Journal of Computers, Mechanical and Management
Journal homepage: www.jcmm.co.in
e-ISSN: 3009-075X

Volume 2 Issue 3

Article Number: 23062

Optimizing Space Utilization In Container Packing: A Comparative Analysis of Packing Algorithms

C. Raghavendra Kamath *

Department of Mechanical and Industrial Engineering, Manipal Academy of Higher Eduction, Manipal, Karnataka, India 576104

Abstract

Container packing presents a complex optimization problem that seeks to efficiently pack diverse items into a fixed-size container. This study provides a comparative analysis of four algorithms – Greedy Shelf, Shelfing with Rotation, Shelfing with Search, and Guillotine Paper Cutting – investigating their proficiency in solving the container packing problem. Utilizing a set of four packages with differing dimensions and IDs, The study evaluated the performance of each algorithm. The results demonstrated that the Shelfing with Search algorithm outperformed its counterparts by yielding a stack height of 3019610 L units. Conversely, the Guillotine Paper Cutting algorithm performed poorly, with a stack height of 7537295 units. This research also explored the impact of different sorting methods on packing efficiency, revealing that sorting packages in descending order of height yields superior results. Consequently, this study provides an extensive evaluation of the various algorithms used for container packing, while suggesting promising directions for future research to enhance packing efficiency.

Keywords: Container Packing Problem; Optimization; Shelfing With Rotation; Shelfing With Search; Guillotine Algorithm

1 Introduction

Container packing is a significant and complex optimization problem that has substantial implications in logistics and supply chain management, necessitating efficient solutions [1, 2]. It involves packing cuboid-shaped containers into a given space, with the dual aims of maximizing space utilization and minimizing the number of containers used [3–5]. In real-world terms, effective container packing can translate into tangible benefits, including reducing logistics costs, decreasing environmental impact, and increasing productivity [6]. Examples of these benefits can be seen in everyday operations of shipping companies and warehouses, where optimal container packing can reduce loading and unloading times, minimize transportation costs, and prevent port congestion [7–9]. Furthermore, containerization, the use of standard intermodal containers, has dramatically transformed international commerce by enabling efficient long-distance goods transportation [10, 11]. Yet, due to product variability and limited space, optimal container packing remains a significant challenge [12]. The present work addresses the bin packing problem, wherein the goal is to pack either a single container as densely as possible or use as few containers as possible to pack all objects [13]. The study devises and compares various algorithms aimed at maximizing the use of available space within containers. The remainder of this article provides a detailed account of the problem statement, the data generation and algorithm development methodologies, and the results obtained from the evaluation of various algorithms. Each algorithm's effectiveness is assessed through a comparison of their pseudocodes. The study also present a thorough discussion of the findings, potential limitations, and suggestions for future research.

*Corresponding author: cr.kamath@manipal.edu
Received: 14 June 2023; **Revised:** 20 July 2023; **Accepted:** 29 July 2023; **Published:** 31 August 2023

DOI: 10.57159/gadl.jcmm.2.3.23062.

2 Related Work

Substantial research has been conducted on the container packing problem, resulting in the development of various algorithms, each offering unique strengths and limitations [14–16]. This section reviews some of these algorithms, as shown in Table 1, and their applications to the container packing problem, setting the stage for the current study's objectives and methods.

Table 1: Summary of Various Algorithms for the Container Packing Problem

Algorithm	Strengths	Weaknesses
Next-Fit Decreasing Algorithm	Simple and efficient to implement [14–16]	Does not always produce optimal results [17]
Best-Fit Algorithm	Yields more optimal results than the Next-Fit Algorithm [18]	Computationally complex [19]
Shelf Algorithms	Various iterations have been developed, offering flexible solutions [20]	Some versions may not optimally utilize the available space [21]
Genetic Algorithm	Uses evolutionary principles to find optimal packing solutions [22]	Complexity and computational requirements can be high [22]
Simulated Annealing Algorithm	Mimics the process of annealing in metallurgy to search for the best packing solution [23]	Could be computationally expensive and results are not always consistent [23]

As shown in Table 1, each algorithm presents unique advantages but also exhibits certain limitations. Therefore, the quest for comprehensive solutions to the container packing problem continues, emphasizing the importance and relevance of the present study.

