. Using the corners of the bin as departure points allows CA to reach certain gaps between pieces.
- **Constructive Approach (Minimum Area) (CAA)** This variant of CA, selects positions that minimize the area of the bounding rectangle enclosing all placed pieces. The bounding rectangle area can be computed with the product of the maximum horizontal coordinate and the maximum vertical coordinate of all placed pieces, including the newly placed piece. This criterion aims to compact the pieces tightly together, reducing wasted space within the bin.

3.5 Solution methods/algorithms

Over the years, researchers have proposed various solution methods for tackling the complex two-dimensional irregular bin packing problem (2DIBPP), ranging from heuristic algorithms to exact methods. While exact methods often involve mathematical mixed-integer linear programming models and some even consider non-convex shapes, the complexity of 2DIBPP renders exact solutions nearly infeasible within reasonable time frames, especially for larger instances. Hence, heuristics and meta-heuristics typically take precedence over the exact methods (Guo et al., 2022). This section outlines the most relevant algorithms for our problem.

3.5.1 Exact methods

Various mathematical approaches have been explored to tackle (irregular) cutting and packing problems. Linear programming (LP) techniques, as demonstrated by Silva et al. (2010), and mixed-integer linear programming (MIP) methods, used in studies by Santoro and Lemos (2015), Cherri et al. (2016,2018), among others, aim to precisely represent the packing process, ensuring compliance with constraints that prevent overlaps between parts while fitting them within the sheet. Examples such as the Dotted-Board model by Toledo et al. (2013) and the QP-nest algorithm by Jones (2014) illustrate different strategies in this domain, while incorporating non-convex shapes. However, despite their accuracy, these exact methods face challenges with larger problem instances, more than > 21 items, due to the complexity of determining constraints, especially for shapes with numerous vertices.
Furthermore, all these MIP models are designed to solve irregular strip packing problems, revealing a disparity in research emphasis compared to LP methods addressing the cutting stock/bin packing problem. This difference in focus might stem from the inherent complexity of 2DIBPP, which are generally more complex than open dimension irregular cutting problems due to the additional constraints imposed by the bin boundaries and the need to efficiently utilize available space within each bin. Consequently, the mathematical modeling and optimization techniques required for 2DIBPP are more demanding.

While exact methods promise optimal solutions, their computational demands often make them impractical for real-world applications. To address this, exact methods are often complemented by heuristic approaches, which, although less precise, offer more practical solutions for larger or realistic problem sizes. For instance, matheuristics, as proposed by Martinez-Sykora et al. (2017),

integrate mathematical models into heuristic frameworks, providing a balance between accuracy and computational efficiency in tackling the 2DIBPP.

3.5.2 Heuristics

Heuristic techniques offer approximate solutions for the 2DIBPP, prioritizing speed over solution accuracy. Nesting irregular parts presents significant challenges due to its complexity and NP-complete nature. Ensuring geometric feasibility, where pieces must not overlap and fit entirely within the sheet, adds to the complexity, given the irregular and non-convex shapes involved. Bennell and Song (2010) introduced a beam search heuristic, laying the foundation for subsequent research.

While early heuristic methods, such as López-Camacho et al.'s (2013) extension of the Djang and Finch heuristic, contributed to the field, they were limited by their inability to handle piece rotations. Cai et al. (2023) proposed novel heuristics allowing limited rotations, utilizing block-based optimization techniques to enhance space utilization efficiency. These methods combine pieces into blocks to complement the shapes of the pieces and reduce wasted bin space, employing operations like fine-tuning, movement, and swap to further increase the bin utilization.
Martinez-Sykora et al. (2017) pioneered free rotation considerations, offering diverse construction algorithms. They used several integer programming models to assign pieces to bins and an MIP model to place pieces into each bin. Abeysooriya et al. (2018) proposed a heuristic method based on the Jostle algorithm, which simultaneously solved both piece allocation and placement, with two strategies for dealing with piece rotation, namely four rotations and free rotations. Wang et al. (2022) considered additional manufacturing parameters, developing an algorithm that accounts for product spacing and free rotation of pieces. Their approach employs a mathematical model to handle product spacing and various algorithms to solve the piece allocation and packing, concluding with a local search to improve the solution.

3.5.3 Meta-heuristics

Meta-heuristics aim to strike a balance between the speed of heuristics and the precision of exact methods, seeking solutions close to optimal within a reasonable time frame. These high-level heuristics delegate tasks to low-level heuristics to find good, though not necessarily optimal, solutions to optimization problems. Meta-heuristics encompass a wide range of algorithms, including Genetic algorithms (GAs), Simulated Annealing (SA), Tabu Search (TS), and Particle Swarm Optimization (PSO), among others.

GAs, inspired by genetic laws, encode shape sequences as individuals and iteratively generate new solutions through selection, crossover, and mutation operations (Goodman et al., 1994; Mundim et al., 2017; Tay et al., 2002). SA, mimicking the annealing process of metals, optimizes objective functions by simulating particle movement during temperature changes (Mundim et al., 2018). TS, preventing backtracking during search iterations, maintains a list of recent moves to guide exploration (E. Burke et al., 2006; Rao et al., 2021). PSO, modeling organism clustering behavior, emphasizes collaboration and competition among individuals to guide the search towards optimal solutions (D. Liu et al., 2008; Omar & Ramakrishnan, 2013).

While many evolutionary algorithms in the literature are used to solve the regular 2D bin packing problem (Jakobs, 1996) or open dimension problems, their application to the 2DIBPP is less common due to its inherent complexity and additional restrictions. The vast search space and geometrical complexity of the 2DIBPP pose challenges for GAs, often leading to difficulties in escaping local optima (Goodman et al., 1994; Y. Yang et al., 2024), and dealing with non-convex polygon types (Jakobs, 1996; Shalaby & Kashkoush, 2013). Furthermore, the stochastic nature of evolutionary algorithms introduces randomness and unpredictability, leading to different results in identical runs. Achieving high-quality solutions heavily relies on meticulous parameter tuning (López-Camacho et al., 2013).

Notable papers addressing the 2DIBPP with meta-heuristics, including limited rotation, include (Guerriero & Saccomanno, 2023; Q. Liu et al., 2020; López-Camacho et al., 2014; Zhang et al., 2022). López-Camacho et al. (2014) proposed an evolutionary selection and constructive hyper-heuristic approach, combining single heuristics, but without allowing piece rotation. Guerriero and Saccomanno (2023) developed a dynamic hierarchical hyper-heuristic approach that explores the space of low-level existing heuristics and decides when and where to apply each single low-level heuristic based on the problem's characteristics. Liu et al. (2020) proposed a constructive solution approach, including limited rotations, where pieces are assigned to bins using the FFD strategy. The placement problem is addressed by the bottom-left algorithm and the pieces exchange method. Additionally, a greedy local search approach is executed to improve solution quality. An extension of this work is given in Zhang et al. (2022), where a waste least first decreasing is introduced to assign pieces to bins. An overlap minimization approach, based on a separation algorithm, is used to address the placement problem. To further improve the solution, a greedy local search approach relying on pieces swapping between two bins is proposed.

3.6 Method selection

From the proposed methods, we identified the most promising approaches that effectively tackle the 2DIBPP within reasonable time frames while considering shape complexities of the polygons involved. These methods are summarized in Table 3.1. The chosen method for company C will be guided by its alignment with our objectives and company C's critical nesting requirements and its ability to produce good results.

The primary objective of the nesting problem is to minimize the number of sheets used while ensuring that items are fully contained within the sheet and do not overlap. However, solely minimizing this may lead to suboptimal solutions, as will be explained in Sections 3.7 and 4.1. To address this, the referenced papers employ a utilization efficiency function, with some also calculating the fractional number of bins used to evaluate their results.
The criteria in Table 3.1 are crucial for company C's needs. Piece rotation enables more flexible item placement, enhancing material utilization, especially with non-convex shapes. Product spacing is essential for maintaining cut quality, and handling non-convex instances is vital given company C's sometimes complex geometries. Accurately modeling polygon geometry ensures the algorithm aligns with actual production shapes.
These criteria are connected to the features and limitations highlighted in Table 2.1, where different nesting software solutions were evaluated. For example, some commercial and open-source solutions in Table 2.1 do not fully support handling non-convex instances or product shaping, making them less suitable for company C.

Benchmark instances, commonly used for evaluation and comparison among other papers, include those from Terashima-Marin et al. (2010), called JP1, and López-Camacho et al. (2014), called JP2, as well as instances from irregular strip packing benchmarks available on the ESICUP website (Martinez-Sykora et al., 2017). These test instances provide relevant comparisons and will be explained in greater detail in Chapter 5 and Appendix C.
Among these benchmarks, irregular strip packing instances offer the most relevant and realistic comparisons. After comparing the utilization efficiency across the papers for each instance, Zhang et al. (2022) yielded the best overall results, closely followed by Wang et al. (2022), with Wang et al. (2022) occasionally outperforming Zhang et al. (2022) when product spacing was excluded.

Based on these insights, and Table 3.1, it was chosen to adopt and adapt the methodology proposed by Wang et al. (2022). This decision is driven by its alignment with our key criteria and ability to generate high-quality solutions. This paper incorporates several methods from other works, including the PBP assignment strategy from Martinez-Sykora et al. (2017), the overlap minimization method from Zhang et al. (2022), and a two-stage free rotation angle method from Abeysooriya et al. (2018).
By adopting Wang et al.'s approach, we will use the NFP method described by Burke et al. (2007),

which refines Mahadevan's (1984) method. We will also integrate the BL placement heuristic, overlap calculation technique, and an assignment strategy from Martinez-Sykora et al. (2017). We will adapt their product spacing method to work for non-convex cases as well and use a different optimization strategy.
Wang et al. (2022) implemented their algorithm in Python, in stead of C++ as Zhang et al. (2022) did. Their average execution time to find the most optimal solution across all benchmark instances, using the Local-Search step, is approximately 1180 seconds. Translating the program into a language like $C++$, the preferred language for implementation within company C, could potentially further accelerate the algorithm's performance. Overall, Wang's method represents a recent advancement in the field, building on previous techniques like Abeysooriya et al.'s free rotation method and Zhang et al.'s overlap minimization approach.

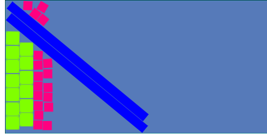
Method \ Criteria	Piece rotation	Free Rotation	Convex instances solved	Non-convex instances solved	Product spacing	Polygon geometry used
López-Camacho et al.(2013)	No	No	Yes	No	No	Yes
López-Camacho et al.(2014)	No	No	Yes	Yes	No	Yes
Martinez-Sykora et al.(2017)	Yes	Yes	Yes	Yes	No	Yes
Abeysooriya et al.(2018)	Yes	Yes	Yes	Yes	No	Yes
Liu et al.(2020)	Yes	No	Yes	Yes	No	Yes
Zhang et al.(2022)	Yes	No	Yes	Yes	No	Yes
Wang et al.(2022)	Yes	Yes	Yes	Yes	Yes	Yes
Guerriero and Saccomanno(2023)	No	No	Yes	No	No	Yes
Cai et al.(2023)	Yes	No	Yes	Yes	No	Yes

TABLE 3.1: Comparison between the 2DIBPP solution methods.

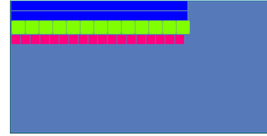
3.7 Research gap

The scientific contribution of this thesis lies in developing a more complete and effective method for distinguishing between solutions that utilize the same number of sheets and picking the best layout that maximizes remnant sheet area. This distinction is crucial because in many industrial contexts, simply minimizing the number of sheets used does not necessarily lead to the most efficient or cost-effective layout. Current research often overlooks the importance of maximizing remnant sheet area, which is directly tied to minimizing waste and improving material utilization.

As discussed in Section 2.4.1, the company's current optimization method can result in significant unusable space, as illustrated in Figure 2.11a. This inefficiency highlights the need for additional metrics beyond just the number of sheets used to better evaluate and compare solution layouts. For example, Figure 2.11b demonstrates how a larger remnant sheet can be achieved, underscoring the potential for more optimal layouts even when the number of sheets remains constant.
Wang et al. (2022) proposed using a utilization efficiency metric during their local search phase to identify the solution with the highest utilization rate among those using the same number of sheets. However, this approach is limited in its ability to distinguish between different layout efficiencies, as shown by the identical efficiency values for the different layouts in Figures 3.5a and 3.5b. The core issues stems from the metric's reliance on the area of items placed in the bin, which fails to account for how efficiently the remaining sheet space is utilized.
To address this shortcoming, another measurement function is necessary. This is where the K-measurement function comes into play. Defined by Han et al. (2013) and mentioned in some of the aforementioned papers and also in Wang et al. (2022), the K-function calculates the fractional number of bins after applying a horizontal ór vertical cut to the least utilized bin. A lower K-value



(A) Solution generated by company software.



(B) Manual adjusted solution maximizing the remnant sheet by employing an L-shaped cut.

FIGURE 3.5: Two solution layouts featuring identical items arranged on the same sheet.

indicates a superior solution layout. However, while Wang et al. (2022) mention this K-value, they do not incorporate it into their optimization phase, limiting its practical utility. Furthermore, even if the K-value were used, it might not guarantee the selection of the most efficient layout, such as the one depicted in Figure 3.5b. Modifying the definition to apply an L-shape cut to the least utilized bin would improve this, leading to better layout selection.
Therefore, the proposed approach entails implementing a three-level heuristic. The first level focuses on minimizing the number of sheets required. The second level aims to maximize utilization rates across all bins, enhancing the overall layout utilization efficiency. Finally, the third level seeks to optimize item placement within specifically the least utilized bin to achieve the lowest possible K-value while preserving the item assignment per bin that yielded the best utilization efficiency. This approach maximizes the remaining sheet area of the least utilized bin and helps favour solutions akin to that depicted in Figure 3.5b over those like Figure 3.5a.

3.8 Conclusion

This chapter reviewed the relevant literature to address the company's plate nesting problem. In Section 3.1, we established that the plate nesting problem can be categorized as a 2D Irregular Bin Packing Problem, specifically a two-dimensional single bin size bin packing problem (SBSBPP) with irregularly shaped pieces. This classification is due to the use of identical sheets in a single nesting scenario. Key characteristics of the problem include the handling of irregular shapes with concavities, piece rotation, and the need for multiple stock sheets to satisfy demand.

Section 3.2 introduced several geometric tools to manage the geometric complexity of convex and non-convex polygons, crucial for determining item overlap and generating candidate placement positions on the sheets. It covered methods such as the pixel/raster method, which simplifies geometric shapes into a grid matrix, and direct trigonometry, which provides precise representations albeit with higher computational demand. Additionally, the No Fit Polygon method and ϕ-function were explored to describe relative positions of two polygons.

Sections 3.3 and 3.4 examined common heuristics for piece allocation and packing, respectively. Section 3.3 discussed various selection heuristics, from basic approaches like Next Fit and First Fit to more advanced approaches like Partial Bin Packing. Section 3.4 focused on placement heuristics used in 2D bin packing problems, including Bottom-Left and Constructive-Approach, which are essential for determining the final placement of items within the bin.

In Section 3.5, we reviewed various solution methods proposed in the literature, including exact methods like mixed-linear programming, as well as heuristics and meta-heuristic approaches such as Genetic Algorithms and Tabu Search. Based on a thorough review and comparison of existing methods, a method selection process is proposed in Section 3.6, culminating in the adoption and adaptation of the algorithm proposed by Wang et al.(2022). This decision is informed by its alignment with critical criteria and ability to produce high-quality solutions, as evidenced by its performance on benchmark instances. The chapter closes by identifying a research gap pertaining to the optimization of solution layouts in irregular bin packing, particularly in maximizing remnant sheet area. To address this gap, a three-level heuristic approach is proposed, integrating additional measurements like the K-function to distinguish between solution layouts.

Chapter 4: Solution design

This chapter outlines the developed algorithm for the 2D irregular bin packing problem, addressing the research question: "*What should be the design of the algorithm?*". It begins with a formal problem statement and then provides an overview of the model, followed by a detailed explanation of each main step, including the optimization phase in Section 4.2.

4.1 Formal problem statement

This section presents a formal description of the 2DIBPP to be solved. In this 2DIBPP, the main elements are the pieces to be packed and the identical rectangular bins used for packing. Let $\mathcal{P} = \{P_i | i = 1, \cdots, n\}$ be the set of n pieces to be packed, each with an associated area s_i. The rectangular bins have fixed dimensions with length L and width W, and we assume there are enough bins to accommodate all pieces. Let $B = \{b_j | j = 1, \cdots, N\}$ be the bin set, with n^j pieces packed into the j_{th} bin. The reference point for piece P_i is r_{P_i} and $R_{\mathcal{P}} = \{r_{P_1}, r_{P_2}, \cdots, r_{P_n}\}$, $i = 1, \cdots, n$ is the set of these reference points. The reference point is designated as the bottom-left vertex of the piece, including when a piece is rotated by an angle θ_i, see Figure 4.1. We always prioritize the leftmost vertex and, if multiple vertices share the same leftmost position, we select the lowest among them.

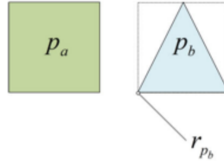


FIGURE 4.1: The reference point r_{P_b} of piece P_b is defined as the bottom-left corner of the piece when rotated to a specific angle (Wang et al., 2022).

Before formally describing the 2DIBPP, we introduce some definitions, where the bins will refer to the sheets for packing the pieces and the pieces are represented by polygons.

Definition 4.1.1 (Translational operator $\oplus$)**.** Given a polygon, P_i, and a translation vector, $v_t = (v_{tx}, v_{ty})$, the operator $\oplus$ describes the translation of the polygon along the vector. For point p_0 on polygon P_i, the translation operator $\oplus$ is defined as: $P_i \oplus v_t = \{(p_{0x} + v_{tx}, p_{0y} + v_{ty} | p_0 \in P_i\}$.

Definition 4.1.2 (Rotation operator $P_i(\theta_i)$)**.** [1] Let the set of allowable rotation angles for piece P_i be ϑ_i, then the rotation angle set for n polygons is $O = \{\vartheta_1, \vartheta_2, \cdots, \vartheta_n\}$, $i = 1, \cdots, n$. Let the coordinates of point p_0 on polygon P_i, with the origin $(0, 0)$ as the reference point, rotated by angle ϑ_i be $p_0^{\theta_i} = (p_{0x}^{\theta_i}, p_{0y}^{\theta_i})$, in which the rotation angle $\theta_i \in \vartheta_i$ and $\theta_i \in [0, 2\pi]$, then we can get the following equation:

$$\begin{bmatrix} p_{0x}^{\theta_i} \\ p_{0y}^{\theta_i} \end{bmatrix} = \begin{bmatrix} cos\,\theta_i & -sin\,\theta_i \\ sin\,\theta_i & cos\,\theta_i \end{bmatrix} \begin{bmatrix} p_{0x} \\ p_{0y} \end{bmatrix} = \begin{bmatrix} p_{0x}\,cos\,\theta_i - p_{0y}\,sin\,\theta_i \\ p_{0x}\,sin\,\theta_i + p_{0y}\,cos\theta_i \end{bmatrix} \tag{4.1}$$

As mentioned, piece packing is the basis for cutting operations. To meet the requirements of the cutting process, a designated slit distance[2] should be reserved between pieces and between pieces and the bin boundaries.
Let the slit distance between any two pieces P_a and P_b be $d_1 = dist(P_a, P_b)$. Let $\partial rect(W, L)$ denote

[1] $\begin{bmatrix} cos\,\theta_i & -sin\,\theta_i \\ sin\,\theta_i & cos\,\theta_i \end{bmatrix}$ is the rotation matrix used to perform rotation in Euclidean space, that rotates points in the xy-plane counterclockwise through an angle θ, about the origin of a 2D Cartesian coordinate system.
[2] Can be seen as product spacing.

the edge of the bin and $intrect(W, L)$ represent the bin's interior. The distance between a piece P_a and the bin's edge is expressed as $d_2 = dist(P_a, \partial rect(W, L))$. The procedure for determining the distance between two pieces and between pieces and the bin is outlined in Section 4.2.1.
Given these definitions, a piece P_a with its reference point r_{P_a} positioned at v_{t_a} with a rotation angle θ_a, can be expressed as $P_a^{v_{t_a}}(\theta_a) := P_a^{\theta_a} \oplus v_{t_a}$. Thus, the solution to the 2DIBPP entails four key elements: the required number of bins, the type and quantity of pieces allocated to each bin, the rotation angle, and the placement position of each piece's reference point. The primary evaluation measure is the number of bins used: $N = |B|$.

Formally, a 2DIBPP problem can be defined as minimizing N subject to the following constraints:

$$\min N \tag{4.2}$$

s.t

$$\text{dist}(P_a^{\theta_a} \oplus v_{t_a}, P_b^{\theta_b} \oplus v_{t_a}) \geq d_1, \qquad 1 \leq a \leq b \leq n \tag{4.3}$$

$$\text{dist}(P_a^{\theta_a} \oplus v_{t_a}, \partial rect(W, L)) \geq d_2, \qquad 1 \leq a \leq n \tag{4.4}$$

$$P_a^{\theta_a} \oplus v_{t_a} \subseteq intrect(W, L), \qquad 1 \leq a \leq n \tag{4.5}$$

$$\theta_a \in \vartheta_a \quad \text{and} \quad \theta_a \in [0, 2\pi], \qquad 1 \leq a \leq n \tag{4.6}$$

$$r_{P_a} \in \mathbb{R}^2, \qquad 1 \leq a \leq n \tag{4.7}$$

$$N \in \mathbb{Z}^+ \tag{4.8}$$

Constraints 4.3 and 4.4 ensure that the distance between any two pieces, and between a piece and the edge of the sheet are greater than or equal to the specified distances d_1 and d_2 respectively. Constraint 4.5 ensures that a piece fits entirely within the sheet, without thus overlapping its boundaries. Constraint 4.6 ensures that the rotation angle of each piece lies within its allowable range. Constraints 4.7 and 4.8 ensure that the reference points are real numbers and the number of bins are positive integers.

However, it is foreseeable that solutions with an identical number of bins may arise. Consequently, merely counting the bins does not provide a means to differentiate between such solutions in our nesting problem. To address this, we need some additional functions that account for the reuse of residual space by partitioning the least utilized bins horizontally and vertically, thereby isolating the unused portions for future use, see Equations (4.9) - (4.11).

$$U_j = \frac{\sum_{j_m=1}^{n^j} s_{j_m}}{L \times W} \tag{4.9}$$

$$F = \frac{\sum_{j=1}^{N} U_j^2}{N} \tag{4.10}$$

$$K = N - 1 + P^* \tag{4.11}$$

Here, F serves as the utilization efficiency metric, maximizing the percentage of each bin's utilization and therefore can also be used to aid in maximizing the residual material available for reuse. U_j denotes the utilization rate of the j_{th} bin, where s_{j_m} is the area of the m_{th} piece within the j_{th} bin. If the m_{th} piece in the j_{th} bin has t vertices, namely $p_{j_m}^1(x_1, y_1), p_{j_m}^2(x_2, y_2), \cdots, p_{j_m}^t(x_t, y_t)$, the formula for s_{j_m} is based on the Shoelace formula (AoPSOnline, 2024) and looks as follows:

$$s_{j_m} = \frac{1}{2}\left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} + \cdots + \begin{vmatrix} x_{t-1} & y_{t-1} \\ x_t & y_t \end{vmatrix} + \begin{vmatrix} x_t & y_t \\ x_1 & y_1 \end{vmatrix} \right) \tag{4.12}$$

Here $|.|$ represents the determinant of a $2x2$ matrix formed by the coordinates of consecutive vertices of the polygonal piece.
P^* is the percentage of utilization corresponding to the least utilized bin after it has been vertically and horizontally partitioned. K then quantifies the fractional number of bins.

So, to account for cases where solutions with an identical number of bins arise, we introduce

the following lexicographic model to differentiate between such solutions.

$$\min N \tag{4.13}$$

$$\max F = \frac{\sum_{j=1}^{N} U_j^2}{N} \tag{4.14}$$

$$\min K = N - 1 + P^* \tag{4.15}$$

s.t constraints 4.3 – 4.8.

When faced with solutions containing an equal number of bins, prioritizing further on material reuse leads to favoring solutions with higher F values and lower K values. Essentially, the final aim is to minimize $P^* = K - (N - 1)$ to select the most efficient solution. U_j incentivizes highly utilized bins while promoting the emptying of less utilized ones, aiding in the reduction of total bins[3]. To facilitate the reuse of residual material, horizontal and vertical cuts are implemented to isolate unused portions of bins for future use, focusing on the least utilized bin.

4.2 The developed algorithm

This section provides a comprehensive overview of the developed model for 2D irregular bin packing, illustrated through a flowchart in Figure 4.2. Each step of the algorithm is explained in more detail to elucidate its functionality.

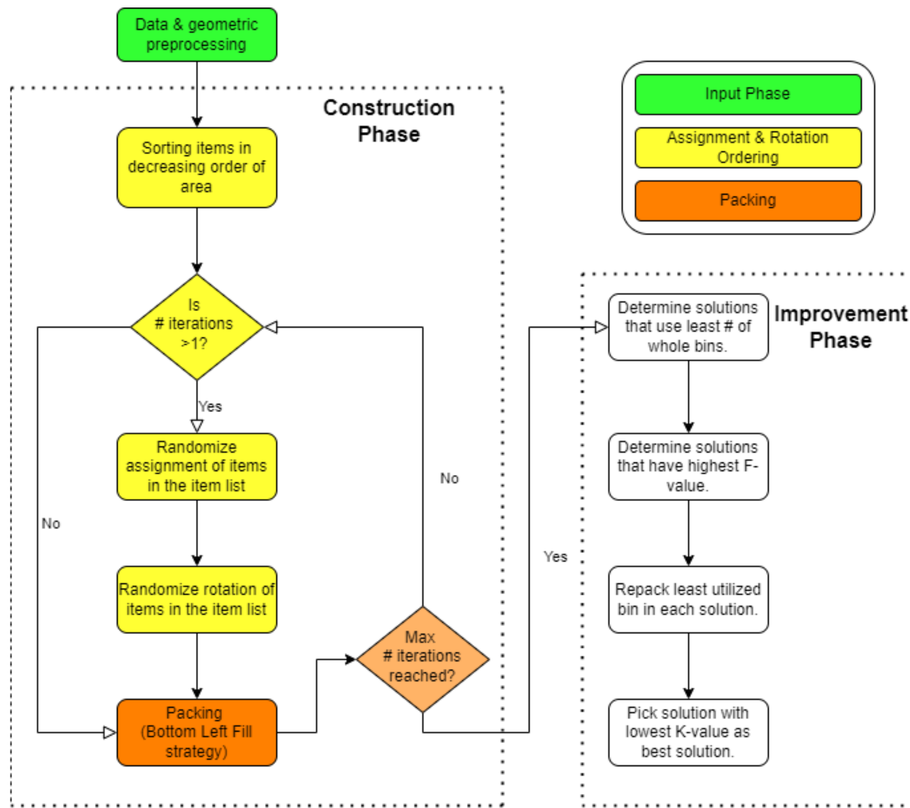


FIGURE 4.2: Flowchart of the developed algorithm for 2D irregular bin packing.

4.2.1 Data input phase

The input phase is a critical step in the developed model, setting the foundation for all subsequent processes. It involves the collection, organization, and preliminary geometric processing of data essential for computational analysis.

[3]When applied in a local search.

Data collection and input parameters

Data collection initiates with gathering the polygons to be packed along with their respective geometries. For one packing solution, we use one dataset. Each dataset comprises multiple polygons defined by their vertex coordinates $[x, y]$, accompanied by information on the quantity of each polygon type to be packed. For example, Dataset A may include two polygons with vertices $[[0.0, 86.0], [966.0, 142.0], ...]$, while Dataset B could contain four polygons with vertices $[[0.0, 173.0], [1761.0, 0.0], [2183.0, 650.0], ...]$ and one polygon with vertices $[[10.0, 50.0], [200.0, 3.0], ...]$. The datasets can include regular and irregular shapes that can be both convex and concave.
Furthermore, the model requires inputting the dimensions of identical rectangular packing sheets where the polygons will be placed. The model dynamically generates new sheets of the same dimensions as required during the packing process, ensuring flexibility to accommodate varying numbers of polygons.

Geometric preprocessing

To incorporate product spacing during the packing process, denoted by slit distances d_1 and d_2, the polygon edges are expanded outward by $\frac{1}{2}d_1$, inspired by the method described by Wang et al. (2022). This process termed the equidistant method, ensures that adjacent edges of each polygon intersect at consistent points, thereby achieving uniform spacing between polygons on the sheet.
The equidistant method takes an original polygon defined by its vertex coordinates from the dataset as input. It also requires an offset distance d_1, which dictates the spacing between polygons on the sheet. The output is a new set of vertices defining the offset polygon, ensuring a uniform distance of $\frac{1}{2}d_1$ between corresponding points of the original and offset polygons. This will further ensure that two offset polygons are exactly d_1 apart from each other and that one offset polygon is exactly $d_2 = \frac{1}{2}d1$ apart from the boundaries of the sheet. See Figure 4.3 depicts the result of a piece after the equidistant offset operation.

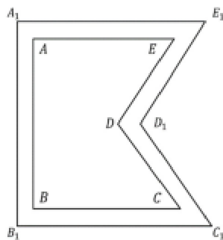


FIGURE 4.3: Illustration of equidistant offset, $A_1B_1C_1D_1E_1$, of a polygon, $ABCDE$, taken from Wang et al. (2022).

Implementation of the equidistant method involves some geometric and trigonometric calculations:

- Traversing through each vertex coordinate of the original polygon;
- Computing vectors for each pair of adjacent edges and determining their cross product to ascertain their orientation; If the cross product of vector 1 and 2 is negative, vector 1 lies in the counterclockwise direction of vector 2. The opposite accounts if the cross product is positive. If the cross product is equal to 0, the vectors are co-linear.
- Calculating the angle bisector between adjacent edges to derive the direction of the offset;
- Adjusting vertex coordinates based on the computed offsets to generate the equidistant polygon vertices.

The complete pseudo-code and additional explanation of this equidistant method can be found in Appendix D.

Geometric preprocessing also includes adopting the NFP method from Burke et al. (2007) to prevent overlap between pieces and ensure that they do not extend beyond the boundaries of the

packing sheet. The necessary code and additional geometric features and show options are based on the implementations from S. Yang (2024).

Our algorithm provides flexibility to optimize computational efficiency by offering two options for handling NFPs. Users can choose to precompute the NFPs and store them in a list for quick access during packing, using the NFPAssistant program. Alternatively, NFPs can be calculated dynamically as needed during the packing phase, reducing the overall number of NFP calculations. Additionally, the Inner Fit Polygon (IFP), also referred to as the Inner Fit Rectangle (IFR), can be computed similarly to NFPs. This involves translating the sliding polygon around the interior boundary of the sheet. The path traced by the polygon's reference point defines the IFP/IFR, see Section 3.2.
These preprocessing steps ensure that each polygon is correctly prepared with the required spacing, facilitating efficient packing without overlap and ensuring that pieces do not extend beyond the boundaries of the packing sheet(s).

4.2.2 Assignment strategy

After inputting the data and performing the necessary geometric calculations, we proceed to the assignment and rotation phase. The method of assigning items to bins significantly influences the packing outcome. One common approach is to pack items into a bin until it is full, then close it and open a new one (Parreño et al., 2010). Martinez-Sykora et al. (2017) use the 'Simple Construction Heuristic' (SCH), which sorts items by non-decreasing area and places them sequentially in bins using the Next-Fit Decreasing strategy. If none of the items can be placed in the current bin, a new bin is opened for the remaining items. This sorting strategy prioritizes larger items, potentially leading to more efficient packing.

We initially adopt this strategy for the first iteration, as shown in the flowchart. However, to explore alternative packing arrangements with rotations and avoid getting stuck in local optima, we introduce randomness in subsequent iterations. This involves randomizing the order of the initially sorted list and applying random rotations from a predefined set or allowing free rotations. This variability helps the model explore diverse configurations and discover better global solutions.

The randomization degree is controlled by a probability parameter, which governs the likelihood of swapping items in the list. A higher probability encourages exploration of new solutions, while a lower one favors the exploitation of promising current solutions, maintaining closer adherence to the initial order. For example, a low probability (e.g. 0.01) results in minimal exchange, while a high probability (e.g. 0.99) induces substantial exchange, leading to significantly different solutions.
Given the effectiveness of the SCH method as highlighted by Martinez-Sykora et al. (2017), the first iteration often yields satisfactory results.[4] However, to escape local optima and explore markedly different solutions, we progressively increase the randomization probability with each iteration following the formula:

$$1 - e^{\left(-5 * \frac{\text{iteration}}{\text{total_iterations}}\right)}.$$

To justify the choice of -5 as the decay rate, a series of experiments were conducted across a range of alternative decay rates, recording the F-values over 100 iterations. The reasoning behind this iteration count can be found in Section 5.2. Figure 4.4 shows the distribution of F-values across different decay rates for the poly2b dataset with its allowed rotations. The boxplot shows that decay rates corresponding to e-values between e^{-1} and e^{-6} generally yield the highest F-values, indicating more efficient packing. Specifically, the e^{-5} category consistently achieves the highest F-values with minimal variance, suggesting robust performance across different runs. In contrast, higher decay rates (e.g. e^{-10} and beyond) result in sharply decreasing F-values and stabilization around lower values, indicating less efficient packing. This suggests that too rapid an increase in randomization disrupts the algorithm's ability to exploit promising solutions found earlier.

[4] This is also our conclusion from running different experiments

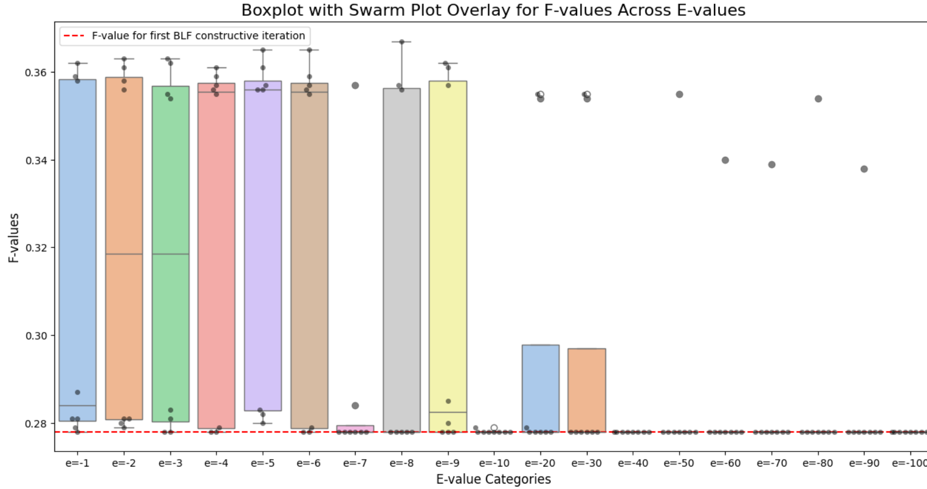


FIGURE 4.4: Impact of different decay rates on packing efficiency.

The experimental data support selecting -5 as the optimal decay rate for randomization probability. This rate furthermore strikes a balance between exploration and exploitation, achieving around 95% randomness at 60% of total iterations, allowing the algorithm to explore diverse packing configurations while converging toward high-efficiency solutions.

Throughout each iteration, the assignment of items, their positions in the bins, and the number of bins used are recorded and stored. This iterative process continues until the maximum number of iterations, a user-defined parameter, is reached. Detailed algorithms for the randomization and rotations are provided in Appendix F.

4.2.3 Packing strategy

Following the assignment phase, the packing phase focuses on efficiently placing these polygons into bins using the Bottom Left Fill (BLF) algorithm. To implement our version of BLF, we used the bottom left point determination, implemented by S.Yang (2024), who in turn got his inspiration from (SVGnest, 2024a). This section details the operational methodology of the BLF heuristic and its implementation specifics.

The BLF is a well-established method for polygon packing that ensures no overlap and optimal space utilization within defined bin dimensions. As discussed in Section 3.4, the performance of BLF relies heavily on the initial ordering of polygons, making exploration of various ordering strategies crucial, hence the randomization.

Initialization and Parameters

The BLF heuristic is initialized with parameters including the bin dimensions and a list of polygons to be packed. Optional parameters may include the NFPassistant for pre-calculating NFPs.

Packing methodology

The BLF heuristic proceeds with the 'place_polygons' method, see Algorithm 3, which iterates through the list of polygons, attempting to place each polygon into the current bin. If the placement of any polygon is no longer feasible due to overlap or boundary constraints, a new bin is initiated.

Algorithm 1 Bottom Left Fill

Input: bin-width, bin-height, polygons, optional-params(NFPAssistant, vertical)
Output: Placed polygons in bins with reference point positions
bins = [ ]
current_bin = [ ]
positions = [ ]
place_polygons()

Algorithm 2 placeFirstPoly

Input: polygon
Output: current_bin, positions
Find the leftmost and bottommost points of the polygon
Slide poly to $(0, 0)$ based on the leftmost and bottommost points
Append poly to current_bin
Record poly reference point in positions

Detailed placement strategy

The actual placing of the polygons is done in Algorithm 4. The method attempts to place subsequent polygons by calculating the IFR and using NFPs to determine available placement areas without overlap. It then slides the polygon to the bottom-left found position. To place the very first polygon in a bin, Algorithm 2 is called. It places the first polygon in the bottom-left corner of the bin to initiate the packing process.

Algorithm 3 place_polygons

```
Input: polygons, current_bin, bins
Output: Modifies bins & positions
Initialize remaining_polygons to a copy of polygons
while remaining_polygons is not empty do
    Set placed_any to False
    for each polygon in remaining_polygons do
        if placePoly(polygon) then
            Remove polygon from remaining_polygons
            Set placed_any to True
        end if
    end for
    if placed_any is False then
        Append current_bin to bins
        Initialize current_bin as empty list
    end if
end while
```

Algorithm 4 placePoly

```
Input: polygon
Output: boolean, current_bin, positions
if current_bin is empty then
    if fits_in_bin(poly) then
        Call placeFirstPoly(poly)
        Return True
    end if
end if
Set adjoin to poly
Calculate Inner Fit Polygon (ifr) for adjoin
Convert ifr to Polygon(differ_region)
for each main_polygon in current_bin do
    Calculate or get No Fit Polygon (nfp)
    Update differ_region by subtracting nfp
end for
if differ_region is empty or invalid then
    Return False
end if
Slide adjoin to Bottom-Left-Point of differ_region
if fits_in_bin(adjoin) then
    Append adjoin to current_bin
    Record adjoin reference point in positions
    Return True
end if
Return False
```

Utility methods

Supporting functions like 'fits_in_bin' and 'getBottomLeft' ensure polygons are checked for fitting within bin boundaries and positioned optimally. The 'fits_in_bin' method is especially useful when dealing with relatively large items that cannot fit in the bin when in a certain rotation.
The 'getBottomLeft' method is crucial in determining the bottom-left point of a polygon based on its coordinates, prioritizing the leftmost and bottommost points for placement.
Every iteration of the BLF concludes by saving the packing results, for further analysis and visualization, using the instance variables *self.bins* and *self.positions*.

Algorithm 5 fits_in_bin

```
Input: polygon
Output: boolean
for each point in poly do
    if point is outside bin boundaries, return False then
    end if
end for
Return True
```

Algorithm 6 getBottomLeft

```
Input: polygon
Output: index of bottom-left point
Initialize bottom_left_points as an empty list to store potential bottom_left_points
Set _min to a large number
for each point pt in poly along with its index i do
    Create a dictionary pt_object with keys: index, x (initialised to pt[0]), y (initialized to pt[1])
    Set target = pt[1] if polygon is oriented vertically, otherwise target = pt[0]
    if target < _min then
        Update _min to target
        Clear bottom_left_points and add pt_object to bottom_left_points
    else if target == _min then
        Append pt_object to bottom_left_points
    end if
end for
if bottom_left_points contains only one point then
    Return the index stored in bottom_left_points[0][index]
else
    Set target to x if polygon is oriented vertically, otherwise to y
    Initialize _min with the target coordinate of the first point in bottom_left_points
    Initialize selected_point with the first point in bottom_left_points
    for remaining points in bottom_left_points do
        if current bottom-left point coordinate < _min then
            Update selected_point to the current point
            Update _min to the primary coordinate of the selected_point
        end if
    end for
    Return selected_point[index] as the index of the determined bottom-left point
end if
```

4.2.4 Optimization phase

The optimization phase focuses on finding the best solution out of all the run iterations, regarding utilization percentages, and trying to optimize it a bit further by trying to maximize the remnant sheet area of the least utilized sheet.
As suggested in Section 3.7, we have implemented a three-level heuristic. The first level aims to locate the solutions that used the least number of whole sheets. The second steps is then calculating the F-values for those solutions that used the minimum number of whole sheets and selecting the one(s) with the highest F-value to continue with. Finally, the third level seeks to optimize the placement of items within specifically the least utilized bin while preserving the item assignment per bin that yielded the best F-value. This last step helps optimize the best-found solutions by maximizing the remnant sheet area of this least utilized bin. The solution that in the end has the lowest K-value, after repacking the least utilized bin to optimize the remnant sheet area, will be chosen as the best solution among all run iterations.
The rest of this section will explain in more detail how to calculate the F-values, how to repack the least utilized bin and how to calculate the K-values to choose the most optimal solution.

F-value calculation & evaluation

After completing the iterations, and saving the output data as mentioned in section 4.2.2, the first step of the optimization phase is the evaluation of the solutions based on the number of whole sheets used. The solution outputs are sorted in ascending order of used number of whole sheets and the solutions with the minimum number of sheets are saved to continue to the next step of the optimization phase.
In this second step, from the saved solutions, the F-values are calculated according to the formula presented in 4.10. The pseudo-code can be found in Appendix G.
The solution(s) with the highest F-value are saved to continue to the next step of the optimization phase.

Repacking

The repacking strategy is part of the last level of the optimization phase. It focuses on optimizing the packing layout in the least utilized bin of every solution saved in the previous step. To achieve this it iteratively tries to maximize the remnant sheet area of the least utilized bin in every solution. This is done by trying to find packing layouts that minimize the P^* metric from the K-function, see equation 4.11. All steps of repacking will be explained in more detail below.

Identifying the Least Utilized Sheet
To start repacking, first the sheet with the lowest utilization rate is identified in each solution from *max F solutions*, using the calculated utilization rates from the F-value calculation step.

Repacking the Least Utilized Sheet
Once the least utilized sheet is identified, it undergoes the repacking algorithm. This algorithm iterates through the possible permutations of the items within the sheet, applying the BLF heuristic to each permutation to repack the items. This approach aims to find the arrangement that maximizes the remnant sheet area, thereby minimizing P^* and, consequently, reducing the K-value.
As you might remember, P^* represents the fraction of the bin effectively utilized by the placed items, determined by isolating the used part of the sheet with vertical and horizontal cuts, resulting in a bounding box. The P^* value is then calculated as the ratio of the bounding box area to the total bin area. You can say that P^* is defined as the proportion of the bin after applying an L-shaped cut to separate the non-utilized part of the bin for future use.

After each iteration of the BFL heuristic, a check is performed to ensure no overlap occurs with the newly packed items. This step is crucial for avoiding unintended overlaps, which occasionally arose in this repacking phase without this verification.
To quantify these potential overlaps during the packing process, we introduce concepts of penetration depth and penetration vector, following the methodology by Wang et al. (2022). When two pieces intersect, the penetration depth, $PD(P_a, P_b)$, represents the minimal distance required to separate pieces P_a and P_b.

Definition 4.2.1 (Penetration Depth and Penetration Vector)**.** The penetration depth between intersecting pieces P_a and P_b, denoted as $PD(P_a, P_b)$, is defined by $PD(P_a, P_b) = \min\{\|v\| | P_a \cap (P_b \oplus v) = \emptyset\}$, where v is the penetration vector, and $\|v\|$ denotes the Euclidean norm of vector v.

The penetration depth, $PD(P_a, P_b)$, can be derived from $NFP_{P_aP_b}$. If the reference point r_{P_b} of P_b lies within $NFP_{P_aP_b}$, $PD(P_a, P_b)$ equals the shortest distance from r_{P_b} to $NFP_{P_aP_b}$. When pieces do not intersect, $PD(P_a, P_b) = 0$.
Similarly, $PD(b_j, P_b)$ denotes the penetration depth between a piece P_b and the boundaries of bin b_j. It is computed using the Inner-Fit-Polygon $IFP_{b_jP_b}$ by calculating the shortest distance between reference point r_{P_b} and the bin's boundary.
To calculate the total overlap, the square of the penetration distances is summed. For a set of pieces $\mathcal{P}_j$ packed into bin b_j, with the set of piece reference point placement positions V_j and the set of piece rotation angles R_j, the overlap $Overlap(\mathcal{P}, V_j, R_j, b_j)$ is defined as:

$$Overlap(\mathcal{P}_j, V_j, R_j, b_j) = \sum_{1 \le a \le b \le n^j} h_{ab}(\mathcal{P}_j, V_j, R_j, b_j) + \sum_{1 \le a \le n^j} k_a(\mathcal{P}_j, V_j, R_j, b_j)$$

Here $h_{ab}(\mathcal{P}_j, V_j, R_j, b_j) = PD^2(P_a^{\theta_a} \oplus v_{t_a}, P_b^{\theta_b} \oplus v_{t_b})$ represents the overlap between pieces P_a and P_b, and $k_a(\mathcal{P}_j, V_j, R_j, b_j) = PD^2(b_j, P_a^{\theta_a} \oplus v_{t_a})$ denotes the overlap between piece P_a and bin b_j.

For the pseudo-code of this repacking algorithm, including overlap calculation, refer to Appendix H.

K-value calculation & evaluation

After completing the repacking for the solutions in *max F solutions*, the K-value is calculated for each solution to identify the 'best' layout following the explained procedure. The K-value, defined by equation 4.11, assumes that all bins, except for the least-utilized bin, are fully utilized. For the least-utilized bin, the K calculation considers the fraction of the bin that is utilized, denoted by P^*. This P^* value is derived as described in the repacking strategy, representing the proportion of the bin occupied by the bounding box around the placed items.
After calculating the K-values for all solutions in *max F solutions*, the solution with the lowest K-value is selected as the best. In cases where multiple solutions have the same number of bins, the same F-value, and the same K-value, any of these solutions can be considered the best. To select one, the algorithm will select the first one stored in the ascending K-value list as the best solution.
The pseudo-code for calculating the K-value can be found in Appendix I.

4.3 Conclusion

In this chapter, we detailed the development and design of an algorithm centered around solving the 2D Irregular Bin Packing Problem by packing a set of irregularly shaped polygons into a series of identical rectangular sheets. The algorithm employs a combination of geometric transformations, assignment heuristics, and optimization strategies to achieve optimal packing layouts.
We began the chapter with a formal definition of the 2DIBPP problem, establishing the mathematical and geometric foundations essential for understanding the constraints and objectives. These constraints include the possibility of maintaining specified distances between pieces, ensuring pieces fit within the sheet boundaries, and adhering to allowable rotation angles.

The developed algorithm was detailed in a structured manner, beginning with the data input phase. This phase involves collecting and preprocessing geometric data to prepare the pieces for packing, ensuring proper spacing using methods like equidistant offsetting and NFP calculations. The flexibility to precompute or dynamically compute NFPs and IFPs adds to the algorithm's efficiency and adaptability.

In the assignment strategy section, various approaches to assigning pieces to bins were mentioned, highlighting the initial sorting and subsequent randomization techniques. By employing a combination of deterministic and stochastic approaches for item assignment, the algorithm can avoid local optima and can explore a wider range of potential solutions.
The Bottom Left Fill (BLF) heuristic is then utilized for every assignment ordering to efficiently place items on sheets, prioritizing minimal overlap and optimal space utilization.

Finally, a three-level heuristic optimization process refines the solutions to achieve solutions that also optimize the F and K objectives, by focusing on maximizing sheet utilization and optimizing the remnant area of the least utilized sheet. By evaluating solutions based on the number of bins used and calculating F-values to assess utilization efficiency, the algorithm prioritizes solutions that maximize material usage and minimize waste. The introduction of the K-value further refines the selection process by considering the packing optimization of the least utilized sheet.

Chapter 5: Experiment design

This chapter describes the experimental design used to assess the algorithm's performance in solving the 2D irregular bin packing problem across various benchmark datasets. The chapter starts by describing the datasets used in our experiments. Section 5.2 discusses then algorithm's parametrization, focusing on balancing computation time and solution quality. Finally, Section 5.3 details the execution of our experiments and the methodology for comparing results with state-of-the-art approaches.

5.1 Data instances

The algorithm's performance is evaluated using the three distinct sets of irregular shape instances sourced from literature and benchmark repositories, as mentioned in Section 3.6.