3 Problem Statement

The focus of this study is to develop and compare algorithms for the optimal packing of one million cuboid-shaped packages into a container with a base area of 1000L by 1000L units, where L denotes the basic unit length. The packages must be packed following decreasing order of their ID, without any overlapping. Although packages can be oriented in any direction parallel to the x, y, or z axes (i.e., they can be rotated in multiples of 90 degrees), the constraint lies in maximizing the density of packing and minimizing the number of containers utilized. The problem statement includes several constraints and requirements to emulate real-world conditions. Although there is no limitation on the stack height, the base area of the stack must not exceed 1000 by 1000 units. Additionally, the packing order must be strictly maintained, and package dimensions are interchangeable, given the 90-degree rotation constraint around the x, y, or z axes. Backtracking is not permitted, eliminating the possibility of repacking for optimization once a package has been placed. The motivation behind this problem lies in its practical implications. Consider a scenario where a shipping company needs to transport a large number of packages of varying sizes across the globe. Inefficient packing could lead to higher costs, more containers used, and increased environmental impact due to the larger number of shipments. Hence, optimizing the packing process can provide significant economic and environmental benefits, making it a crucial concern for industries involved in shipping, transportation, and warehousing. This problem is not just an academic exercise, but it also addresses a key logistical challenge faced by industries globally. A successful solution could pave the way for the development of robust, general-purpose packing algorithms for a variety of applications, emphasizing the significance and relevance of the current study.

4 Methodology

The approach in the present study to solving the container packing problem involved two primary steps: data generation and algorithm development.

4.1 Data Generation

Python scripting language was used to generate random numbers within the range of 1 to 250. These random numbers represent the dimensions of the packages. The dimensions, along with the package ID, were stored in a CSV file for further processing.

4.2 Algorithm Development

The algorithm development phase involved the comparison and evaluation of four algorithms to determine their effectiveness in packing the cuboid-shaped packages into the container. The algorithms under investigation were the Basic Greedy Shelf algorithm, the Shelfing with Rotation algorithm, the Shelfing with Internal Search algorithm, and the Guillotine Paper Cutting algorithm. Unlike previous methods, our methodology differs in two significant aspects.

Firstly, the Python scripting language was leveraged for data generation, which helped in creating a large dataset of packages that align with the problem requirements. Secondly, based our evaluation of the algorithms on the height of the stack formed by packing the packages within the provided space, provided a more objective measure for comparing the effectiveness of the algorithms.

5 Algorithms

Four distinct algorithms were examined for resolving the container packing problem: the Basic Greedy Shelf (BGS) algorithm, the Shelfing with Rotation (SwR) algorithm, the Shelfing with Internal Search (SwIS) algorithm, and the Guillotine Paper Cutting (GPC) algorithm. Each algorithm is described below, accompanied by pseudocode and examples.

5.1 Basic Greedy Shelf Algorithm

The BGS algorithm, as a simple yet effective heuristic, commences the packing process from the origin and continues by adding packages along the x-axis (lengthwise). When the end of a shelf is reached, the algorithm moves up to the next level. This process continues until the height limit of the container is reached. The algorithm can be summarised as listing 1.

Listing 1: Pseudocode for Basic Greedy Shelf algorithm

```
def basic_greedy_shelf(packages):
cursor = (0,0,0) # initialize cursor at the origin
for p in packages:
if can_add_to_current_shelf(p, cursor):
add_to_shelf(p, cursor)
cursor = (cursor[0] + p.width, cursor[1], cursor[2])
else:
cursor = (0, get_max_height_of_previous_shelf(), cursor[2])
add_to_shelf(p, cursor)
```

5.2 Shelfing with Rotation Algorithm

The SwR algorithm extends the BGS algorithm by incorporating the rotation of packages, ensuring that the package's height does not exceed the level height or shelf height. The pseudocode for the SwR algorithm is given in Listing 2.