5.1.1 Jigsaw puzzle instances (JP1 and JP2)

The JP1 and JP2 datasets, proposed by López-Camacho et al. (2013), consist of jigsaw puzzle instances where the optimal solution is known, achieving 100% utilization of each bin. The first set, JP1, includes 540 instances featuring convex pieces, categorized into 18 classes, each with 30 problems of varying piece counts per bin. The second set, JP2, consists of 480 instances, similarly categorized, including pieces with concavities. The number of pieces in JP2 varies per instance, depending on the amount of non-convex pieces in the instance. This number increases with the instance number, resulting in more (non-convex) pieces to pack in later instances. Typically, the instances fluctuate between 5 to 20 non-covex pieces per type.
Both sets are publicly available on the ESICUP website (The Association of European Operational Research Societies, 2024). For JP1 and JP2, each instance is packed into standardized bins of size 1000x1000 units. Further details on these datasets, such as the optimal number of sheets needed, rectangularity factors, and concavity degrees can be found in Appendices C.1 and C.2.

5.1.2 Irregular strip packing instances

The third set of instances is derived from well-known irregular strip packing benchmarks, also available on the ESICUP website. Unlike the jigsaw puzzle sets, these instances feature a wide variety of shapes, including convex and non-convex pieces that do not fit together exactly.
In strip packing problems, only the width of the stock sheet is constrained, with the objective to minimize the total length required to pack all pieces. To adapt these strip packing instances to our bin packing problem, Martinez-Sykora et al. (2017) defined a fixed square stock sheet size (width and length) as follows:

- *Nest-SB* (small bins). The bin dimensions are $W = L = 1.1m_d$.
- *Nest-MB* (medium bins). The bin dimensions are $W = L = 1.5m_d$.
- *Nest-LB* (large bins). The bin dimensions are $W = L = 2m_d$.

Here, m_d represents the maximum length or width across all pieces in their initial orientation for a given instance. This adaptation results in 23 irregular strip-packing instances, translating to 69 bin packing instances with specific piece counts, bin sizes, and permitted rotations. Further details on these datasets, such as the permitted rotations and bin sizes, can be found in Appendix C.3.
The rationale behind using three different bin sizes is to explore the relative significance of the assignment and packing phases. Intuitively, instances categorized as Nest-SB prioritize the assignment phase due to the limited number of pieces per bin, making the packing problem comparatively easier. Conversely, Nest-LB instances place greater importance on the packing phase, where efficient packing strategies may yield superior solutions even with sub-optimal assignments.

The irregular strip packing instances closely resemble the items that company C and its clients cut from metal sheets. For example, the jakobs1 instance is particularly relevant for steel construction tasks, considering the rectangularity of the items. Other instances like albano, trousers, han, mao, swim, shirts, shapes, and jakobs align well with scenarios in (mechanical) engineering, because of their varying shapes with a lot of vertices and non-convexity.

5.2 Parameterization

In addition to selecting the correct dataset with accompanying sheet dimensions and possible product spacing, several general parameters need to be configured for the algorithm to run effectively.

5.2.1 Iterations for initial packing solutions

First, the total number of iterations to find initial packing solutions must be determined. As discussed in Chapters 3 and 4, the BLF heuristic can be time-consuming. To balance computation time with solution quality, we set the total number of iterations to 100 for all approaches. This choice is based on observed computation times and results across various datasets, which can take several hours to solve. Limiting iterations to 100 ensures manageable computation times while still generating effective solutions.

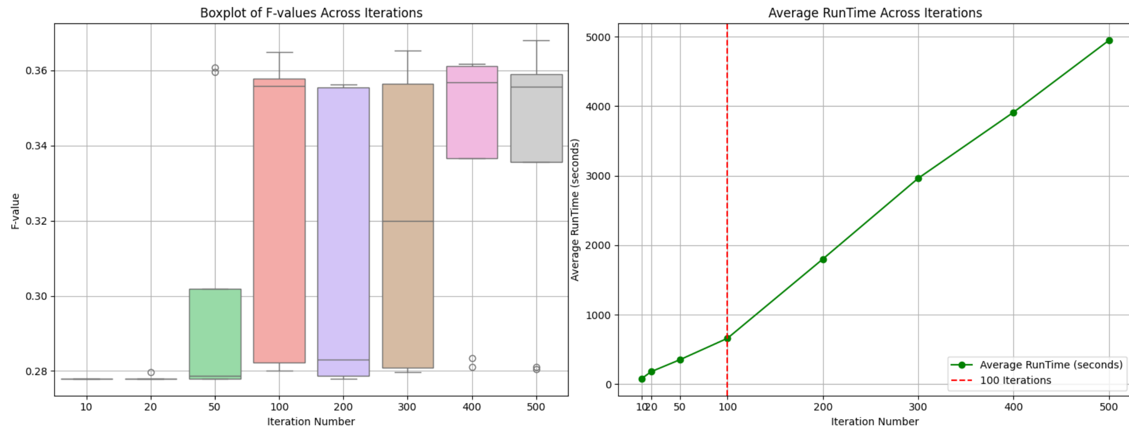


FIGURE 5.1: Sensitivity analysis of F-values to iteration counts: balancing solution quality and computation time.

The decision to cap iterations at 100 is supported by an analysis of F-values and corresponding computation times, as illustrated in Figure 5.1. The boxplot shows how F-values improve with more iterations, with noticeable gains up to 100 iterations. Beyond this point, improvements in F-values become less significant, showing a diminishing return on additional computation time. The line graph further supports this, showing that the average runtime increases linearly, reaching over 4000 seconds (approximately 66 minutes) for 500 iterations on a relatively fast dataset, namely the poly2b with its allowed rotations.
However, at 100 iterations, the average runtime is about 660 seconds (11 minutes), which is manageable considering the Python implementation. F-values at this point are significantly improved compared to lower iteration counts and approach those achieved with higher iteration counts. This suggests that 100 iterations strike an effective balance between solution quality and runtime.

An absolute time limit of two hours for the BLF iterations is also implemented. After this period, the algorithm stops generating more iterations and proceeds to the optimization phase with the solutions found within the time limit. This measure prevents the program from running indefinitely.

5.2.2 Permutations in the repacking phase

The next parameter concerns the repacking phase. As described in Chapter 4, this algorithm maximizes the remnant sheet area of the least utilized bin by repacking items in various orders, permutations of the initial ordering, using the BLF heuristic. These permutations are generated using a built-in Python function. Running the BLF for every possible permutation can result in thousands of iterations, each taking a few seconds. To maintain a manageable runtime, we limit the process to a maximum of 1200 permutation runs, following the order in which they were generated. Since the permutations are exhaustive and the order does not favor any particular arrangement, the first 1200 permutations cover a wide range of possible configurations. This ensures that you are getting a representative sample of the full permutation space, similar to what you would expect from a random selection.
The limit ensures that bins with up to 6 items can explore all possible permutations (720 permutations for 6 items), while bins with 7 or more items will be capped at 1200 permutations. This choice balances thoroughness and runtime, as 1200 BLF runs take approximately 20 to 60 minutes, given the 1 to 3 seconds per run.
Moreover, datasets such as JP1, JP2, and strip packing instances for nest-SB often have a maximum of 6 items in the least utilized bin, making 1200 permutations sufficient. For nest-LB instances, which often have more items in this bin, this cap may still help find better solutions within a reasonable time frame. The same consideration applies to the other datasets.

It is evident that running more iterations or permutations typically results in better solutions. However, these parameter values are chosen to strike a reasonable balance between computation time and solution efficiency for the current algorithm.

5.2.3 Rotation handling

The handling of rotations within the packing model can be seen as a parameter in our algorithm. As illustrated in Figure 4.2, we first randomize the assignment order of items in the list and then apply random rotations before packing. In scenarios where items can freely rotate, their orientations could be determined using the 'random_rotation' model detailed in Chapter 4, similar to cases where only a fixed set of allowed rotation angles is used. After the rotation angles are determined, the items are packed using the BLF heuristic.

However, enabling free rotations introduced significant computational challenges. The extensive processing time required for free rotations often led to hours of computation, primarily due to memory constraints and the necessity of multiple repacking attempts when some rotated items could not fit on the sheet with others. For relatively larger items, we even encountered situations where certain packings could not be generated at all because the rotated item could not even fit on a whole empty sheet. These time constraints made it impractical to include free rotations in our experimental design, as there would not be enough time to run free rotations for all datasets, hindering fair comparisons across all datasets.

It is important to note that this issue did not really arise when selecting from a fixed set of rotations, which still allowed for item rotation without infeasible packings and the associated computational burden.

5.3 Experiment execution

The algorithm is programmed in Python 3.12.2 64-bit and implemented using Visual Studio Code 1.86.1 2024. The computational tests are run on a PC with an Intel Core i7- 1355U processor and 16 gigabytes of memory.

Each strip-packing instance is executed three times with the parameters and rotation options as indicated above. For each jigsaw puzzle class, we randomly selected five instances to run. For the JP2 cases, we selected instances with a higher number of non-convex items to assess how well

the algorithm handles concavities.
For instances with explicitly mentioned allowed piece orientations, we strictly adhere to these predefined orientations. In the case of JP1 and JP2 instances, where no rotation requirements are specified, our experiments are conducted using their original orientations.
All results will be averaged and compared, using the resulting N-, F-, and K-values, against those of Martinez-Sykora et al. (2017), Zhang et al. (2022), and Wang et al. (2022), which are state-of-the-art in literature, as mentioned in Section 3.6.
Additionally, the experiment design includes some test runs of nest-LB instances using product spacing. However, these results will be analyzed separately due to the absence of comparable literature benchmarks. Furthermore, we will run each instance for 10 BLF iterations, since the goal is to show it works.

Lastly, it is important to note that for the strip packing instances, the poly2a through poly5a datasets could not be found on the ESICUP website and therefore have been excluded from the experiment design and the results chapter.

5.4 Conclusion

We will test and evaluate the designed algorithm utilizing diverse datasets available in the literature, including the jigsaw puzzle instances (JP1 and JP2) and an adaptation of the well-known irregular strip packing instances. The total number of bins, F- and K-values, resulting from our computations, will be compared to those of Martinez-Sykora et al. (2017), Zhang et al. (2022, and Wang et al. (2022), which are state of the art in literature.
The parametrization of the algorithm, including iterations for initial packing solutions, permutations in the repacking phase, and handling of rotations, was carefully chosen and explained to balance computation time and solution quality.
The experiment execution section outlines the setup and methodology for running the algorithm, emphasizing adherence to predefined orientations and the averaging of results for robust comparison. The exclusion of certain datasets due to unavailability is also noted, ensuring transparency in our experimental design.

Chapter 6: Analysis of the results

In this chapter, we compare our results with those of Martinez-Sykora et al. (2017) and Zhang et al. (2022), and also mention the general results of Wang et al. (2022) for context, as this paper does not provide detailed individual results. These papers represent the state of the art in 2DIBPP research and report on the metrics N and F, which are directly comparable to our own results. However, their definition of the K-value differs from ours.

Martinez-Sykora et al. (2017) introduced five methods for assigning irregular items to sheets: Bin Packing with Greedy Decisions (BPGD), First Fit Algorithm (FF), Partial Bin Packing (PBP), Two Phases Strategy (TPS), and Simple Construction Heuristic (SCH). In our comparison, we focus on PBP and SCH. PBP was the most effective among their methods, closely followed by SCH, which resembles the initial assignment strategy used in our algorithm.
Zhang et al. (2022) employs the same overlap minimization method as Wang et al. (2022) combined with a Least-Waste First strategy for generating initial packing solutions and a local search for improvement. Their local search was executed 10 times with different random seeds.
For the nesting instances, all methods consider allowed rotation angles. For the jigsaw puzzle instances, PBP allowed free rotation, while the others considered only four specific rotation angles. However, allowing the four rotation angles for the jigsaw puzzles in our algorithm resulted in prohibitively long computation times, making it impractical to run each instance for 100 iterations. Therefore, we ran these instances in their original orientations to maintain a generally equal test run for all the instances. Average values for all dataset tests are recorded in Tables 6.1 - 6.5.

The cited papers primarily focused on optimizing the N and F values and only calculated the K-value as a supplementary metric. In contrast, our approach integrates the K metric directly into the optimization process, aiming to maximize the efficiency of the sheet usage.
Moreover, it is important to recognize that the K-value in our study is defined differently. While the previous papers calculated K based on horizontal or vertical cuts, our K-value is determined by cutting off the least utilized sheet in an L=shape, making direct comparisons of K-values unfeasible. Despite these difference, we include the K-values in our results for a comprehensive overview, acknowledging that the metric, while defined differently, plays a crucial role in our optimization process.

Finally, because the algorithms from the reviewed papers were implemented in different languages and on different platforms, their execution times are not directly comparable. That said, our algorithm has a considerably longer runtime, several tens of minutes, making the computation times of the other papers significantly better, which run in several minutes or even seconds. Details on our algorithm's computation times are provided in Section 6.4.

We conclude this chapter with a discussion of some overlapping issues encountered during testing (Section 6.5) and present results for the Nest-LB instances using product spacing (Section 6.6).

6.1 Results of the JP1 instances

The JP1 instances, discussed in Chapter 5, involve convex pieces with 3 to 8 sides, where the optimal solutions are known since all pieces can fit together perfectly. Table 6.1 shows that none of the algorithms consistently achieved the optimal number of sheets, see for comparison Appendix C.1, except for the LocalSearch algorithm by Zhang et al. (2022), which found the optimal layout for types B, H, and Q.

Our algorithm performed relatively poorly on type Q, needing six more sheets than the other algorithms, resulting in a 54.3% utilization efficiency compared to 100% for LocalSearch and 95.6% for

Subset	LS2-PBP	LJS1-MU	LJS3-MU	LS-4R	LocalSearch -WLFD	LS-PBP (slit)	LS-PBP (no slit)
JP1	0.723	0.691	0.732	0.704	0.744	0.705	**0.746**
JP2	0.729	0.701	0.747	0.701	**0.777**	0.719	0.768
Nest-SB	0.432	0.393	0.403	0.442	0.445	0.417	**0.448**
Nest-MB	0.452	0.418	0.409	0.445	**0.482**	0.431	**0.482**
Nest -LB	0.403	0.425	0.410	0.438	0.443	0.397	**0.446**

FIGURE 6.1: Comparison of Wang et al.'s results (LS-PBP with and without slit) to other literature, including LocalSearch-WLFD from Zhang et al (2022), and LS2-PBP from Martinez-Sykora et al. (2017). The bold values denote the best solutions (Wang et al., 2022).

Martinez-Sykora et al.'s strategies. Appendix C.1 indicates that type Q has a relatively high rectangularity factor, as do types B and H, which also underperformed, suggesting our algorithm may struggle with rectangular items. However, our results for type I, which has 100% rectangularity, are relatively good compared to the other algorithms and the optimality of 3 sheets. This indicates that our algorithm can perform well with rectangular items under certain conditions. Additionally, types G and J, also high in rectangularity, performed well, suggesting that poor results for types Q, B, and H must be due to other factors.

Instance	SCH			PBP			LocalSearch			Own Results		
	N	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**
A	4	0.613	3.498	4	0.614	3.49	4	0.630	3.435	4.4	0.498	3.986
B	11.833	0.761	11.406	11.70	0.781	11.281	10	1.000	10.000	13.2	0.591	12.502
C	7.233	0.731	6.838	7.23	0.729	6.858	7.01	0.775	6.580	9	0.486	8.667
D	4	0.591	3.638	4	0.59	3.636	4	0.599	3.596	4.4	0.496	4.136
E	4.4	0.523	4.098	4.40	0.522	4.098	4	0.579	3.836	6	0.293	5.825
F	3	0.516	2.412	3	0.515	2.414	3	0.536	2.381	3	0.465	2.873
G	14.533	0.716	14.158	14.47	0.724	14.05	13.51	0.822	13.131	14.4	0.628	13.713
H	14.267	0.751	13.859	14.13	0.766	13.693	12	1.000	12.000	15	0.682	14.307
I	4	0.629	3.383	4	0.628	3.382	4	0.663	3.250	4	0.595	3.612
J	5	0.671	4.578	5	0.672	4.57	5	0.686	4.467	5	0.656	4.883
K	7.033	0.760	6.730	7.03	0.761	6.712	7	0.771	6.641	8	0.601	7.754
L	4.067	0.589	3.751	4.07	0.588	3.754	4	0.609	3.667	5	0.376	4.891
M	6.4	0.658	6.102	6.40	0.655	6.111	6.18	0.690	5.936	8	0.444	7.503
N	3	0.518	2.388	3	0.517	2.392	3	0.528	2.360	3	0.472	2.801
O	7.967	0.826	7.641	7.93	0.831	7.596	7.08	0.986	7.005	9	0.633	8.548
P	9.867	0.713	9.420	9.80	0.72	9.383	9.57	0.742	9.190	11.4	0.537	11.020
Q	15.633	0.941	15.338	15.50	0.956	15.263	15	1.000	15.000	21	0.543	20.293
R	10.567	0.766	10.165	10.57	0.767	10.165	10.40	0.784	10.064	12	0.615	11.413
Avg.	7.600	0.682	7.189	7.57	0.685	7.159	7.15	0.744	6.808	8.656	0.534	8.263

TABLE 6.1: Results for JP1 instances.

For type B (see Figure 20), the packing could have been improved with a different assignment ordering. The BLF heuristic is dependent on good assignment ordering, which is crucial when the average piece area is high. In such cases, items do not fit on the same sheet if not fitted as a jigsaw puzzle. Types G and J, similar in rectangularity to type B, with G also having a similar average piece area, benefiting from better assignment ordering, allowing some sheets to be filled optimally. Type H, while not performing as well as Martinez-Sykora et al.'s strategies, could improve with better assignment ordering.
The poor performance for type Q can also be attributed to bad assignment ordering. The packing (see Figure 35) shows that while some sheets are filled optimally, others are far from optimal. The high average piece area exacerbates this issue, requiring extra sheets because items cannot fit on the same sheet when not packed efficiently in a certain rotation.

In summary, the poor performance for types Q, B, and H is due to a combination of bad assignment ordering, high average piece area, and a lack of rotation options. Running additional iterations, with the option to freely rotate, to explore more assignment orderings could potentially improve the layout.

Our results for type E show a significantly lower F-value compared to the other algorithms. Type E is less rectangular than the aforementioned types, and contains many long, narrow elements (see Figure 23) that are positioned at an angle in the original rotation. This makes packing these elements difficult with the current rotation. Allowing (free) rotation could have enabled these elements to fit better, likely reducing the number of sheets needed.

In contrast, types I, J, N, F, and D performed well. These types share a common characteristic of having a relatively small average piece area, which allows smaller elements to fill gaps between larger rectangular items. Additionally, the best layouts of these types often followed the assignment ordering of the first iteration, consistent with the SCH method by Martinez-Sykora et al. (2017).

In conclusion, LocalSearch outperformed all other algorithms, including ours, with an average of 11% better overall utilization efficiency. Implementing a Local Search that adjusts the assignment per sheet by swapping items between sheets and moving them from lesser to higher utilized sheets could significantly optimize the packing results.[1]

6.2 Results of the JP2 instances

The JP2 instances include both convex and non-convex pieces, with known optimal solutions for all cases. Table 6.2 shows that none of the algorithms consistently achieved the optimal number of sheets, except for Zhang et al.'s (2022) LocalSearch, which found the optimal layout for type V. A key observation is the large gap in F-values between our results and Martinez-Sykora et al., compared to Zhang et al's LocalSearch. Zhang's method improved utilization by up to 30% for type S, highlighting the efficiency of local search. Incorporating Wang et al.'s local search method could potentially yield similar improvements for our algorithm.

Our algorithm struggled with type T, requiring 3 extra sheets and resulting in a 54.4% utilization efficiency, compared to 99.7% for LocalSearch and 89% for Martinez-Sykora et al.'s strategies. Similarly, our packing for type V is much worse, with LocalSearch achieving optimality. Type M also performed poorly in terms of F-values, though it yielded the highest F-value for us, and the F-ratio with the LocalSearch is quite high.
Type T has a high concavity degree and several large shapes, see Appendix C.2 and Figure 46, making assignment ordering and rotations crucial for good packing, similar to the JP1 instances. Due to computation time constraints, we did not allow rotations, but incorporating them, along with a local search method, could significantly improve packing. The same applies to type V, which has a high average piece size as well, but a low concavity degree.
Type M has a relatively low concavity degree and rectangularity, with many long, narrow elements skewed in their original rotation, much like type E from JP1. Allowing rotations could improve packing, potentially reducing the number of required sheets.

Our best results are for types F, L, A, and X, with an average utilization gap of just 17%. These types share a low average piece size, relatively low rectangularity, and moderate concavity. Narrow items helped fill gaps more easily, making packing less sensitive to a 'poor' assignment ordering.

Among all types, types B and T have the largest average piece size and the highest concavity degree. The results for type B are not as poor as for type T, largely due to a reasonable good assignment ordering and fewer very large items, allowing some sheets to be filled nearly optimally,

[1] The Local Search from Wang et al. (2022) actually does this, just like the local searches from Zhang et al. (2022) and Martinez-Sykora et al. (2017)

with gaps filled by smaller items.

In conclusion, as with the JP1 instances, our algorithm struggles with instances that have a high average piece size, especially when there are few small items to fill gaps. Allowing rotations and improving assignment ordering through local search could greatly enhance our results, as demonstrated by Zhang et al. (2022) and the average results of Wang et al. (2022), see Figure 6.1. Finally, it is worth noting that our results might be skewed since we did not run all instances that exist for every case and specifically selected those with more non-convex items.

Instance	SCH			PBP			LocalSearch			Own Results		
	N	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**
A	4	0.626	3.435	4	0.605	3.579	4	0.613	3.527	5	0.430	4.397
B	12	0.728	11.644	11.97	0.731	11.596	11.16	0.841	10.808	14	0.564	13.382
C	7.7	0.665	7.217	7.52	0.69	7.094	7.43	0.702	7.007	9	0.474	8.693
F	3	0.510	2.488	3	0.511	2.468	3	0.526	2.413	3.4	0.428	2.866
H	14.433	0.730	13.985	14.43	0.73	13.995	12.88	0.895	12.654	16	0.605	15.425
L	4.2	0.563	3.883	4.17	0.571	3.823	4.04	0.591	3.771	5	0.384	4.898
M	6.633	0.627	6.264	6.53	0.641	6.194	6.33	0.666	6.073	8.6	0.376	8.171
O	8.3	0.769	7.857	8.23	0.774	7.843	7.66	0.879	7.488	10.4	0.476	10.396
S	3	0.517	2.633	2.97	0.521	2.605	2.39	0.817	2.185	3.4	0.409	3.083
T	10.833	0.887	10.556	10.83	0.89	10.542	10.02	0.997	10.019	14	0.544	13.340
U	6.133	0.732	5.830	6.10	0.737	5.826	5.42	0.894	5.242	8.4	0.415	7.981
V	5.233	0.942	5.115	5.20	0.951	5.091	5	1.000	5.000	6.4	0.717	5.905
W	5.133	0.646	4.790	5.07	0.656	4.712	4.73	0.783	4.364	6	0.493	5.417
X	4.033	0.586	3.727	4	0.593	3.691	3.98	0.613	3.579	5	0.420	4.321
Y	7.300	0.720	6.992	7.30	0.72	6.958	7.11	0.743	6.870	9.6	0.462	8.757
Z	13.333	0.847	12.924	13.23	0.856	12.866	13.14	0.865	12.843	14.8	0.602	14.473
Avg.	7.204	0.693	6.834	7.16	0.699	6.805	6.77	0.777	6.490	8.688	0.487	8.219

Table 6.2: Results for JP2 instances.

6.3 Results of the strip packing instances

The strip packing instances use realistic industrial shapes, commonly encountered in the industry, so there are no known optimal solutions. As discussed in Appendix C.3, three different bin sizes are used to examine the relative importance of assignment and packing strategies. Martinez-Sykora et al. (2017) noted that for Nest-SB instances, the assignment problem is more crucial due to the limited bin capacity, while for Nest-LB instances, efficient packing plays a greater role, potentially leading to superior solutions even with sub-optimal assignments.

The following sections will outline the key results and conclusions for each of the different bin sizes, taking the observation from Martinez-Sykora et al. (2017) into account.

6.3.1 Results of the Nest-SB instances

Analyzing the results for the Nest-SB instances, see Table 6.3, our algorithm struggled with the swim instance, requiring three more sheets than the other algorithms, resulting in a utilization gap of 18.9%. The poly2b and shirts instances also performed poorly, with about a 14% utilization efficiency gap.

For poly2b, allowing more or better rotation options could help items fit together more efficiently, reducing the large gaps between items. Running additional iterations to enhance the likelihood of finding better rotations might thus help yield better results. This applies to all poly instances, as they share many items. For instance, poly2b contains all items from poly1a plus 15 more, while poly3b includes poly2b items plus an additional 15.

The swim instance, with complex items having up to 36 vertices, made NFP generation and packing challenging and time-consuming, resulting in several hours to finalize a layout. Additionally, the relatively large items left gaps that increased the need for extra sheets. If the algorithm was faster, additional iterations could be run to try to improve the assignment ordering and thus packing.
The shirts instance faced similar issues, with quite a few large items leaving gaps in later sheets, see Figure 57, as there are not enough small items to fill the gaps. With 99 items, most of any instance, the algorithm took several hours to generate a final layout, preventing running extra iterations for improvement.

In contrast, instances like albano, dighe2, fu, and jakobs1 performed well, with an average utilization efficiency gap of 7.3% compared to the LocalSearch and just 3% compared to SCH and PBP. These instances have fewer items than shirts and swim and simpler shapes (up to 14 vertices), reducing solution generation times to 2-20 minutes. The good results stem from a balanced mix of small and large items, allowing small items to fill gaps, and the relatively rectangular corners of many items, causing nice fittings.

The results for the shapes instances may appear even better, with higher F-values compared to the other algorithms. However, this is misleading due to overlapping issues in some solution layouts (see Figures 55 and 56), which is why they are marked red. In shapes0, overlap occurs in the first sheet between two non-convex items with notches. Similar overlap are present in the first two sheets of shapes1, and in the han and jakobs2 instances (see Figures 61 and 63). This issue was anticipated and will be discussed further in Section 6.5. While these overlaps may render some of the reported values infeasible, we still include the results because there are always feasible solutions available. Notably, about 68% of shapes0, 34% of shapes1, 36% of jakobs2, and 19% of han instances exhibited overlap.
To conclude, the Nest-SB instances where the final solution closely followed the first iteration, like fu, dighe2, jakobs1, and poly4b, yielded good results. These findings suggest that an assignment ordering similar to SCH or PBP could improve layouts for the other instances. Thus, we agree with Martinez-Sykora et al. (2017) so far that assignment plays a significant role when the number of pieces per bin is small.

Instance	SCH			PBP			LocalSearch			Own Results		
	N	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**
albano	6	0.474	5.216	6	0.474	5.216	5	0.600	4.677	6	0.420	5.594
trousers	5	0.671	4.909	5	0.671	4.909	5	0.675	4.850	6	0.485	5.459
shapes0	14	0.245	13.390	14	0.245	13.39	12	0.330	11.534	14	0.308	13.510
shapes1	13	0.272	12.714	13	0.272	12.714	12	0.330	11.649	13	0.344	12.649
shirts	14	0.571	13.839	14	0.571	13.839	14	0.579	13.876	16	0.446	15.616
dighe2	3	0.359	2.654	3	0.359	2.654	3	0.397	2.390	3	0.329	2.670
dighe1	3	0.406	2.771	3	0.406	2.771	3	0.457	2.472	4	0.257	3.368
fu	8	0.362	7.455	8	0.362	7.455	8	0.366	7.455	8	0.349	7.708
han	5	0.435	4.237	5	0.435	4.237	4.5	0.529	4.000	5	0.383	4.788
jakobs1	9	0.371	8.341	9	0.371	8.341	9	0.375	8.341	9	0.347	8.129
jakobs2	7	0.402	6.682	7	0.402	6.682	7	0.406	6.568	7	0.393	6.665
mao	4	0.432	3.706	4	0.432	3.706	4	0.449	3.623	4.333	0.391	3.827
poly1a	3	0.438	2.892	3	0.438	2.892	3	0.456	2.997	4.333	0.246	3.974
poly2b	7	0.409	6.632	7	0.41	6.506	7	0.413	6.850	9	0.268	8.274
poly3b	9	0.444	8.916	9	0.444	8.916	9	0.451	8.889	10.333	0.323	10.173
poly4b	12	0.430	11.210	12	0.43	11.21	12	0.431	11.000	13	0.335	12.765
poly5b	14	0.428	13.426	14	0.431	13.284	14	0.454	13.350	15.333	0.332	15.103
swim	9	0.416	8.877	9	0.418	8.806	9	0.397	8.771	12	0.229	11.776
Avg.	8.056	0.420	7.659	8.056	0.421	7.640	7.806	0.450	7.405	8.852	0.344	8.447

TABLE 6.3: Results for nesting instances with small bins.

6.3.2 Results of the Nest-MB instances

In the Nest-MB instances (Table 6.4), our algorithm underperformed for dighe2, trousers, fu, and albano compared to the other algorithms. This result is interesting, as fu, dighe2, and albano were among the best performers in Nest-SB. The underperformance of albano in Nest-MB seems linked to assignment ordering. In Nest-SB, albano often yielded final results from the initial iterations, adhering to the SCH method. However, in Nest-MB, the final results came from much later iterations, leading to larger items being packed inefficiently in the last sheets, resulting in significant unused space. Incorporating a local search approach, such as Zhang et al.'s (2022), could improve assignment per sheet and reduce this unused space. Additionally, adding rotations beyond 0° and 180°, as done by Martinez-Sykora et al. (2017) and Wang et al. (2022), might improve our results, see Figure 6.2. These additional rotations are now meaningful, given the larger bins in Nest-MB. In Nest-SB, there was not enough room for the larger items to be rotated differently.

Instance	SCH			PBP			LocalSearch			Own Results		
	N	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**
albano	3	0.480	2.873	3	0.48	2.873	3	0.527	2.497	4	0.323	3.176
trousers	3	0.555	2.667	3	0.555	2.667	3	0.569	2.635	4	0.342	3.315
shapes0	7	0.271	6.762	7	0.271	6.762	6	0.398	5.381	8	0.247	7.476
shapes1	7	0.282	6.476	7	0.282	6.476	6	0.398	5.381	8	0.255	7.444
shirts	8	0.518	7.844	8	0.518	7.844	7	0.666	6.740	9	0.411	8.615
dighe2	1	0.823	0.952	1	0.823	0.952	1	0.823	1.000	2	0.279	1.350
dighe1	2	0.368	1.333	2	0.368	1.333	2	0.374	1.394	2	0.298	1.694
fu	4	0.443	3.571	4	0.443	3.571	4	0.452	3.571	5	0.275	4.619
han	3	0.387	2.203	3	0.387	2.203	3	0.420	2.232	3	0.340	2.497
jakobs1	4	0.570	3.333	4	0.57	3.333	4	0.577	3.250	4.333	0.447	3.778
jakobs2	4	0.401	3.250	4	0.401	3.25	3.6	0.513	3.000	4	0.354	3.698
mao	3	0.271	2.328	3	0.271	2.328	2	0.492	1.852	3	0.272	2.165
poly1a	2	0.308	1.536	2	0.308	1.536	2	0.344	1.497	2.333	0.261	1.974
poly2b	4	0.389	3.436	4	0.389	3.436	4	0.413	3.420	4	0.356	3.901
poly3b	5	0.434	4.582	5	0.432	4.582	5	0.450	4.515	6	0.313	5.209
poly4b	6	0.465	5.801	6	0.465	5.801	6	0.475	5.751	7	0.351	6.621
poly5b	7	0.478	6.936	7	0.478	6.936	7	0.488	6.555	8	0.358	7.648
swim	5	0.397	4.891	5	0.397	4.891	5	0.392	4.411	6	0.279	5.573
Avg.	4.333	0.436	3.931	4.333	0.435	3.932	4.089	0.487	3.616	4.981	0.320	4.486

TABLE 6.4: Results for nesting instances with medium bins.

For dighe2, the large utilization efficiency gap of 54.4% stems from needing an extra sheet to accommodate just two items (see Figure 76). In Nest-SB, the final results often came from the first iteration(s), but this was not the case in Nest-MB. Prioritizing larger items in the assignment order might yield better outcomes. Local search techniques, such as swapping and reallocating items from lesser utilized sheets to more utilized ones, could potentially eliminate the need for an extra sheet.

Interestingly, the performance of the poly instances improved significantly compared to Nest-SB and the other algorithms. The relatively smaller items compared to sheet size and assignment ordering following the SCH method often resulted in much better solutions. It resulted in using the same number of sheets as the other algorithms, although gaps between items still exist.

For the first time, we achieved better results than Martinez-Sykora et al. (2017) for the mao instance, despite only using the allowed rotations instead of Martinez's free rotations. The difference is minimal, with a 0.1% utilization, likely due to a slightly larger item being placed in the last sheet compared to ours. Nonetheless, there is still significant room for improvement, as the LocalSearch nearly doubles total utilization efficiency by reducing the number of required sheets.

Furthermore, there was no overlap in the shapes instances, indicating that good results can be achieved with an assignment ordering similar to the SCH method, as these solutions often emerged from the first iteration(s). However, overlap issues persist in the jakobs2 and han instances due to notches in non-convex shapes, with approximately 38% of jakobs2 and 20% of han instances experiencing overlap.

To conclude, compared to Nest-SB, the best results in Nest-MB, relative to the other algorithms, were achieved without relying on an assignment ordering close to the SCH method. Instances like dighe1, mao, and poly2b allowed for placing several large items in the last few bins while still achieving good final layouts. However, many instances still require an assignment ordering similar to the SCH method to avoid gaps between later-placed large items.

Instances	Limited angles		Free rotation		Instances	Limited angles		Free rotation	
	N	F	N	F		N	F	N	F
albano	3	0.467	3	**0.492**	poly5a	8	0.420	8	**0.426**
dighe1	2	**0.354**	2	0.271	poly2b	4	0.371	4	**0.378**
dighe2	1	**0.823**	2	0.222	poly3b	5	0.416	5	**0.422**
fu	4	0.421	4	0.421	poly4b	6	0.445	6	**0.451**
han	3	0.362	3	**0.366**	poly5b	7	0.459	7	**0.464**
jakobs1	4	**0.510**	4	0.482	shapes0	8	0.502	8	**0.546**
jakobs2	4	**0.383**	4	0.358	shapes1	7	0.255	6	**0.368**
mao	3	0.250	3	**0.280**	shapes2	10	**0.358**	10	0.350
poly1a	2	0.291	2	**0.300**	shirts	7	0.264	6	**0.368**
poly2a	4	0.323	4	**0.325**	swim	5	0.377	5	0.377
poly3a	5	0.399	5	**0.401**	trousers	3	0.535	3	**0.559**
poly4a	7	0.377	7	**0.381**	Average	4.87	0.407	4.826	0.392

FIGURE 6.2: Results for Nest-MB instances from Wang et al. (2022).

6.3.3 Results of the Nest-LB instances

As shown in Table 6.5, our algorithm performs exceptionally well on Nest-LB instances, with an average utilization gap of just 7.2% across all instances and all methods, significantly closer to those of the competing algorithms than in our cases of Nest-SB, Nest-MB, JP1, and JP2.
Among the less favorable results are the poly2b and shirts instances. For poly2b, better rotations could immediately enhance the layout, and running additional iterations, to explore better rotation choices, might have yielded better results.
For the shirts instance, the packing efficiency is quite high with only minimal gaps between items. While the current assignment ordering achieves relatively effective packing, a slightly adjusted ordering, particularly for sheet 4, could lead to notable improvements by eliminating existing gaps and bringing our solution closer to those achieved by the other algorithms. Utilizing local search techniques, like those from Wang et al. (2022) or Zhang et al. (2022) could further improve results.

Our algorithm shows comparable or even superior N- and F-values for the poly1a, dighe2, dighe1, and mao instances. Poly1a, in particular, demonstrates impressive results, matching Zhang et al.'s solution and outperforming Martinez-Sykora et al.'s by 1.3%, with a computation time of just over 2 minutes. These results were achieved without following the assignment strategy of placing larger items first, as seen in the initial iterations. Similarly, we improved the mao instance's packing relative to Martinez-Sykora et al.'s, although significant potential for further enhancement remains with local search adaptation.
The dighe instances have identical F-values over all methods, as they utilize just one sheet for packing all items. Without access to layout images from LocalSearch or Martinez's strategies, it is hard to determine which algorithm performed better, as the K-values are defined differently and thus not comparable.

For jakobs1, jakobs2, and the shapes instances, overlaps hinder clear conclusions (see Figures 91, 92, 98, and 99). Specifically, overlap occurs in about 65% of shapes0 instances, 32% of shapes1, 49% of jakobs2, and 45% of jakobs1 instances. While these overlaps affect feasibility, we include these results because feasible solutions can always be derived.

Regarding assignment orderings in general, some instances achieved the best layouts from the first iteration(s), consistent with the SCH method. However, others, like albano, both dighe instances, poly1a, poly4b, and fu, performed nearly as well or equally to other algorithms even with large items placed in the final sheets. This highlights the effectiveness of our packing strategy, even when assignment orderings are not optimal.
In summary, our findings support Martinez-Sykora et al.'s conclusions: for smaller sheets (Nest-SB and other instances with small bin dimensions relative to item sizes) an effective assignment strategy is crucial due to limited bin capacity. Conversely, for Nest-LB and other instances with larger bins, the packing strategy itself becomes more important, leading to excellent solutions even with sub-optimal assignments.

Instance	SCH			PBP			LocalSearch			Own Results		
	N	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**	**N**	**F**	**K**
albano	2	0.387	1.472	2	0.387	1.472	2	0.416	1.374	2	0.383	1.645
trousers	2	0.445	1.500	2	0.445	1.5	2	0.454	1.405	2	0.420	1.611
shapes0	4	0.290	3.393	4	0.29	3.393	3	0.462	3.000	4	0.278	4
shapes1	4	0.302	3.286	4	0.302	3.286	3	0.462	3.000	4	0.283	3.929
shirts	4	0.644	3.926	4	0.644	3.926	4	0.642	3.916	5	0.467	4.237
dighe2	1	0.260	0.590	1	0.26	0.587	1	0.260	0.701	1	0.260	0.750
dighe1	1	0.329	0.835	1	0.329	0.835	1	0.329	0.707	1	0.329	0.818
fu	2	0.505	1.681	2	0.505	1.681	2	0.526	1.571	2.333	0.410	1.925
han	2	0.300	1.217	2	0.3	1.217	2	0.332	1.283	2	0.273	1.443
jakobs1	2	0.608	1.763	2	0.608	1.763	2	0.602	1.778	3	0.368	2.102
jakobs2	2	0.450	1.875	2	0.45	1.875	2	0.464	1.686	2	0.441	1.944
mao	2	0.231	1.346	2	0.231	1.346	1	0.610	0.961	2	0.255	1.087
poly1a	1	0.355	0.907	1	0.355	0.907	1	0.368	0.861	1	0.368	0.975
poly2b	2	0.442	1.898	2	0.442	1.898	2	0.475	1.870	3	0.271	2.240
poly3b	3	0.396	2.590	3	0.396	2.59	3	0.408	2.448	3	0.354	2.830
poly4b	4	0.389	3.265	4	0.389	3.255	4	0.415	3.231	4	0.342	3.660
poly5b	4	0.472	3.754	4	0.472	3.754	4	0.476	3.742	5	0.344	4.192
swim	3	0.364	2.593	3	0.364	2.593	3	0.361	2.414	3	0.329	2.945
Avg.	2.5	0.398	2.105	2.5	0.398	2.104	2.333	0.448	1.997	2.741	0.343	2.352

TABLE 6.5: Results for nesting instances with large bins.

6.4 Analysis of computation times

As discussed in Chapter 5, we opted for 100 iterations of the BLF heuristic combined with the repacking algorithm, limited to 1200 permutations, to keep computation times manageable. This setup resulted in an average overall computation time of around 20 minutes for the JP1 and JP2 datasets, with occasional spikes to $30-40$ minutes for the J-instance in JP1 and the F-instance in JP2, which yielded the best results compared to the other algorithms in their respective datasets.

For the strip packing instances, average computation times were significantly higher, partially prompting, as mentioned before, the choice of our parameter values. Specifically, the Nest-SB instances averaged 16 minutes, excluding the 'swim' instance, which can be seen as a bottleneck instance as it took over 200 minutes due to its complex pieces with up to 36 vertices and many line segments. The Nest-MB and Nest-LB instances averaged around 53 minutes, with the 'swim' instance again being the most time-consuming, requiring approximately 190 and over 200 minutes, respectively.

Instances with complex shapes or a large number of items had longer computation times. For

example, datasets like poly5b, trousers, and shirts had overall times ranging from 50 to 120 minutes in the Nest-LB case. Specifically, the number of items and the complexity of shapes (e.g. higher number of vertices) directly impacted computation time. The dighe instances in Nest-LB, characterized by high packing density and numerous items, had average computation times slightly exceeding 105 minutes, mainly due to the repacking algorithm reaching its maximum iterations to optimize the layout for the least utilized sheets. Generally, when the repacking algorithm used its maximum iterations, the process took between 20 minutes and a few hours, while the initial BLF iterations were computed relatively quickly, usually within 10 minutes.

The swim instance, with pieces having up to 36 vertices, highlights how complexity affects computation time, as NFP generation and packing was particularly challenging, leading to extended computation times. In contrast, instances with fewer or simpler items, such as albano, dighe2, fu, and jakobs1, were completed in a more efficient $2 - 20$ minutes, showing that simpler shapes and fewer items generally result in faster computations while still providing effective solutions.

Overall, the BLF initial iterations were efficient, typically finishing in about 3 minutes for the jigsaw datasets and 10 to 20 minutes for the strip packing instances. However, they took longer for instances with complex shapes or a large number of items, such as swim, shirts, and trousers. Instances with overlap, like shapes, han, and jakobs, also extended the BLF time, because of the complex NFP generation and the need for additional packing configurations. The repacking algorithm was the main contributor to extended computation times, especially for datasets with high item counts, due to the large number of permutations and overlap checks required.

6.5 Overlapping issues

This section addresses the overlapping issues observed in certain instances (shapes0, shapes1,han, jakobs1, and jakobs2), as noted in Appendices L-N. These overlaps were anticipated due to the current implementation of the NFP generation in combination with the packing algorithm in Python.

Our NFP generation is based on the method by Burke et al. (2007), up to the 'Start Points' section, which was not implemented in our work due to its complexity. Instead, our approach, similar to the algorithm developed by Mahadevan (1984), includes enhancements to reduce computation time. However, both Mahadevan's and our approach can encounter issues when generating NFPs for shapes with interlocking concavities, jigsaw pieces, or holes, as seen in the han, jakobs, and shapes instances. This is why Burke et al. (2007) implemented the 'Start Points' method.

The 'Start Points' method addresses potential overlap issues and the inability to generate complete NFPs for shapes involving interlocking concavities, jigsaw pieces, or holes by identifying feasible touching positions for the moving polygon. These positions serve as starting points for generating the remaining parts of the NFP, ensuring all potential fits are considered and, more importantly, overlaps with other polygon edges are avoided. This final check helps resolve overlaps for complex interlocking shapes or non-convex shapes with holes.

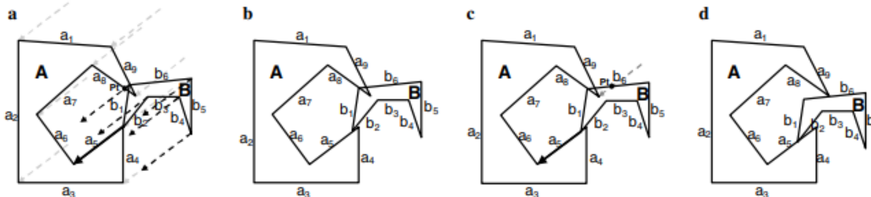
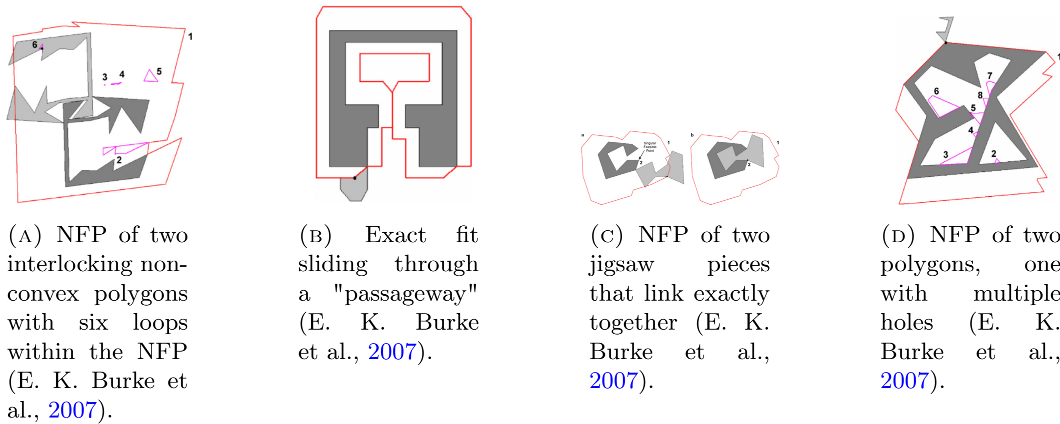


FIGURE 6.3: The start point generation process (E. K. Burke et al., 2007).

Figure 6.3 illustrates this method, showing how the iterative process trims translation vectors to prevent overlaps, ultimately generating a non-overlapping NFP. The first step involves identifying the closest intersection point and trimming the translation vector accordingly, see Figure **a**.

Polygon B, the moving polygon, is then translated by this trimmed vector. If overlap persists, as shown in Figure **b**, the process is repeated, with the subsequent translation vector determined from the touching point to the end vertex of edge a_5 and then trimmed again, see Figure **c**. This iterative approach continues until the polygons no longer intersect, allowing for the generation of a new, non-overlapping NFP (E. K. Burke et al., 2007). Combined with the packing algorithm and a separation algorithm, such as the approach developed by Wang et al. (2022), this method resolves edge intersections, prevents overlap, and produces more complete NFPs enabling (non-convex) polygons to interlock and/or fit in holes of other polygons.
Examples of NFPs generated using the 'Start Points' method can be seen in Figure 6.4, where the moving polygon is depicted in light grey with a reference point marked by a black dot.



(A) NFP of two interlocking non-convex polygons with six loops within the NFP (E. K. Burke et al., 2007).

(B) Exact fit sliding through a "passageway" (E. K. Burke et al., 2007).

(C) NFP of two jigsaw pieces that link exactly together (E. K. Burke et al., 2007).

(D) NFP of two polygons, one with multiple holes (E. K. Burke et al., 2007).

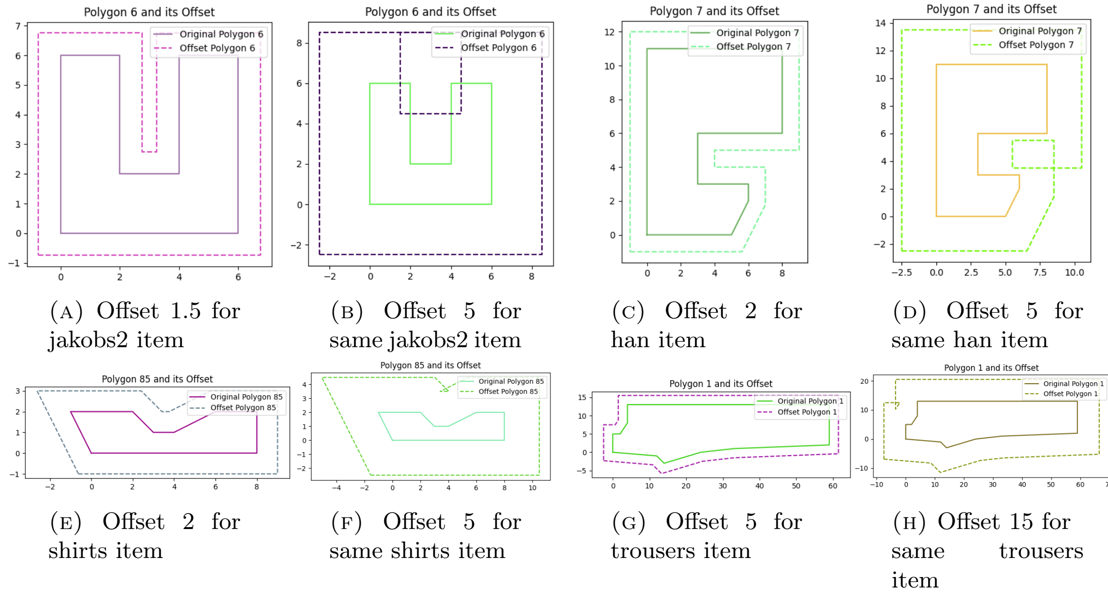
FIGURE 6.4: NFP for shapes with interlocking concavities, jigsaw pieces, and holes.

In summary, the Start Points method by Burke et al. (2007) is a sophisticated technique that improves the accuracy and efficiency of NFP generation, particularly for complex geometries. It ensures that all potential fits are considered, overlaps are avoided, and the overall packing solution is optimized.

6.6 Offset generation results and handling complex polygons

This section discusses the results of generating product spacing around polygons and the challenges encountered with very complex shapes and larger offset distances. Initially, an equidistant method, based on geometric calculations, was developed as described in Chapter 4. This method, detailed in Appendix D, improves upon Wang et al.'s (2022) to work well for concave polygons by distinguishing the angle calculations between convex and concave polygons. Handling concave angles correctly is crucial to ensure the correct offset direction and magnitude, especially when implementing it in programming environments like Python.
However, last-stage testing revealed limitations of this method when applied to highly intricate polygons with closely spaced convex and concave vertices in combination with relatively large offset distances. To overcome these limitations, an additional approach, the buffer method, was implemented.

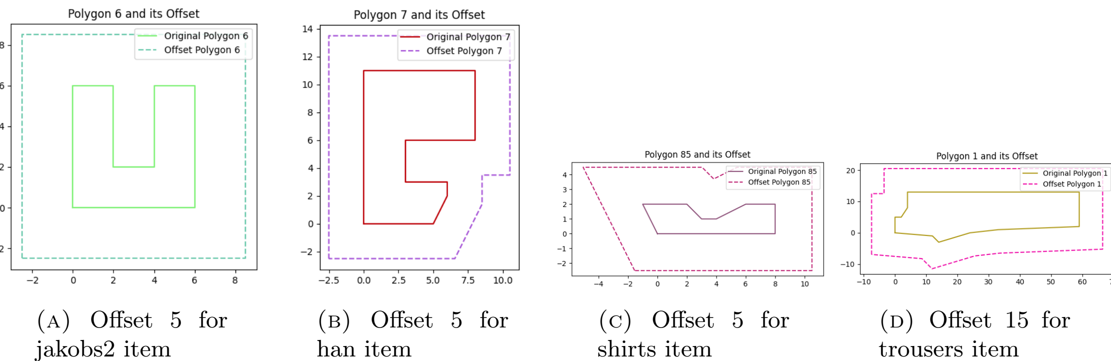
The equidistant offset method, inspired by Wang et al. (2022), uses angular bisectors and vector operations to maintain the geometric properties of the original shape when inducing product spacing. While effective with small offset distances, it struggles when the offset distance exceeds the size of a notch in the original polygon, as shown in Figure 6.5.
To overcome this, the buffer method, utilizing Shapely's buffer function, was introduced. This method approximates the offset polygon by creating a buffer around the original shape and deliberately skipping closely located vertices when their distance is too small relative to the offset. This results in a slightly simplified shape, see Figures 6.6, but ensures reliable offset generation even for



(A) Offset 1.5 for jakobs2 item
(B) Offset 5 for same jakobs2 item
(C) Offset 2 for han item
(D) Offset 5 for same han item
(E) Offset 2 for shirts item
(F) Offset 5 for same shirts item
(G) Offset 5 for trousers item
(H) Offset 15 for same trousers item

FIGURE 6.5: Offset examples for complex shapes using the original offset method.

larger offset distances. Although this may lead to slightly larger offset areas and potentially more wasted space, the primary goal is to ensure sufficient product spacing for high-quality product cuts, which is more important for the company than strictly minimizing waste (when choosing this option). The pseudo-code for this simplified method can be found in Appendix E.



(A) Offset 5 for jakobs2 item
(B) Offset 5 for han item
(C) Offset 5 for shirts item
(D) Offset 15 for trousers item

FIGURE 6.6: Offset generation using the buffer method for the various shapes.

Appendix O presents the nesting results for items with applied product spacing. These results use the equidistant method with carefully chosen spacing distances to avoid any issues. Notably, none of the test instances exhibited overlapping problems.

6.7 Conclusion

The performance analysis revealed valuable insights into our algorithm's strengths and areas for improvement. It performed less efficiently for some JP1 and JP2 instances, particularly where average piece sizes were larger, as highlighted by the B, H, and Q classes in JP1 and class T in JP2, leading to suboptimal sheet utilization and utilization efficiency gaps of approximately 40%. Additionally, type E in JP1 showed poor performance, suggesting that rotation could improve packing for narrow elements. However, the algorithm excelled for classes with smaller piece sizes, achieving average utilization efficiency gaps of 17% for the JP2 cases and just 6.6% for the JP1 cases. Overall, the methods of LocalSearch and Martinez-Sykora et al. outperformed ours, indi-

cating a need for better assignment strategies and rotation options.

In the strip packing Nest-SB category, our algorithm encountered challenges with highly complex instances that have a high number of vertices and intricate item shapes, like swim and shirts, but performed well on simpler ones, such as albano, dighe2, and fu, with average utilization efficiency gaps of 7.3% and 3% compared to LocalSearch and SCH & PBP, respectively. This indicates that, while our algorithm is effective in handling more balanced item sizes and simpler shapes, there is room for improvement, especially in addressing more complex geometries. Improved rotation and assignment strategies, like PBP and SCH, could enhance performance for intricate shapes and larger item sets.
For Nest-MB, our results for dighe2, trousers, and albano were less effective than in Nest-SB. Albano's lower performance likely stems from sub-optimal assignment ordering. This is further highlighted by the significant improvements in the poly instances, resulting from assignment orderings like SCH and PBP. The results indicate that effective assignment ordering is still crucial in Nest-MB.
In Nest-LB, our algorithm performed competitively, with an average utilization efficiency gap of just 7.2% across all instances and all methods. It matched Zhang et al.'s results for poly1a and outperformed Martinez-Sykora et al.'s strategies by 1.3% and even by 2.4% for mao. These findings support the notion that in larger bins, efficient packing strategies can outweigh the importance of assignment strategies, achieving high utilization efficiency even with sub-optimal assignments.

Computation times varied significantly across different datasets and instances, with the JP1 and JP2 datasets averaging 20 minutes, while Nest-SB, Nest-MB, and Nest-LB had notably longer computation times, especially for complex instances like swim, which took over 200 minutes. Generally, the initial BLF heuristic iterations were efficient, only taking a long time for instances with complex shapes and NFPs or a large number of items to place. The repacking algorithm, due to its iterative nature and overlap checks, was the primary contributor to extended computation times.

Overlap issues in some strip-packing instances were encountered. To address these issues, implementing the Start Points method by Burke et al. (2007) was suggested, which improves NFP generation by considering feasible touching positions and ensures overlaps are avoided through iterative checks.

Finally, our offset generation analysis showed that while the equidistant method works well for smaller offsets, it faces limitations with more complex shapes in combination with larger offsets. The buffer method, which simplifies the offset process by approximating offsets around the original shape, addresses these issues, ensuring reliable product spacing even with intricate polygons and larger offsets despite potentially more waste. The buffer method prioritizes high-quality cut products over waste minimization. The equidistant offset method remains very capable for generating product spacing when product spacings are carefully chosen.

Overall, as a foundational and initial algorithm for company C, it performs quite well for more realistic (non-)convex items when placed on relatively larger sheets. Significant improvements can be made when placing items on relatively smaller sheets by incorporating (free) rotation options and local search techniques, like those from Wang et al. (2022), to optimize assignment ordering and enhance packing efficiency to maybe even reduce the number of sheets required.

Chapter 7: Implementation, Conclusions, and Future directions

This final chapter will wrap up the thesis by first outlining practical steps for implementing the developed algorithm in the steel manufacturing process. Section 7.2 will then present final conclusions and recommendations. Section 7.3 will present a discussion of the study's limitations, and the last section will give suggestions for future work.

7.1 Implementation

Typically, an entire chapter could be dedicated to outlining an implementation plan for a developed solution design. However, in this case, our algorithm serves as a solver and is just one part of the complete nesting process. As described in Chapter 2, the nesting process in the steel manufacturing sector begins with work order management and sheet management, where different items and types of sheets required for nesting are selected. Following this, the necessary CAD drawings are imported into CAM software, where the nesting algorithm is applied to generate an efficient nesting layout for machine cutting.

The algorithm, being a solver, is specifically tasked with optimizing the arrangement of parts on steel sheets, but it does not encompass the entire production workflow. Once the algorithm determines the optimal nesting layout, several additional steps must be taken to integrate it into the production process. These steps include:

- **NC-Code Generation:** After the nesting layout is established, the CAM software must generate NC-code, which provides the necessary tool paths for machine operations. This code is critical for dictating the exact movements and actions the machinery will undertake.
- **Post-Processing:** The generated NC-code is then translated into G-code by a post-processor. G-code is the language understood by CNC machines, allowing them to execute the specific cutting instructions derived from the nesting layout.
- **Machine Execution:** With the G-code ready, it can be sent to the CNC machines for execution, where the physical cutting of materials takes place according to the optimized layout.

Therefore, while our algorithm plays a vital role in optimizing the nesting layout, it is essential to recognize that it is one component of the entire nesting process. For the company to fully utilize this algorithm in their manufacturing operations, the surrounding infrastructure of the CAD-to-Cut process must be established and integrated.

7.2 Conclusion and recommendations

We have developed a plate nesting algorithm designed to automatically create efficient nesting layouts for cutting operations. The algorithm addresses the 2D irregular bin packing problem by determining the optimal arrangement of irregularly shaped parts on identical rectangular sheets, thereby minimizing the number of sheets used and reducing material waste. The solution involves a three-phase heuristic algorithm that determines the best placement and orientation for each part to maximize sheet utilization while adhering to constraints such as part orientation, sheet dimensions, and non-overlapping requirements, and allowing for the option of product spacing, which was a requirement of company C for industrial use.

The core objective is to achieve the most efficient nesting layout that utilizes the minimum number of sheets possible, translating to reduced material costs and improved operational efficiency.
In the first phase, parts are placed on sheets using the Bottom Left Fill heuristic, which prioritizes the placement of items at the most possible bottom left position of the sheet without overlapping with the sheet's boundaries or other placed items. This allows for the quick generation of initial feasible layouts. Several iterations of the BLF are performed using different assignment orderings, where the first iteration follows an assignment ordering like SCH and PBP, by prioritizing the larger items being placed first to let the smaller items fill in the gaps. The other assignment orderings are randomized based on the probability function: $1 - e^{(-5 * \frac{\text{iteration}}{\text{total_iterations}})}$, to escape local optima and pursue markedly different packing layouts, considering the BLF heuristic is quite sensitive to initial assignment ordering.
In the second phase, for the selected layouts that use the least amount of whole sheets, the overall utilization efficiencies are computed and those with the highest overall utilization efficiencies are selected as potential best layouts. The third and last phase focuses on refining these layouts to minimize scrap by exploring alternative placements and reconfiguring parts to enhance space utilization for the least utilized sheets. The layout resulting in the lowest K-value is in the end chosen as best layout.

For the JP1 and JP2 instances, particularly where average piece sizes were larger, our algorithm was less efficient compared to existing methods, with utilization gaps reaching up to 40% in certain cases. However, when dealing with smaller average piece sizes, the results improved significantly, reducing gaps to 17% in JP2 and 6.6% in JP1. Despite these improvements, LocalSearch and Martinez-Sykora et al.'s methods outperformed ours, highlighting a need for improved assignment strategies and rotation options.
In the Nest-SB category, our algorithm struggled with complex instances like swim and shirts, which had intricate shapes and numerous vertices. It performed well on simpler instances like albano, dighe2, and fu, with average utilization gaps of 7.3% compared to LocalSearch and 3% compared to Martinez-Sykora et al.'s strategies. These results suggest that while our algorithm excels with simpler shapes, better rotation and assignment strategies could improve performance with complex geometries.
For the Nest-MB category, the algorithm's performance on instances like dighe2, trousers, and albano was hindered by suboptimal assignment orderings. These conclusions were highlighted by the significant improvements observed in the poly instances, resulting largely from the assignment orderings like SCH and PBP, underscoring the importance of effective assignment ordering in situations with relatively smaller sheet dimensions. Although we achieved slightly better result for the mao instance compared to Martinez-Sykora et al. (2017), the LocalSearch by Zhang et al. (2022) still demonstrated superior overall performance.
In the Nest-LB category, our algorithm generally performed well with an average utilization gap of just 7.2%. We matched Zhang et al.'s results for poly1a and outperformed Martinez-Sykora et al.'s strategies by 1.3% and mao by even 2.4%. These results align with the idea that, for larger sheets, packing strategies can outweigh the importance of assignment strategies.

Based on all these findings, we recommend adopting assignment strategies like PBP and SCH, as we did for our very first iteration, which have consistently yielded better nesting layouts for cases with relatively smaller sheets. These strategies can enhance the order in which items are placed, leading to more efficient use of sheets when combined with the BLF placement heuristic.

Additionally, integrating local search techniques, such as those proposed by Zhang et al. (2022) and Wang et al. (2022), could further improve packing efficiency. These methods refine the initial assignment by strategically swapping and relocating items between sheets, typically shifting items from less utilized sheets to higher utilized sheets. This approach optimizes item placement and can sometimes reduce the number of sheets required. Adopting such a method can improve the overall utilization efficiency and lead to more efficient nesting solutions, as demonstrated in the studies by Zhang et al. (2022) and Wang et al. (2022).

Incorporating rotation flexibility, as suggested by Wang et al. (2022) and Martinez-Sykora et al. (2017), is also advisable. Allowing items to rotate freely can address layout gaps more effectively, especially for larger or more complex shapes that have a lower rectangularity factor.

Our analysis of offset generation showed that the equidistant method is effective for creating small product spacing around shapes, but it struggled with shapes that have notches smaller than the desired spacing. To address this, we implemented an equidistant buffer method that simplifies the offset process by approximating the offset shape around the original shape and ensuring reliable product spacing, though it may slightly increase waste. The buffer method prioritizes product quality over strict waste minimization. We recommend using the equidistant method whenever it can successfully generate the desired offsets, and the buffer method for cases requiring larger product spacing in combination with shapes that have smaller cut-outs to maintain spacing reliability.

7.3 Discussion

The developed nesting algorithm shows promising performance and provides a solid foundation for future improvements. However, several areas present opportunities for refinement, which we discuss in this section.

While no company-specific test cases were included to evaluate the algorithm, this decision was based on two factors. First, the irregular strip packing instances used in this study closely resemble the types of items that company C and its clients commonly cut from metal sheets. Instances like jakobs1 are particularly relevant for steel construction tasks, while others such as albano, trousers, hand, mao, swim, shirts, shapes, and jakobs align well with scenarios encountered in (mechanical) engineering. These similarities allowed for effective testing without needing company-specific cases. Second, our algorithm relies on the coordinates of the vertices of the items and sheet dimensions, whereas company C uses DXF files and CAD software, which are not directly compatible with our algorithm. Converting the company's test cases into a usable format would have required significant time to extract the coordinate vertices of every item. Given the resemblance of existing instances to the company's typical items and time constraints, it was determined that company-specific test cases were not essential for this study.

One area for improvement is the computation time. While the algorithm is effective, it currently requires more time to arrive at solutions compared to state-of-the-art algorithms from the 2DIBPP literature. While other algorithms often arrive at solutions in mere seconds or minutes, our algorithm typically requires tens of minutes and occasionally over an hour to deliver a solution. Python, which is used for the current implementation, offers ease of development and debugging but is generally slower in execution compared to lower-level programming languages like C++. These languages are known for their faster execution speeds, which can significantly reduce computation times. Thus, rewriting the algorithm in a more efficient language could reduce the computation time significantly.

Additionally, there is potential for optimizing the code structure itself. While the current implementation is functional and produces good results, further optimization could streamline processes and enhance performance. Implementing techniques such as parallel processing could also offer substantial improvements in computation speed, especially for more complex or dense instances.
That brings us to the repacking algorithm, which is an important component of our nesting solution, aimed at refining the initial packing arrangement of the least utilized sheet by iteratively adjusting item positions to minimize overlap and maximize space utilization. Even though the current implementation works, the iterative nature can lead to significant computation times, especially when dealing with complex or dense instances, such as the Nest-LB's dighe instances.
Parallelizing the process could reduce computation time, by splitting the computational workload across multiple processing units, without sacrificing accuracy. Additionally, exploring alternative strategies, such as simulated annealing or other meta-heuristics, might offer faster convergence to optimal packing arrangements, balancing solution quality and computation time. Simulated an-

nealing, for example, could provide a way to explore the search space more effectively, avoiding the exhaustive iterative search while still maintaining a good chance of finding a near-optimal solution.

Second, in some strip-packing instances, overlapping issues were encountered in some of their iterations, particularly in complex shapes with interlocking concavities or jigsaw-like features. These overlaps are due to the current limitations in the NFP generation. The existing method, inspired by Mahadevan (1984), struggles with shapes involving interlocking concavities or holes, as observed in the 'han' and 'jakobs' instances. This limitation leads to an inability to generate complete NFPs, causing overlaps and inefficient packing.
To address these issues, we should implement the Start Points method developed by Burke et al. (2007). This method improves NFP generation by identifying feasible touching positions and preventing overlaps through iterative checks. Integrating this approach with our current algorithm, and possibly incorporating a separation algorithm like the one by Wang et al. (2022), can generate more accurate NFPs, avoid overlap issues, and enhance packing efficiency.

Parameter tuning also plays a significant role in the algorithm's performance. While the current parameters have been chosen to balance solution quality and computation time, a more comprehensive sensitivity analysis could refine this balance even further, optimizing performance across a wider range of instances.

The algorithm currently also has limited capabilities in terms of free piece rotation, impacting its performance in instances where such a rotation could significantly improve packing efficiency, such as the JP1's type E performance. The lack of effective (free) rotation handling can lead to inefficient use of space and increased material waste.

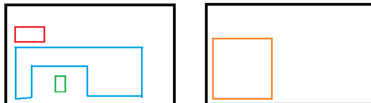


FIGURE 7.1: Repacking multiple sheets for better nesting solutions.

The current algorithm focuses on repacking only the least utilized sheet to maximize the remnant sheet area. While this approach can lead to efficient layouts, it may not always yield the best solutions. In some cases, a different sheet that is not the least utilized could be poorly packed and might benefit from repacking, potentially leading to better overall solutions. For example, Figure 7.1 shows that swapping the red and green items could enhance packing efficiency by increasing remnant sheet space. Repacking all sheets could thus potentially result in more optimal layouts, creating more remnant sheets to be used for future orders, or creating enough room to place extra items. However, performing repacking across all sheets could lead to an exploding in runtime due to the increased computational complexity. This prevented further exploration of this approach within the current timeframe of this research, indicating a need for future research to address and refine this aspect.

Finally, the algorithm currently assumes that all sheets are of identical rectangular shape, a simplification that may not be realistic in industrial contexts. In practice, industries often deal with remnant sheets of varying shapes and sizes, which are used for nesting future orders. The algorithm does not support the combination of different sheet sizes in one nesting, resulting in remnant sheets being left unused and not considered for future packing when applied to industrial use.

7.4 Further research

In this section, we propose several areas for future research, building upon the findings and insights gained from this research.

Future research should focus on incorporating the local search strategy by Wang et al. (2022).

Due to time constraints and the complexity of implementing the separation algorithm, this aspect was not fully explored in the current study. Integrating this separation algorithm would allow for effective implementation of the local search, which, as concluded earlier, is expected to improve overall utilization efficiency and nesting layout solutions. The strategy seeks to optimize nesting layouts by moving and swapping items from less utilized sheets to others in such a way no overlap occurs and the F-value increases. By repeatedly applying this method, the algorithm can increase material utilization and potentially reduce the number of sheets used. Apart from the separation algorithm, the other foundational algorithms for this local search have already been developed, providing a solid base for future enhancements.

Another promising area for future research is the effective incorporation of (free) rotations into the algorithm. The method proposed by Abeysooriya et al. (2018), and subsequently adopted by Wang et al. (2022), offers a systematic approach to applying a fixed set of rotations to items and reducing the infinite set of possible orientations, when opting for free rotation of items, to a finite set of promising angles. By identifying angles based on the arrangement of pieces in a partial solution, this method defines new promising angles of orientation resulting from each touching point or edge with the already placed pieces or sheet edges. It continues this process until no further improvements in packing layout are identified. Implementing this approach could enhance both speed and efficiency, allowing for more optimal utilization of sheet space. Future research could thus explore developing a method to focus on the most promising rotation angles, rather than relying on the current randomization approach.

While this research primarily focused on developing and optimizing the nesting algorithm itself, several practical manufacturing elements were beyond its scope. These include considerations such as lead-in and lead-out, bevel, part-specific zoning, and prioritization of parts. Future research should aim to incorporate these practical elements into the algorithm, as well as the development of surrounding software. By enabling features that allow for customization based on client needs, the algorithm can be made more applicable to real-world industrial contexts, ultimately providing the company's clients with the ability to generate nestings that are fully tailored to their specific requirements.

In Chapter 2, we described two types of cutting machines currently in use at company C: the flatbed machines and pass-through machines. The current algorithm was designed with flatbed machines in mind, as these machines have fewer restrictions and requirements compared to pass-through machines, which involve sheet movement and necessitate nesting in columns. Future research could explore expanding the algorithm to accommodate the additional constraints imposed by pass-through machines, potentially broadening the algorithm's applicability across different machine types.

Lastly, as mentioned in the Discussion, our current algorithm assumes that all sheets are rectangular and of identical dimensions. This assumption poses limitations for industries that wish to utilize remnant sheets of varying shapes and sizes for nesting orders, as is also the case for company C. To address this limitation, it is worthwhile to explore the 2D irregular multiple bin-size packing problem. This problem involves packing irregularly shaped items into rectangular bins with different dimensions, a challenge that has been explored in some existing literature. Furthermore, Yao et al. (2024) have conducted research that considers an extra complexity, namely packing irregular items into irregularly shaped bins. For future research, investigating these problem types- nesting on irregularly shaped sheets or on sheets with differing rectangular dimensions- presents an opportunity to significantly enhance the algorithm's practical applicability. By developing a solution that can efficiently handle a diverse range of sheet sizes and shapes, industries would benefit from improved material utilization, reduced waste, and increased flexibility in manufacturing processes.

Bibliography

Abeysooriya, R. P., Bennell, J. A., & Martinez-Sykora, A. (2018). Jostle heuristics for the 2d-irregular shapes bin packing problems with free rotation. *International Journal of Production Economics*, *195*, 12–26.

Adamowicz, M., & Albano, A. (1976). Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design*, *8*(1), 27–33.

Alma. (2024). Alma is launching nest&cut, web-based application for automatic nesting [Accessed on March 7]. https://almacam.com/nest-and-cut-web-based-application-for-automatic-nesting/

Alvarez-Valdes, R., Martinez, A., & Tamarit, J. (2013). A branch & bound algorithm for cutting and packing irregularly shaped pieces. *International Journal of Production Economics*, *145*(2), 463–477.

Andy. (2021). Nesting - everything you need to know | scan2cad. https://www.scan2cad.com/blog/cnc/everything-about-nesting/

AoPSOnline. (2024). Shoelace theorem [Accesed on July 7]. https://artofproblemsolving.com/wiki/index.php/Shoelace_Theorem

Babu, A. R., & Babu, N. R. (2001). A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, *33*(12), 879–891.

Bennell, J. A., Dowsland, K. A., & Dowsland, W. B. (2001). The irregular cutting-stock problem—a new procedure for deriving the no-fit polygon. *Computers & Operations Research*, *28*(3), 271–287.

Bennell, J. A., & Oliveira, J. F. (2008). The geometry of nesting problems: A tutorial. *European journal of operational research*, *184*(2), 397–415.

Bennell, J. A., & Song, X. (2008). A comprehensive and robust procedure for obtaining the nofit polygon using minkowski sums. *Computers & Operations Research*, *35*(1), 267–281.

Bennell, J. A., & Song, X. (2010). A beam search implementation for the irregular shape packing problem. *Journal of Heuristics*, *16*, 167–188.

Burke, E., Hellier, R., Kendall, G., & Whitwell, G. (2006). A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations research*, *54*(3), 587–601.

Burke, E. K., Hellier, R. S., Kendall, G., & Whitwell, G. (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, *179*(1), 27–49.

Cadix. (2024). Wat is cad - cam en wat kunt u er mee? | cadix [Accessed on February 12]. https://www.cadix.nl/specialisaties/werktuigbouw/cad-cam

Cai, S., Deng, J., Lee, L. H., Chew, E. P., & Li, H. (2023). Heuristics for the two-dimensional irregular bin packing problem with limited rotations. *Computers & Operations Research*, *160*, 106398.

Cherri, L. H., Cherri, A. C., & Soler, E. M. (2018). Mixed integer quadratically-constrained programming model to solve the irregular strip packing problem with continuous rotations. *Journal of Global Optimization*, *72*, 89–107.

Cherri, L. H., Mundim, L. R., Andretta, M., Toledo, F. M., Oliveira, J. F., & Carravilla, M. A. (2016). Robust mixed-integer linear programming models for the irregular strip packing problem. *European Journal of Operational Research*, *253*(3), 570–583.

Coffman, E. G., Csirik, J., Galambos, G., Martello, S., Vigo, D., et al. (2013). Bin packing approximation algorithms: Survey and classification. In *Handbook of combinatorial optimization* (pp. 455–531). Springer.

CUNINGHAMEGREEN, R. (1989). Geometry, shoemaking and the milk tray problem. *New Scientist*, *123*(1677), 50–53.

Dean, H. T., Tu, Y., & Raffensperger, J. F. (2006). An improved method for calculating the no-fit polygon. *Computers & operations research*, *33*(6), 1521–1539.

ESAB. (2024). What is a nesting system and why do you need one? https://esab.com/ae/mea_en/esab-university/blogs/what-is-a-nesting-system-and-why-do-you-need-one/

Eziil. (2024). The best nesting software for laser and plasma cutting [Accessed on March 4]. https://eziil.com/best-nesting-software-for-laser-cutting

Ghosh, P. K. (1991). An algebra of polygons through the notion of negative shapes. *CVGIP: Image Understanding*, *54*(1), 119–144.

Goodman, E. D., Tetelbaum, A. Y., & Kureichik, V. M. (1994). A genetic algorithm approach to compaction, bin packing, and nesting problems. *Case Center for Computer-aided Engineering and Manufacturing. Michigan State University*, *940702*.

Guerriero, F., & Saccomanno, F. P. (2023). A hierarchical hyper-heuristic for the bin packing problem. *Soft Computing*, *27*(18), 12997–13010.

Guo, B., Zhang, Y., Hu, J., Li, J., Wu, F., Peng, Q., & Zhang, Q. (2022). Two-dimensional irregular packing problems: A review. *Frontiers in Mechanical Engineering*, *8*, 966691.

Han, W., Bennell, J. A., Zhao, X., & Song, X. (2013). Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints. *European journal of operational research*, *230*(3), 495–504.

Hansen, P., & Mladenović, N. (2006). First vs. best improvement: An empirical study. *Discrete Applied Mathematics*, *154*(5), 802–817.

Hu, X., Li, J., & Cui, J. (2020). Greedy adaptive search: A new approach for large-scale irregular packing problems in the fabric industry. *IEEE Access*, *8*, 91476–91487.

HyperTherm. (2024). How cam software works. https://www.hypertherm.com/en-US/solutions/technology/cam-software/

Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European journal of operational research*, *88*(1), 165–181.

Jetcam. (2024). What is nesting software, how to improve efficiency? [Accessed on February 12]. https://www.jetcam.net/nesting-software-101.htm

Jones, D. R. (2014). A fully general, exact algorithm for nesting irregular shapes. *Journal of Global Optimization*, *59*(2), 367–404.

Lambora, A., Gupta, K., & Chopra, K. (2019). Genetic algorithm- a literature review. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 380–384.

Leao, A. A., Toledo, F. M., Oliveira, J. F., Carravilla, M. A., & Alvarez-Valdés, R. (2020). Irregular packing problems: A review of mathematical models. *European Journal of Operational Research*, *282*(3), 803–822.

Li, Z., & Milenkovic, V. (1995). Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operational Research*, *84*(3), 539–561.

Liu, D., Tan, K. C., Huang, S., Goh, C. K., & Ho, W. K. (2008). On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research*, *190*(2), 357–382.

Liu, Q., Zeng, J., Zhang, H., & Wei, L. (2020). A heuristic for the two-dimensional irregular bin packing problem with limited rotations. *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices: 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020, Proceedings 33*, 268–279.

López Camacho, E., et al. (2012). An evolutionary framework for producing hyper-heuristics for solving the 2d irregular bin packing problem [Ph.D thesis].

López-Camacho, E., Ochoa, G., Terashima-Marín, H., & Burke, E. K. (2013). An effective heuristic for the two-dimensional irregular bin packing problem. *Annals of Operations Research*, *206*, 241–264.

López-Camacho, E., Terashima-Marin, H., Ross, P., & Ochoa, G. (2014). A unified hyper-heuristic framework for solving bin packing problems. *Expert Systems with Applications*, *41*(15), 6876–6889.

Mahadevan, A. (1984). *Optimization in computer-aided pattern packing (marking, envelopes)*. North Carolina State University.

Martinez-Martinez, G., Sanchez-Romero, J., Jimeno-Morenilla, A., & Mora-Mora, H. (2021). An improved nesting algorithm for irregular patterns [Preprints]. https://doi.org/10.20944/preprints202110.0334.v1

Martinez-Sykora, A., Alvarez-Valdés, R., Bennell, J. A., Ruiz, R., & Tamarit, J. M. (2017). Matheuristics for the irregular bin packing problem with free rotations. *European Journal of Operational Research*, *258*(2), 440–455.

Mathew, T. (2012). Genetic algorithm. *Report submitted at IIT Bombay*, *53*.

Milenkovic, V., Daniels, K., & Li, Z. (1991). Automatic marker making. *Proceedings of the Third Canadian Conference on Computational Geometry*, 243–246.

Mundim, L. R., Andretta, M., Carravilla, M. A., & Oliveira, J. F. (2018). A general heuristic for two-dimensional nesting problems with limited-size containers. *International Journal of Production Research*, *56*(1-2), 709–732.

Mundim, L. R., Andretta, M., & de Queiroz, T. A. (2017). A biased random key genetic algorithm for open dimension nesting problems using no-fit raster. *Expert Systems with Applications*, *81*, 358–371.

Nest&Cut. (2024). Nest&cut | cloud-based 2d nesting software | cnc online cutting optimization [Accesed on March 7]. https://nestandcut.com/why-use-nest-and-cut/

Oliveira, J. F. C., & Ferreira, J. A. S. (1993). Algorithms for nesting problems. *Applied simulated annealing*, 255–273.

Omar, M. K., & Ramakrishnan, K. (2013). Solving non-oriented two dimensional bin packing problem using evolutionary particle swarm optimisation. *International Journal of Production Research*, *51*(20), 6002–6016.

Parreño, F., Alvarez-Valdés, R., Oliveira, J. F., & Tamarit, J. M. (2010). A hybrid grasp/vnd algorithm for two-and three-dimensional bin packing. *Annals of Operations Research*, *179*, 203–220.

Rao, Y., Wang, P., & Luo, Q. (2021). Hybridizing beam search with tabu search for the irregular packing problem. *Mathematical Problems in Engineering*, *2021*, 1–14.

Santoro, M. C., & Lemos, F. K. (2015). Irregular packing: Milp model based on a polygonal enclosure. *Annals of Operations Research*, *235*(1), 693–707.

Shalaby, M. A., & Kashkoush, M. (2013). A particle swarm optimization algorithm for a 2-d irregular strip packing problem.

Silva, E., Alvelos, F., & De Carvalho, J. V. (2010). An integer programming model for two-and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, *205*(3), 699–708.

Stoyan, Y., Scheithauer, G., Gil, N., & Romanova, T. (2004). -functions for complex 2d-objects. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, *2*(1), 69–84.

Stoyan, Y., Terno, J., Scheithauer, G., Gil, N., & Romanova, T. (2002). Phi-functions for primary 2d-objects. *Stud. Inform. Univ.*, *2*(1), 1–32.

SVGnest. (2024a). Github - jack000/svgnest: An open source vector nesting tool [Accessed on February 5]. https://github.com/Jack000/SVGnest?tab=readme-ov-file%5C#what-is-nesting

SVGnest. (2024b). Svgnest - free and open source nesting for cnc machines, lasers and plasma cutters [Accessed on March 5]. https://svgnest.com/

Tay, F. E., Chong, T., & Lee, F. (2002). Pattern nesting on irregular-shaped stock using genetic algorithms. *Engineering Applications of Artificial Intelligence*, *15*(6), 551–558.

Terashima-Marín, H., Ross, P., Farías-Zárate, C., López-Camacho, E., & Valenzuela-Rendón, M. (2010). Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, *179*, 369–392.

The Association of European Operational Research Societies. (2024). 2d irregular datasets of esicup cutting and packing [Accessed on July 26]. https://www.euro-online.org/websites/esicup/data-sets/#1535972088237-bbcb74e3-b507

Toledo, F. M., Carravilla, M. A., Ribeiro, C., Oliveira, J. F., & Gomes, A. M. (2013). The dotted-board model: A new mip model for nesting irregular shapes. *International Journal of Production Economics*, *145*(2), 478–487.

Tormach. (2024). What is cad, cam and g-code? [Accessed on February 12]. https://tormach.com/articles/what-is-cad-cam-gcode/

University, E. (2024). What is cutting kerf and why is it important? [Accessed on February 26]. https://esab.com/us/nam_en/esab-university/blogs/what-is-cutting-kerf-and-why-is-it-important/

Wang, Z., Chang, D., & Man, X. (2022). Optimization of two-dimensional irregular bin packing problem considering slit distance and free rotation of pieces. *International Journal of Industrial Engineering Computations*, *13*(4), 491–506.

Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European journal of operational research*, *183*(3), 1109–1130.

Watts, L. (2024). 8 best nesting software for laser cutters (free & paid) - cncsourced. https://www.cncsourced.com/rankings/best-nesting-software-for-laser-cutters/

Xie, S. Q., Wang, G. G., & Liu, Y. (2007). Nesting of two-dimensional irregular parts: An integrated approach. *International Journal of Computer Integrated Manufacturing*, *20*(8), 741–756.

Xu, Y.-x., et al. (2016). An efficient heuristic approach for irregular cutting stock problem in ship building industry. *Mathematical problems in engineering*, *2016*.

Yang, S. (2024). Seanys/2d-irregular-packing-algorithm: Realize 2d irregular packing algorithm with python [Accessed on July 7]. https://github.com/seanys/2D-Irregular-Packing-Algorithm/tree/master

Yang, Y., Liu, B., Li, X., Jia, Q., Duan, W., & Wang, G. (2024). Fidelity-adaptive evolutionary optimization algorithm for 2d irregular cutting and packing problem. *Journal of Intelligent Manufacturing*, 1–19.

Yao, S., Tang, C., Zhang, H., Wu, S., Wei, L., & Liu, Q. (2024). An iteratively doubling binary search for the two-dimensional irregular multiple-size bin packing problem raised in the steel industry. *Computers & Operations Research*, *162*, 106476.

Zhang, H., Liu, Q., Wei, L., Zeng, J., Leng, J., & Yan, D. (2022). An iteratively doubling local search for the two-dimensional irregular bin packing problem with limited rotations. *Computers & Operations Research*, *137*, 105550.

Appendices

A Problem cluster

Figure 2 depicts the problem cluster, illustrating our progression from the action problem to the core problems and their associated consequences.

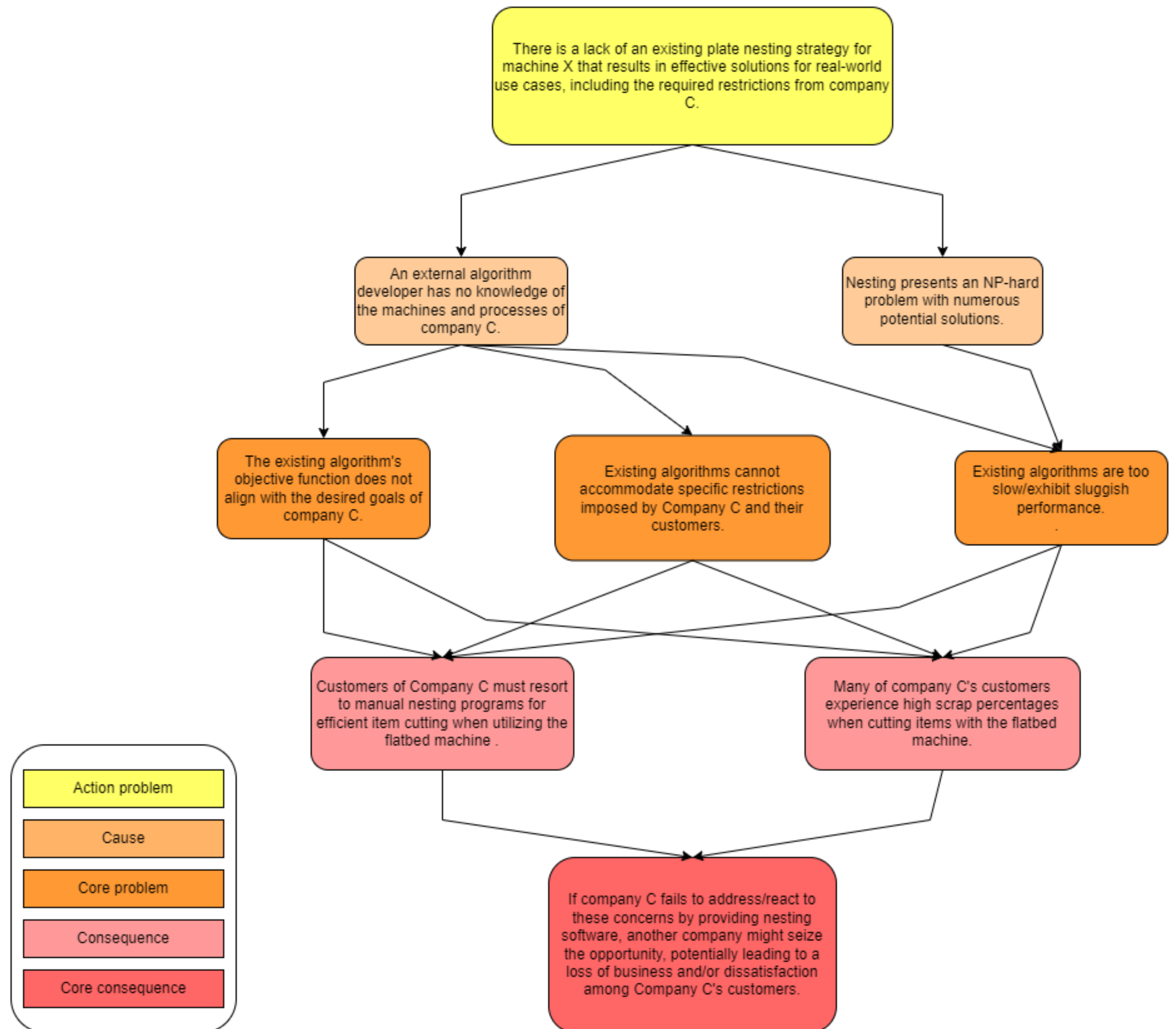


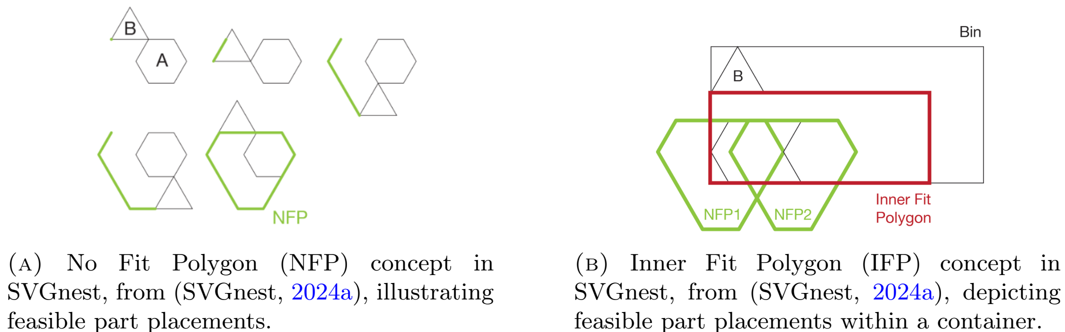
FIGURE 2: Problem cluster displaying the interconnected causes, core problems, and consequences stemming from the action problem.

B Overview of nesting software

B.1 SVGnest

As mentioned before, SVGnest is a browser-based vector nesting tool that offers a free and open-source alternative for resolving nesting challenges. Utilizing a genetic algorithm for global optimization, it competently addresses nesting problems, including arbitrary containers and concave edge cases, placing it on par with commercial counterparts. Notably, SVGnest supports part-in-part functionality, allowing parts to be positioned within the voids of other parts.

The nesting strategy of SVGnest comprises two fundamental aspects: placement strategy and optimisation strategy. For part placement, SVGnest utilizes the NFP and IFP concepts to determine feasible part placements. By orbiting one polygon around another, ensuring they touch but do not intersect, SVGnest derives the NFP, indicating feasible part placements, see Figure 3. Similarly, an "Inner Fit Polygon" (IFP) can be constructed for the part and the bin, serving a similar purpose, see Figure 3b. When multiple parts are placed, the union of the NFPs of previously placed parts is computed to facilitate subsequent placements.



(A) No Fit Polygon (NFP) concept in SVGnest, from (SVGnest, 2024a), illustrating feasible part placements.

(B) Inner Fit Polygon (IFP) concept in SVGnest, from (SVGnest, 2024a), depicting feasible part placements within a container.

FIGURE 3: NFP and IFP in SVGnest.

In terms of optimization, SVGnest adopts the "First-Fit-Decreasing" heuristic, prioritizing larger parts during placement. This heuristic is augmented by a genetic algorithm, which explores the solution space iteratively to refine the nesting layout (SVGnest, 2024a).
A genetic algorithm (GA) operates by evolving a population of potential solutions over successive generations, akin to natural selection. Individuals representing candidate solutions undergo genetic operations such as selection, crossover, and mutation to produce new offspring with favorable characteristics (i.e. those that perform well according to a predefined fitness function). These offspring gradually refine the population toward optimal solutions, making GAs particularly suitable for complex optimization problems (Lambora et al., 2019), (Mathew, 2012). In SVGnest the insertion order and the rotation of the parts form the gene for the GA. Furthermore, the fitness function for the GAs minimizes unplaceable parts, the number of bins used, and the width of all placed parts. While small mutations in the gene can lead to significant fitness changes, caching NFPs facilitates rapid evaluation of new individuals.

Additionally, SVGnest offers configurable parameters such as space between parts, curve tolerance, part rotations, GA population size, mutation rate, part-in-part functionality, and exploration of concave areas, depicted in Figure 4.
Despite its strengths, SVGnest has notable drawbacks. It exclusively supports SVG (Scalable Vector Graphics) file formats, may limit compatibility with CAD programs and other file formats such as DXF (Drawing Exchange Format). Moreover, it lacks advanced configuration options, preventing users from specifying thickness dimensions or preventing individual items from rotating. Additionally, users cannot adjust individual part clearance, which can be crucial for preventing warping due to heat. The software also occasionally encounters overlapping issues, particularly when using the "Explore concave areas" configuration.
Furthermore, performance may degrade when adding space between parts, and the software may

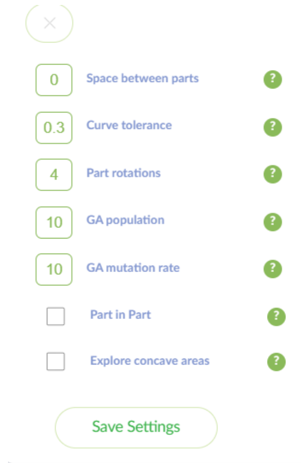


FIGURE 4: Configuration parameters in SVGnest, from (SVGnest, 2024b), highlighting adjustable settings for nesting optimization.

not provide clear feedback when dealing with oversized items or failing to generate a nesting layout.

However, one notable feature of SVGnest is its ability to automatically minimize the number of sheets required to efficiently nest all items, without the need to specify a minimum number of sheets beforehand. It also depicts the material utilization while nesting.

Regarding the optimization strategy for the remnant sheet, SVGnest aims to minimize the overall horizontal width, similar to the approach used by company C's software, as can be seen in Figure 5. This is also the only example that can be given for SVGnest, since SVGnest struggled to find nesting layouts for both examples.

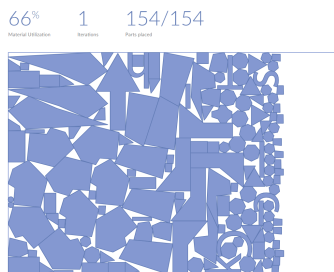


FIGURE 5: SVGnest output of the demo from (SVGnest, 2024b), after 5 minutes, showcasing the nesting layout generated by the software.

B.2 Deepnest

Deepnest is another open-source nesting software that combines desktop and online technologies. While it may lack some of the flashy features found in professional software, it excels in simplicity and optimization. Developed by the same team behind SVGnest, Deepnest offers enhanced sophistication and a wider array of configuration options. One of Deepnest's notable features it its comprehensive set of nesting configuration parameters:

- **Space between parts**: This parameter allows users to specify the minimum space between each part.
- **Curve tolerance**: Users can determine the maximum acceptable error when approximating curved sections into line segments, influencing precision and speed.

- **Part rotations**: Deepnest enables users to specify the number of rotations to attempt when inserting a part, providing flexibility for irregular shapes.
- **Optimization type**: Deepnest offers three optimization types to choose from, each with its own advantages, see Figure 6.

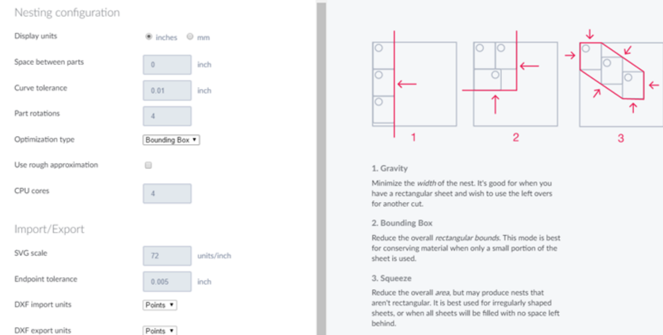


FIGURE 6: The different optimization types in Deepnest, providing users with various strategies for nesting optimization.

- **Use rough approximation**: Allow users to opt for a simpler polygon approximation to expedite the nesting process, albeit at the cost of material efficiency.

In addition to nesting configuration, Deepnest provides options for laser settings and meta-heuristic fine-tuning:

- **Merge common lines**: Deepnest automatically merges part edges that touch, reducing cut time and heat warping.
- **Optimization ratio**: This parameter allows users to balance between saving time and material, providing flexibility in decision-making.
- **Genetic population size**: Users can adjusts the size of the population for GAs, impacting processing time and diversity.
- **Genetic mutation rate**: This parameter controls the degree of mutation in each trial, affecting the variety of nest arrangements explored.

One standout feature of Deepnest is its user-friendly interface, which simplifies the nesting process into three straightforward steps: importing the file, selecting the largest part as the sheet, and starting the optimization process. It will run an iterative algorithm that keeps reducing material waste until you hit stop or it reaches the number of iterations you intended. The software automatically merges common lines, when set on, and nests smaller parts inside larger cavities for optimal material usage.

Deepnest supports both SVG and DXF file formats, making it compatible with a wide range of design software. However, it does come with some drawbacks. For example, importing multiple files simultaneously isn't supported, requiring users to manually select each file. Additionally, it doesn't automatically select between different sheet sizes and scraps. Instead, users must define the shape and size of the sheet and mark them accordingly for nesting.
Furthermore, Deepnest doesn't generate extra sheets automatically when needed, necessitating users to specify the number of sheets it can use beforehand. If only one sheet is designated and it proves insufficient for nesting all items, the program won't generate additional sheets but simply indicate that several parts remain un-nested, see Figure 7.
Moreover, Deepnest lacks clear error messages when items are too large to nest on the sheet. Compared to SVGnest, it also lacks a material utilization/efficiency button and fails to specify which items are not nested when unable to complete the process. Additionally, it lacks features such as part mirroring, fixing individual items from rotating, or specifying part clearance per item.

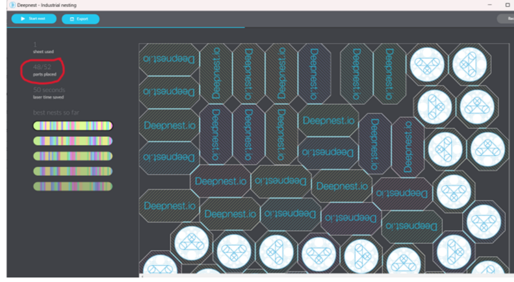


FIGURE 7: Several parts haven't been nested in Deepnest, indicating that one sheet isn't sufficient for the nesting process.

Although it can perform part-in-part nesting, it struggles with larger item quantities or specific part gap configurations in some cases.

B.3 Nest&Cut

Nest&Cut, developed by Alma, is a sophisticated web-based application designed to optimize the arrangement of shapes on a given sheet or material to minimize waste and maximize material utilization. Leveraging advanced nesting algorithms, Nest&Cut offers subscription-based access to high-performance automatic nesting functions for various complex 2D shapes, making it an ideal choice for companies involved in cutting flat materials.

The application is user-friendly, requiring only a few clicks to send DXF geometric formats, specify quantities of parts and material formats, and initiate automatic nesting in the cloud. Once initiated, Nest&Cut delivers optimized nesting layouts in DXF format, including the cutting order of parts, see Figure 8, ready for use in cutting software or numerical control machines (Nest&Cut, 2024).