Listing 2: Pseudocode for Shelfing with Rotation algorithm

```
def shelfing_with_rotation(packages):
cursor = (0,0,0)
for p in packages:
if can_add_to_current_shelf(p, cursor):
add_to_shelf(p, cursor)
cursor = (cursor[0] + p.width, cursor[1], cursor[2])
else:
p.rotate_to_best_fit() # rotate package for optimal fit
if can_add_to_current_shelf(p, cursor):
add_to_shelf(p, cursor)
cursor = (cursor[0] + p.width, cursor[1], cursor[2])
else:
cursor = (0, get_max_height_of_previous_shelf(), cursor[2])
add_to_shelf(p, cursor)
```

5.3 Shelfing with Internal Search Algorithm

The SwIS algorithm further enhances the SwR algorithm by keeping track of packages packed in the z-direction. The algorithm scans for a gap in the current shelf where the package can fit, optimizing the utilization of space. The pseudocode for the SwIS algorithm is given in Listing 3.

5.4 Guillotine Paper Cutting Algorithm

The GPC algorithm adopts a recursive approach to divide the container space into filled and unfilled regions. It handles irregularly shaped packages by repeatedly dividing the container space, ensuring high space utilization. Despite its robustness, it can be computationally intensive, especially when dealing with a large number of packages. The pseudocode for the GPC algorithm is is given in Listing 4.

Listing 3: Pseudocode for Shelfing with Internal Search algorithm

```
def shelfing_with_internal_search(packages):
cursor = (0,0,0)
for p in packages:
if can_add_to_current_shelf(p, cursor):
add_to_shelf(p, cursor)
cursor = (cursor[0] + p.width, cursor[1], cursor[2])
else:
margin = find_margin_in_current_shelf(p)
if margin:
add_to_shelf_at_margin(p, margin)
else:
cursor = (0, get_max_height_of_previous_shelf(), cursor[2])
add_to_shelf(p, cursor)
```

Listing 4: Pseudocode for Guillotine Paper Cutting algorithm

```
def guillotine_paper_cutting(packages):
container = Container()
while packages:
p = select_largest_package(packages)
space = find_best_space(p, container)
if space:
place_in_space(p, space)
else:
p.rotate_to_best_fit()
space = find_best_space(p, container)
if space:
place_in_space(p, space)
else:
smaller_spaces = cut_space(space)
guillotine_paper_cutting(smaller_spaces)
return container
```

6 Data Generation and Algorithm Evaluation

6.1 Python code for data generation

The Python code below generates a list of a predefined number of cuboid-shaped packages. Each package is randomly assigned dimensions between 50L and 200L. These packages are represented as objects of the *Package* class, each with a length, width, height, and package ID. The *Package* class also includes a *rotate* method, which allows for the rotation of the package by 90 degrees, thus facilitating packing in various orientations.

```
import random
class Package:
def __init__(self, length, width, height, package_id):
self.length = length
self.width = width
```

```
self.height = height
self.package_id = package_id
def rotate(self):
# Rotate package by 90 degrees
self.length, self.width = self.width, self.length
def __str__(self):
return f'Package {self.package_id}: {self.length}L x {self.width}L x {self.height}L'
def generate_data(num_packages):
packages = []
for i in range(num_packages):
length = random.randint(50, 200)
width = random.randint(50, 200)
height = random.randint(50, 200)
packages.append(Package(length, width, height, i+1))
return packages
```