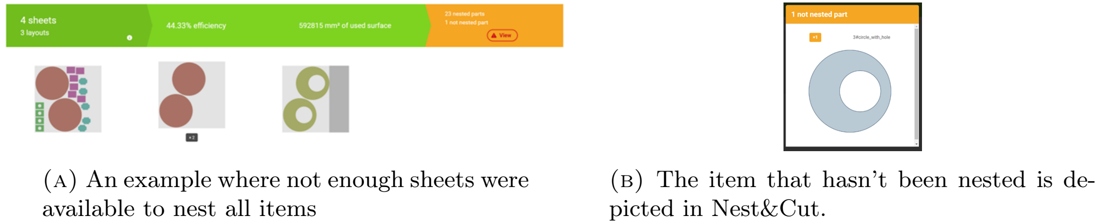
FIGURE 8: The cutting order of the second example, resulting from Nest&Cut, after running the nest for 5 minutes.

Additionally, Nest&Cut features advanced functions for automatic cleaning of DXF or DWG geometries and can improve DXF nesting layouts created using other systems. It also provides features such as CAM support, multi-format nesting, and recognition of various CNC and laser machines for exporting optimized NC files.
Nest&Cut can also be used to accurately estimate material consumption and associated costs (Alma, 2024), (Nest&Cut, 2024).

Nest&Cut offers several nesting settings, including part and sheet border gaps, rotation angle steps, flipping allowance, and three different nesting computation limits (fast, balanced, and quality) to compromise between material yield and available time.

Despite its strengths, Nest&Cut has some limitations. For instance, it requires users to specify the number of sheets that can be used, after which the software determines the minimum

number of sheets needed for the nesting. However, it does reveal which items have not been nested if sufficient sheets are available, see Figure 9.



(A) An example where not enough sheets were available to nest all items

(B) The item that hasn't been nested is depicted in Nest&Cut.

FIGURE 9: Example where not all items are nested on the available sheets in Nest&Cut.

Comparing Nest&Cut to other nesting software, it offers advantages such as the ability to select multiple sheet sizes, allowing for the utilization of remnant sheets, and prioritize sheets. It provides features like re-nesting, order generation, and part-in-part nesting, enhancing its nesting capabilities for various applications.

While the specific algorithms utilized by Nest&Cut are undisclosed, the resulting output closely resembles what the company software could provide. Hence, it is very plausible that the optimization strategy is centered around minimizing horizontal width, akin to the gravity optimization approach of Deepnest.

B.4 Inventor nesting

Inventor Nesting, a component of Autodesk's Inventor software suite, optimizes material usage and streamlines sheet metal fabrication. Employing algorithms, it automatically organizes parts on sheets, minimizing waste and maximizing efficiency. Key features include:

- **Integration with Inventor**: Seamlessly integrating with Autodesk Inventor, it creates a cohesive CAD/CAM environment for sheet metal design and fabrication. Users can effortlessly transfer part designs from Inventor to Inventor Nesting for nesting optimization.
- **Material utilization**: Inventor Nesting considers factors such as material type, size, and cutting parameters like thickness and density. It offers rotating options of 0°, 90°, 180°, and 270°, along with mirror options.
- **Customization options**: Users can tailor nesting layouts to specific manufacturing requirements. Parameters such as part orientation, nesting strategies, and cutting paths can be adjusted to optimize results for diverse applications.
- **Reporting and Analysis**: The software provides reporting and analysis tools to monitor material usage, nesting efficiency, and production metrics. Detailed reports enable users to identify areas for improvement and optimization.
- **Compatibility**: Inventor Nesting is compatible with a variety of cutting technologies, including laser, plasma, waterjet, and CNC punching machines. It supports common file formats for seamless integration with third-party software and equipment, allowing nesting layouts to be exported to formats like DXF and 3D.

Compared to other nesting software, Inventor Nesting offers the advantage of manual adjustments to each individual item, as depicted in Figure 10. Users can also fix certain items from rotating, providing greater control over the nesting process.
Furthermore, Inventor Nesting allows for the selection of multiple sheet sizes, and multi-format nesting is supported. The software automatically sets the number of sheets that can be used to ∞, eliminating the need for manual adjustments compared to other nesting software.

In the "Create Nest Study" tab, users can specify the position of the first nested item, set minimum and maximum computation times and define the yield percentage. Additionally, three remnant optimization types are available: minimizing length, width or both.

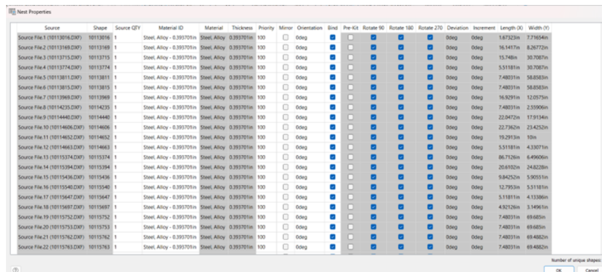


FIGURE 10: In Inventor Nesting you can make manual adjustments to each item in the nest properties tab.

C Benchmark instances explained

C.1 JP1 benchmark test instances

The JP1 benchmark instances, as described by Terashima-Marin (2010), were created to provide a comprehensive evaluation of irregular packing problems. These instances were generated using a specialized algorithm designed to produce irregular pieces, differing from traditional methods used for rectangular pieces.

The algorithm employed in generating the JP1 instances starts by randomly generating a predefined number of rectangles. These rectangles are then subdivided into irregular pieces until the desired total number of pieces is reached. The parameters governing the creation of each instance include the number of bins, their dimensions, the number of pieces per bin, the minimum piece side length, the maximum ratio between the largest and smallest side lengths (determining the irregularity or rectangularity factor), and the initial number of rectangles. Figure 11 shows an example of an instance with one bin and 10 pieces.

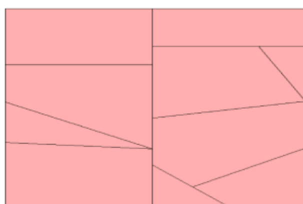


FIGURE 11: Example of irregular generated instance (Terashima-Marín et al., 2010).

The resulting pieces are convex polygons with sides ranging from 3 to 8, providing a diverse set of shapes to challenge packing algorithms. The JP1 instances encompass 540 problem instances distributed across 18 different types, with 30 instances generated for each type. These instances exhibit various characteristics such as object (bin) size, number of pieces, and number of objects, detailed in Figure 12.

Type G instances stand out as they have an unknown optimal number of objects, stemming from alternations made to randomly generated problems with known optima. Instances within each type typically have the same number of pieces per object, except for types C, K, P, and R, which feature varying configurations.

The irregularity factor of each instance type is indicative of its level of irregularity, influenced by factors such as the minimum piece side length and the ratio between the largest and smallest side lengths. Instances with smaller minimum side lengths and larger maximum ratios tend to exhibit higher irregularity, resulting in pieces with smaller rectangularity factors. The instances in problem type I consist solely of rectangles, providing a baseline for comparison.

Type	Objects side	Pieces	Num. of instances	Optimal (num. of objects)
A	1000	30	30	3
B	1000	30	30	10
C	1000	36	30	6
D	1000	60	30	3
E	1000	60	30	3
F	1000	30	30	2
G	1000	36	30	≤ 15
H	1000	36	30	12
I	1000	60	30	3
J	1000	60	30	4
K	1000	54	30	6
L	1000	30	30	3
M	1000	40	30	5
N	1000	60	30	2
O	1000	28	30	7
P	1000	56	30	8
Q	1000	60	30	15
R	1000	54	30	9
Total			540	

Figure 12: Description of the convex problem instances (López-Camacho et al., 2013).

The level of irregularity is further detailed in Figure 13a, illustrating how different problem types exhibit varying degrees of irregularity. Instances with a greater initial number of rectangles tend to feature more pieces with horizontal or vertical sides, potentially simplifying the packing process by increasing the likelihood of finding matches with object edges.

	Average piece area	Piece area standard deviation	Average rectan-gularity	Percentage of right angles	Percentage of vertical/horizontal sides
Minimum	0.033	0.014	0.35	11	34
Total average	0.154	0.100	0.68	42	65
Maximum	0.354	0.280	1	100	100
Average of instances per type					
A	0.100	0.069	0.70	42	68
B	0.333	0.162	0.87	67	84
C	0.167	0.124	0.68	36	63
D	0.050	0.036	0.57	23	51
E	0.050	0.035	0.41	12	38
F	0.067	0.050	0.59	29	57
G	0.332	0.156	0.87	67	83
H	0.333	0.158	0.86	67	83
I	0.053	0.017	1	100	100
J	0.067	0.034	0.83	68	83
K	0.154	0.150	0.63	34	60
L	0.100	0.075	0.51	23	50
M	0.125	0.102	0.55	28	55
N	0.033	0.024	0.62	32	60
O	0.250	0.223	0.57	27	58
P	0.143	0.173	0.49	18	43
Q	0.250	0.053	0.89	51	76
R	0.167	0.153	0.63	36	62

(a) Characteristics of the convex problem instances (López-Camacho et al., 2013).

Type	Minimum side	Maximum ratio (*largest side*)/(*minimum side*)	Initial rectangles per object
Type A	100	3	3
Type B	100	3	1
Type C	100	3	1
Type D	100	3	3
Type E	50	10	1
Type F	50	5	3
Type G	100	3	1
Type H	100	3	1
Type I	100	3	19
Type J	100	3	9
Type K	100	4	1
Type L	10	10	1
Type M	10	10	1
Type N	30	5	8
Type O	10	10	0
Type P	25	5	0
Type Q	200	2	1
Type R	80	4	1

(b) Irregularity in the generated convex problems (Terashima-Marín et al., 2010).

Figure 13: Irregular characteristics of the JP1 benchmark instances.

The JP1 instances serve as a comprehensive testbed for evaluating algorithms tackling irregular packing problems, offering a diverse range of challenges to researches in the field. However, these instances only consider convex polygons. An example of two solutions of two JP1 instances are presented in Figure 14.

C.2 JP2 benchmark test instances

López-Camacho et al. (2014) introduced the JP2 instances, comprising 480 new 2D instances featuring non-convex polygons. These instances were generated with a deliberate focus on diversity and complexity.

To construct these instances, 240 were created by splitting at least five pieces from each instance of types A, B, C, F, H, L, M, O of Figure 12 respectively. In this process, convex pieces were randomly selected and divided into two parts: one convex and one non-convex. The remaining 240

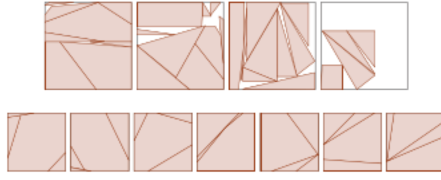


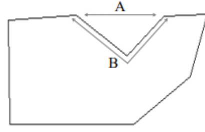
Figure 14: The solution of two JP1 instances of different types, where top solution is instance of class L and bottom solution is instance of class O (Martinez-Sykora et al., 2017).

non-convex instances were formed by initially creating convex instances and subsequently splitting some pieces into non-convex polygons. Types U, W, and X were produced by splitting pieces from instances already containing non-convex shapes.

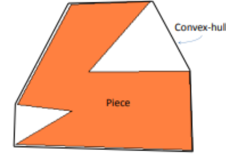
Non Convex 2D				
Type	Objects side	Num. of instances	Num. of pieces	Optimal (num. of objects)
NConv A	1000	30	35-50	3
NConv B	1000	30	40-52	10
NConv C	1000	30	42-60	6
NConv F	1000	30	35-45	2
NConv H	1000	30	42-60	12
NConv L	1000	30	35-45	3
NConv M	1000	30	45-58	5
NConv O	1000	30	33-43	7
NConv S	1000	30	17-20	2
NConv T	1000	30	30-40	10
NConv U	1000	30	20-33	5
NConv V	1000	30	15-18	5
NConv W	1000	30	24-28	4
NConv X	1000	30	25-39	3
NConv Y	1000	30	40-50	6
NConv Z	1000	30	60	12
Total	480			

Table 1: Description of the JP2 benchmark test instances (López-Camacho et al., 2014).

The variety in these instances encompasses a wide range of feature values. For instance, some piece have an average size of $\frac{1}{30}$ of the object while others are significantly larger, averaging almost $\frac{2}{3}$ of the object size. Moreover, the optimal number of bins or the best-known results for these instances span between 2 and 273. In terms of rectangularity (the ratio of the area of a piece to the area of the smallest enclosing rectangle), values vary between 0.35 and 1.
The characteristics of the JP2 instances, just like those described for the JP1 instances, are listed in Figure 16.
Following López-Camacho et al. (2012), the degree of concavity is defined by the concaveness of the largest internal angle and can be computed as $DC = \frac{B}{A}$, see Figure 15a. For 1D items and 2D convex polygons, including rectangles, the degree of concavity equals one. For concave polygons, the degree of concavity is greater than one. Following López-Camacho et al. (2012), the concavities are constructed using a triangle.
The convex hull of a set of points, $\mathcal{S}$, in the plane is the smallest convex polygon that contains all the points in $\mathcal{S}$. This can be visualized by imagining an elastic band stretched around the object. The convex hull of a polygon is defined as the convex hull of all its vertices. For a convex polygon, its convex hull is the polygon itself. However, for a non-convex polygon, the convex hull has a



(A) Degree of concavity (López Camacho et al., 2012).



(B) Convex hull of a non-convex piece (López Camacho et al., 2012).

FIGURE 15: Comparison of concavity degree and convex hull of non-convex piece.

greater area, as shown in Figure 15b. The ratio of the area of the piece to the area of its convex hull is thus less than one for non-convex polygons.

	Average piece size	Piece size standard deviation	Average rectan-gularity	Percentage of right angles	Percentage of orthogonal sides	Average of concavity degree	Average of ratio area / convex hull	Optimal (number of objects)
minimum	0.044	0.036	0.38	6	27	1.004	0.834	2
total average	0.160	0.135	0.59	26	50	1.13	0.930	5.9
maximum	0.333	0.314	0.84	60	74	1.56	0.987	12
Average of instances per type								
Nconv A	0.074	0.062	0.60	28	52	1.12	0.935	3
Nconv B	0.214	0.158	0.69	38	58	1.22	0.923	10
Nconv C	0.123	0.111	0.59	25	49	1.11	0.939	6
Nconv F	0.051	0.045	0.53	20	46	1.10	0.940	2
Nconv H	0.245	0.163	0.73	46	64	1.15	0.944	12
Nconv L	0.076	0.065	0.47	16	41	1.10	0.941	3
Nconv M	0.099	0.092	0.50	20	46	1.07	0.956	5
Nconv O	0.186	0.190	0.51	19	46	1.10	0.940	7
Nconv S	0.106	0.097	0.45	10	33	1.16	0.918	2
Nconv T	0.293	0.239	0.60	26	51	1.24	0.916	10
Nconv U	0.197	0.161	0.55	17	44	1.19	0.888	5
Nconv V	0.306	0.236	0.62	27	54	1.09	0.936	5
Nconv W	0.155	0.097	0.78	53	69	1.12	0.931	4
Nconv X	0.097	0.072	0.66	32	53	1.17	0.895	3
Nconv Y	0.135	0.129	0.61	25	51	1.09	0.943	6
Nconv Z	0.200	0.234	0.54	19	45	1.09	0.940	12

FIGURE 16: Characteristics of the JP2 instances.

The creation process for instances with non-convex pieces involves several steps. Initially, a set of convex pieces, each defined by integer coordinates of corners, is selected. Subsequently, a chosen convex piece is split into two parts, one convex and the other concave, through a series of steps involving edge selection and interior point determination.

(a) Choose two edges;

(b) For each of these two edges, either select an integer-valued interior point or, if none exists, choose one of the endpoints of the edge. This results in obtaining two points, Q and R, on the border of the piece;

(c) Select an integer-valued interior point, P, and connect Q to P to R, thereby splitting the piece into two, with one of them being concave.

Finally, the order of all pieces is randomized to ensure the two parts of a split piece are unlikely to be adjacent in the list of pieces. This process not only allows for the creation of various non-convex shapes, but also enables the generation of U-shaped polygons or shapes with internal empty spaces. However, shapes with holes are not produced by this algorithm.

For further information and access to the JP2 benchmark instance set, interested parties can refer to the provided link: https://www.euro-online.org/websites/esicup/data-sets/#1535972088237-bbcb74e3-b507.

C.3 Irregular strip packing benchmark test instances

The irregular strip packing benchmark instances, introduced in the study by Martinez-Sykora et al. (2017), offer a diverse array of shapes, including both convex and non-convex polygons, available on the ESICUP website.
Unlike the previous sets, these instances do not guarantee exact piece fitting, adding complexity to the problem.

In strip packing problems, as should be known to the reader by now, only the width of the stock sheet is constrained, with the objective of minimizing the total length required to pack all the pieces. To adapt these instances to the bin packing problem, a fixed stock sheet size (width and length) was defined, resulting in three sets of instances based on bin size:

- **Nest-SB (small bins)**: Bin dimensions are $W = L = 1.1m_d$.
- **Nest-MB (medium bins)**: Bin dimensions are $W = L = 1.5m_d$.
- **Nest-LB (large bins)**: Bin dimensions are $W = L = 2m_d$.

Here m_d represents the maximum length (or width) across all the pieces in their initial orientation for a given instance, and by defining the bins as above, they are all fixed-dimensional squares (in the same units as the pieces).

These test instances consist of 23 irregular strip packing instances, yielding 69 bin packing instances across the three bin sizes. The rationale for employing three different bin sizes is to investigate the relative importance of assignment and packing approaches. In Nest-SB instances, the assignment problem is expected to be more crucial due to the limited capacity of each bin. Conversely, in Nest-LB instances, efficient packing assumes greater significance, potentially leading to superior solutions even with sub-optimal assignments.

The instances, along with corresponding piece counts, bin sizes, and permitted rotations, are detailed in Figure 17.

	N	SB	MB	LB	Rot
albano	24	3337.40	4551.00	6068.00	0–180
shapes2	28	5.50	7.50	10.00	0–180
trousers	64	64.90	88.50	118.00	0–180
shapes0	43	15.40	21.00	28.00	0–90–180–270
shapes1	43	15.40	21.00	28.00	0–180
shirts	99	26.00	19.50	14.30	0–180
dighe2	10	77.00	105.00	140.00	0
dighe1	16	72.60	99.00	132.00	0
fu	12	15.40	21.00	28.00	0–90–180–270
han	23	25.30	34.50	46.00	0–90–180–270
jakobs1	25	8.80	12.00	16.00	0–90–180–270
jakobs2	25	17.60	24.00	32.00	0–90–180–270
mao	20	1206.70	1645.50	2194.00	0–180
poly1a	15	14.30	19.50	26.00	0–90–180–270
poly2a	30	14.30	19.50	26.00	0–90–180–270
poly3a	45	14.30	19.50	26.00	0–90–180–270
poly4a	60	14.30	19.50	26.00	0–90–180–270
poly5a	75	14.30	19.50	26.00	0–90–180–270
poly2b	30	14.30	19.50	26.00	0–90–180–270
poly3b	45	14.30	19.50	26.00	0–90–180–270
poly4b	60	14.30	19.50	26.00	0–90–180–270
poly5b	75	14.30	19.50	26.00	0–90–180–270
swimm	48	2133.61	2909.47	3879.29	0–180

FIGURE 17: Irregular strip packing instances (Martinez-Sykora et al., 2017).

D Pseudo-code of the equidistant method

The equidistant offset method aims to compute a new set of polygons that are uniformly offset from the original polygons by a specified distance $\frac{1}{2}d_1$, creating spacing of d_1 in the packing phase. An illustration of the offset method can be found in Figure 4.3.

To explain the method and the mathematics behind it, let's consider an example of calculating the offset vertex of vertex B. We start by identifying the previous vertex A, denoted by $p2$, the next vertex C, denoted by $p3$, and surrounding vertex B, denoted by $p1$. This helps us determine the vectors of the adjacent edges: $v1$ (from $B = p1$ to $A = p2$) and $v2$ (from $B = p1$ to $C = p3$). Next, we calculate the bisector vector of the angle between these adjacent edges and normalize it for further calculations. This bisector plays a crucial role in determining the direction of the offset vertex.
Based on the cross product of $v1$ and $v2$, we determine whether $v1$ lies in the counterclockwise direction of $v2$ (convex case) or in the clockwise direction of $v2$ (concave case). This distinction is essential because it affects how we must calculate the angle between the edges. This is where we differ from the method by Wang et al. (2022), since they did not separate the angle calculation for both cases, resulting in an algorithm that would only work well for convex cases.
For concave cases, we have to adjust the angle calculation by subtracting it from 360° to obtain the correct angle. This has to do with the interpretation by Python, which always takes the smallest angle as the angle between two vectors.
Using trigonometry, see Figure 18, specifically offset_length $= \frac{\text{offset_distance}}{\sin(\alpha/2)}$, where α is the computed angle, we calculate the length of the offset needed. In concave situations, we have to multiply this offset length by -1 to ensure the offset direction is correct relative to the polygon's curvature.

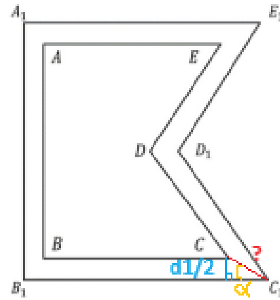


FIGURE 18: Trigonometry applied to the ABCDE piece.

Finally, we translate $B = p1$ along the normalized bisector by the calculated offset_length. This gives us the new coordinate points, which are appended as the offset vertex to help form the offset polygon.

Algorithm 7 Equidistant method

Input: polygons (list of polygons represented as lists of vertices), d1 (product spacing parameter)
Output: List of polygons with their product-spaced vertices
offset_distance = $0.5 * d1$
offset_polygons = [ ]
for each polygon in polygons **do**
  offset_polygon = [ ]
  for i from 0 to len(polygon) - 1 **do**
    p1 = polygon[i]
    p2 = polygon[(i-1) % len(polygon)]
    p3 = polygon[(i+1) % len(polygon)]
    $v1 = [p2[0] - p1[0], p2[1] - p1[1]]$
    $v2 = [p3[0] - p1[0], p3[1] - p1[1]]$
    unit_bisector = angular_bisector($v1, v2$)
    if cross product of v1 & v2 > 0 **then**
      $\alpha = \arccos(\frac{\text{dot product of v1 and v2}}{\text{norm(v1) * norm(v2)}})$
      length = $\frac{\text{offset_distance}}{sin(\alpha/2)}$
    else
      $\alpha = 2 * \pi - \arccos(\frac{\text{dot product of v1 and v2}}{\text{norm(v1) * norm(v2)}})$
      length = $-1 * \frac{\text{offset_distance}}{sin(\alpha/2)}$
    end if
    offset_translation = length * unit_bisector
    offset_point = [offset_translation[0] + p1[0], offset_translation[1] + p1[1]]
    Append offset_point to offset_polygon
  end for
  Append offset_polygon to offset_polygons
end for
Return offset_polygons

Algorithm 8 angular_bisector

Input: the vectors, v1 and v2
Output: normalized vector representing the angular bisector
Normalize vector v1
Normalize vector v2
bisector = v1_normalized + v2_normalized
Normalize the bisector
Return bisector_normalized

E Pseudo-code of the simplified equidistant method

The 'equidistant_offset_with_buffer' algorithm is designed as a quick fallback method for the 'equidistant_offset_method' to ensure product spacing can always be achieved even for larger required product spacing, as discussed in Section 6.6.

The function takes a list of polygons and a desired offset distance, d_1. It then creates the offset for each polygon using the Shapely library's buffer method. This method is particularly useful for handling various types of polygons, including concave polygons. The buffer method accepts a resolution parameter that controls the number of points used to approximate curved segments. Higher resolutions can result in more accurate offsets but may also increase computational cost. In our algorithm, we have chosen for a resolution parameter equal to 16.
The buffer method also accepts a 'join style' parameter, which affects how corners are treated when

creating the offset. It has three options:

- 'join_style = 1': Specifies round joins to round the corners of the offset polygon.
- 'join_style = 2': Specifies mitre joins to maintain sharp corners.
- 'join_style = 3': Specifies bevel joins to cut off the corner at a flat edge.

In our algorithm, we have chosen a join style equal to 2 to maintain the straight lines and handle sharp corners, as we did in the original equidistant method.

After the buffer has created a new polygon offset by the given distance, the algorithm converts the offset polygon to a list of coordinates using the helper function 'polygon_to_list_of_lists'. This is required to append the resulting vertices to the final coordinate list from which the program can read and plot the product-spaced polygon.

Algorithm 9 Equidistant Offset with Buffer

Input: *polygons* (List of polygons, each represented as a list of points),
  $d1$ (Desired offset distance)
Output: List of offset polygons, each represented as a list of points
offset_distance ← 0.5 × $d1$
offset_polygons ← []
for each poly in polygons **do**
  afpoly ← Polygon(*poly*)
  poffafpoly ← afpoly.buffer (offset_distance, 16, 2)
  poffafpoly ← polygon_to_list_of_lists(poffafpoly)
  Append poffafpoly to offset_polygons
end for
return offset_polygons

Algorithm 10 Polygon to List of Lists

Input: *polygon* (Shapely Polygon object)
Output: List of lists, where each inner list represents a coordinate $[x, y]$
coords ← polygon.exterior.coords[:-1]
coord_list ← []
for each coord in coords **do**
  Append list(*coord*) to coord_list
end for
return coord_list

F Pseudo-code of the randomisation and rotation for the assignment order

The 'randomize_list' function takes a list of polygons and a probability as inputs. The function then iterates through each polygon in the list. For each polygon, there is a certain probability that it will be swapped with another randomly selected polygon from the list. This introduces randomness into the order of the polygons.

Algorithm 11 Randomize_list

Input: polygons (list of polygons), probability (float)
Output: randomized_polygons (List of polygons)
Initialize randomized_polygons as a copy of the input polygons
Set n to the length of randomized_polygons
for each polygon, i, in randomized_polygons **do**
    if random() $<$ probability **then**
        Set j to a random integer between 0 and n-1
        Swap randomized_polygons[i] with randomized_polygons[j]
    end if
end for
Return randomized_polygons

The 'get_randomized_probability' function computes a probability values based on the current iteration number and the total number of iterations planned for the algorithm. This probability value is used to control the likelihood of randomizing the order of polygons to pack. The formula $1-e^{-5*\frac{\text{iteration}}{\text{total_iterations}}}$ generates a probability that increases as the iteration number progresses towards the total iterations, allowing for more randomization in later stages of the algorithm. Increasing the total number of iterations will allow for more exploitation around the initially sorted iteration. In contrast, fewer total iterations will allow for more exploration earlier in the iteration count. For example, taking 50 iterations as the total number of iterations will result in a probability value of 50% at iteration 7, while taking 500 iterations as the total number of iterations, will result in a 50% probability value at iteration 70.

Algorithm 12 get_randomized_probability

Input: iteration: current iteration number (starting from 1)
Input: total_iterations: total number of iterations planned for the algorithm
Output: randomized_probability: probability value between 0 and 1
Return $1 - e^{-5*\frac{\text{iteration}}{\text{total_iterations}}}$

The 'randomRotate' function performs random rotations on a list of polygons. Each polygon in the list is rotated by a random angle chosen from a specified set of allowed rotation angles or within a specified range, allowing free rotation.

Algorithm 13 randomRotate(poly_list, min_angle = None, allowed_angles = None)

```
Input: poly_list (list of polygons), min_angle (float, optional), allowed_angles (list of floats, optional)
Output: new_poly_list (list of rotated polygons)
new_poly_list = deep copy of poly_list
for index, poly_obj in enumerate(new_poly_list) do
    if allowed_angles is not None then
        rotation_angle = random choice from allowed_angles
        rotation_poly = RatotionPoly(rotation_angle)
        rotation_poly.rotation(poly_obj)
    else
        if min_angle is not None then
            rotation_angle = random integer between -min_angle and min_angle
        else
            rotation_angle = 0
        end if
        if rotation_angle is not 0 then
            rotation_poly = RatotionPoly(rotation_angle)
            rotation_poly.rotation(poly_obj)
        end if
    end if
end for
Return new_poly_list
```

The ‘RatotionPoly’ class is needed for the ‘randomRotate’ function and is responsible for handling the rotation of polygons by specified angles. It includes methods for initializing the class with an angle and rotating a polygon by the angle.

Algorithm 14 RatotionPoly Class and rotation Method

```
Class RatotionPoly
  Method __init__(self, angle)
    Input: self, angle
    self.angle = angle
    self._max = ⌊360 / angle⌋
  End Method
  Method rotation(self, poly)
    Input: self, poly
    If self._max > 1 then
      rotation_res = random integer between 1 and (self._max - 1)
      Poly = create Polygon object from poly
      new_Poly = rotate Poly by (rotation_res × self.angle)
      mapping_res = convert new_Poly to dictionary representation
      new_poly = mapping_res["coordinates"][0]
      For index = 0 to len(poly) - 1 do
        poly[index] = [new_poly[index][0], new_poly[index][1]]
      End For
    End If
  End Method
End Class
```

G Pseudo-code of the utilisation efficiency metric

The ‘calculate_objective_F’ algorithm is designed to evaluate the utilization efficiency of space within a set of bins, where each bin can contain multiple items/polygons. The objective is to

compute the F-value, which reflects the distribution of utilization rates across these bins.

Initially, all necessary parameters are set up, including the assignment of items to each bin, the total number of bins used, and the bin dimensions. The algorithm then proceeds to loop through each bin, calculating the total area of the items packed within it.
After summing the areas of all items in the current bin, the utilization rate is computed as the ratio of the item area to the product of the bin's height and width, using formula 4.9.
By performing this calculation for each bin, the algorithm computes the F-value as the average of the squared utilization rates, in accordance with formula 4.10.

Algorithm 15 calculate_objective_F

Input: bins (list of list of polygons), bin dimensions
Output: objective F (float), utilization rates of every bin
$L \leftarrow bin_height$
$W \leftarrow bin_width$
$N \leftarrow$ number of bins
$utilization_rates \leftarrow []$
for each bin in bins **do**
  Initialize used_area
  for each item in bin **do**
    Update used_area with area of the item
  end for
  U_bin = $\frac{\text{used_area}}{L*W}$
  Append U_bin to $utilization_rates$
end for
$F \leftarrow \sum(U_bin \in utilization_rates) \quad \frac{U_bin^2}{N}$
Return $F, utilization_rates$

H Pseudo-codes for the repacking strategy

The 'repack_least_utilized_bin' function attempts to find the best packing configuration resulting in maximum remnant sheet area, using permutations of the items and the BLF heuristic. It evaluates each permutation based on overlap and calculates P^* to optimize the packing efficiency within specified a specified number of iterations.

Algorithm 16 repack_least_utilized_bin

Input: bin_polygons (list of polygons), bin-dimensions, max_iterations (integer)
Output: best_configuration (list of packed polygons with corresponding coordinates)
Initialize items_to_pack as bin_polygons
Generate all permutations of items_to_pack
Initialize best_configuration as items_to_pack
Initialize best_P_star as ∞
Initialize iteration_count as 0
Ensure packed items retain their orientations
for each permutation in permutations **do**
    if iteration_count $\geq$ max_iterations **then**
        Print "Maximum number of iterations reached. Stopping."
        break
    end if
    Create a Bottom-Left Fill (BLF) object with the current permutation
    Check overlap using calculateOverlap function
    if overlap > 0 **then**
        Continue to next permutation
    end if
    Calculate P_star using calculate_P_star with BLF bins and utilization_rates
    if P_star $<$ best_P_star **then**
        Update best_configuration with adjusted positions of current permutation
        Update best_P_star to P_star
    end if
    Increment iteration_count by 1
end for
Return best_configuration

The 'calculate_P_star' function calculates the P^* metric, representing the percentage of the minimal bounding box area of items within the least utilized bin relative to the total area of the bin. It iterates through polygons to find the maximum x and y coordinates of all the placed polygons within the bin. These coordinates define a virtual bounding box that tightly encompasses all the items in the bin. The bounding box area is calculated as the product of these maximum x and y coordinates. P^* can then be calculated as the ratio of the bounding box area to the total bin area.

Algorithm 17 calculate_P_star

Input: bins (list of list of polygons), utilization_rates, bin-dimensions
Output: P_star (float)
Find the index of the least utilized bin based on utilization_rates
Retrieve the least utilized bin from bins
Initialize max_x and max_y as 0
for each polygon in the least utilized bin **do**
    for each vertex in the polygon **do**
        if vertex.x $>$ max_x **then**
            max_x $\leftarrow$ vertex.x
        end if
        if vertex.y $>$ max_y **then**
            max_y $\leftarrow$ vertex.y
        end if
    end for
end for
Calculate bounding_box_area as max_x $\times$ max_y
Calculate P_star as bounding_box_area / (bin_height $\times$ bin_width)
Return P_star

The 'calculateOverlap' function computes the total overlap score between polygons and between a polygon and the sheet's boundaries. It iterates through pairs of polygons and between each polygon and the sheet's boundaries to calculate overlap using the functions 'getDepth' and 'getDepthBin' respectively.

Algorithm 18 calculateOverlap

Input: polygons, reference points of polygons (list of positions), orientations, bin-dimensions
Output: total_overlap (float)
Initialize total_overlap as 0.0
Set num_polygons as the number of polygons
for i from 0 to num_polygons - 1 **do**
  for j from i + 1 to num_polygons - 1 **do**
    Calculate pairwise overlap between polygons as getDepth2(polygons[i], polygons[j], orientations[i], orientations[j], positions, i, j)
    Increment total_overlap by (depth)$**2$
  end for
end for
for i from 0 to num_polygons - 1 **do**
  Calculate overlap between sheet's boundaries for every polygon as getDepthBin2(polygons[i], polygons[i], orientations[i], orientations[i], bin_width, bin_height, positions, i, i)
  Increment total_overlap by (depth)$**2$
end for
Return total_overlap

The 'getDepth2' function calls the 'getDepth' function to calculate the penetration depth between two polygons after the items have been packed in the accompanying sheet. To make sure the input polygon coordinates align with the packing position, the polygons are slid to the right location using the 'adjust_polygon' function, where after the polygons are converted to a list of lists of coordinates to get the right formatting for the getDepth function. This second getDepth function calculates the actual penetration depth between two polygons, by calculating the distance from the reference point of the sliding polygon, P_b, to the $NFP_{P_a P_b}$. If d_1, the distance from the reference point to the NFP is 0, it means the reference point lies inside the NFP and thus the distance from the reference point to the nearest boundary of the NFP can be calculated, resulting in the required penetration depth.

Algorithm 19 getDepth2

Input: poly1, poly2, ori1 (orientation), ori2 (orientation), positions (list of reference points output from BFL), idx1 (integer), idx2 (integer)
Output: depth
Adjust poly1 position using positions[idx1]
Adjust poly2 position using positions[idx2]
Convert poly1 to a list of lists of coordinates
Convert poly2 to a list of lists of coordinates
Calculate depth using NFP(poly1, poly2).getDepth()
Return depth

Algorithm 20 getDepth

Input: NFP, original_top sliding polygon
Output: penetration depth between two polygons
Calculate distance $d1$ from original_top to NFP
if $d1 == 0$ **then**
  Calculate $d2$ as Polygon(NFP).boundary.distance(Point(original_top))
  Return $d2$
else
  Return 0
end if

The 'getDepthBin2' function calls the 'getDepthBin' function to calculate the penetration depth between a polygon and the bin's boundaries after the polygons have been packed in the accompanying sheet. It follows the same logic as the 'getDepth2' function, by first aligning the input polygon coordinates to the packing positions, and converting them to the right format to calculate

the penetration depth. The penetration depth can then be calculated by calling upon the 'getDepthBin' function. This function first checks if the polygon is completely contained in the bin, resulting in a penetration depth of 0. If the polygon is not completely contained in the bin, it calculates the distance from the reference point of the polygon to IFR_{bP_b}, resulting in the required penetration depth.

Algorithm 21 getDepthBin2

Input: poly1, poly2, ori1 (orientation), ori2 (orientation), bin-dimensions, positions (list of reference points output from BFL), idx1 (integer), idx2 (integer)
Output: depth
Adjust poly1 position using positions[idx1]
Adjust poly2 position using positions[idx2]
Convert poly1 to a list of lists of coordinates
Convert poly2 to a list of lists of coordinates
Create inner_fit_rectangle
Calculate depth using NFP(poly1, poly2, sheet_dimensions=(bin_width, bin_height)).getDepthBin(inner_fit_rectangle, bin_width, bin_height)
Return depth

Algorithm 22 getDepthBin

Input: inner_fit_rectangle, bin-dimensions, original_top (point)
Output: depth
Create a Polygon object from the bin as Polygon([(0, 0), (bin_width, 0), (bin_width, bin_length), (0, bin_length)])
Create reference point as Point(original_top)
Create piece_polygon as Polygon(self.sliding)
Convert inner_fit_rectangle to Polygon if it's a list
Calculate $d1_bin$ as bin_polygon.distance(reference point)
if bin_polygon.contains(piece_polygon) **then**
    Return 0
else
    if inner_fit_rectangle **then**
        Return inner_fit_rectangle.distance(reference point)
    else
        Return 0
    end if
end if

The 'find_bottom_left_corner' and 'adjust_polygon' functions are utility functions needed for 'getDepth2' and 'getDepthBin2' functions to slide the polygons to the right positions, aligning with the BLF packing layout.
The 'find_bottom_left_corner' algorithm identifies the bottom-left corner of a polygon. It is the point with the smallest x-coordinate. However, if there are multiple points with the same x-coordinate, the point with the smallest y-coordinate among them is chosen.
The 'adjust_polygon' algorithm adjusts the positions of a polygon based on the reference point location outputted from the BLF heuristic. The adjustment involves translating the polygon such that its bottom-left corner aligns with the reference point.

Algorithm 23 find_bottom_left_corner

Input: poly (list of points)
Output: bottom_left (tuple)
$min_x \leftarrow$ infinity
$min_y \leftarrow$ infinity
for each point in poly **do**
  if point[0] $<$ min_x **or** (point[0] $==$ min_x **and** point[1] $<$ min_y) **then**
    $min_x \leftarrow$ point[0]
    $min_y \leftarrow$ point[1]
  end if
end for
Return (min_x, min_y)

Algorithm 24 adjust_polygon

Input: poly (list of points), position (tuple of length 2)
Output: adjusted_polygon (Polygon object)
bottom_left $\leftarrow$ find_bottom_left_corner(poly)
bottom_left $\leftarrow$ tuple(bottom_left)
position $\leftarrow$ tuple(position)
if length of position $\neq 2$ **or** length of bottom_left $\neq 2$ **then**
  Raise ValueError('position and bottom_left must be tuples of length 2')
end if
x_offset $\leftarrow$ position[0] - bottom_left[0]
y_offset $\leftarrow$ position[1] - bottom_left[1]
Return affinity.translate(Polygon(poly), xoff=x_offset, yoff=y_offset)

I Pseudo-code of the K-value

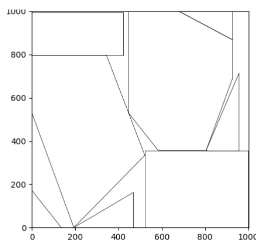
This appendix explains the calculation of the K-value, supplemented with the pseudo-code, which is part of the third level in the optimization phase of our algorithm.

The first step is to identify the bin with the lowest utilization rate from the list of bins. The index of the least utilized bin is found by locating the minimum value in the 'utilization_rates' list.
For the least utilized bin, the algorithm determines the minimal bounding box that can enclose all the placed items. This involves iterating through each point of each polygon within the bin and identifying the maximum x and y coordinates. These coordinates define the dimensions of the bounding box.
The area of the bounding box can then be calculated. With this information we can calculate P^* as the ratio of the bounding box area to the total bin area.
After determining the total number of bins used in the solution, the K-value can be calculated using the formula: $K = N - 1 + P^*$.

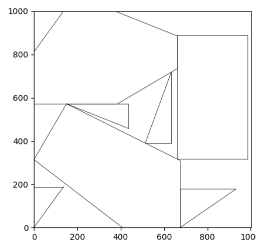
Algorithm 25 calculate_objective_K

```
Input: bins, utilization_rates, bin-dimensions
Output: K-value
Identify the least utilized bin
max_x ← 0
max_y ← 0
for each poly in least_utilized_bin do
    for each point in poly do
        if point[0]>max_x then
            max_x ← point[0]
        end if
        if point[1]>max_y then
            max_y ← point[1]
        end if
    end for
end for
bounding_box_area ← max_x * max_y
if bin_height * bin_width > 0 then
    P_star ← bounding_box_area / (bin_height * bin_width)
else
    P_star ← 0
end if
N ← number of bins
K ← N-1+P_star
Return K
```

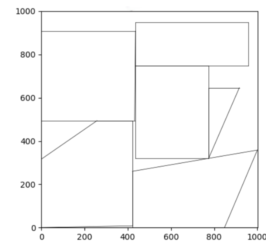
J Nesting results for the JP1 instances



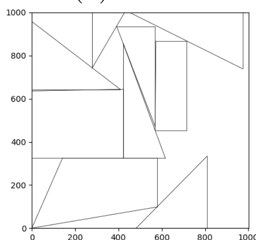
(A) Sheet 1



(B) Sheet 2

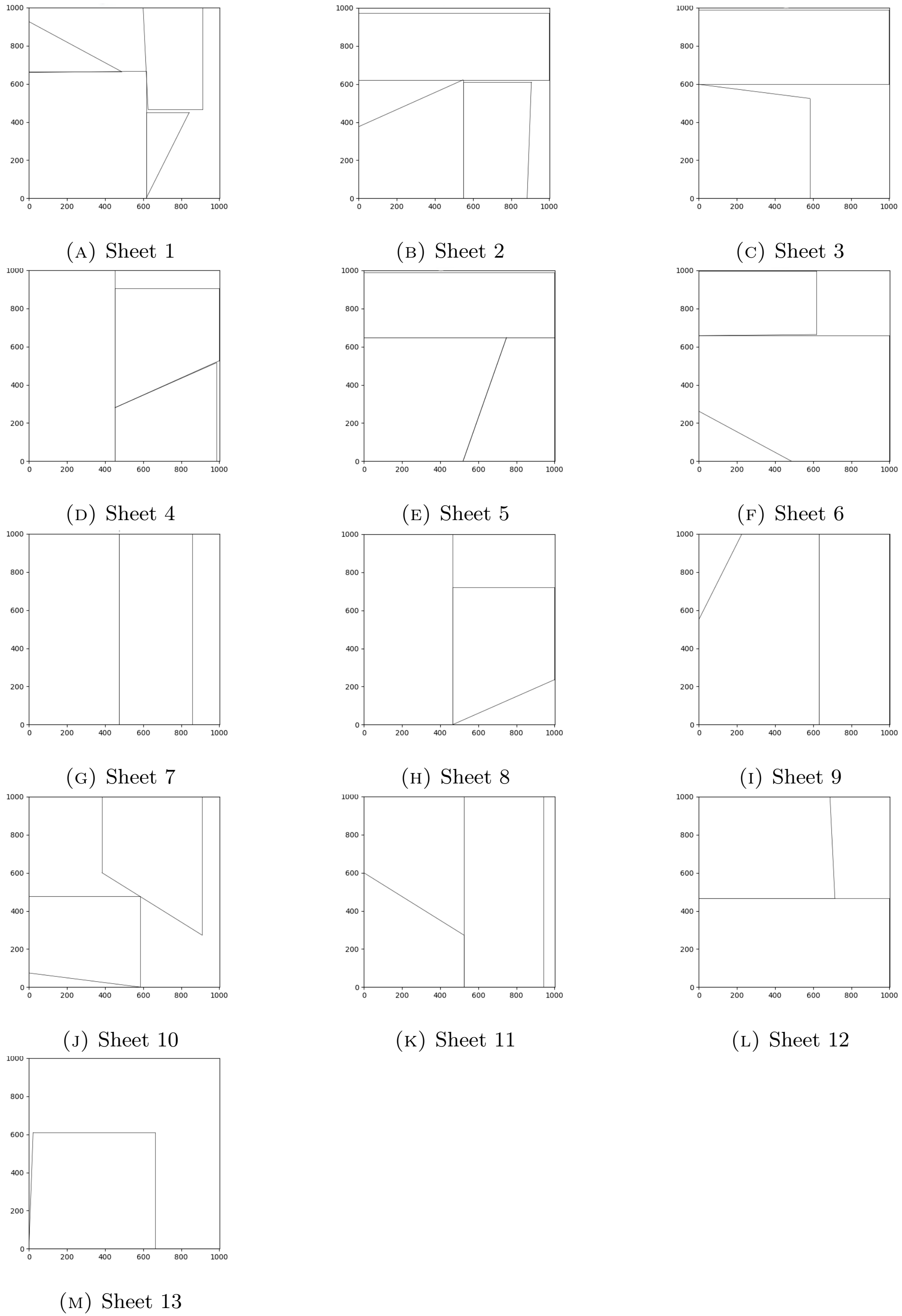


(C) Sheet 3



(D) Sheet 4

FIGURE 19: Nesting solution layout of type A of JP1.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

(J) Sheet 10 (K) Sheet 11 (L) Sheet 12

(M) Sheet 13

FIGURE 20: Nesting solution layout of type B of JP1.

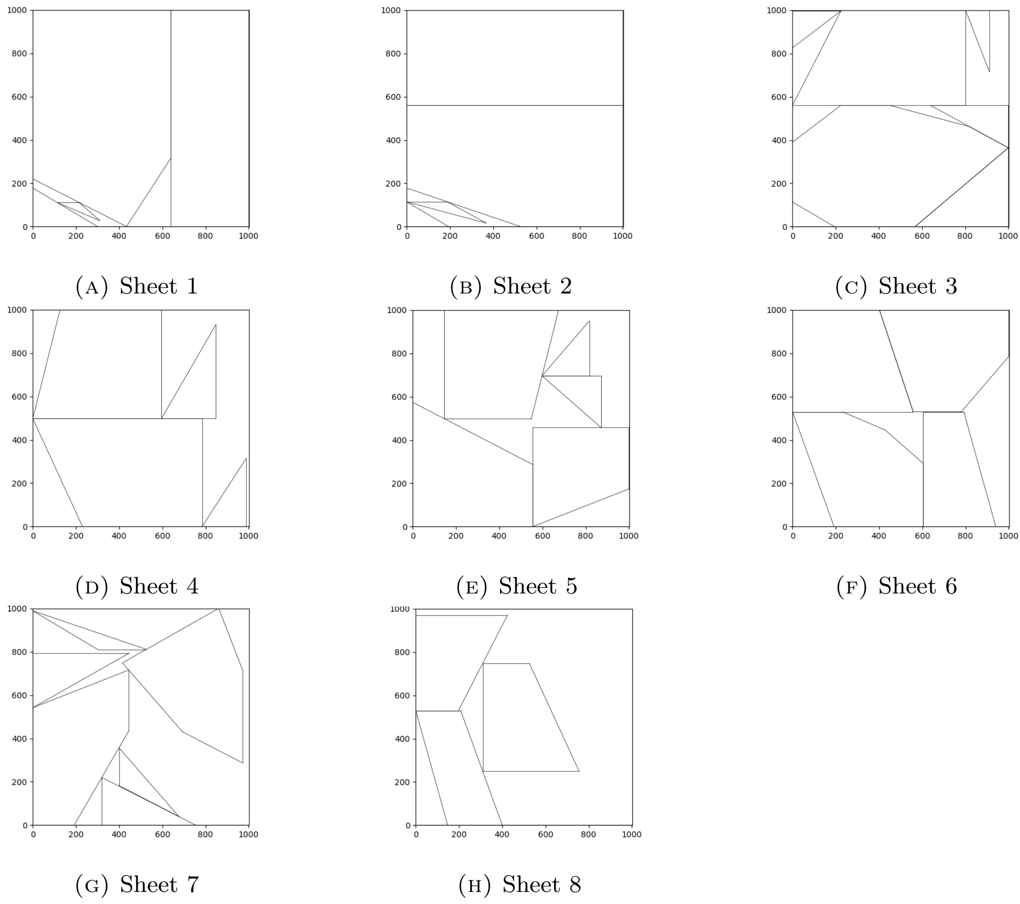


Figure 21: Nesting solution layout of type C of JP1.

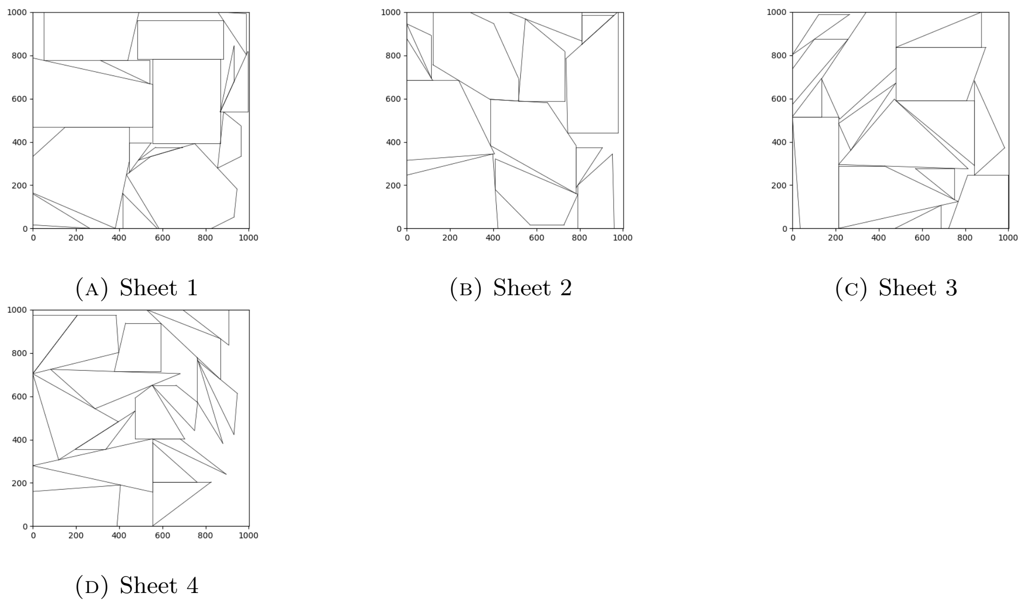
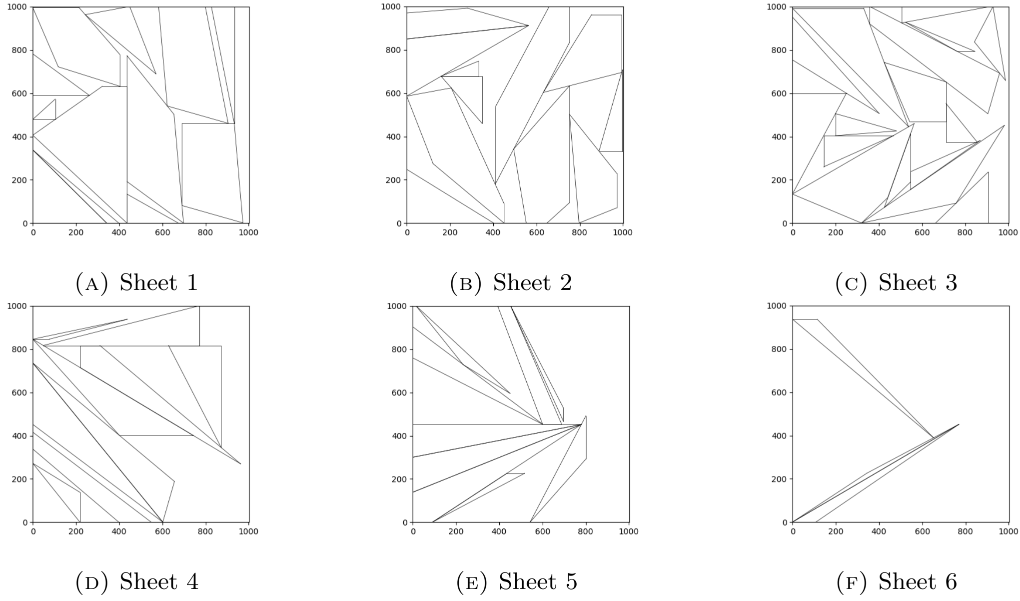


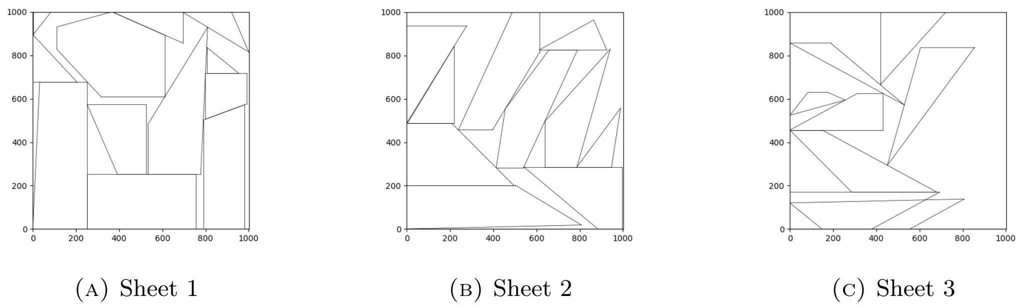
Figure 22: Nesting solution layout of type D of JP1.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

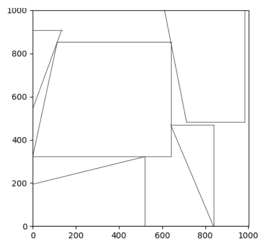
(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

FIGURE 23: Nesting solution layout of type E of JP1.

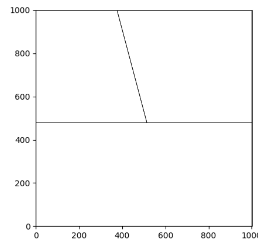


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

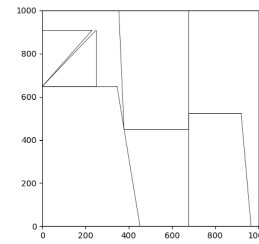
FIGURE 24: Nesting solution layout of type F of JP1.



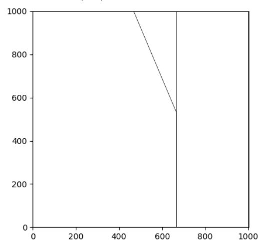
(A) Sheet 1



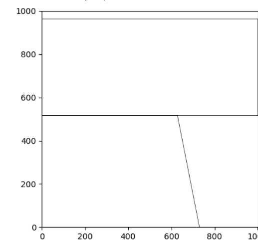
(B) Sheet 2



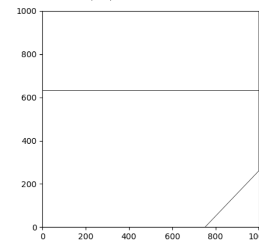
(C) Sheet 3



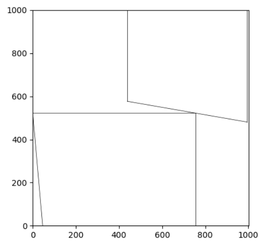
(D) Sheet 4



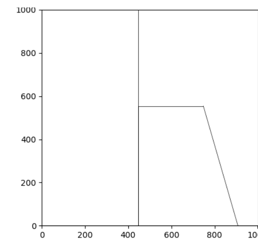
(E) Sheet 5



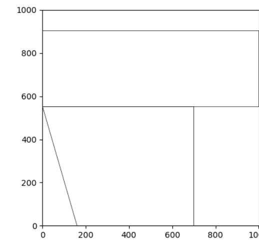
(F) Sheet 6



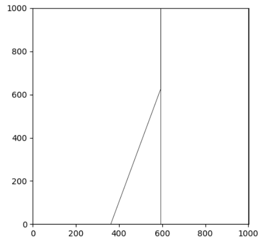
(G) Sheet 7



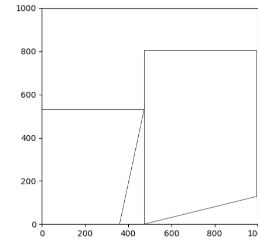
(H) Sheet 8



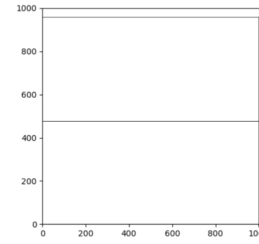
(I) Sheet 9



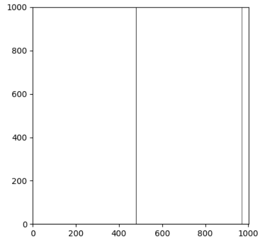
(J) Sheet 10



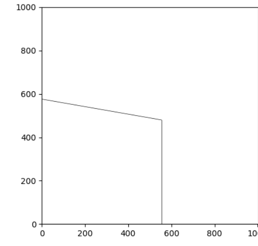
(K) Sheet 11



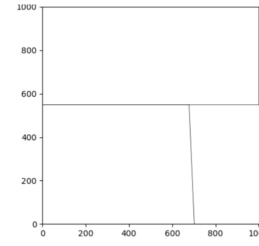
(L) Sheet 12



(M) Sheet 13

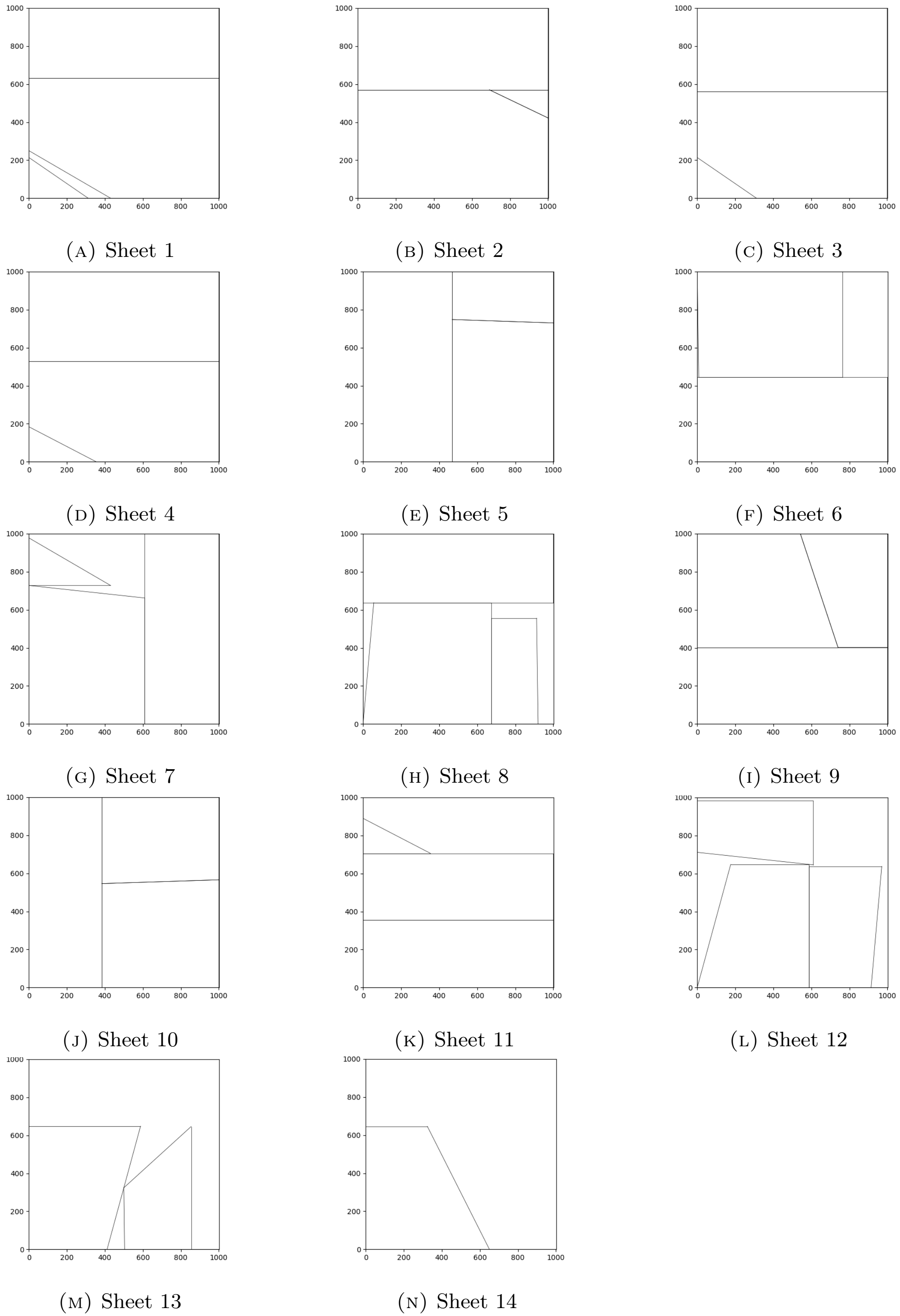


(N) Sheet 14



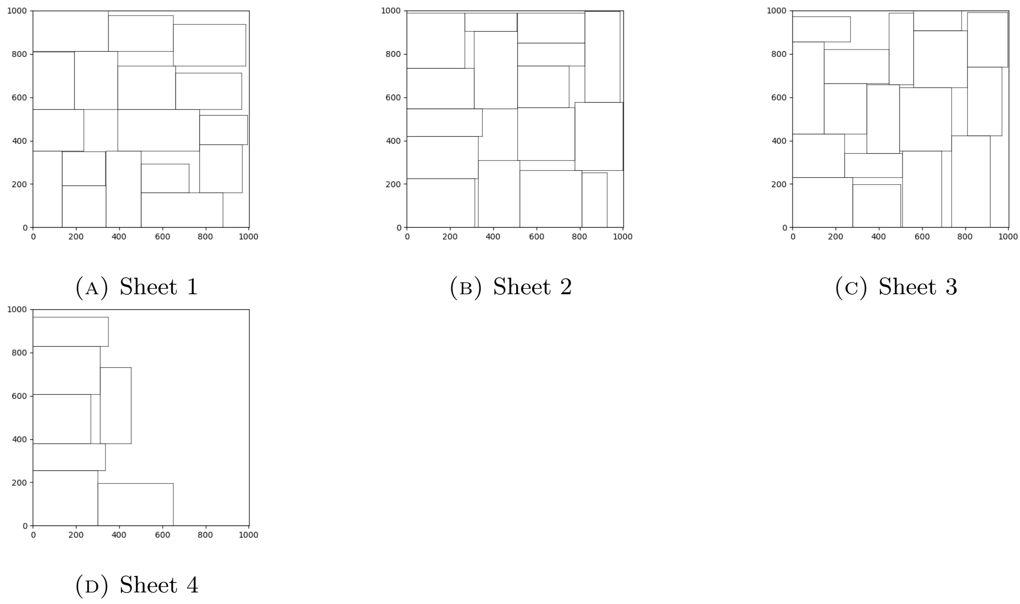
(O) Sheet 15

Figure 25: Nesting solution layout of type G of JP1.



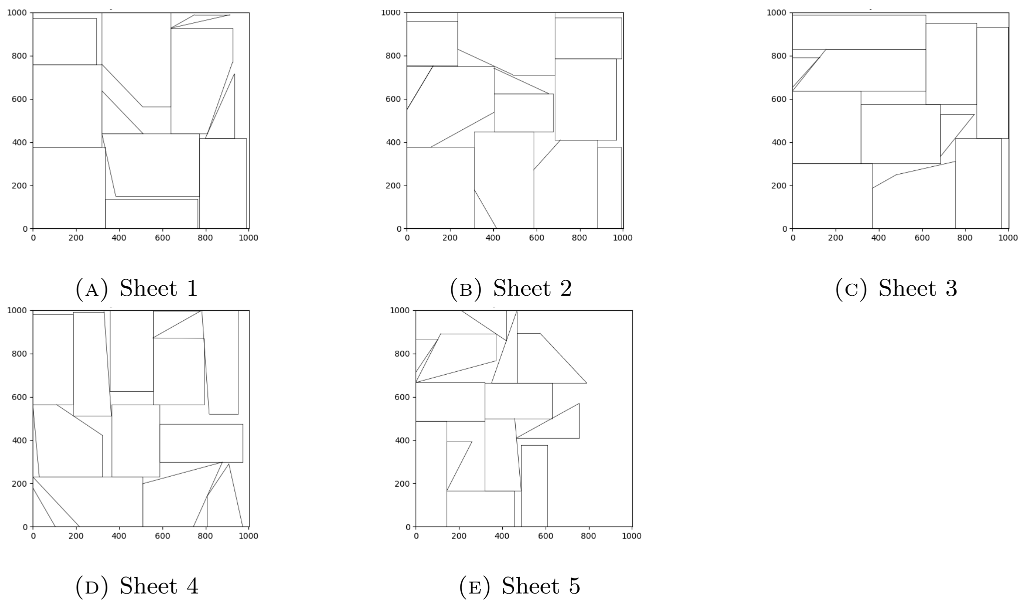
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4 (E) Sheet 5 (F) Sheet 6 (G) Sheet 7 (H) Sheet 8 (I) Sheet 9 (J) Sheet 10 (K) Sheet 11 (L) Sheet 12 (M) Sheet 13 (N) Sheet 14

Figure 26: Nesting solution layout of type H of JP1.



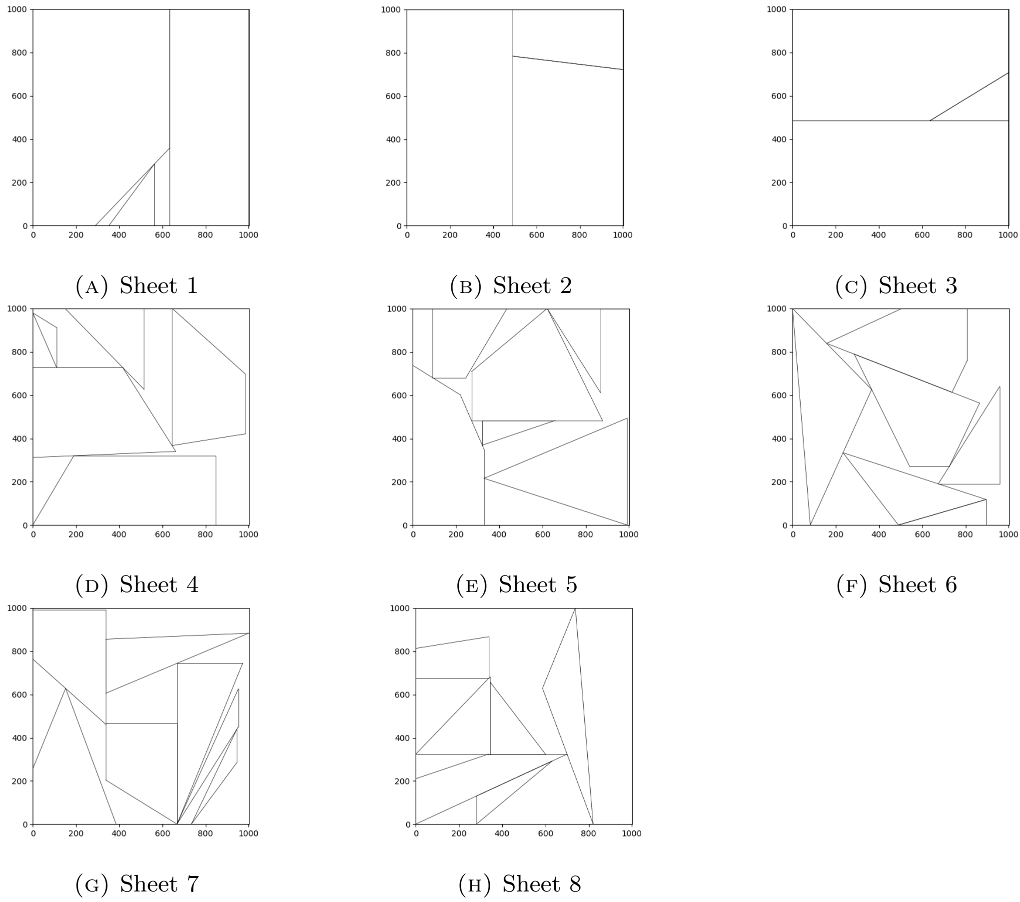
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4

Figure 27: Nesting solution layout of type I of JP1.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4 (E) Sheet 5

Figure 28: Nesting solution layout of type J of JP1.

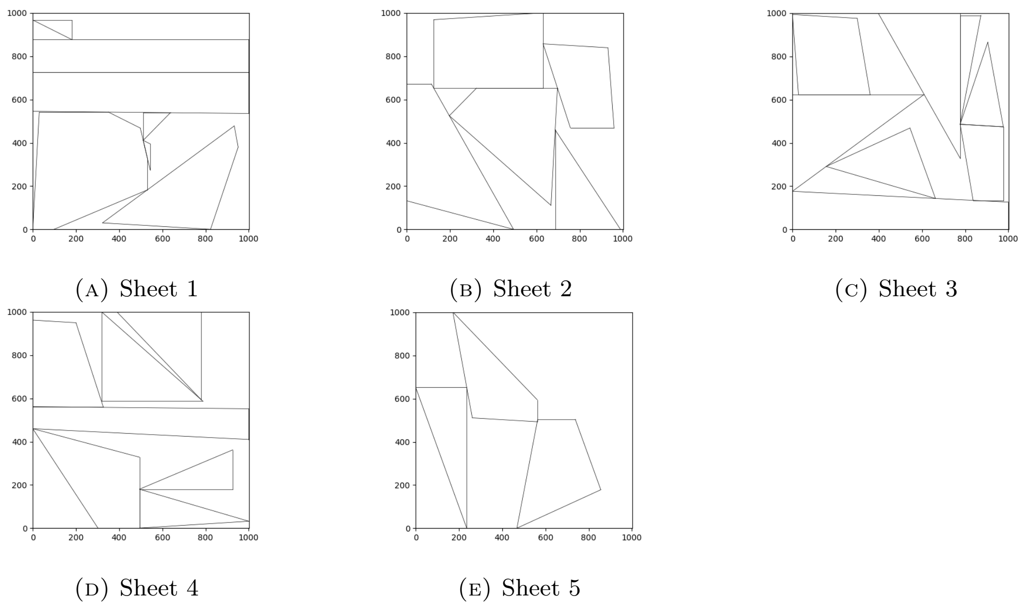


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8

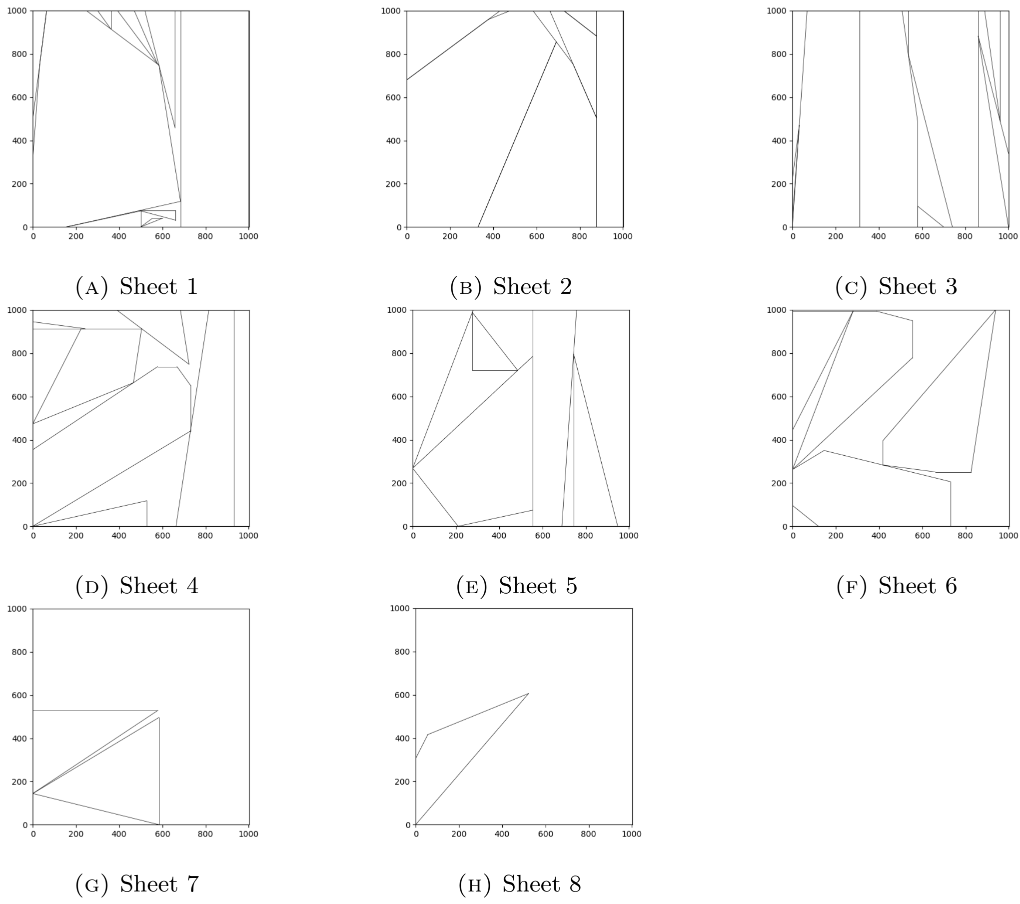
Figure 29: Nesting solution layout of type K of JP1.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5

Figure 30: Nesting solution layout of type L of JP1.

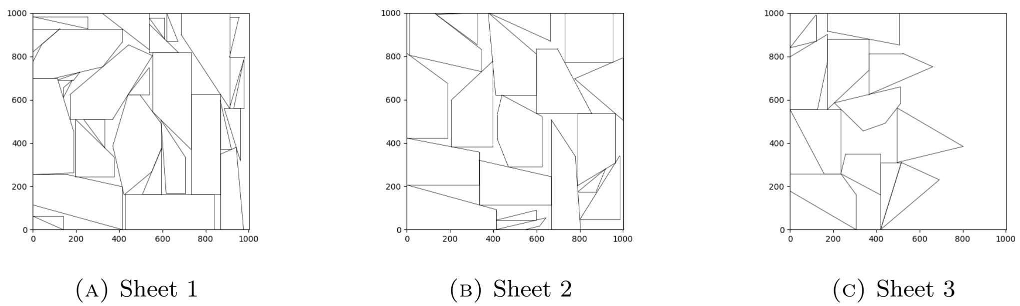


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

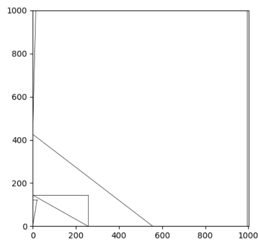
(G) Sheet 7 (H) Sheet 8

FIGURE 31: Nesting solution layout of type M of JP1.

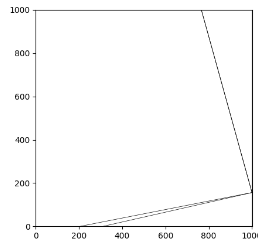


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

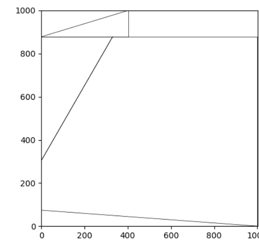
FIGURE 32: Nesting solution layout of type N of JP1.



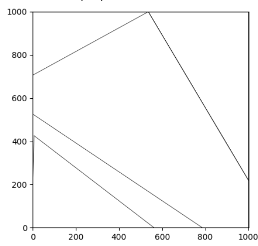
(A) Sheet 1



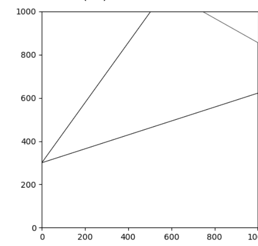
(B) Sheet 2



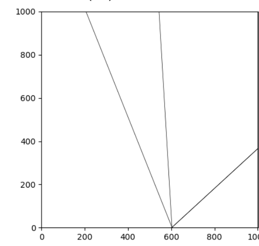
(C) Sheet 3



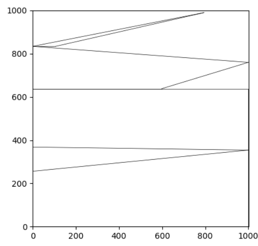
(D) Sheet 4



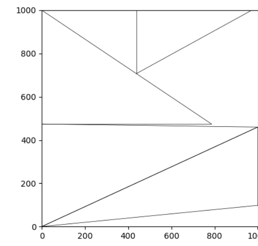
(E) Sheet 5



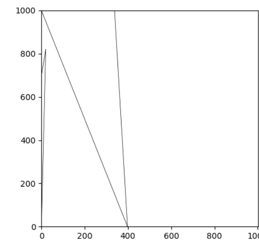
(F) Sheet 6



(G) Sheet 7

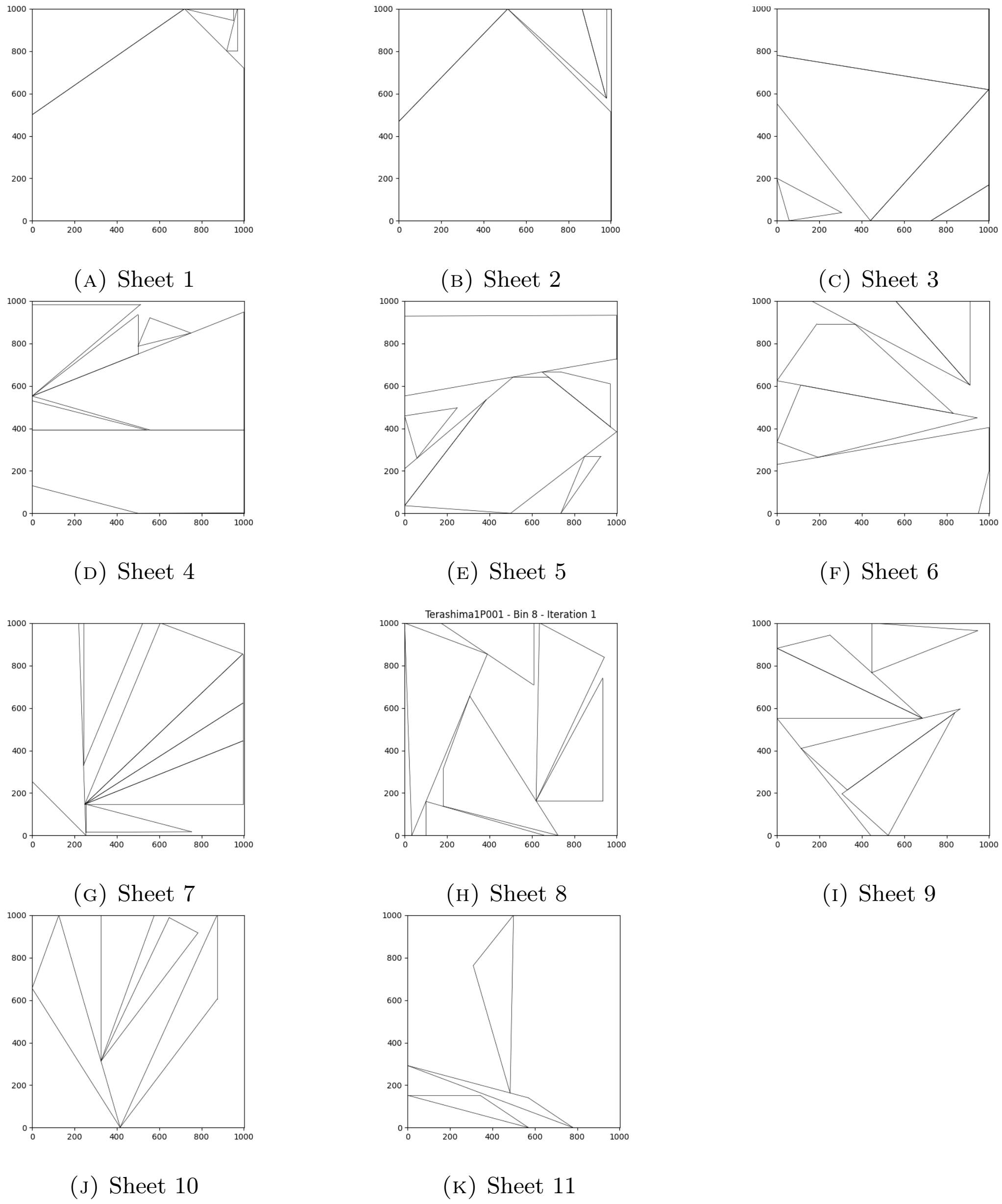


(H) Sheet 8



(I) Sheet 9

FIGURE 33: Nesting solution layout of type O of JP1.



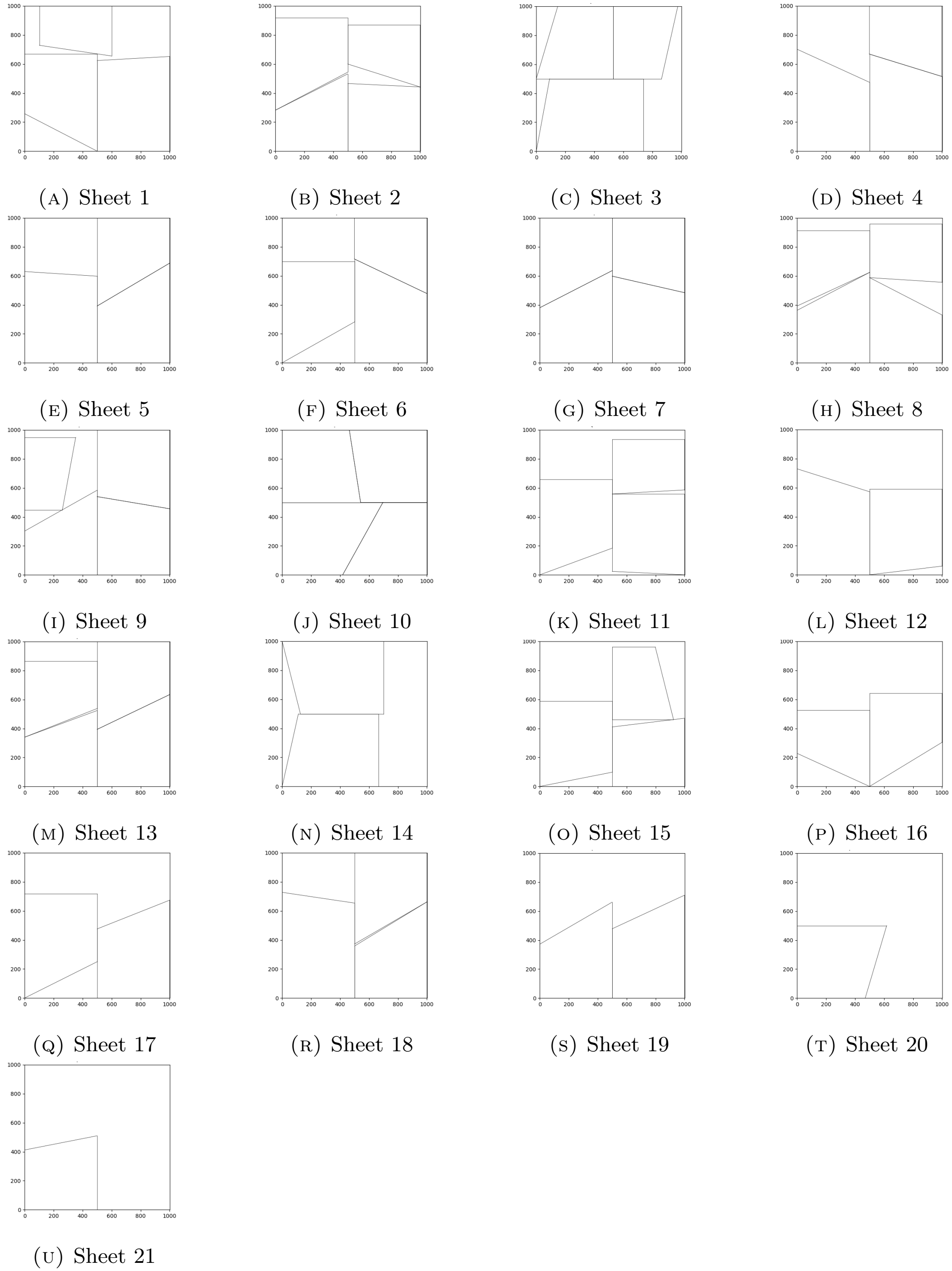
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

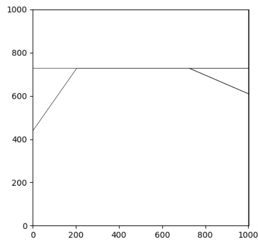
(J) Sheet 10 (K) Sheet 11

FIGURE 34: Nesting solution layout of type P of JP1.

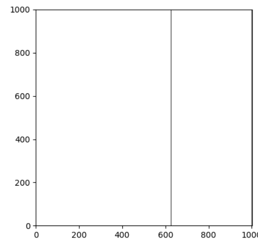


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4
(E) Sheet 5 (F) Sheet 6 (G) Sheet 7 (H) Sheet 8
(I) Sheet 9 (J) Sheet 10 (K) Sheet 11 (L) Sheet 12
(M) Sheet 13 (N) Sheet 14 (O) Sheet 15 (P) Sheet 16
(Q) Sheet 17 (R) Sheet 18 (S) Sheet 19 (T) Sheet 20
(U) Sheet 21

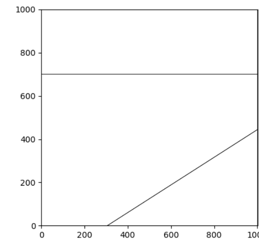
FIGURE 35: Nesting solution layout of type Q of JP1.



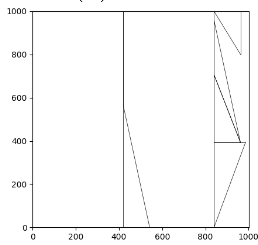
(A) Sheet 1



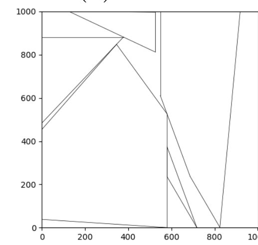
(B) Sheet 2



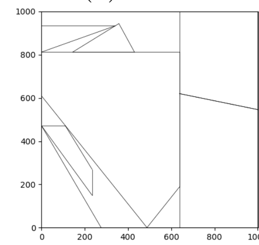
(C) Sheet 3



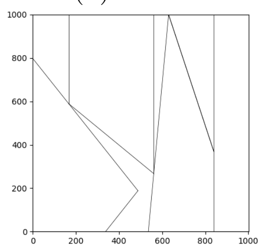
(D) Sheet 4



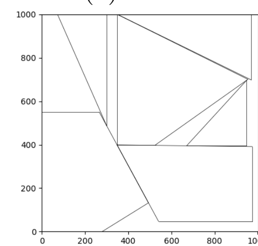
(E) Sheet 5



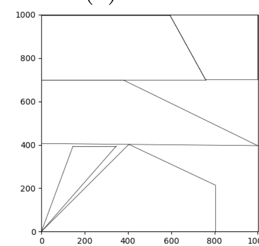
(F) Sheet 6



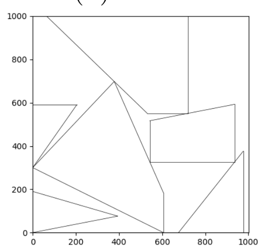
(G) Sheet 7



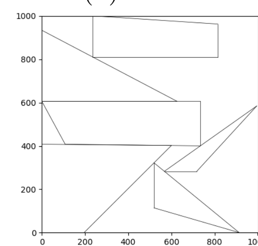
(H) Sheet 8



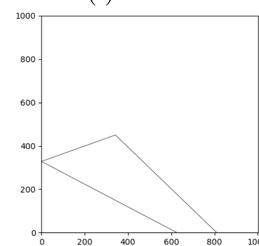
(I) Sheet 9



(J) Sheet 10



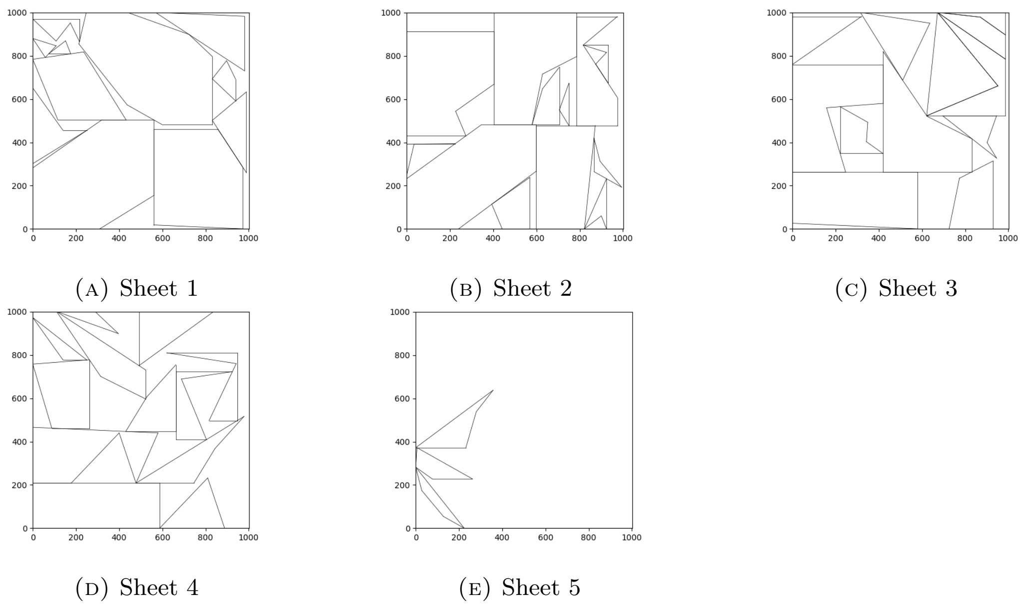
(K) Sheet 11



(L) Sheet 12

Figure 36: Nesting solution layout of type R of JP1.

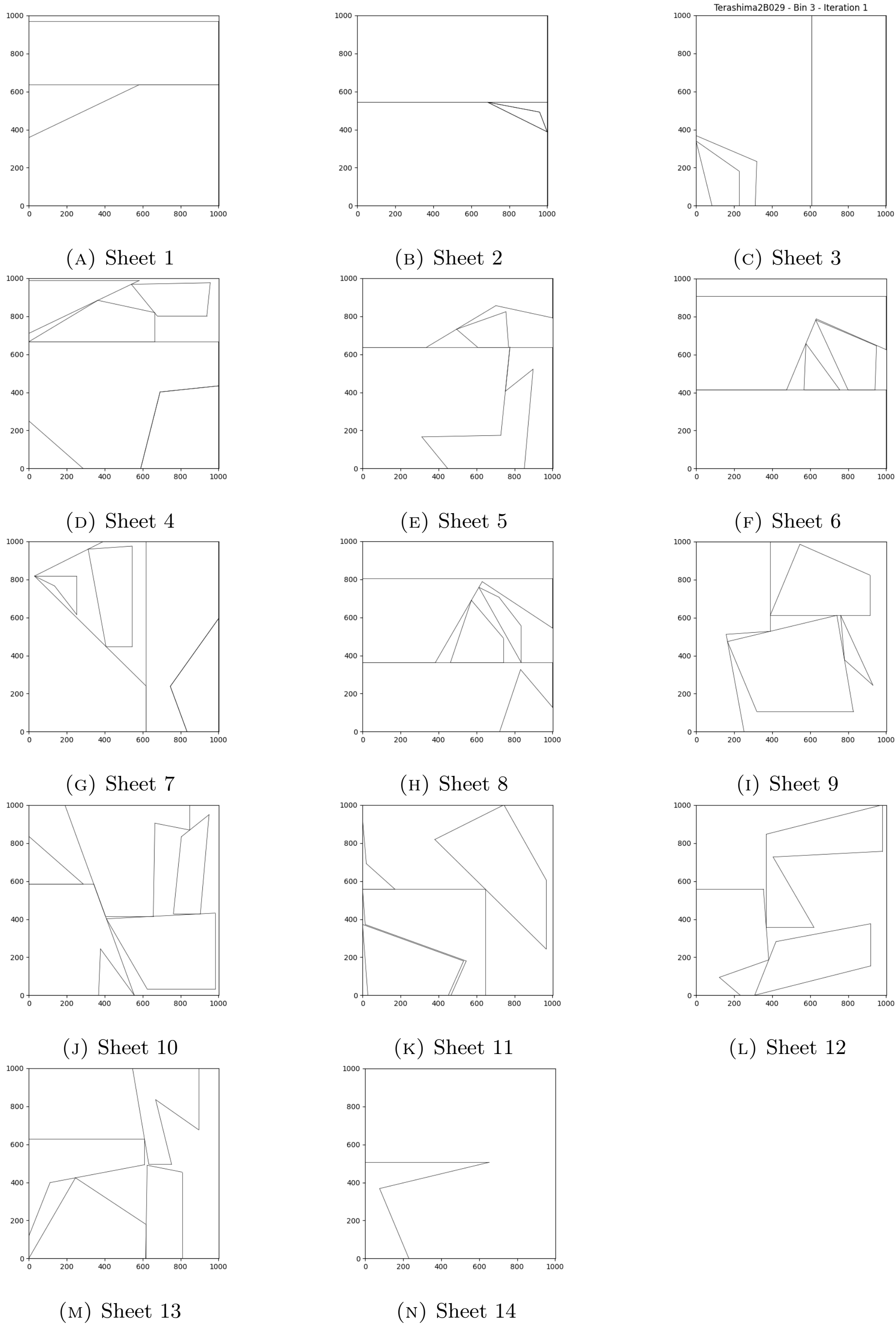
K Nesting results for the JP2 instances



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5

FIGURE 37: Nesting solution layout of type A of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

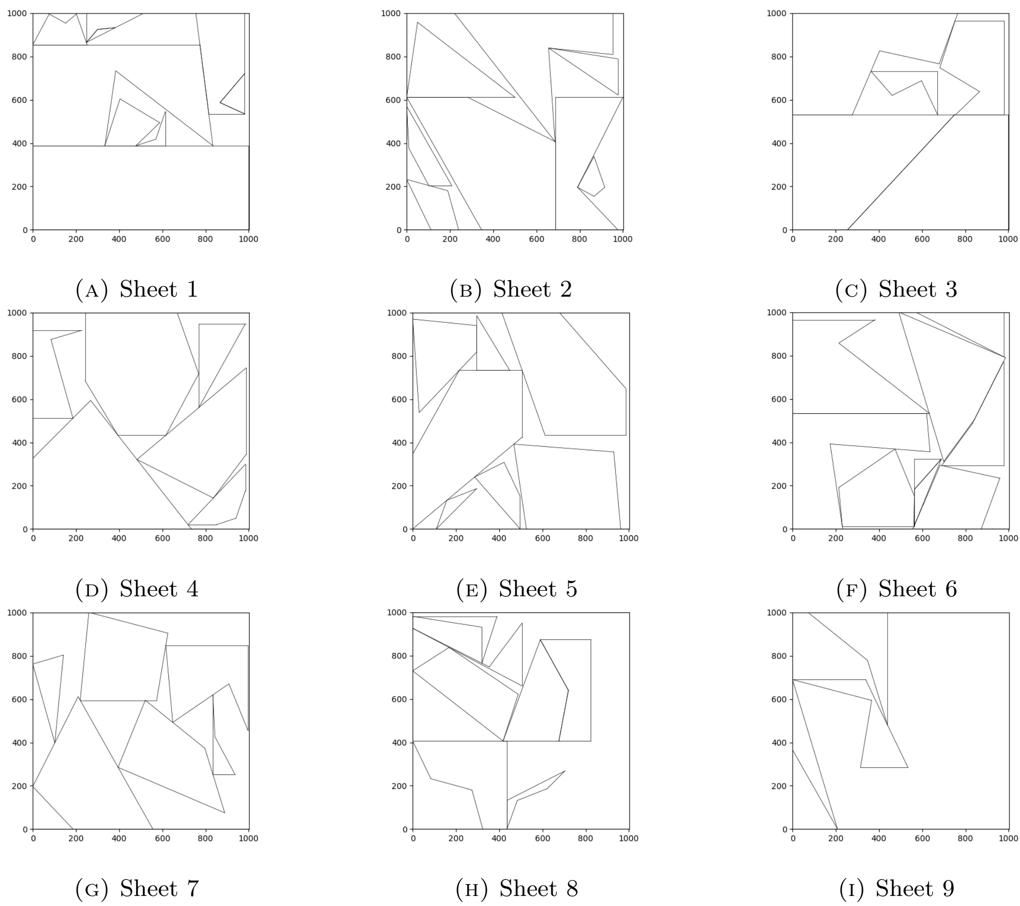
(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

(J) Sheet 10 (K) Sheet 11 (L) Sheet 12

(M) Sheet 13 (N) Sheet 14

FIGURE 38: Nesting solution layout of type B of JP2.