6.2 Python code for algorithm evaluation

The provided Python code can be used to evaluate the height of the stack created by various packing algorithms for a given set of packages. The *pack_packages* function is designed to accept a list of packages and a packing algorithm, subsequently returning a list of packed packages along with their respective coordinates. The *calculate_stack_height* function takes this list of packed packages and computes the maximum height of the stack. This combined usage enables us to evaluate the stack height resultant from our algorithm choices.

```
def pack_packages(packages, algorithm):
# Implement the selected packing algorithm here
# and return a list of packed packages with their coordinates
def calculate_stack_height(packed_packages):
max_height = 0
for package in packed_packages:
if package[3] > max_height:
max_height = package[3]
return max_height

# Example usage
packages = [(1, 200, 100, 50), (2, 100, 100, 50), (3, 50, 50, 50), (4, 50, 50, 50)]
packed_packages = pack_packages(packages, 'Shelfing with Rotation')
stack_height = calculate_stack_height(packed_packages)
print("Stack height:", stack_height)
```

7 Results and Discussion

This study compared the performance of four algorithms for solving the container packing problem: the Greedy Shelf algorithm, Shelfing with Rotation, Shelfing with Search, and the Guillotine Paper Cutting algorithm. The efficacy of each algorithm was gauged on the basis of the height of the resulting stack. Table 2 provides a comparative analysis of the effects of different sorting methods on the input packages.

Table 2: Comparison of various sorting methods and their effects

Sorting Method	Area Usage %	Did Not Fit
Height	90	NA
Width	88	40x50, 40x50, 40x50
Random	66	100x200, 250x80, 100x200, 40x50

In the case where packages were organized in descending order based on their height, area utilization was roughly 90% of the total available area. However, random arrangement led to a decrease in area utilization, amounting to around 65% of the total area. Width-based sorting resulted in 88% area utilization, with some packages not fitting into the area. Among the algorithms compared, the Greedy Shelf algorithm yielded a stack of height 5270836 L units, while Shelfing with Rotation and Shelfing with Search resulted in stacks of height 3929432 L and 3019610 L units, respectively.

Notably, the Guillotine Paper Cutting algorithm produced the highest stack, with a height of 7537295 L units. From these observations, it is clear that the Shelfing with Search algorithm outperforms both the Greedy Shelf algorithm and the Shelfing with Rotation algorithm. It resulted in approximately a 40% reduction in stack height compared to the former two. Contrary to expectations, the Guillotine Paper Cutting algorithm showed inferior performance when compared to the other three algorithms. Figures 1 and 2 illustrate the placement of packages for the cases of random order and descending order based on height, respectively. The former case achieved roughly 65% area utilization, while the latter resulted in 90% utilization. From these findings, it can be concluded that the Shelfing with Search algorithm is the most effective solution for the container packing problem among the tested algorithms. Future studies could focus on exploring variations of the Shelfing with Search algorithm, such as incorporating different heuristics or optimizing for other metrics. Figures 1, 2, and 3 along with Table 2 serve as a strong visual support to this discussion, illustrating the efficacy of various sorting methods and algorithms.

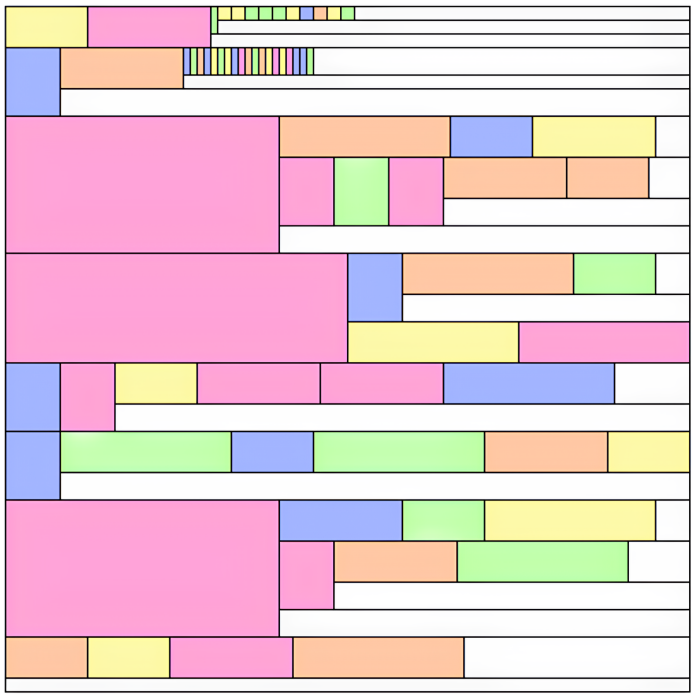

Figure 1: Random order placement: An illustration of package arrangement.