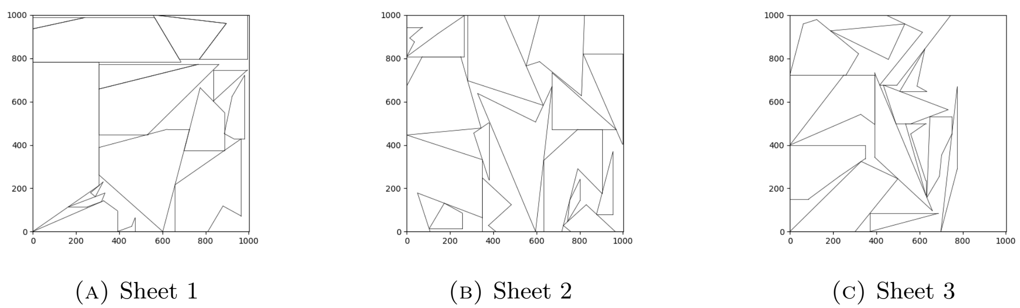


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

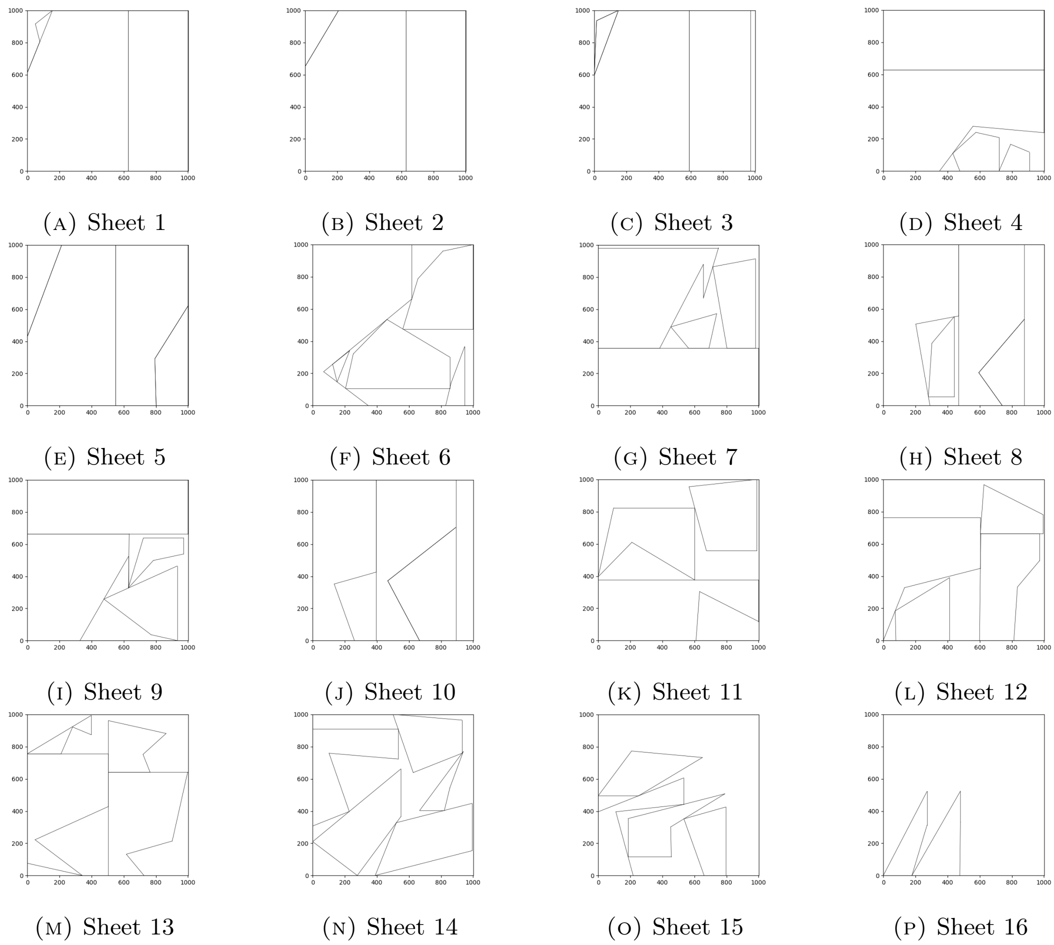
(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

FIGURE 39: Nesting solution layout of type C of JP2.



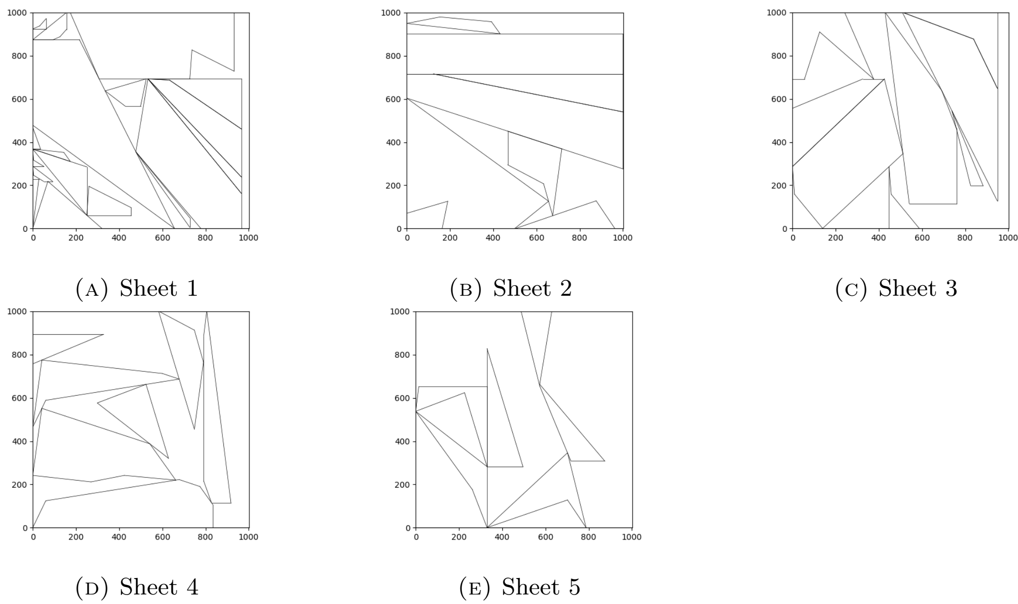
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

FIGURE 40: Nesting solution layout of type F of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4
(E) Sheet 5 (F) Sheet 6 (G) Sheet 7 (H) Sheet 8
(I) Sheet 9 (J) Sheet 10 (K) Sheet 11 (L) Sheet 12
(M) Sheet 13 (N) Sheet 14 (O) Sheet 15 (P) Sheet 16

FIGURE 41: Nesting solution layout of type H of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3
(D) Sheet 4 (E) Sheet 5

FIGURE 42: Nesting solution layout of type L of JP2.

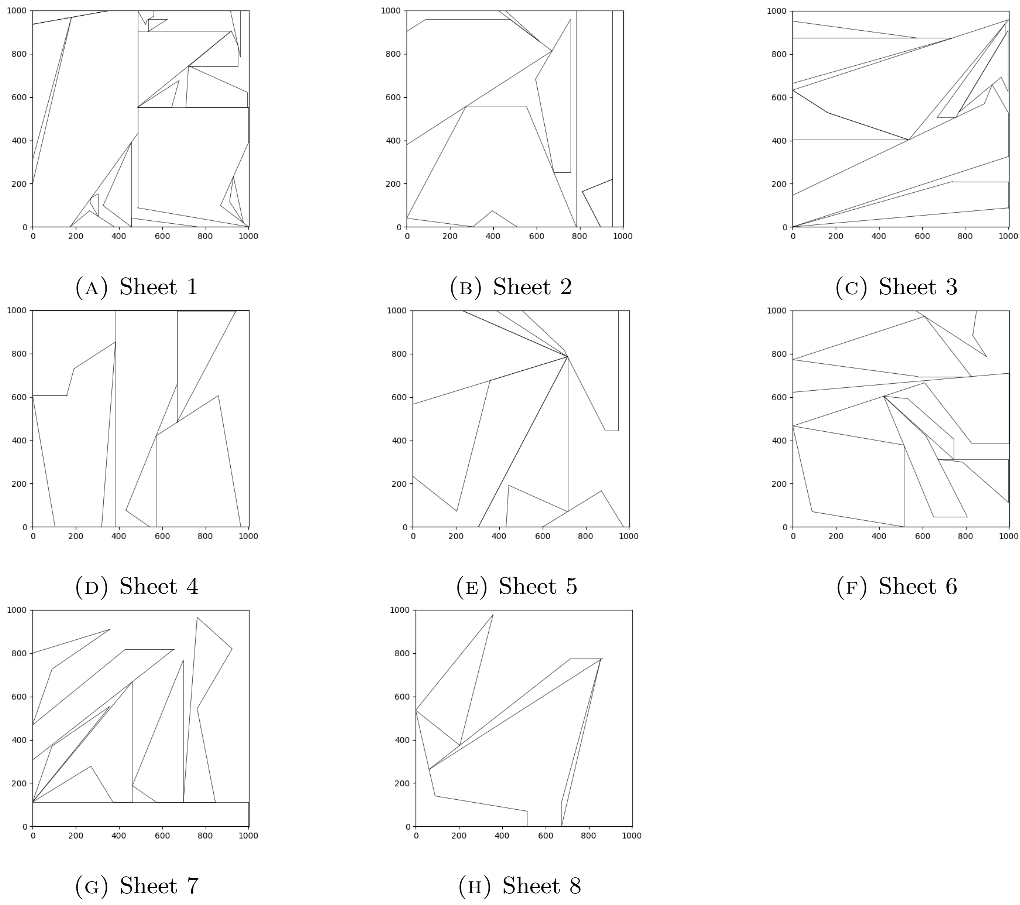
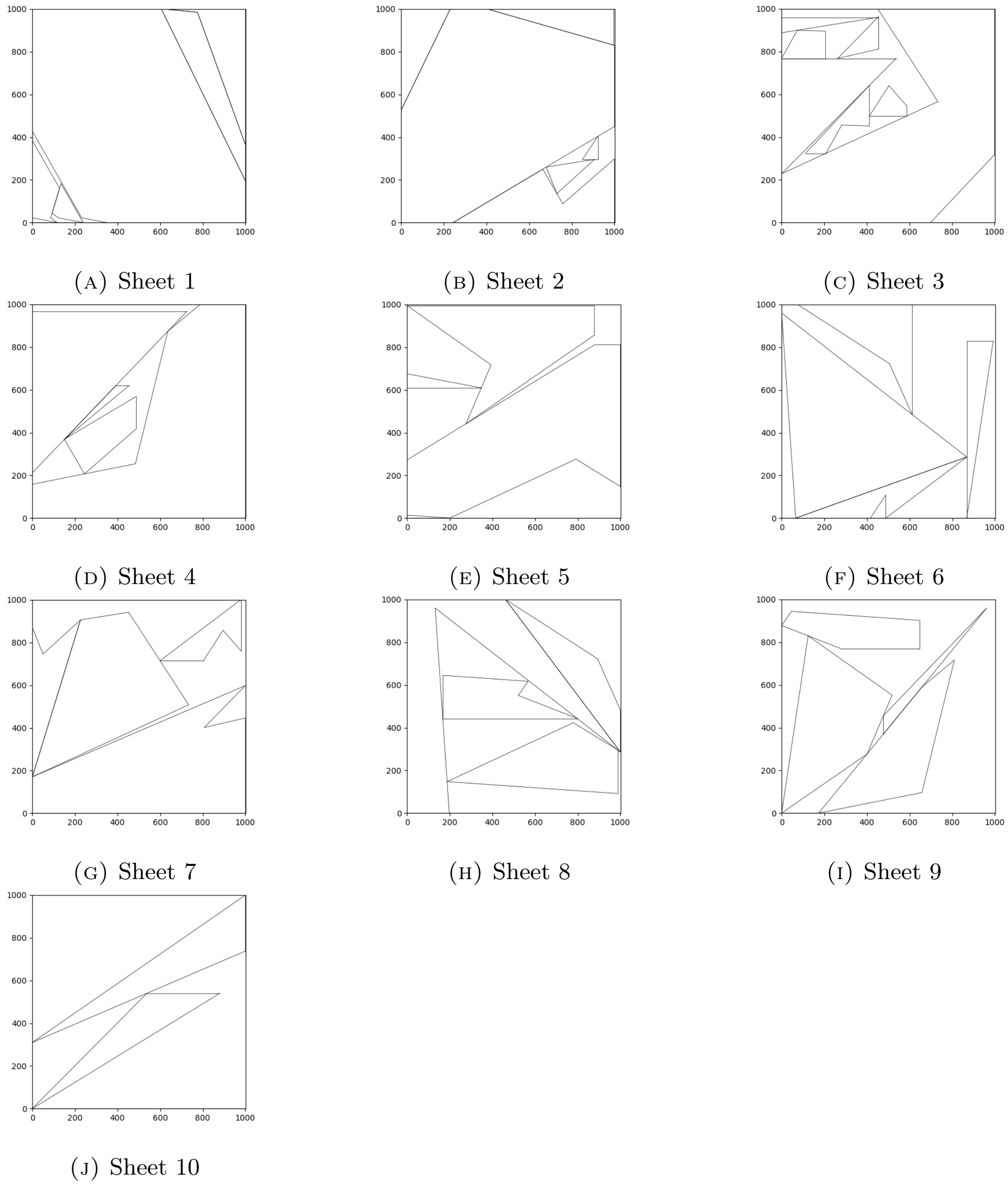

(A) Sheet 1
(B) Sheet 2
(C) Sheet 3
(D) Sheet 4
(E) Sheet 5
(F) Sheet 6
(G) Sheet 7
(H) Sheet 8

FIGURE 43: Nesting solution layout of type M of JP2.



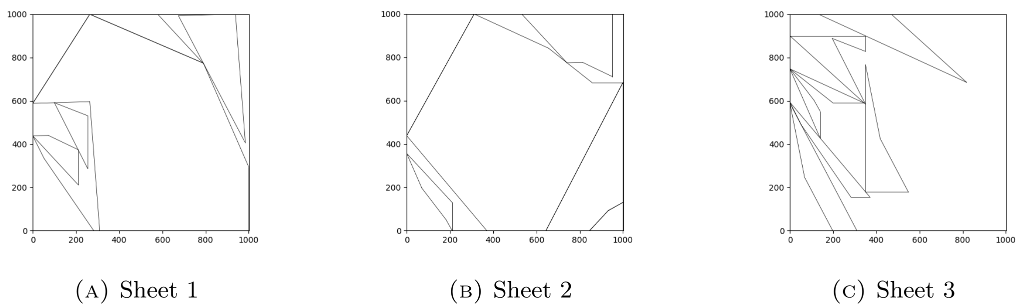
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

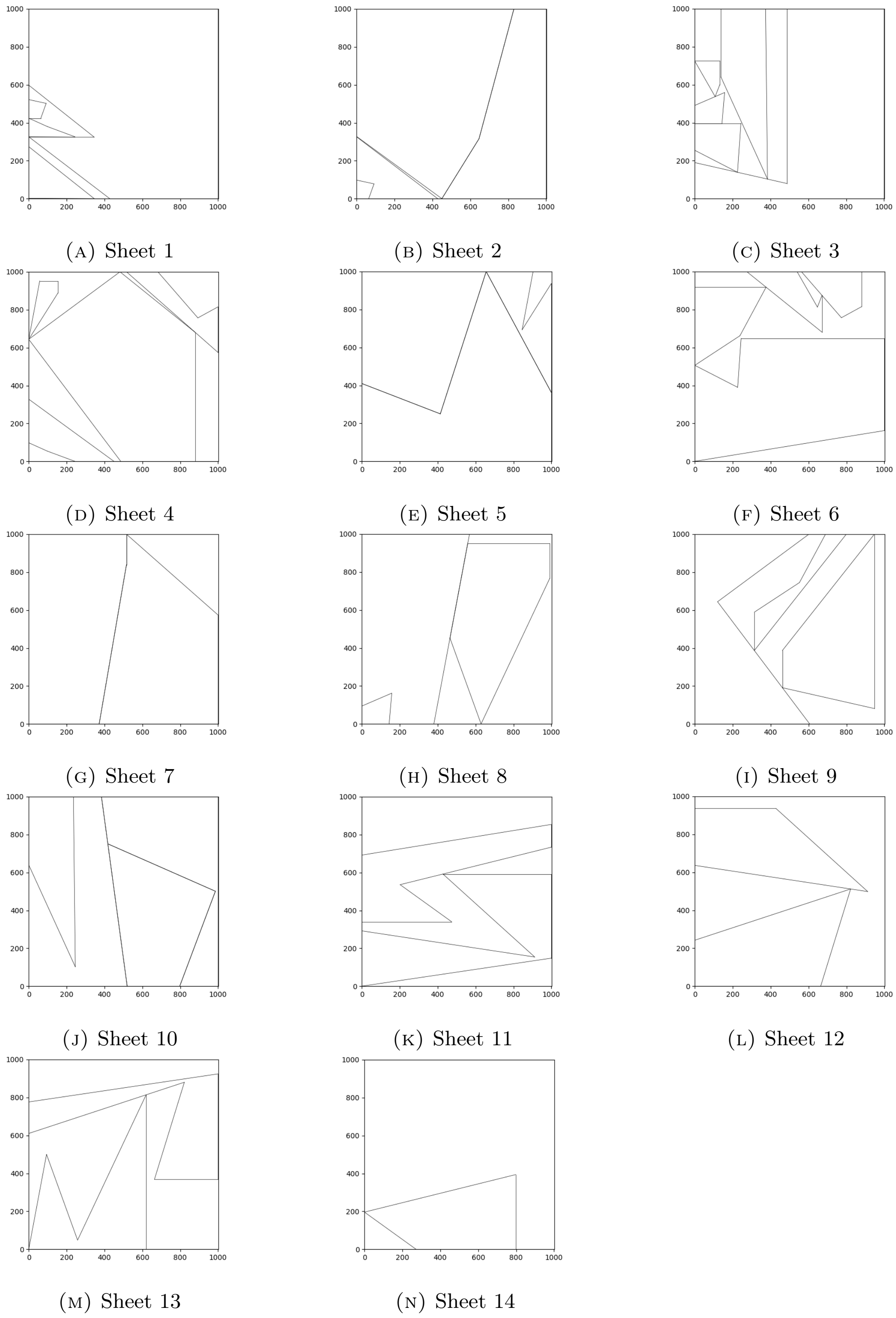
(J) Sheet 10

FIGURE 44: Nesting solution layout of type O of JP2.



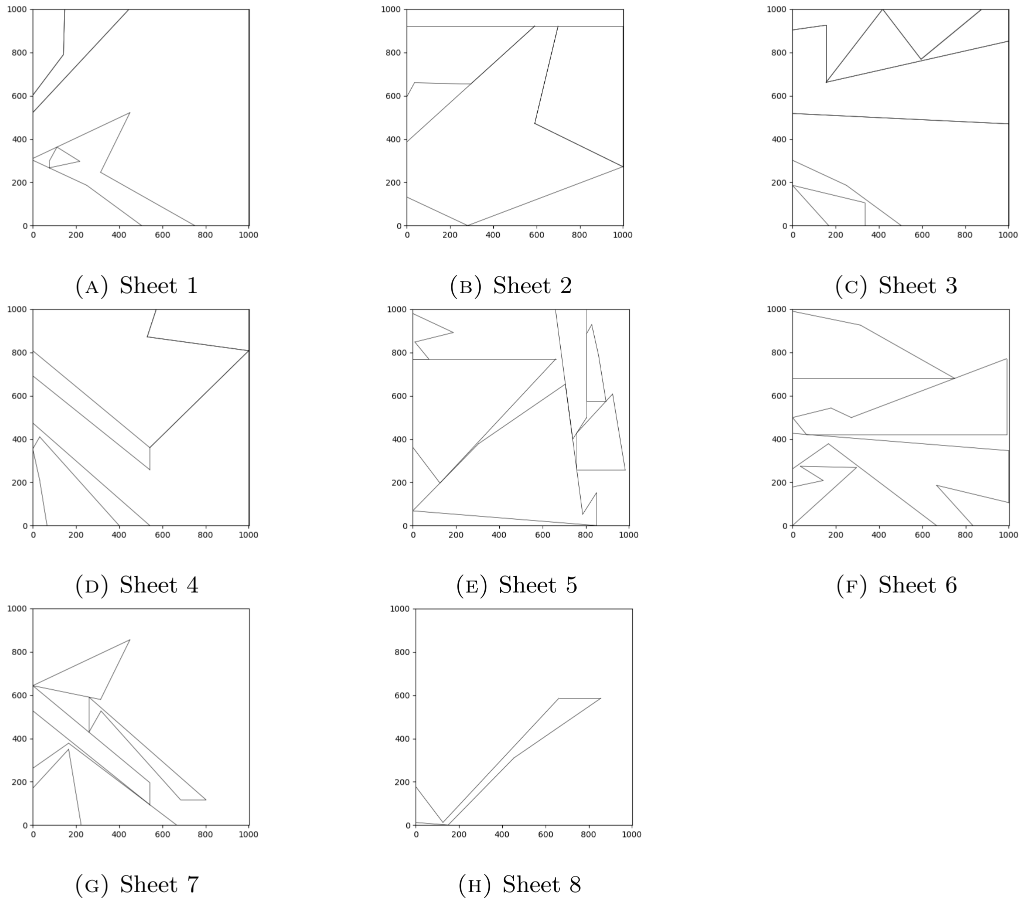
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

FIGURE 45: Nesting solution layout of type S of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3
(D) Sheet 4 (E) Sheet 5 (F) Sheet 6
(G) Sheet 7 (H) Sheet 8 (I) Sheet 9
(J) Sheet 10 (K) Sheet 11 (L) Sheet 12
(M) Sheet 13 (N) Sheet 14

FIGURE 46: Nesting solution layout of type T of JP2.

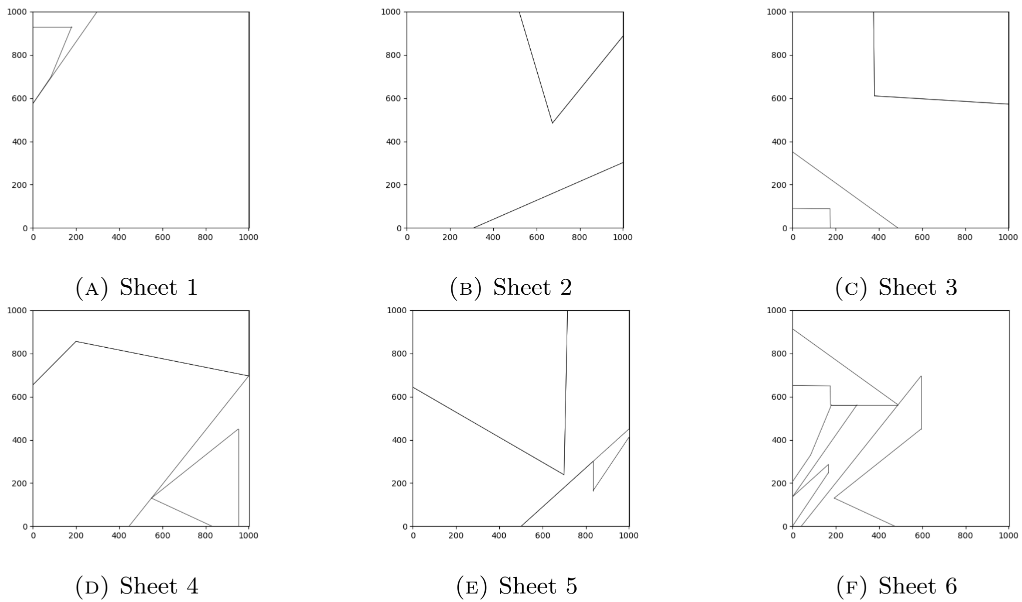


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8

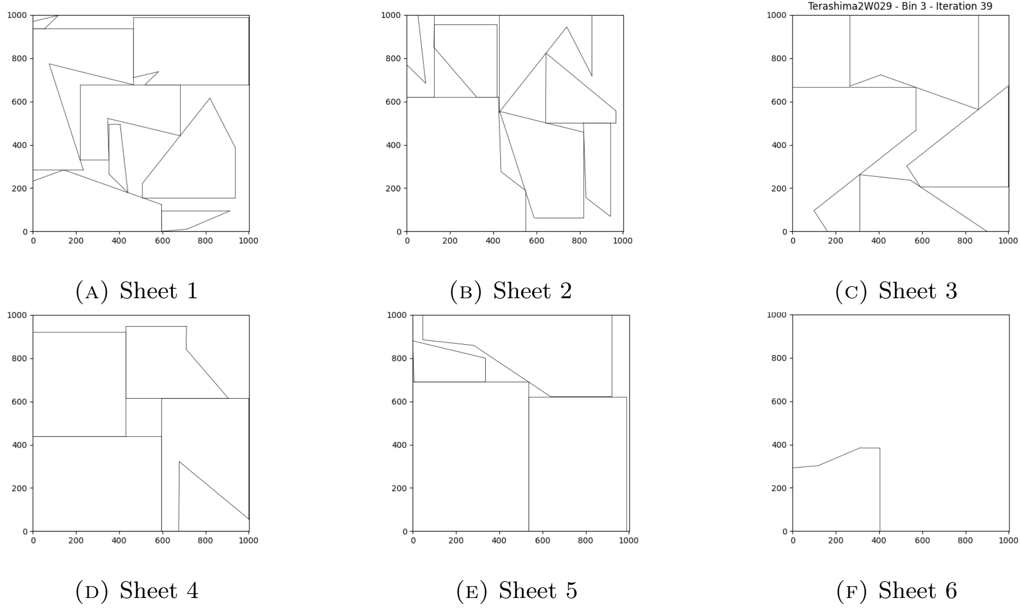
FIGURE 47: Nesting solution layout of type U of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

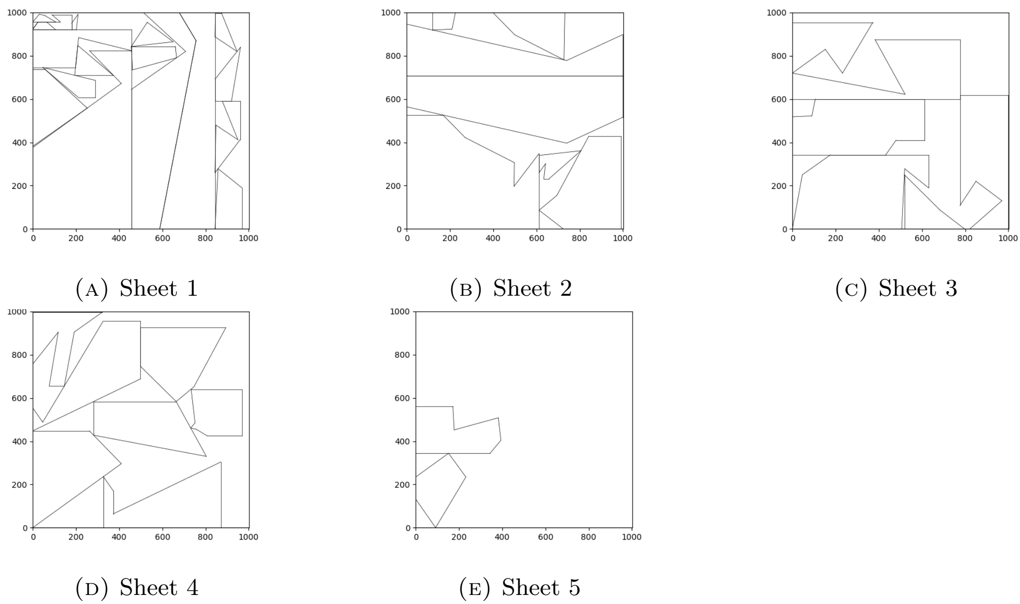
FIGURE 48: Nesting solution layout of type V of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

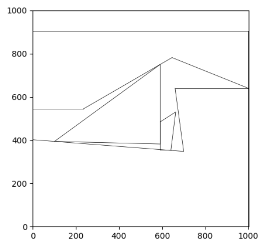
FIGURE 49: Nesting solution layout of type W of JP2.



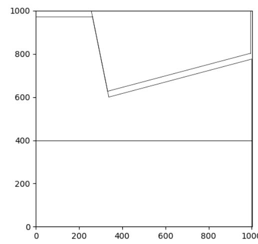
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5

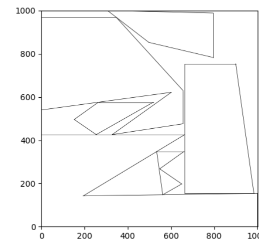
FIGURE 50: Nesting solution layout of type X of JP2.



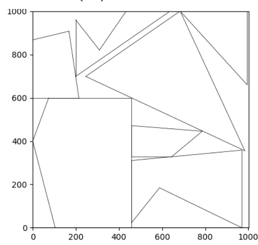
(A) Sheet 1



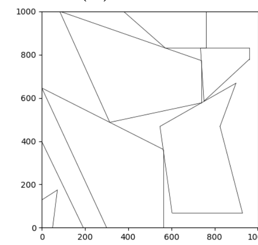
(B) Sheet 2



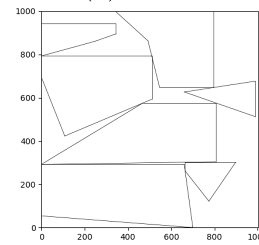
(C) Sheet 3



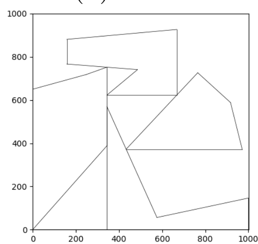
(D) Sheet 4



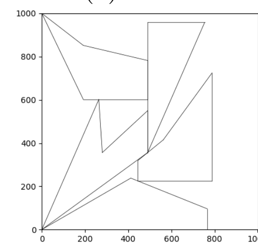
(E) Sheet 5



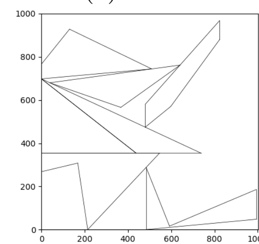
(F) Sheet 6



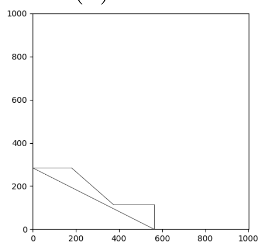
(G) Sheet 7



(H) Sheet 8

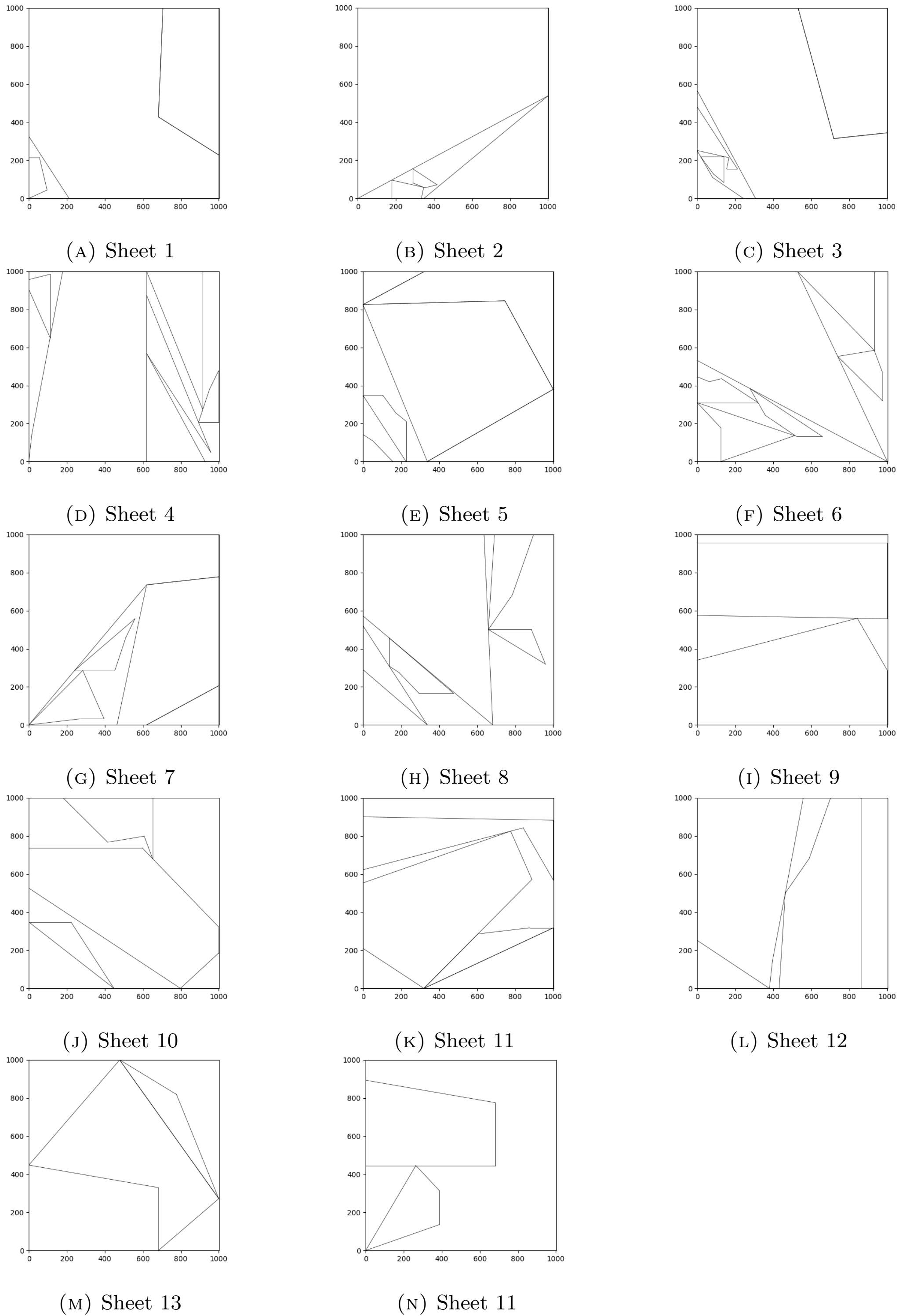


(I) Sheet 9



(J) Sheet 10

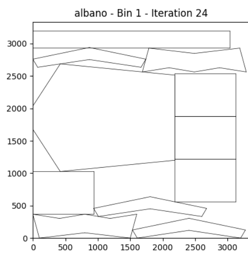
FIGURE 51: Nesting solution layout of type Y of JP2.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3
(D) Sheet 4 (E) Sheet 5 (F) Sheet 6
(G) Sheet 7 (H) Sheet 8 (I) Sheet 9
(J) Sheet 10 (K) Sheet 11 (L) Sheet 12
(M) Sheet 13 (N) Sheet 11

Figure 52: Nesting solution layout of type Z of JP2.

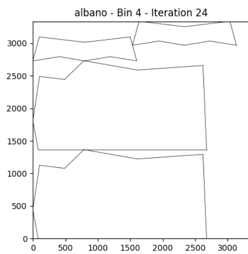
L Nesting results for the Nest-SB instances



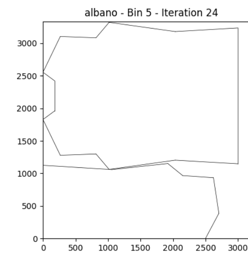
(A) Sheet 1

(B) Sheet 2

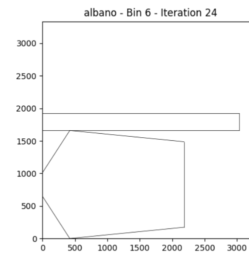
(C) Sheet 3



(D) Sheet 4

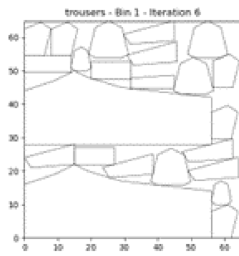


(E) Sheet 5

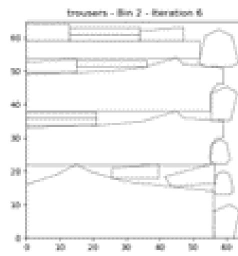


(F) Sheet 6

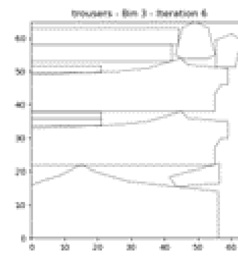
FIGURE 53: Nesting solution layout of albano of the Nest-SB instances.



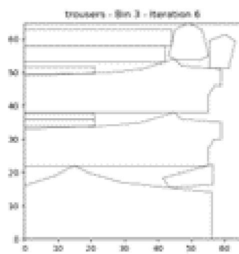
(A) Sheet 1



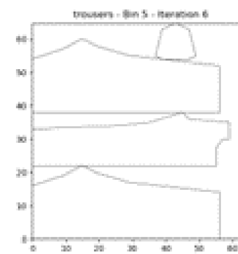
(B) Sheet 2



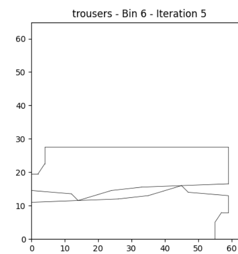
(C) Sheet 3



(D) Sheet 4

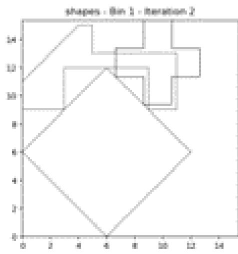


(E) Sheet 5

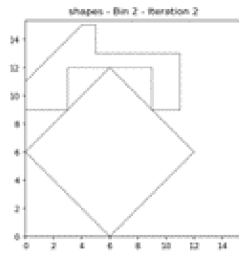


(F) Sheet 6

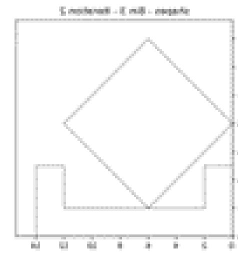
FIGURE 54: Nesting solution layout of trousers of the Nest-SB instances.



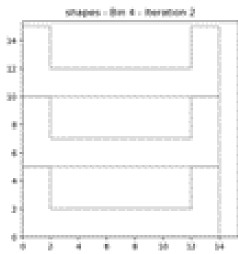
(A) Sheet 1



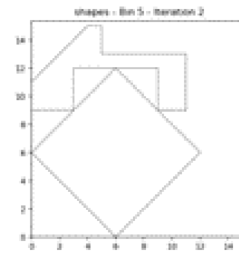
(B) Sheet 2



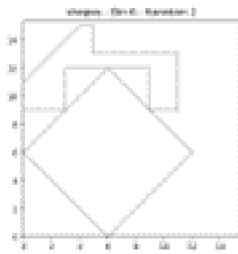
(C) Sheet 3



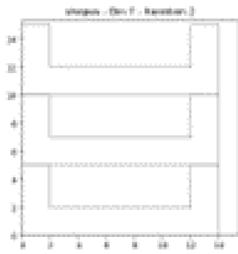
(D) Sheet 4



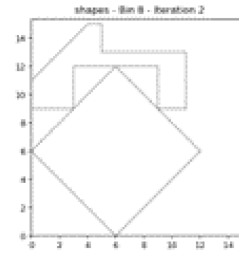
(E) Sheet 5



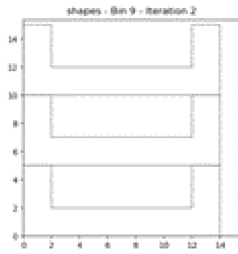
(F) Sheet 6



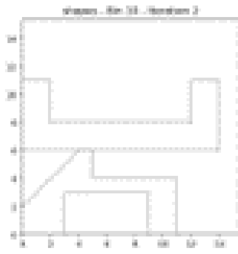
(G) Sheet 7



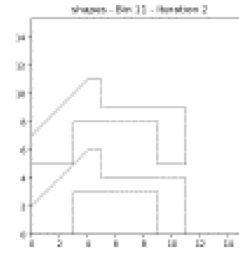
(H) Sheet 8



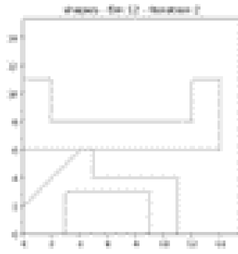
(I) Sheet 9



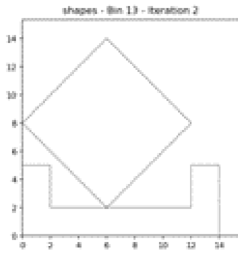
(J) Sheet 10



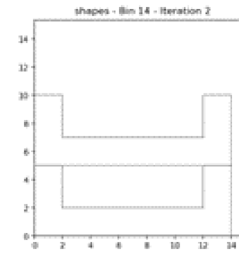
(K) Sheet 11



(L) Sheet 12

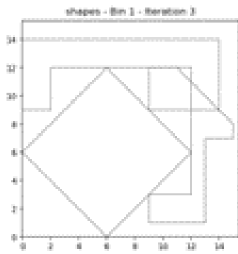


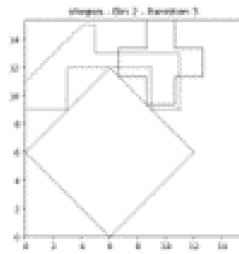
(M) Sheet 13

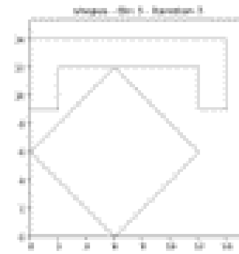


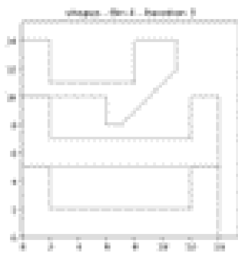
(N) Sheet 14

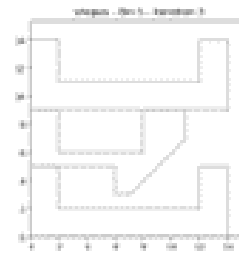
FIGURE 55: Nesting solution layout of shapes0 of the Nest-SB instances.

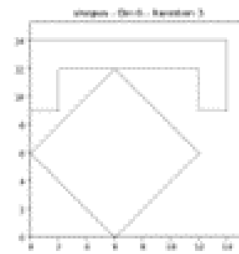

(A) Sheet 1

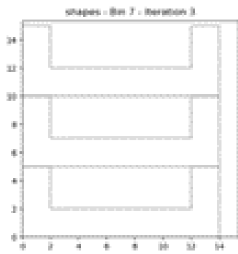

(B) Sheet 2

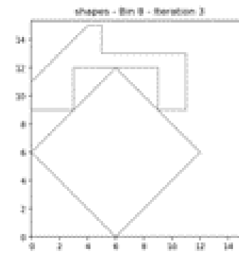

(C) Sheet 3

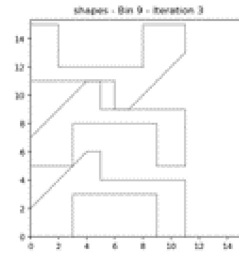

(D) Sheet 4

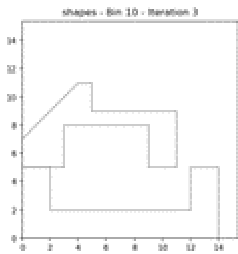

(E) Sheet 5

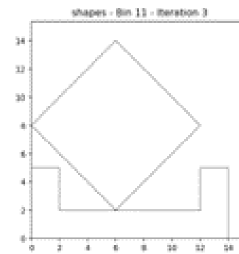

(F) Sheet 6

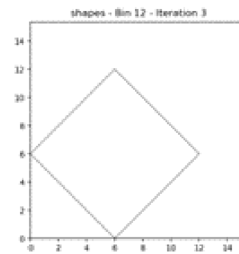

(G) Sheet 7


(H) Sheet 8


(I) Sheet 9


(J) Sheet 10


(K) Sheet 11


(L) Sheet 12

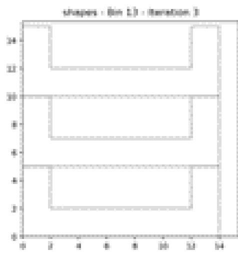
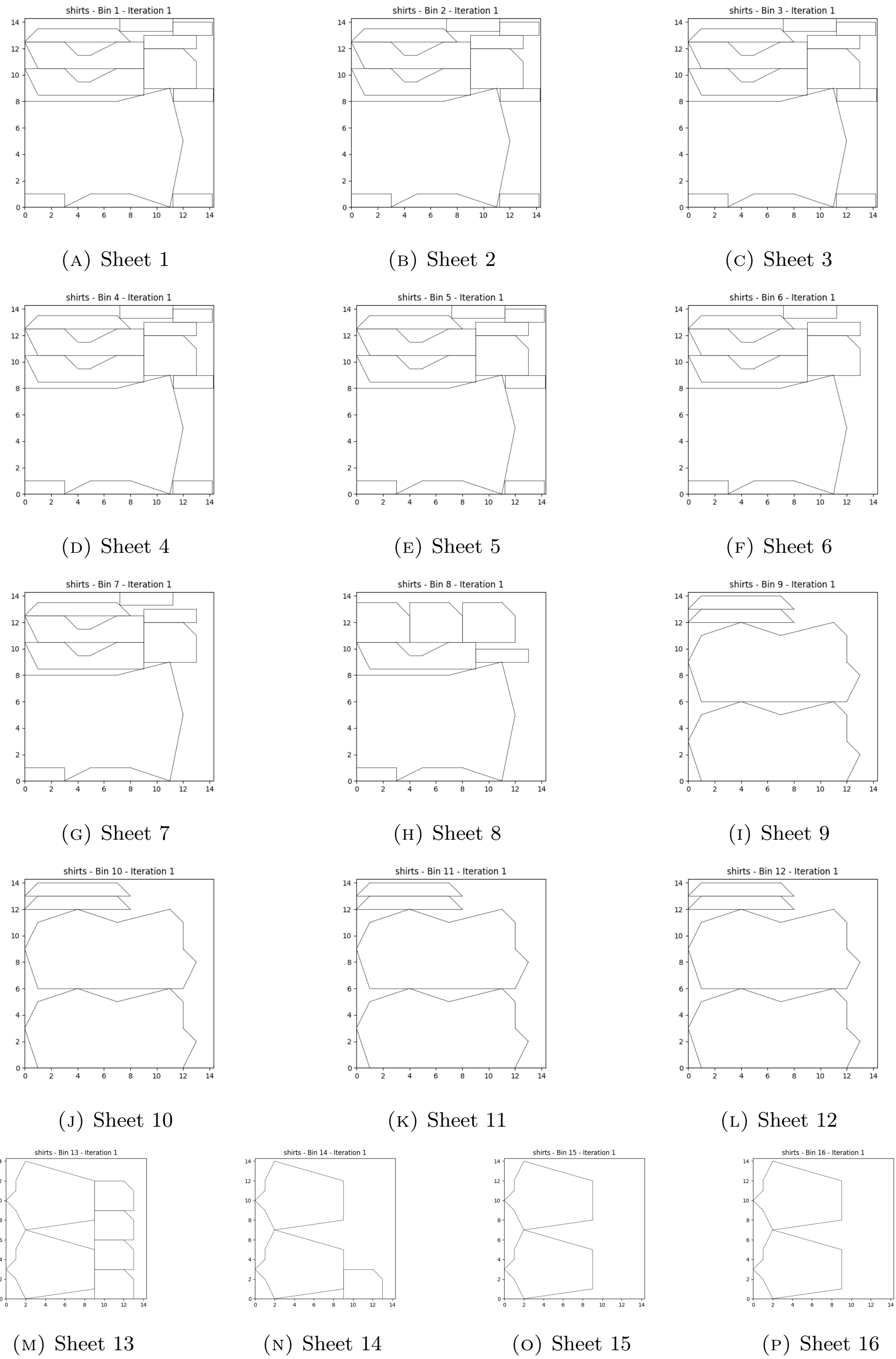

(M) Sheet 13

FIGURE 56: Nesting solution layout of shapes1 of the Nest-SB instances.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

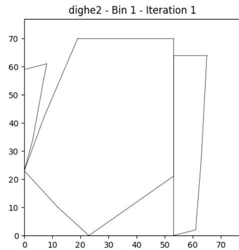
(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

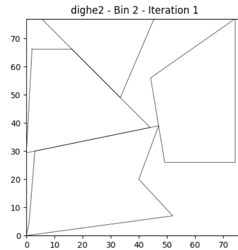
(J) Sheet 10 (K) Sheet 11 (L) Sheet 12

(M) Sheet 13 (N) Sheet 14 (O) Sheet 15 (P) Sheet 16

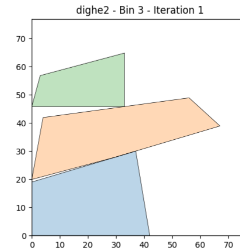
FIGURE 57: Nesting solution layout of shirts of the Nest-SB instances.



(A) Sheet 1

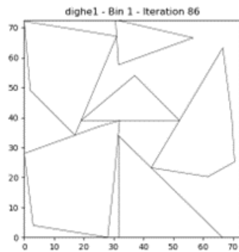


(B) Sheet 2

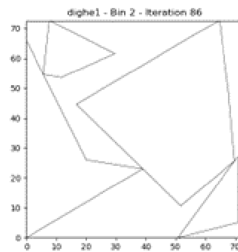


(C) Sheet 3

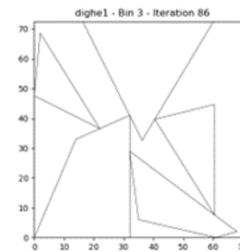
Figure 58: Nesting solution layout of dighe2 of the Nest-SB instances.



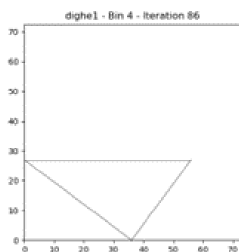
(A) Sheet 1



(B) Sheet 2

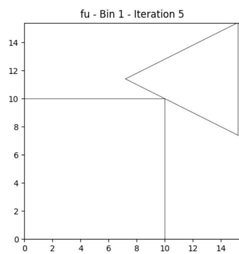


(C) Sheet 3

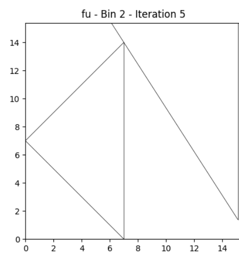


(D) Sheet 4

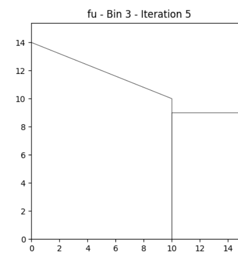
Figure 59: Nesting solution layout of dighe1 of the Nest-SB instances.



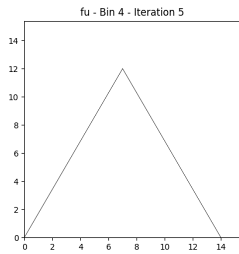
(A) Sheet 1



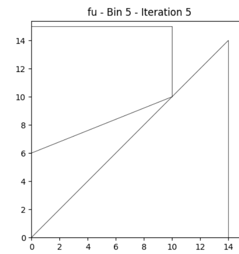
(B) Sheet 2



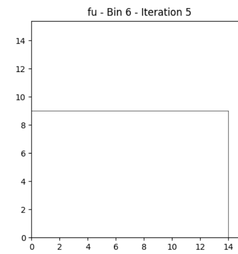
(C) Sheet 3



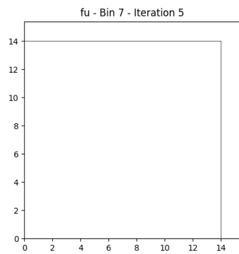
(D) Sheet 4



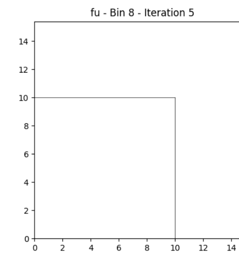
(E) Sheet 5



(F) Sheet 6

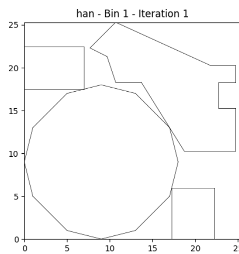


(G) Sheet 7

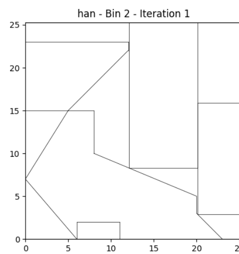


(H) Sheet 8

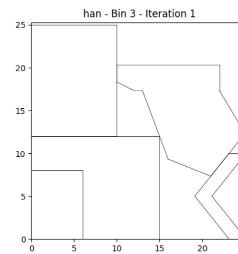
FIGURE 60: Nesting solution layout of fu of the Nest-SB instances.



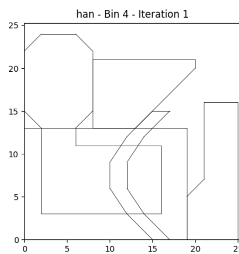
(A) Sheet 1



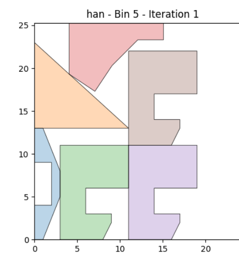
(B) Sheet 2



(C) Sheet 3

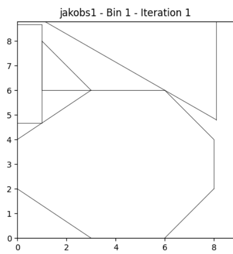


(D) Sheet 4

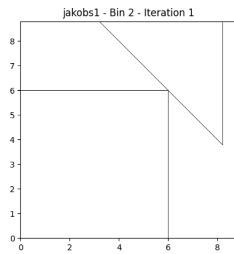


(E) Sheet 5

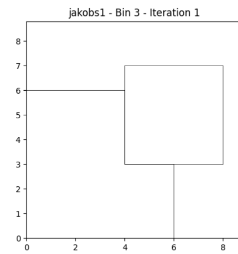
FIGURE 61: Nesting solution layout of han of the Nest-SB instances.



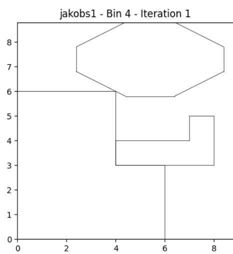
(A) Sheet 1



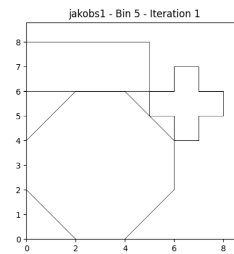
(B) Sheet 2



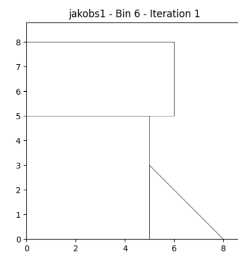
(C) Sheet 3



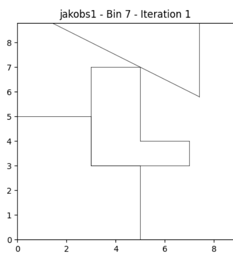
(D) Sheet 4



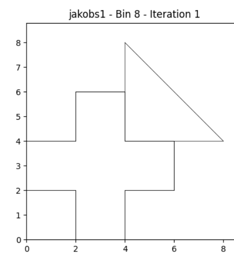
(E) Sheet 5



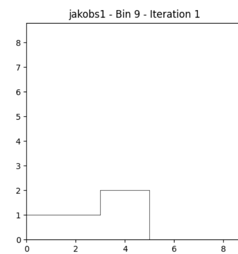
(F) Sheet 6



(G) Sheet 7

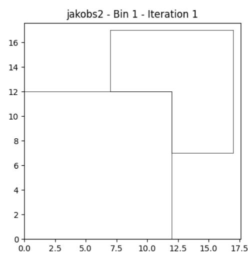


(H) Sheet 8

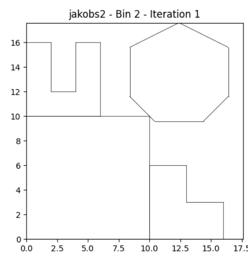


(I) Sheet 9

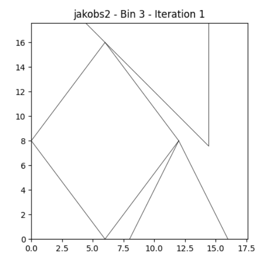
FIGURE 62: Nesting solution layout of jakobs1 of the Nest-SB instances.



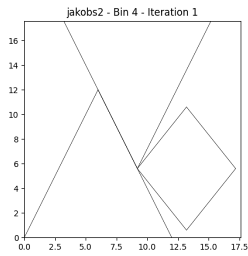
(A) Sheet 1



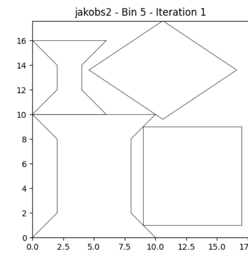
(B) Sheet 2



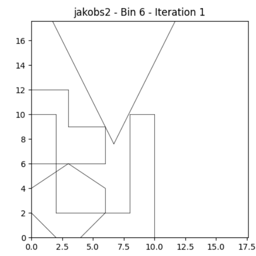
(C) Sheet 3



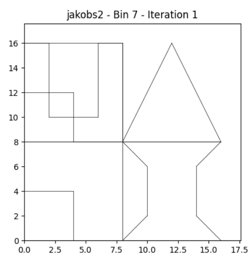
(D) Sheet 4



(E) Sheet 5

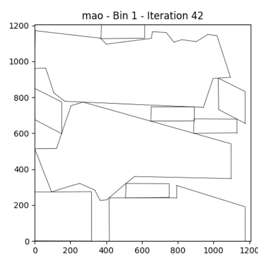


(F) Sheet 6

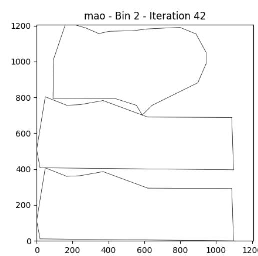


(G) Sheet 7

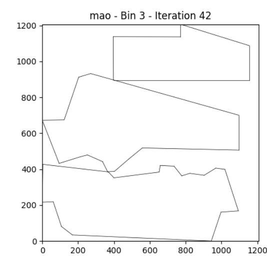
Figure 63: Nesting solution layout of jakobs2 of the Nest-SB instances.



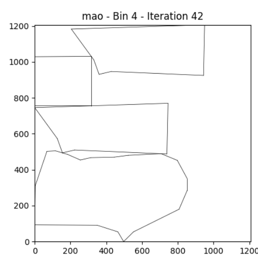
(A) Sheet 1



(B) Sheet 2

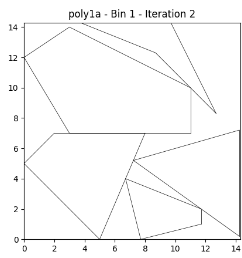


(C) Sheet 3

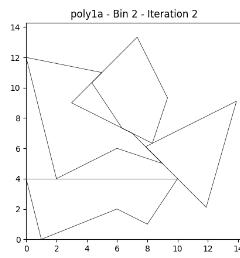


(D) Sheet 4

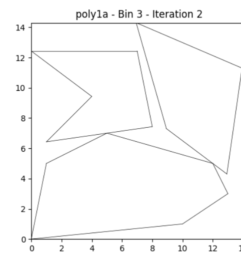
Figure 64: Nesting solution layout of mao of the Nest-SB instances.



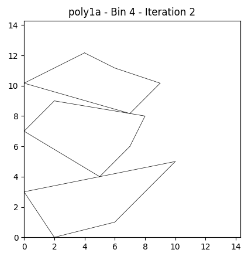
(A) Sheet 1



(B) Sheet 2

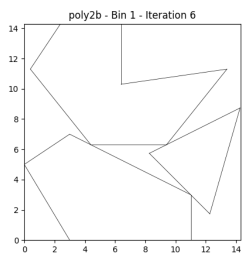


(C) Sheet 3

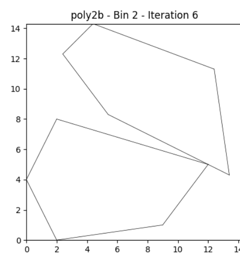


(D) Sheet 4

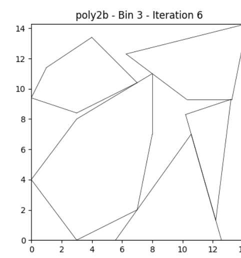
FIGURE 65: Nesting solution layout of poly1a of the Nest-SB instances.



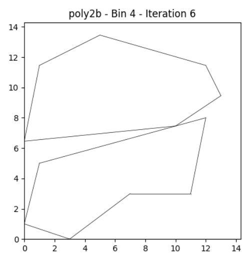
(A) Sheet 1



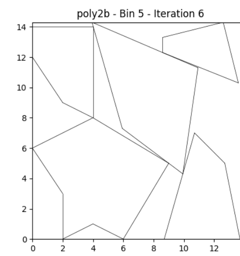
(B) Sheet 2



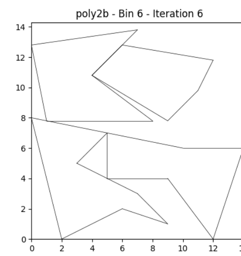
(C) Sheet 3



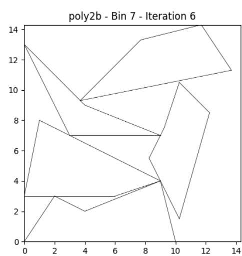
(D) Sheet 4



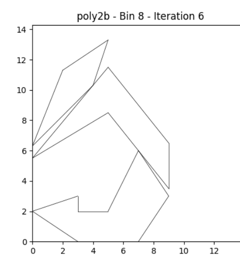
(E) Sheet 5



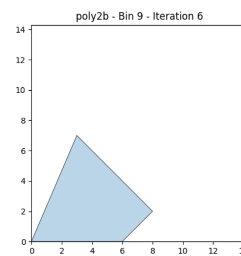
(F) Sheet 6



(G) Sheet 7

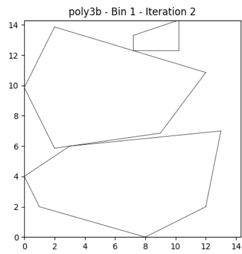


(H) Sheet 8

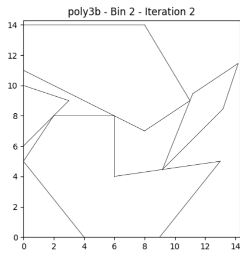


(I) Sheet 9

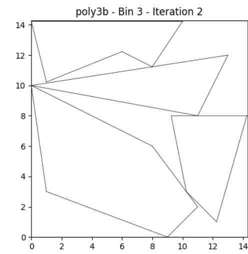
FIGURE 66: Nesting solution layout of poly2b of the Nest-SB instances.



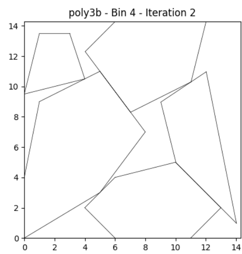
(A) Sheet 1



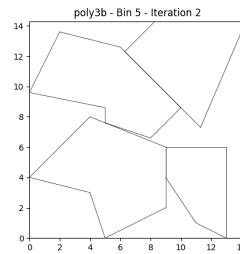
(B) Sheet 2



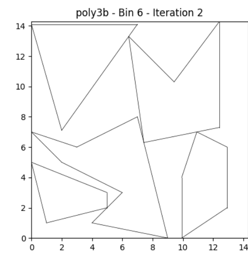
(C) Sheet 3



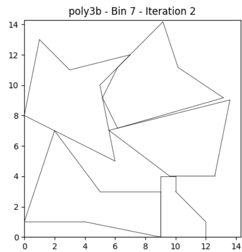
(D) Sheet 4



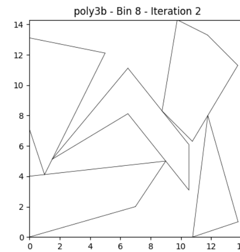
(E) Sheet 5



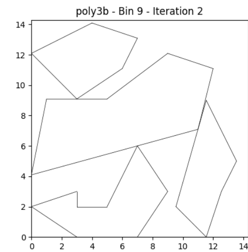
(F) Sheet 6



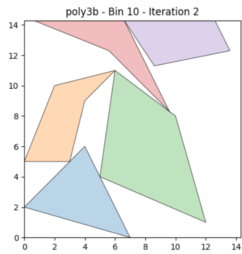
(G) Sheet 7



(H) Sheet 8

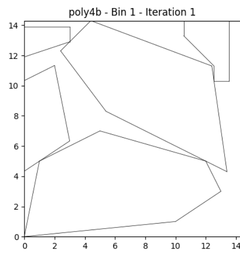


(I) Sheet 9

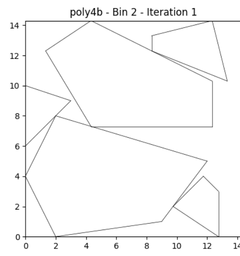


(J) Sheet 10

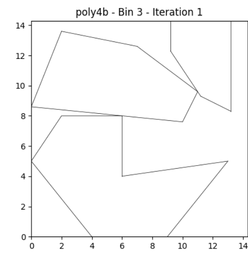
FIGURE 67: Nesting solution layout of poly3b of the Nest-SB instances.



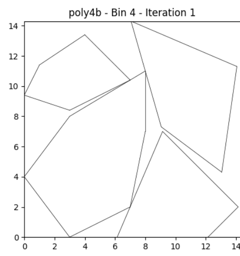
(A) Sheet 1



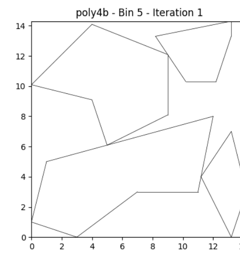
(B) Sheet 2



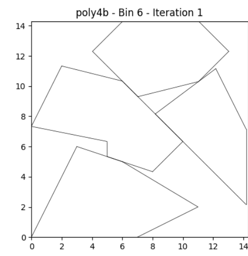
(C) Sheet 3



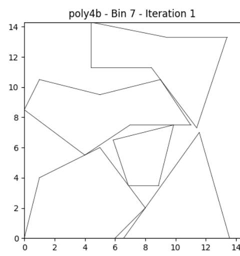
(D) Sheet 4



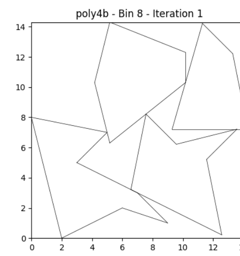
(E) Sheet 5



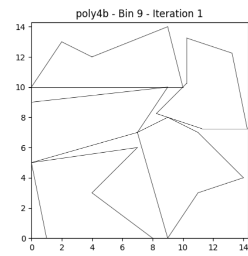
(F) Sheet 6



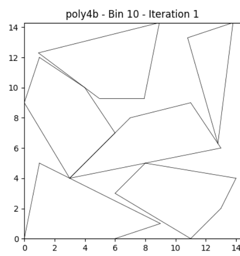
(G) Sheet 7



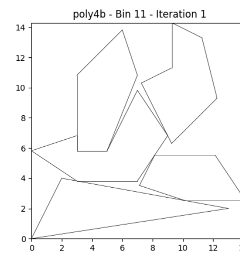
(H) Sheet 8



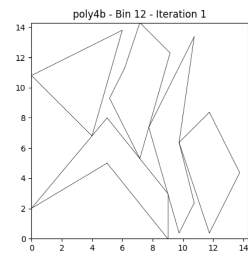
(I) Sheet 9



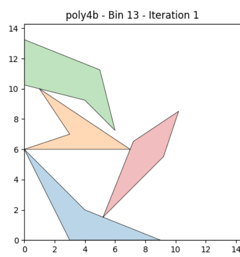
(J) Sheet 10



(K) Sheet 11

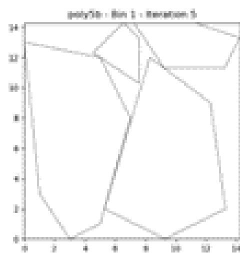


(L) Sheet 12

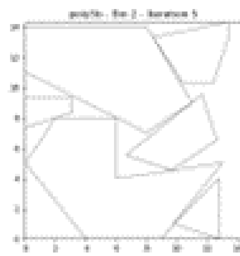


(M) Sheet 13

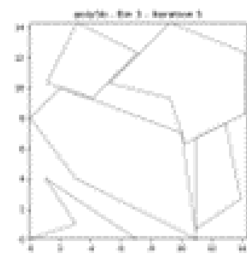
Figure 68: Nesting solution layout of poly4b of the Nest-SB instance.



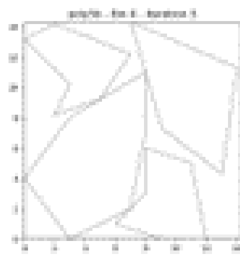
(A) Sheet 1



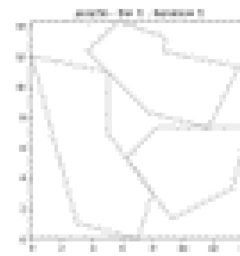
(B) Sheet 2



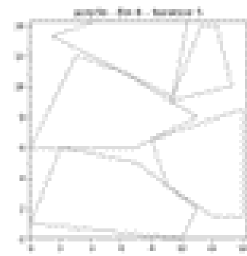
(C) Sheet 3



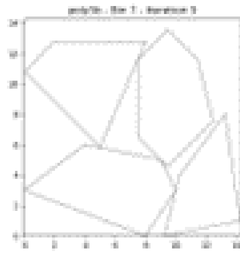
(D) Sheet 4



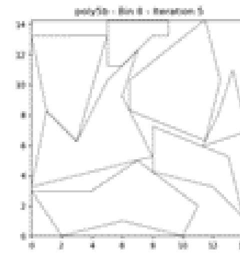
(E) Sheet 5



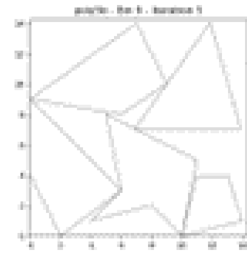
(F) Sheet 6



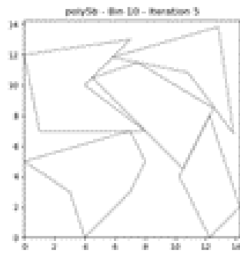
(G) Sheet 7



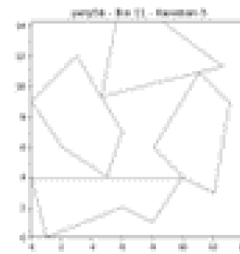
(H) Sheet 8



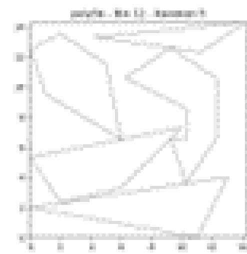
(I) Sheet 9



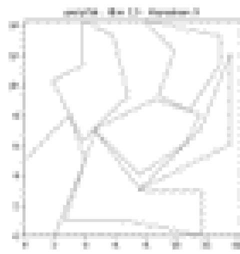
(J) Sheet 10



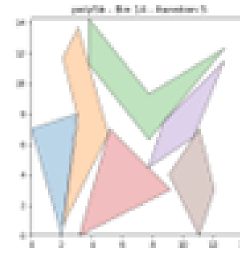
(K) Sheet 11



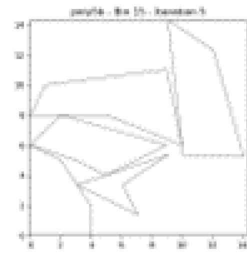
(L) Sheet 12



(M) Sheet 13

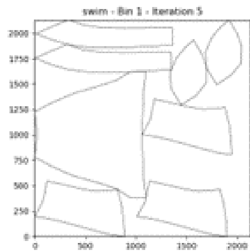


(N) Sheet 14

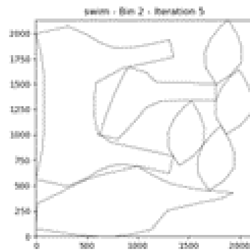


(O) Sheet 15

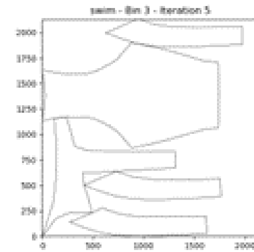
FIGURE 69: Nesting solution layout of poly5b of the Nest-SB instances.



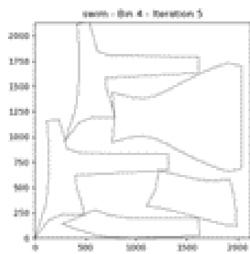
(A) Sheet 1



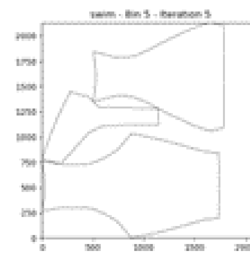
(B) Sheet 2



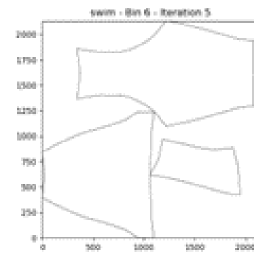
(C) Sheet 3



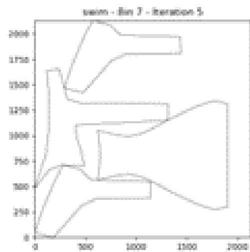
(D) Sheet 4



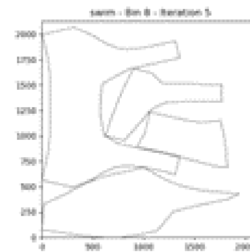
(E) Sheet 5



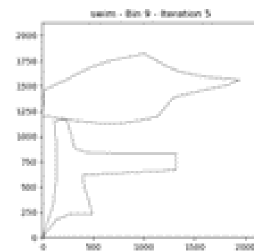
(F) Sheet 6



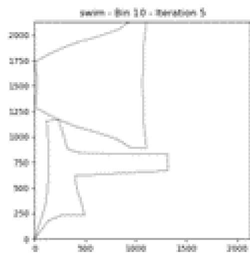
(G) Sheet 7



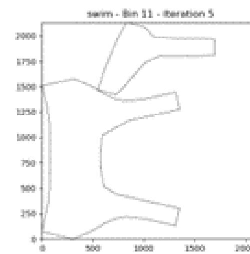
(H) Sheet 8



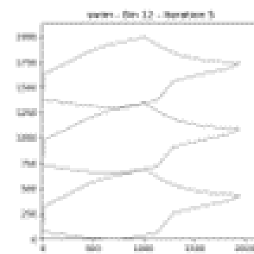
(I) Sheet 9



(J) Sheet 10



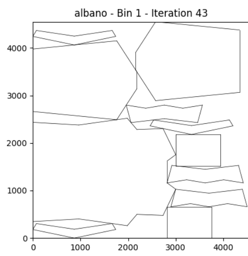
(K) Sheet 11



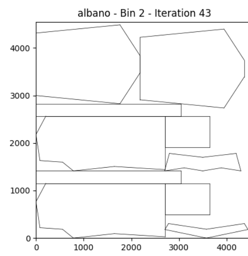
(L) Sheet 12

FIGURE 70: Nesting solution layout of swim of the Nest-SB instances.

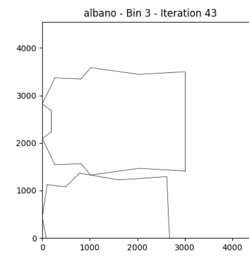
M Nesting results for the Nest-MB instances



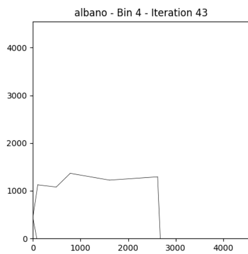
(A) Sheet 1



(B) Sheet 2

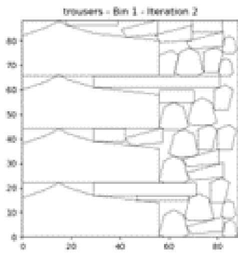


(C) Sheet 3

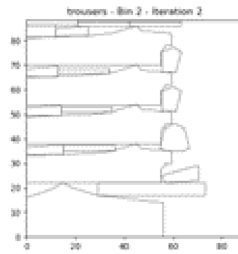


(D) Sheet 4

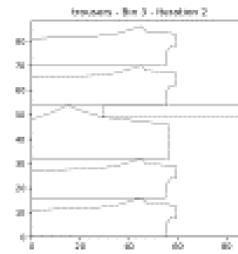
FIGURE 71: Nesting solution layout of albano of the Nest-MB instances.



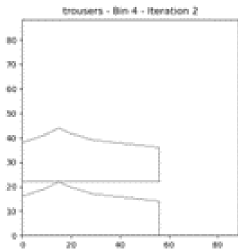
(A) Sheet 1



(B) Sheet 2

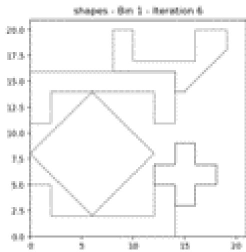


(C) Sheet 3

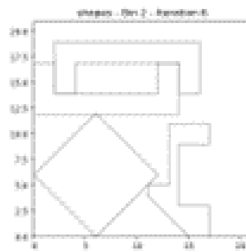


(D) Sheet 4

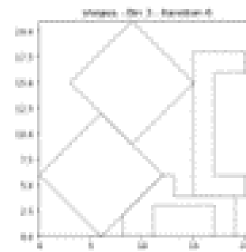
FIGURE 72: Nesting solution layout of trousers of the Nest-MB instances.



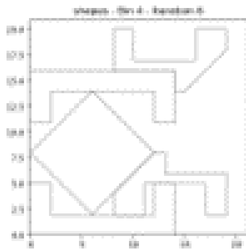
(A) Sheet 1



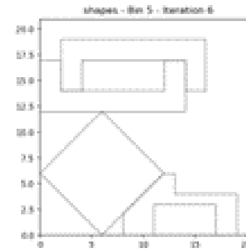
(B) Sheet 2



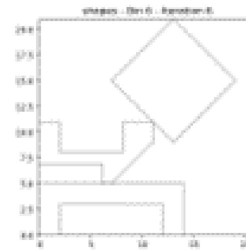
(C) Sheet 3



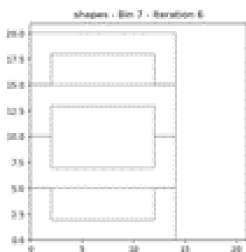
(D) Sheet 4



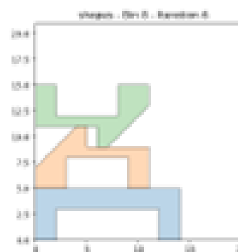
(E) Sheet 5



(F) Sheet 6

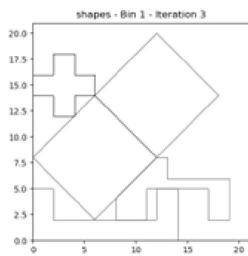


(G) Sheet 7

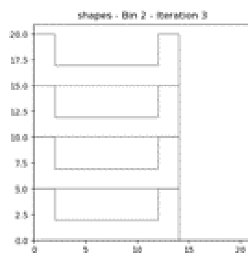


(H) Sheet 8

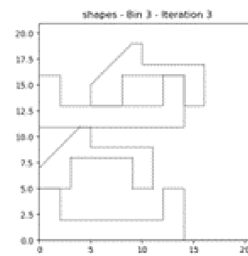
FIGURE 73: Nesting solution layout of shapes0 of the Nest-MB instances.



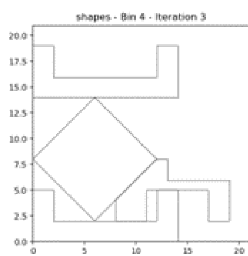
(A) Sheet 1



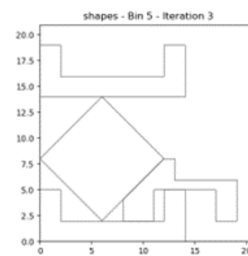
(B) Sheet 2



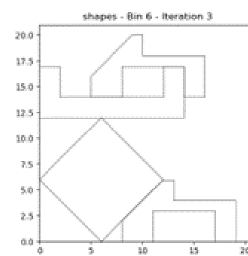
(C) Sheet 3



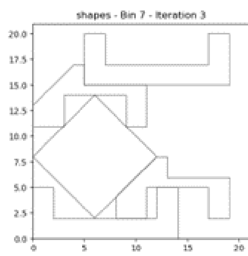
(D) Sheet 4



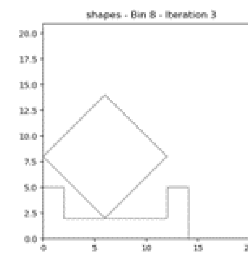
(E) Sheet 5



(F) Sheet 6

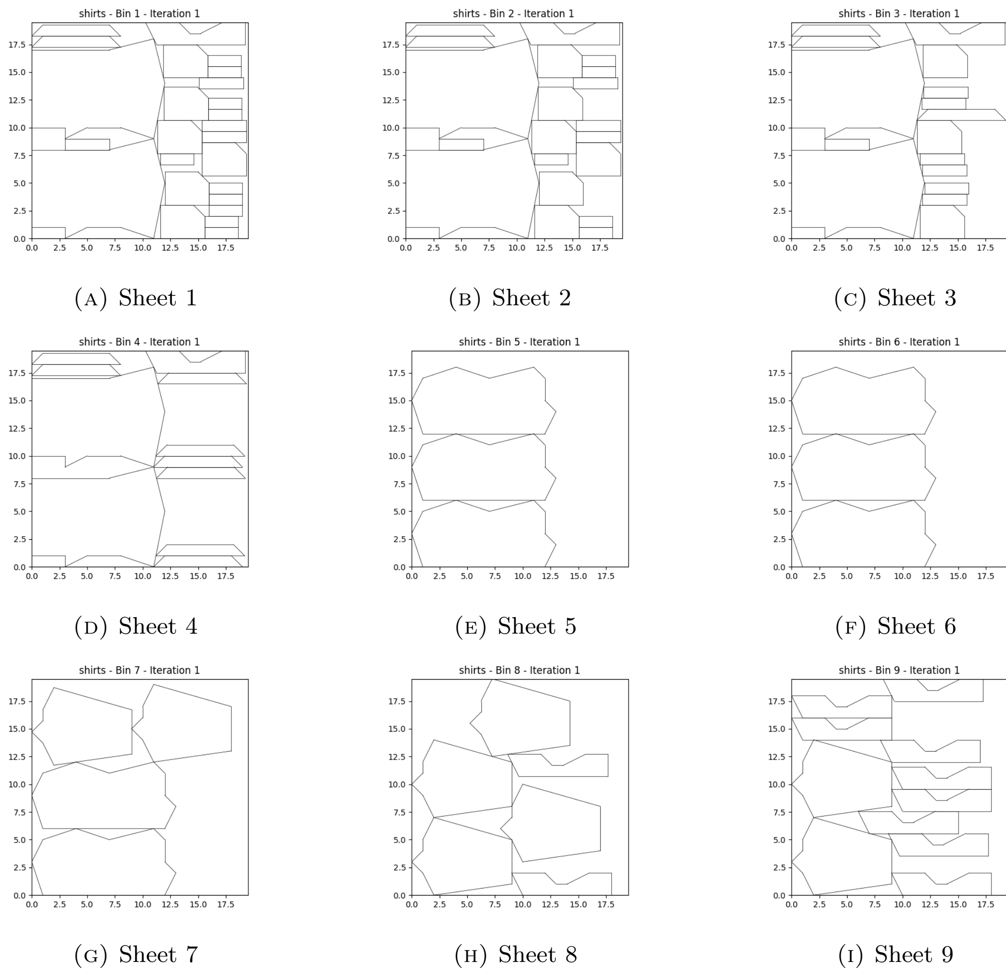


(G) Sheet 7



(H) Sheet 8

FIGURE 74: Nesting solution layout of shapes1 of the Nest-MB instances.

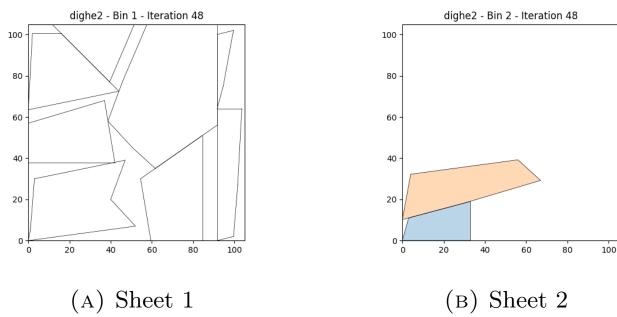


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

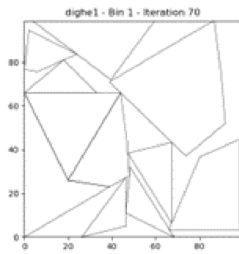
(G) Sheet 7 (H) Sheet 8 (I) Sheet 9

FIGURE 75: Nesting solution layout of shirts of the Nest-MB instances.

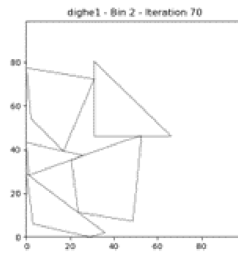


(A) Sheet 1 (B) Sheet 2

FIGURE 76: Nesting solution layout of dighe2 of the Nest-MB instances.

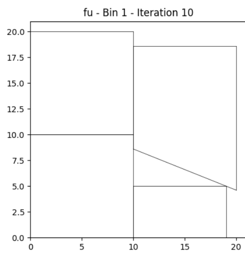


(A) Sheet 1

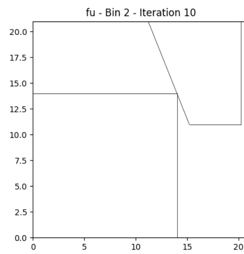


(B) Sheet 2

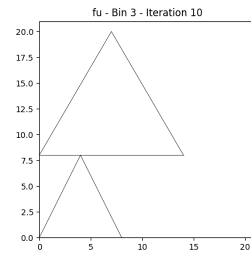
FIGURE 77: Nesting solution layout of dighe1 of the Nest-MB instances.



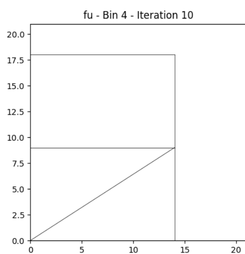
(A) Sheet 1



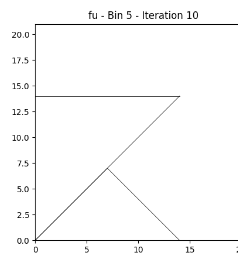
(B) Sheet 2



(C) Sheet 3

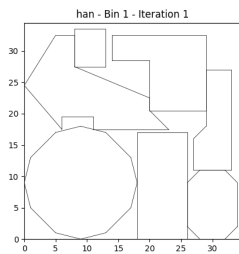


(D) Sheet 4

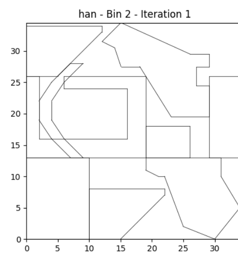


(E) Sheet 5

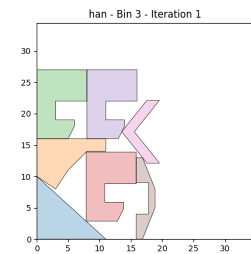
FIGURE 78: Nesting solution layout of fu of the Nest-MB instances.



(A) Sheet 1

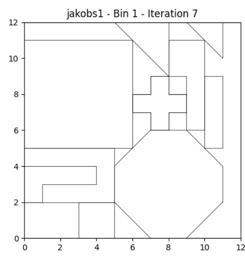


(B) Sheet 2

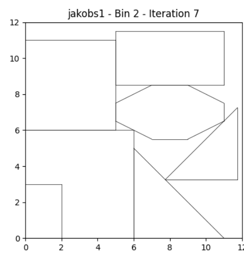


(C) Sheet 3

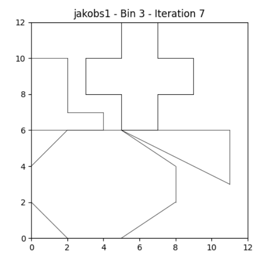
FIGURE 79: Nesting solution layout of han of the Nest-MB instances.



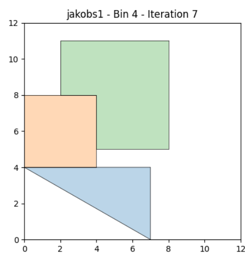
(A) Sheet 1



(B) Sheet 2

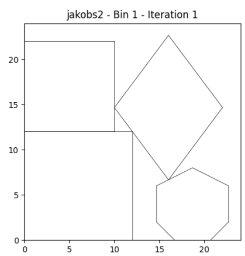


(C) Sheet 3

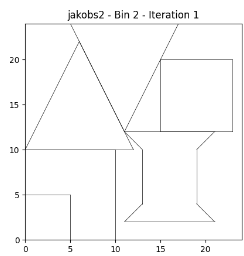


(D) Sheet 4

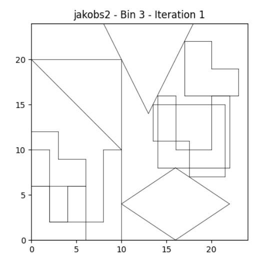
FIGURE 80: Nesting solution layout of jakobs1 of the Nest-MB instances.



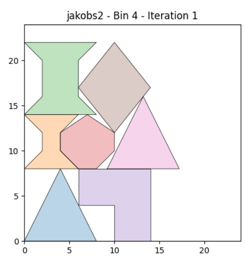
(A) Sheet 1



(B) Sheet 2

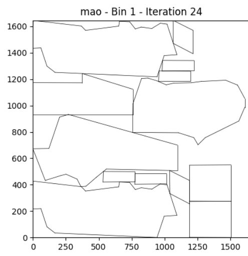


(C) Sheet 3

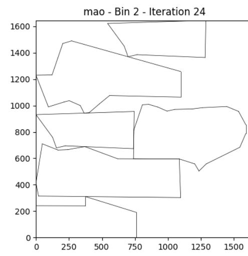


(D) Sheet 4

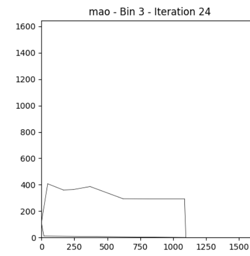
FIGURE 81: Nesting solution layout of jakobs2 of the Nest-MB instances.



(A) Sheet 1

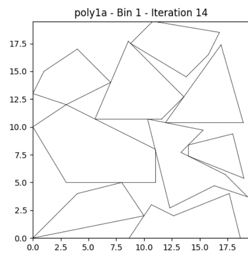


(B) Sheet 2

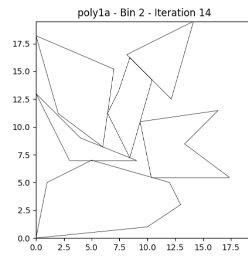


(C) Sheet 3

Figure 82: Nesting solution layout of mao of the Nest-MB instances.

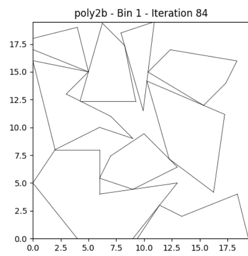


(A) Sheet 1

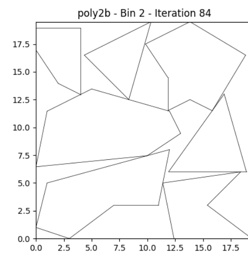


(B) Sheet 2

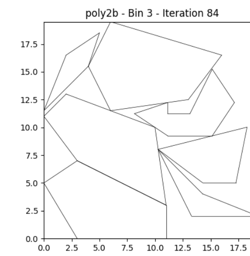
Figure 83: Nesting solution layout of poly1a of the Nest-MB instances.



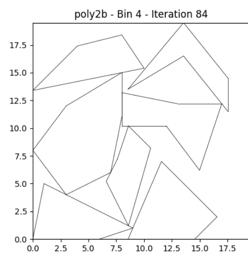
(A) Sheet 1



(B) Sheet 2

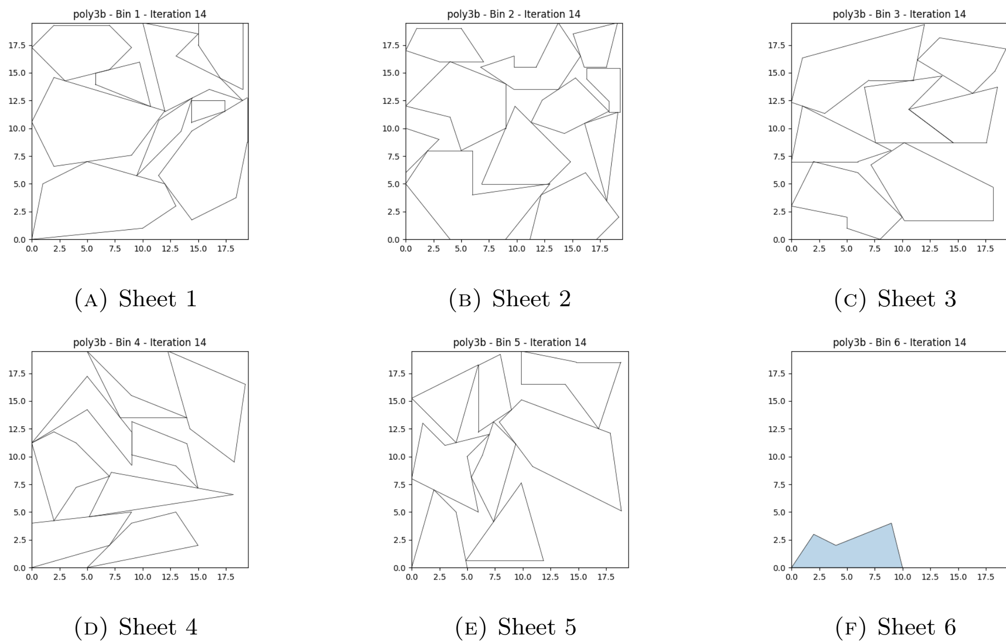


(C) Sheet 3



(D) Sheet 4

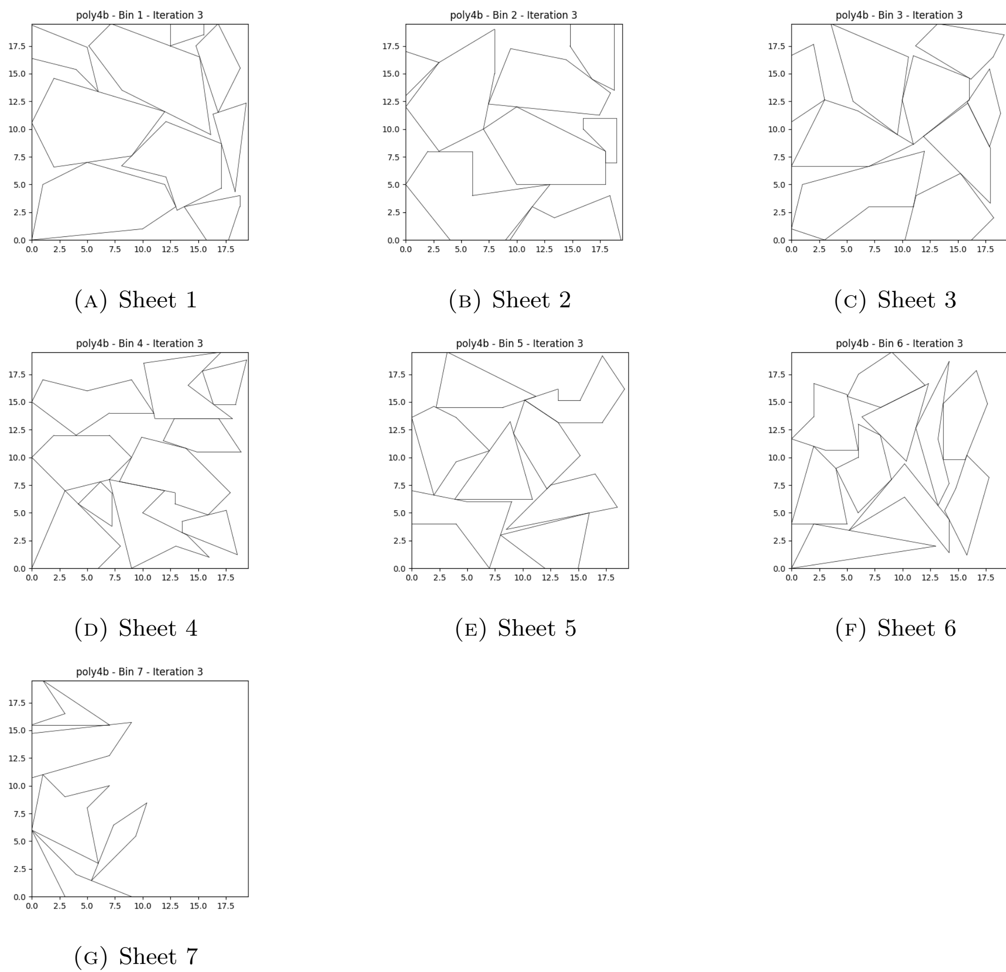
Figure 84: Nesting solution layout of poly2b of the Nest-MB instances.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

FIGURE 85: Nesting solution layout of poly3b of the Nest-MB instances.

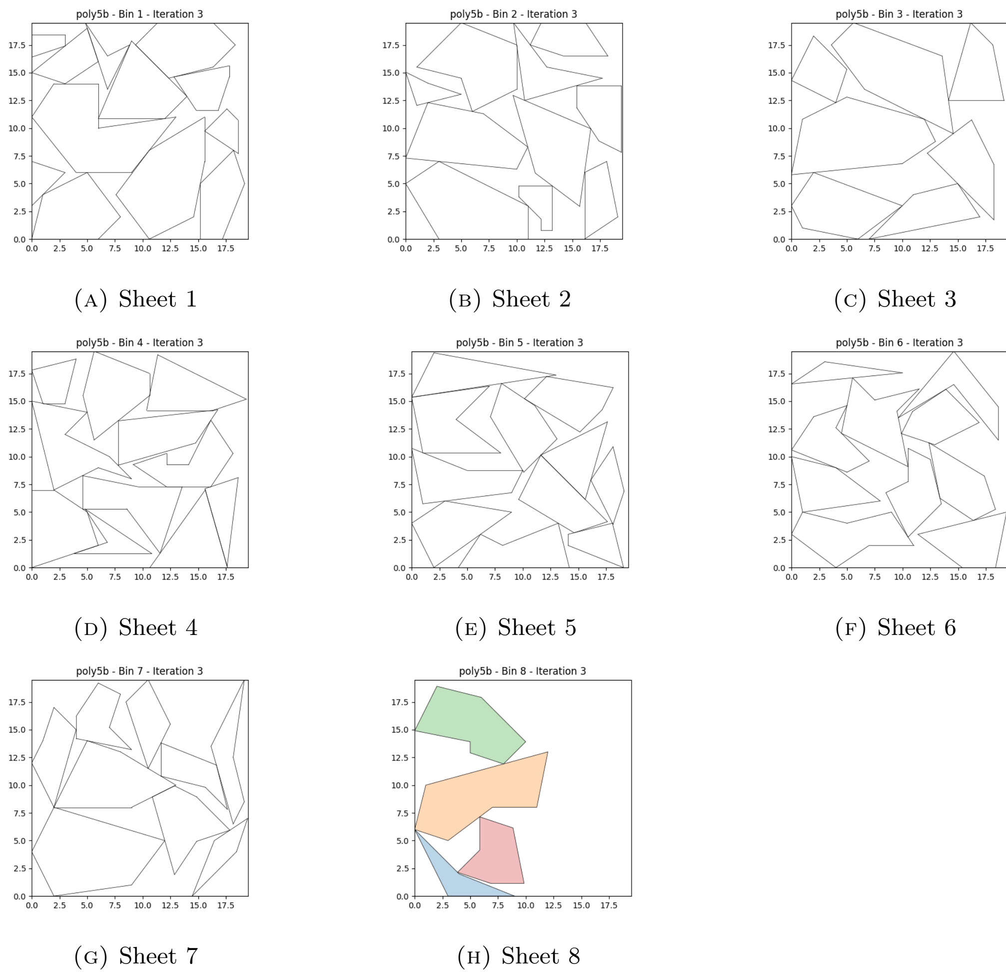


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7

FIGURE 86: Nesting solution layout of poly4b of the Nest-MB instances.

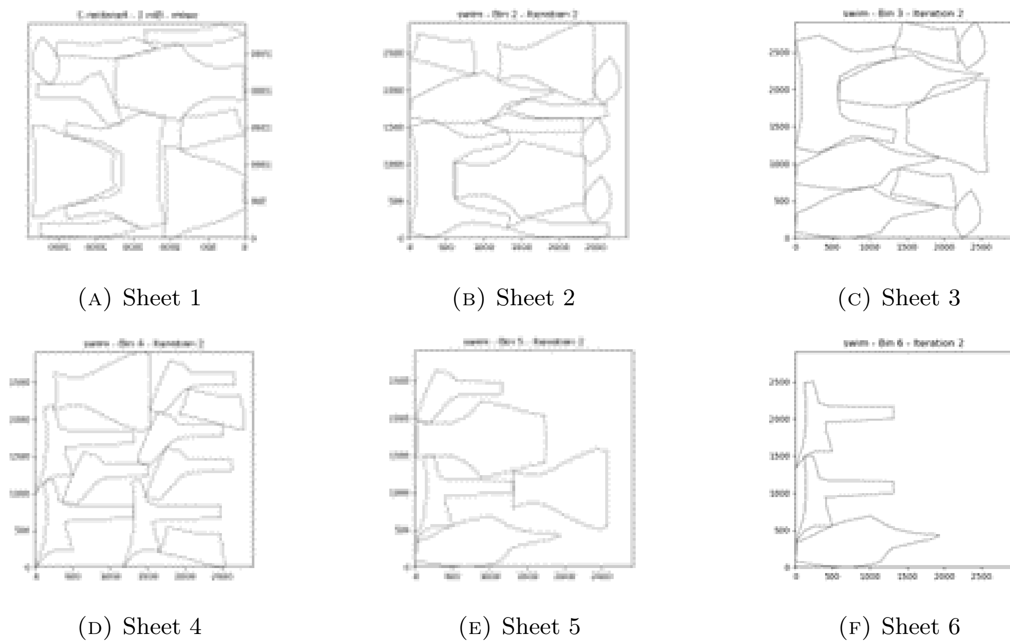


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

(G) Sheet 7 (H) Sheet 8

FIGURE 87: Nesting solution layout of poly5b of the Nest-MB instances.

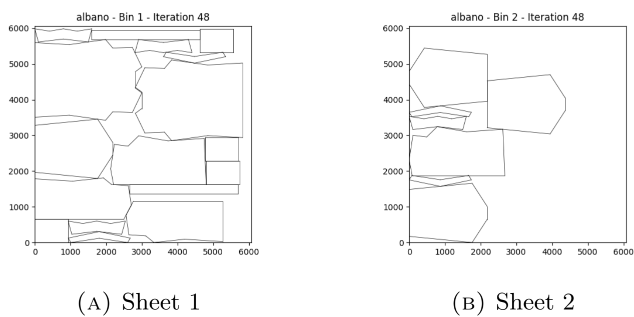


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

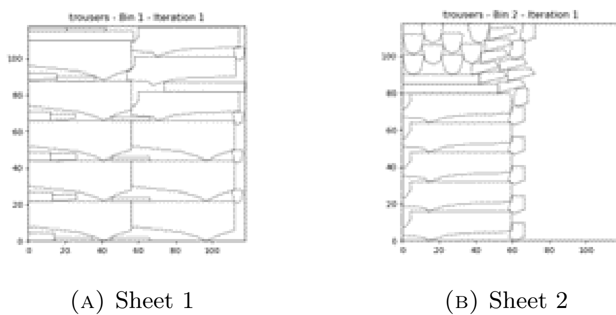
FIGURE 88: Nesting solution layout of swim of the Nest-MB instances.

N Nesting results for the Nest-LB instances



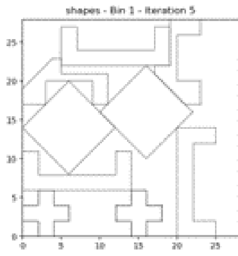
(A) Sheet 1 (B) Sheet 2

FIGURE 89: Nesting solution layout of albano of the Nest-LB instances.

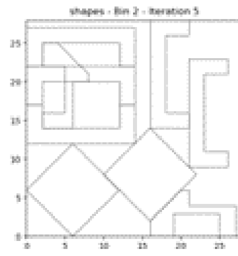


(A) Sheet 1 (B) Sheet 2

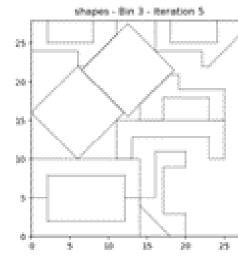
FIGURE 90: Nesting solution layout of trousers of the Nest-LB instances.



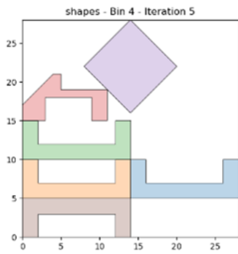
(A) Sheet 1



(B) Sheet 2

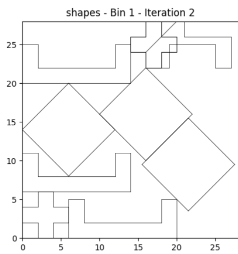


(C) Sheet 3

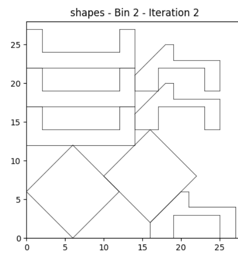


(D) Sheet 4

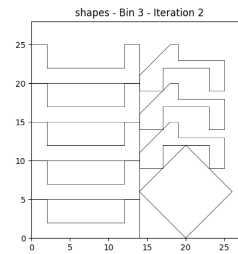
FIGURE 91: Nesting solution layout of shapes0 of the Nest-LB instances.



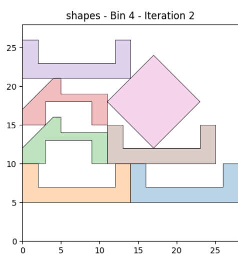
(A) Sheet 1



(B) Sheet 2

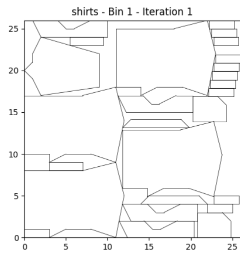


(C) Sheet 3

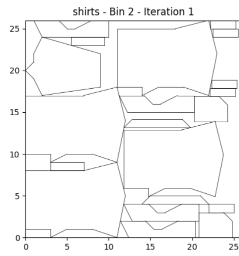


(D) Sheet 4

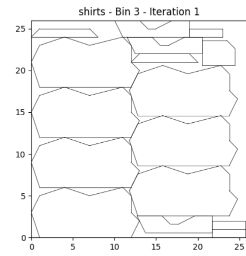
FIGURE 92: Nesting solution layout of shapes1 of the Nest-LB instances.



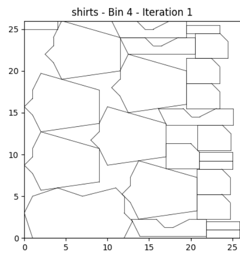
(A) Sheet 1



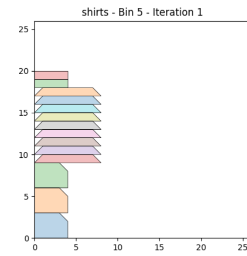
(B) Sheet 2



(C) Sheet 3

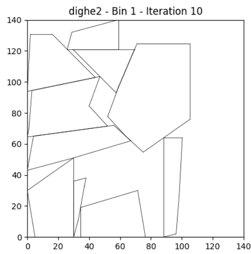


(D) Sheet 4



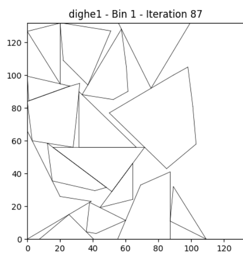
(E) Sheet 5

Figure 93: Nesting solution layout of shirts of the Nest-LB instances.



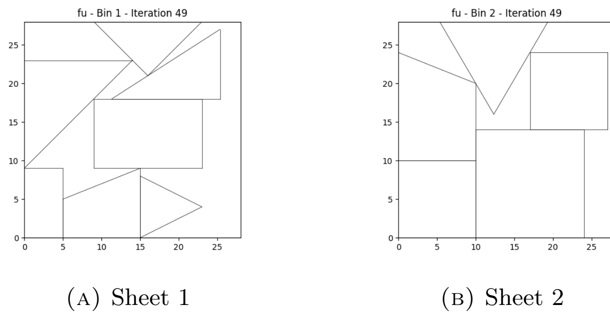
(A) Sheet 1

Figure 94: Nesting solution layout of dighe2 of the Nest-LB instances.



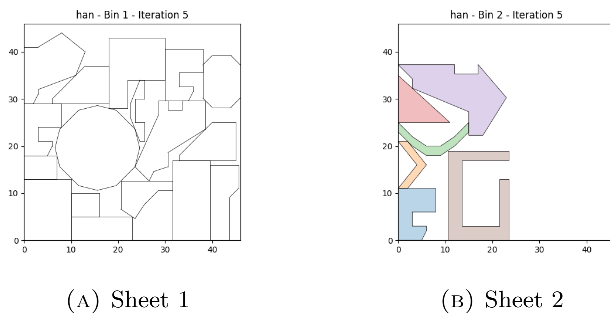
(A) Sheet 1

Figure 95: Nesting solution layout of dighe1 of the Nest-LB instances.



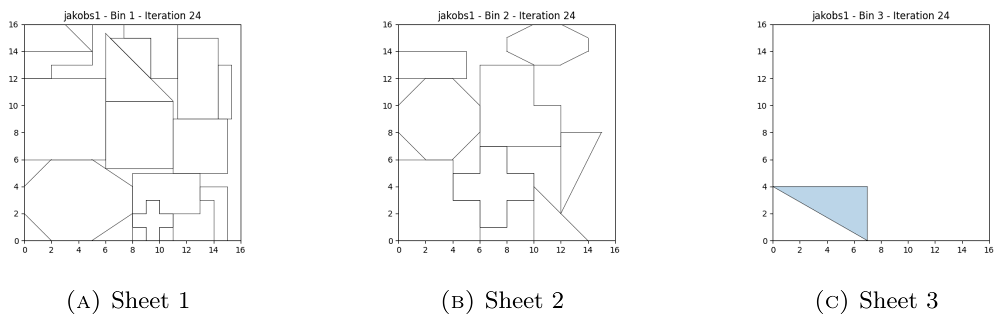
(A) Sheet 1 (B) Sheet 2

FIGURE 96: Nesting solution layout of fu of the Nest-LB instances.



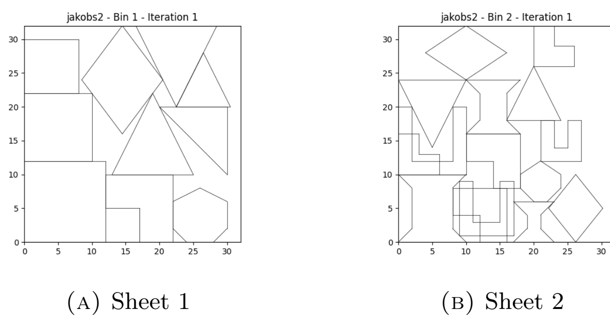
(A) Sheet 1 (B) Sheet 2

FIGURE 97: Nesting solution layout of han of the Nest-LB instances.



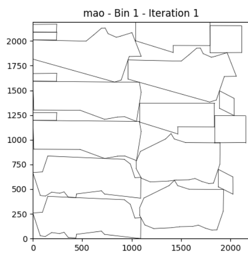
(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

FIGURE 98: Nesting solution layout of jakobs1 of the Nest-LB instances.

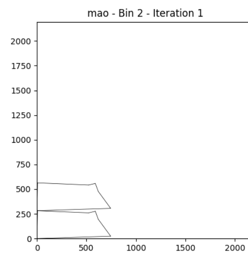


(A) Sheet 1 (B) Sheet 2

FIGURE 99: Nesting solution layout of jakobs2 of the Nest-LB instances.

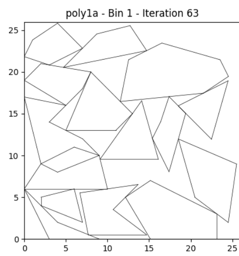


(A) Sheet 1



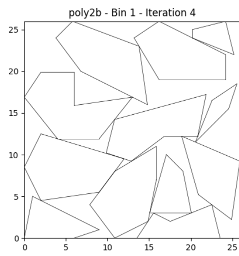
(B) Sheet 2

FIGURE 100: Nesting solution layout of mao of the Nest-LB instances.

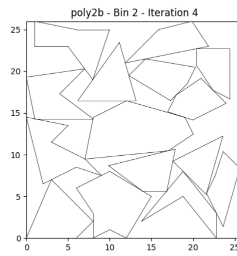


(A) Sheet 1

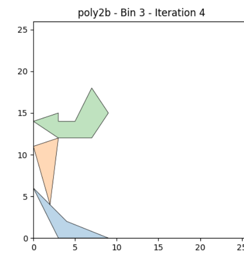
FIGURE 101: Nesting solution layout of poly1a of the Nest-LB instances.



(A) Sheet 1

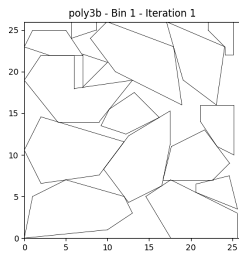


(B) Sheet 2

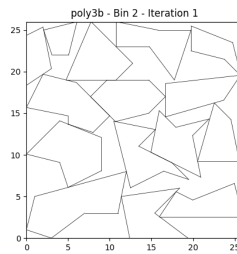


(C) Sheet 3

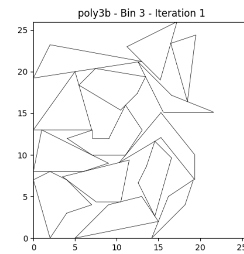
FIGURE 102: Nesting solution layout of poly2b of the Nest-LB instances.



(A) Sheet 1

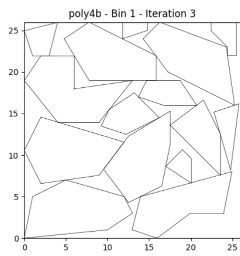


(B) Sheet 2

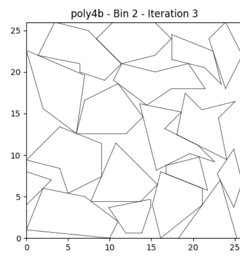


(C) Sheet 3

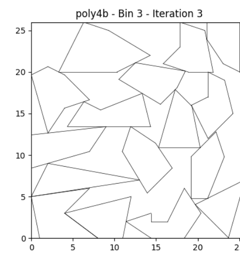
FIGURE 103: Nesting solution layout of poly3b of the Nest-LB instances.



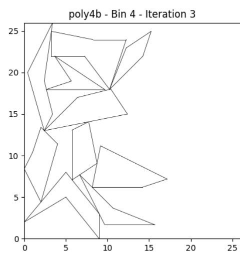
(A) Sheet 1



(B) Sheet 2

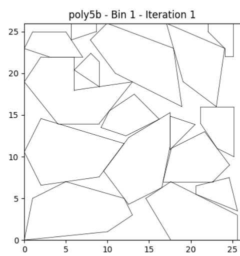


(C) Sheet 3

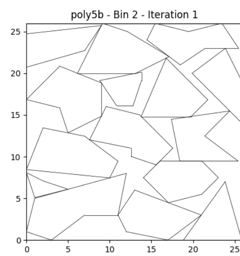


(D) Sheet 4

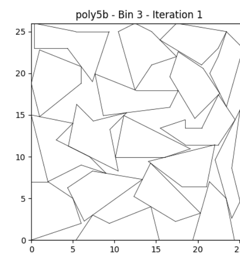
FIGURE 104: Nesting solution layout of poly4b of the Nest-LB instances.



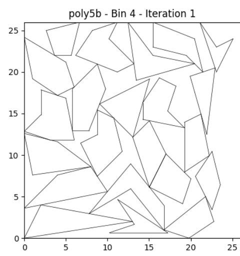
(A) Sheet 1



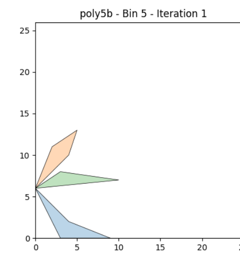
(B) Sheet 2



(C) Sheet 3

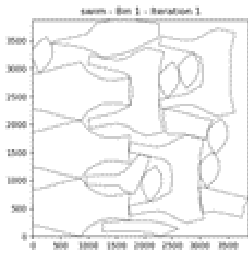


(D) Sheet 4

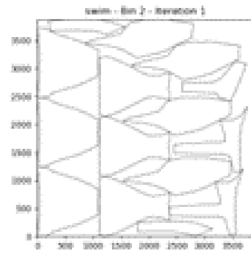


(E) Sheet 5

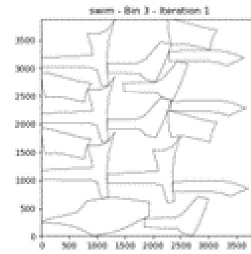
FIGURE 105: Nesting solution layout of poly5b of the Nest-LB instances.



(A) Sheet 1



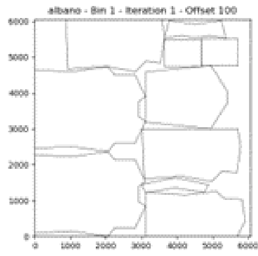
(B) Sheet 2



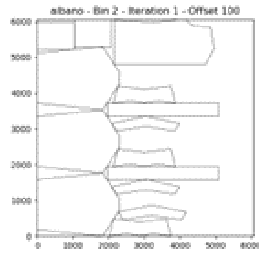
(C) Sheet 3

FIGURE 106: Nesting solution layout of swim of the Nest-LB instances.

O Nesting results of product spaced items

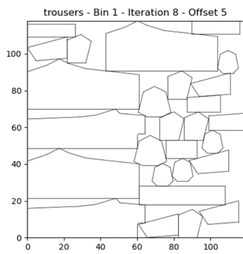


(A) Sheet 1

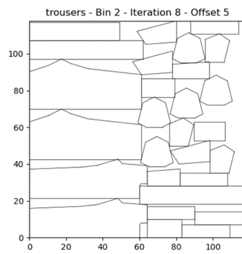


(B) Sheet 2

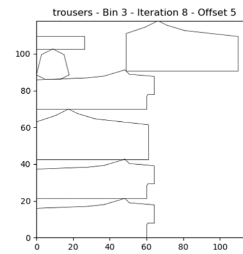
FIGURE 107: Nesting solution layout of albano of the Nest-LB instances with product spacing.



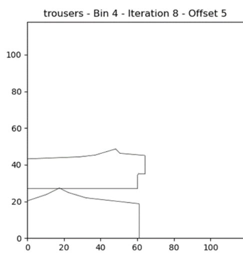
(A) Sheet 1



(B) Sheet 2



(C) Sheet 3



(D) Sheet 4

FIGURE 108: Nesting solution layout of trousers of the Nest-LB instances with product spacing.

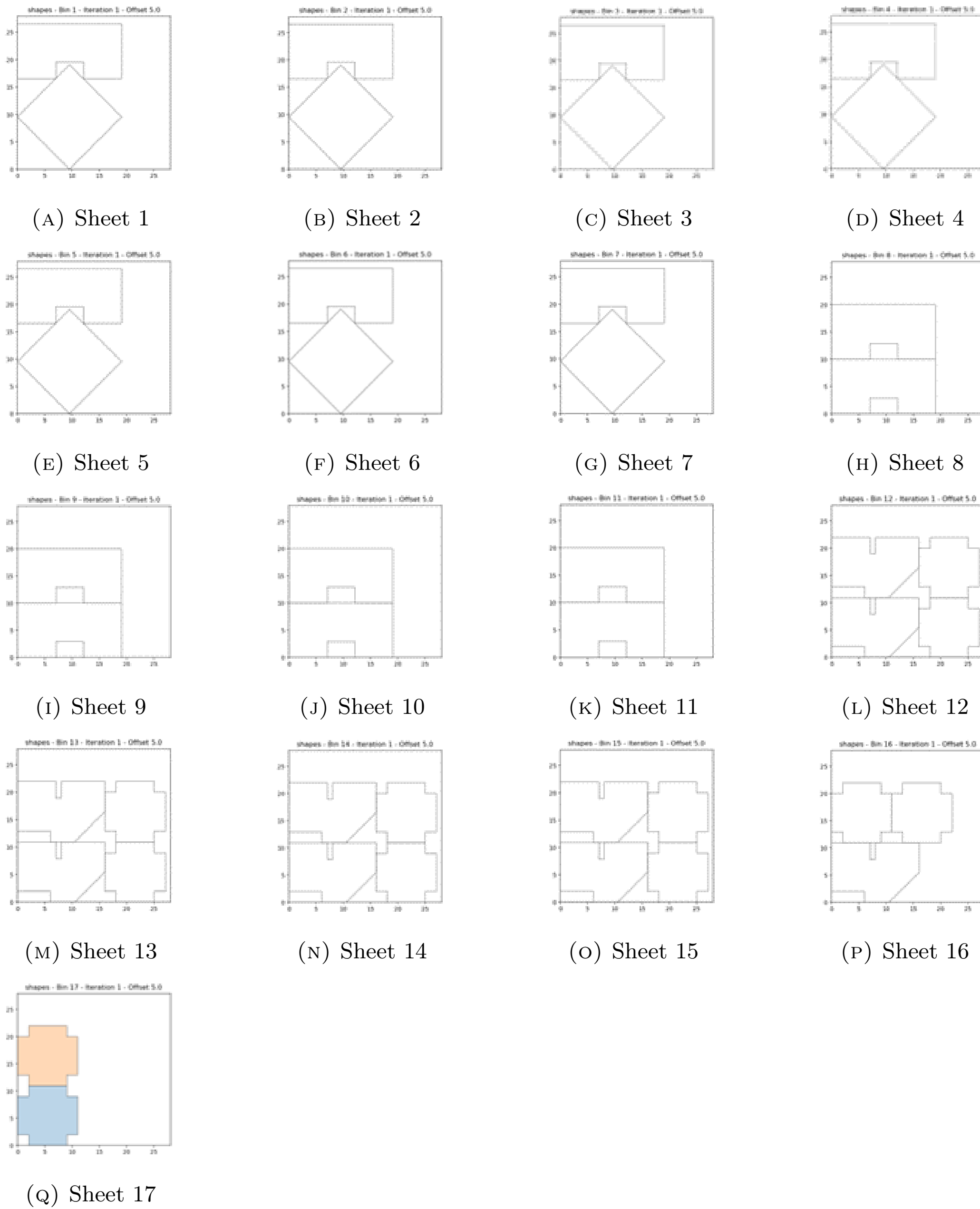
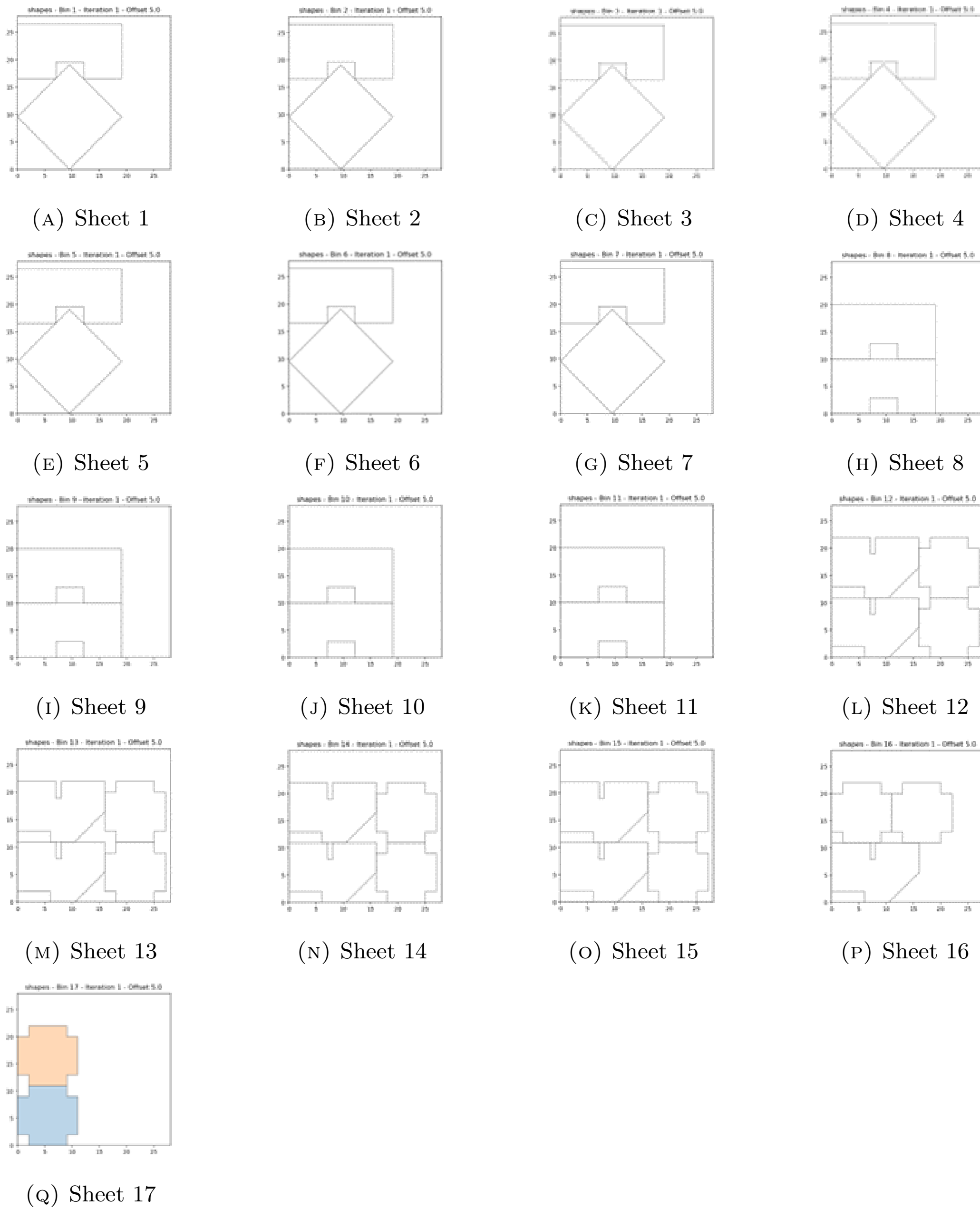


Figure 109: Nesting solution layout of shapes0 of the Nest-LB instances with product spacing.



(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4

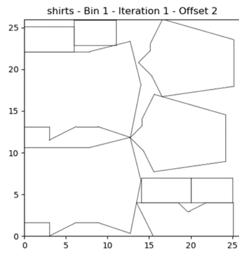
(E) Sheet 5 (F) Sheet 6 (G) Sheet 7 (H) Sheet 8

(I) Sheet 9 (J) Sheet 10 (K) Sheet 11 (L) Sheet 12

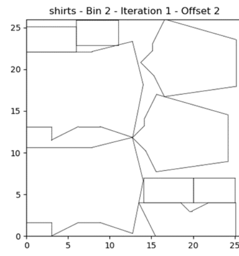
(M) Sheet 13 (N) Sheet 14 (O) Sheet 15 (P) Sheet 16

(Q) Sheet 17

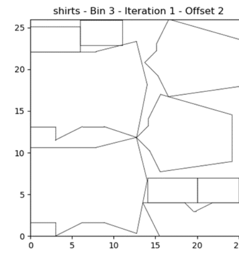
FIGURE 110: Nesting solution layout of shapes1 of the Nest-LB instances with product spacing.



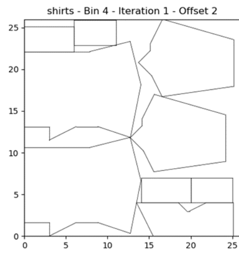
(A) Sheet 1



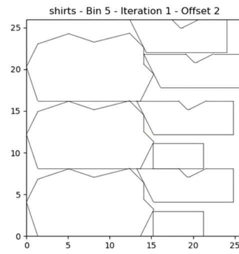
(B) Sheet 2



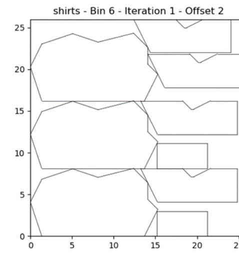
(C) Sheet 3



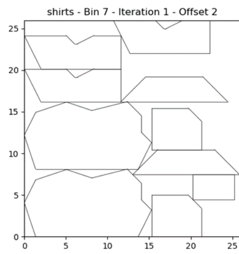
(D) Sheet 4



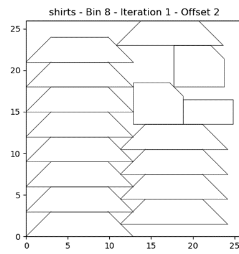
(E) Sheet 5



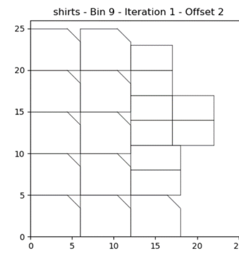
(F) Sheet 6



(G) Sheet 7

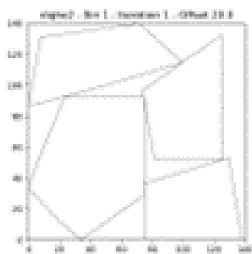


(H) Sheet 8

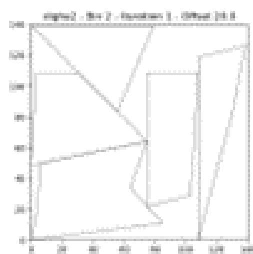


(I) Sheet 9

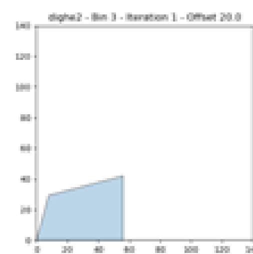
FIGURE 111: Nesting solution layout of shirts of the Nest-LB instances with product spacing.



(A) Sheet 1

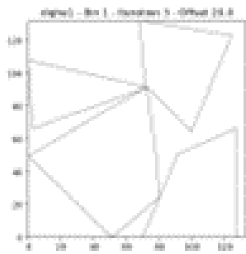


(B) Sheet 2

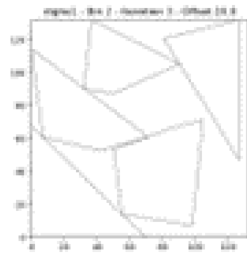


(C) Sheet 3

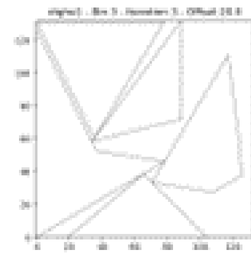
FIGURE 112: Nesting solution layout of dighe2 of the Nest-LB instances with product spacing.



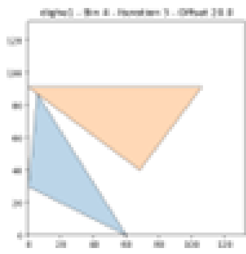
(A) Sheet 1



(B) Sheet 2

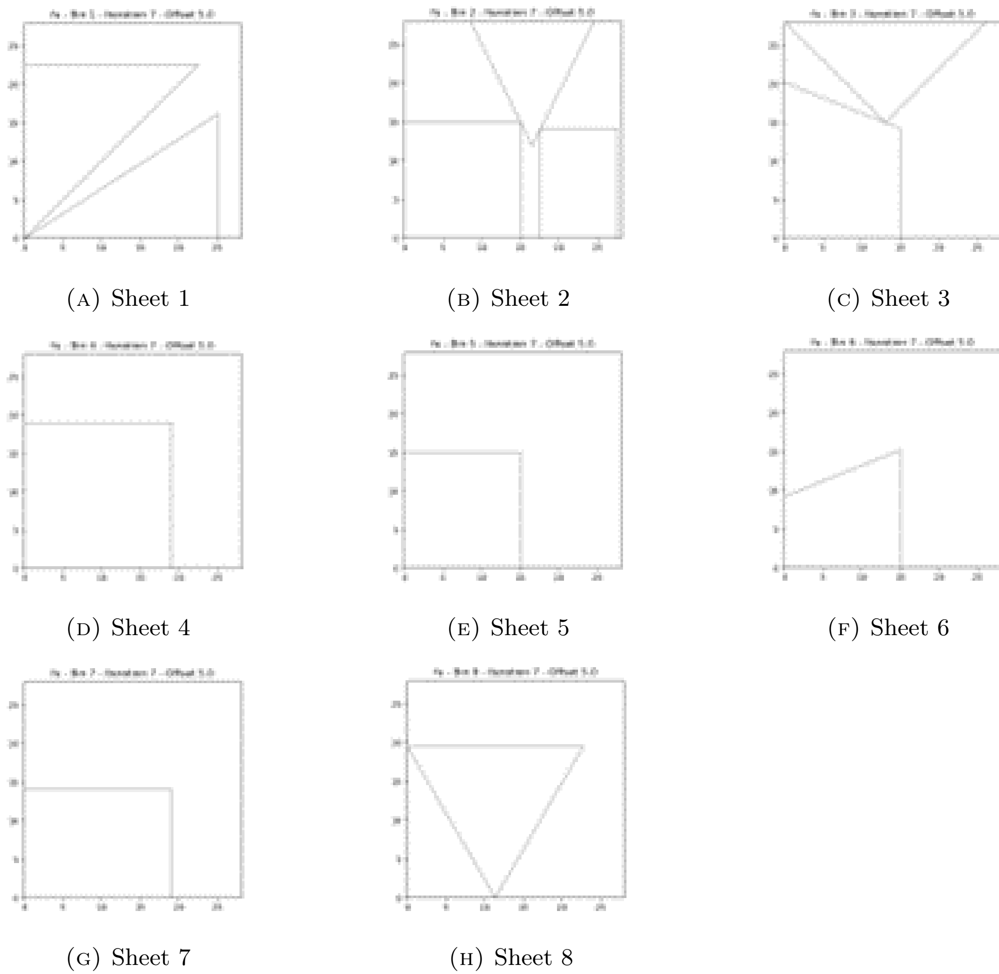


(C) Sheet 3



(D) Sheet 4

FIGURE 113: Nesting solution layout of dighe1 of the Nest-LB instances with product spacing.

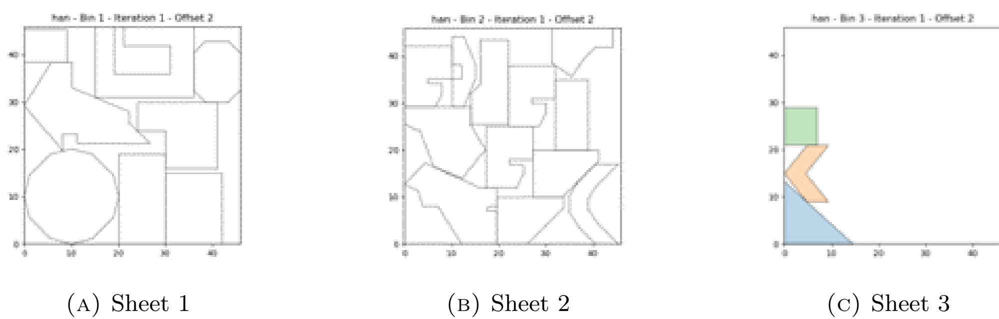


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

(D) Sheet 4 (E) Sheet 5 (F) Sheet 6

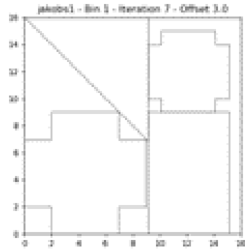
(G) Sheet 7 (H) Sheet 8

Figure 114: Nesting solution layout of fu of the Nest-LB instances with product spacing.

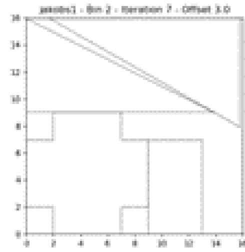


(A) Sheet 1 (B) Sheet 2 (C) Sheet 3

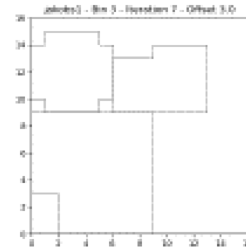
Figure 115: Nesting solution layout of han of the Nest-LB instances with product spacing.



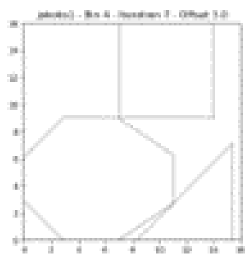
(A) Sheet 1



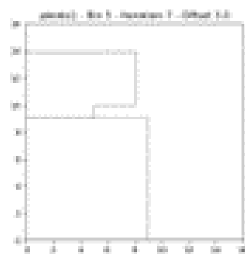
(B) Sheet 2



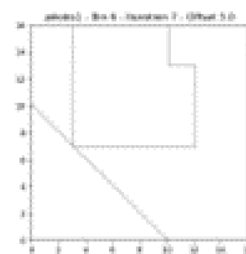
(C) Sheet 3



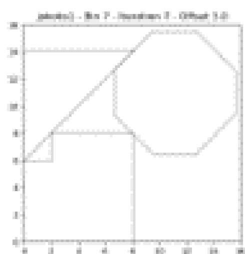
(D) Sheet 4



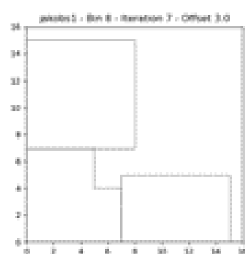
(E) Sheet 5



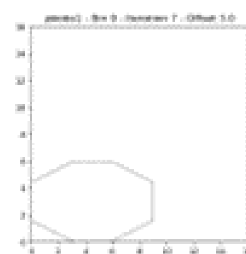
(F) Sheet 6



(G) Sheet 7

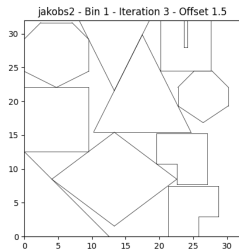


(H) Sheet 8

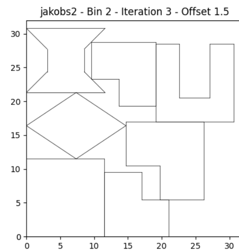


(I) Sheet 9

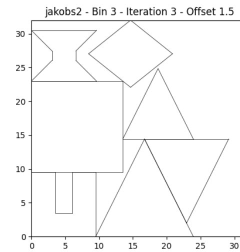
FIGURE 116: Nesting solution layout of jakobs1 of the Nest-LB instances with product spacing.



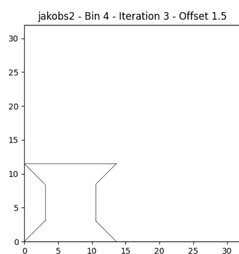
(A) Sheet 1



(B) Sheet 2

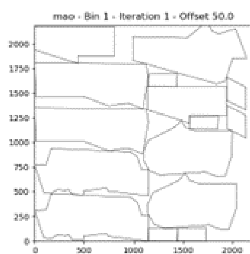


(C) Sheet 3

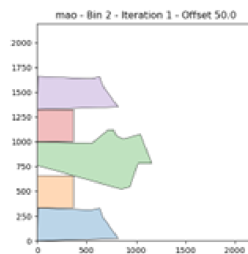


(D) Sheet 4

FIGURE 117: Nesting solution layout of jakobs2 of the Nest-LB instances with product spacing.

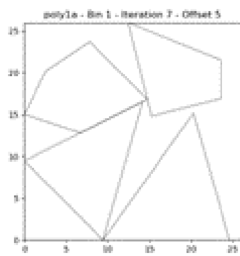


(A) Sheet 1

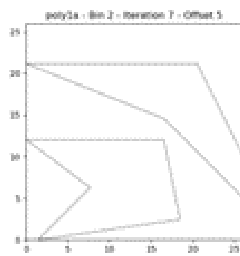


(B) Sheet 2

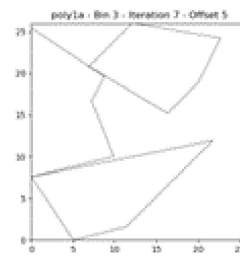
FIGURE 118: Nesting solution layout of mao of the Nest-LB instances with product spacing.



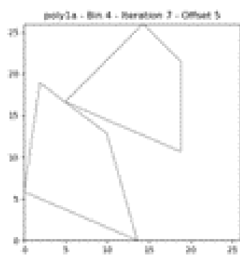
(A) Sheet 1



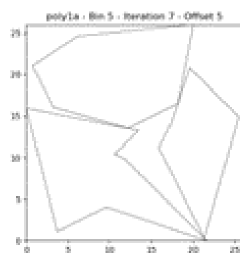
(B) Sheet 2



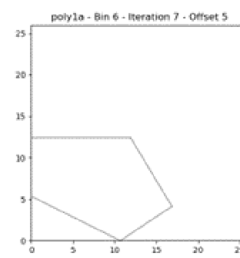
(C) Sheet 3



(D) Sheet 4

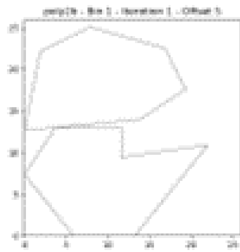


(E) Sheet 5

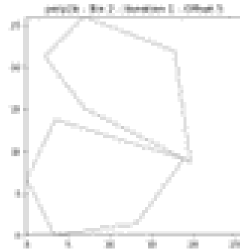


(F) Sheet 6

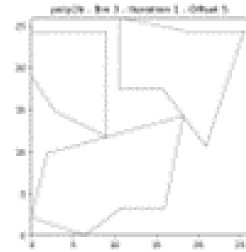
FIGURE 119: Nesting solution layout of poly1a of the Nest-LB instances with product spacing.



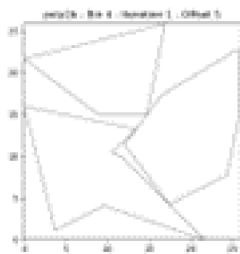
(A) Sheet 1



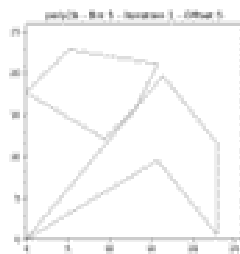
(B) Sheet 2



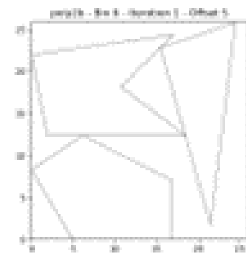
(C) Sheet 3



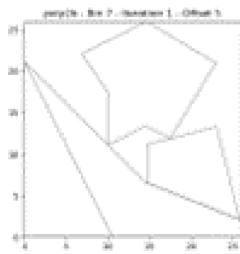
(D) Sheet 4



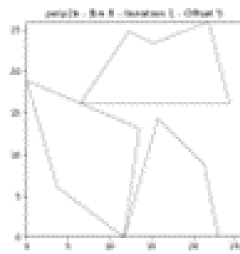
(E) Sheet 5



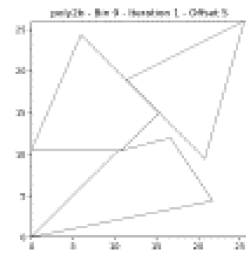
(F) Sheet 6



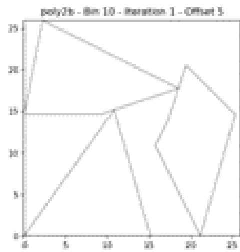
(G) Sheet 7



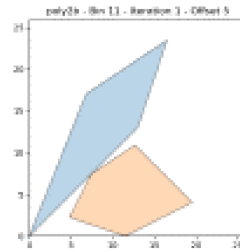
(H) Sheet 8



(I) Sheet 9

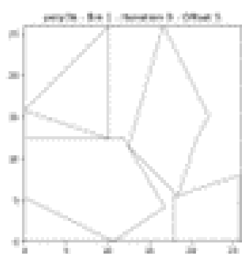


(J) Sheet 10

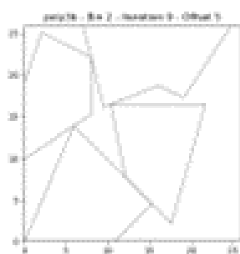


(K) Sheet 11

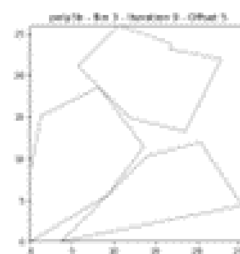
FIGURE 120: Nesting solution layout of poly2b of the Nest-LB instances with product spacing.



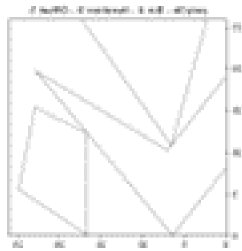
(A) Sheet 1



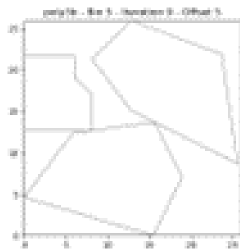
(B) Sheet 2



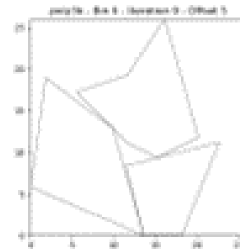
(C) Sheet 3



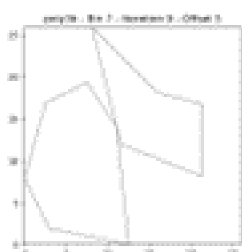
(D) Sheet 4



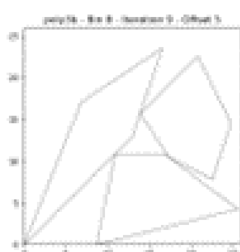
(E) Sheet 5



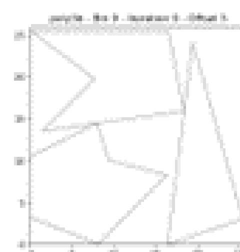
(F) Sheet 6



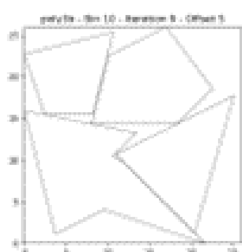
(G) Sheet 7



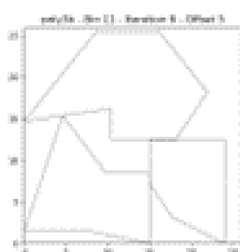
(H) Sheet 8



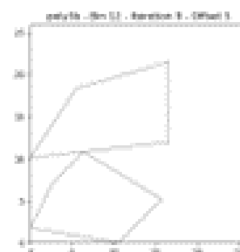
(I) Sheet 9



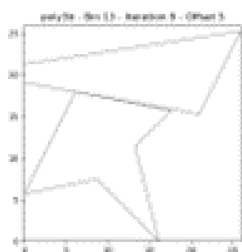
(J) Sheet 10



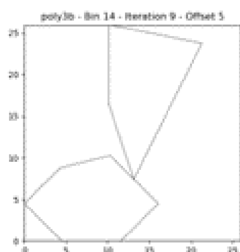
(K) Sheet 11



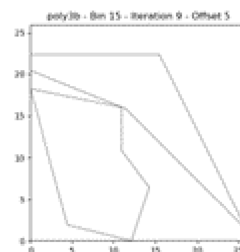
(L) Sheet 12



(M) Sheet 13



(N) Sheet 14



(O) Sheet 15

FIGURE 121: Nesting solution layout of poly3b of the Nest-LB instances with product spacing.

(A) Sheet 1 (B) Sheet 2 (C) Sheet 3 (D) Sheet 4

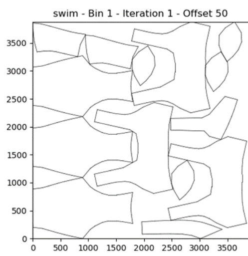
(E) Sheet 5 (F) Sheet 6 (G) Sheet 7 (H) Sheet 8

(I) Sheet 9 (J) Sheet 10 (K) Sheet 11 (L) Sheet 12

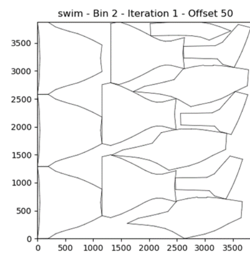
(M) Sheet 13 (N) Sheet 14 (O) Sheet 15 (P) Sheet 16

(Q) Sheet 17 (R) Sheet 18 (S) Sheet 19 (T) Sheet 20

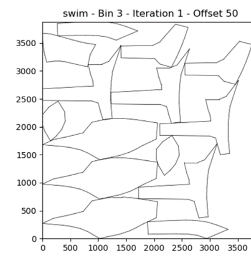
FIGURE 122: Nesting solution layout of poly4b of the Nest-LB instances with product spacing.



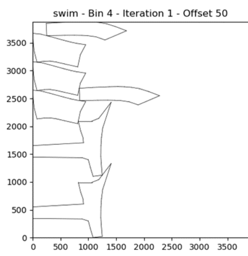
(A) Sheet 1



(B) Sheet 2



(C) Sheet 3



(D) Sheet 4

FIGURE 123: Nesting solution layout of swim of the Nest-LB instances with product spacing.