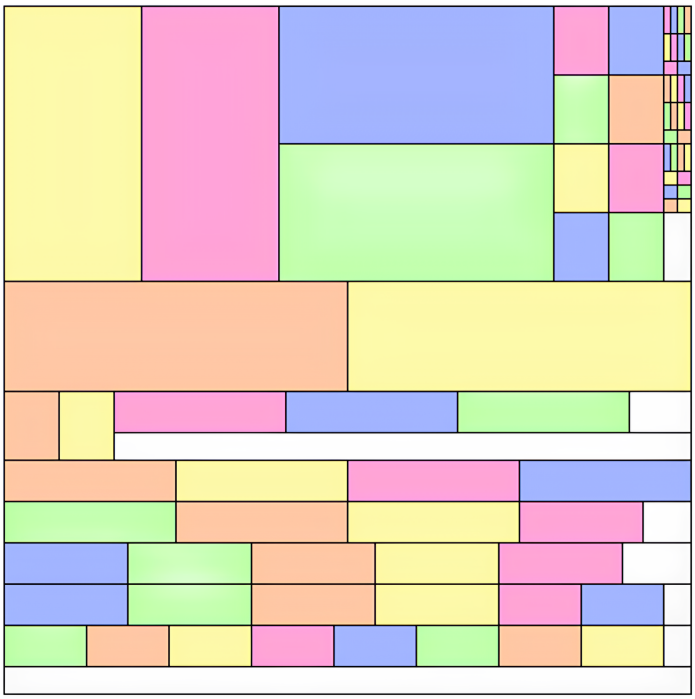

Figure 2: Decreasing height order placement: An effective strategy for container packing

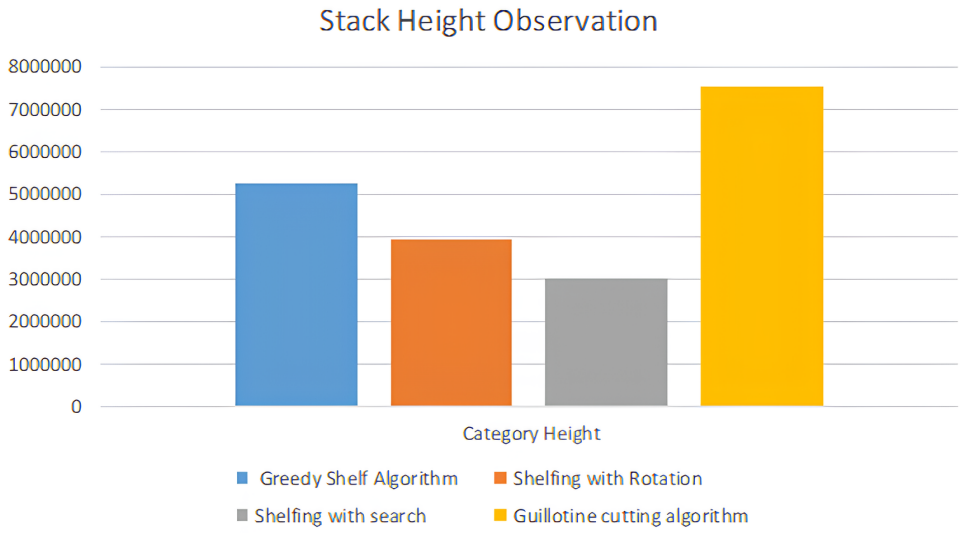

Figure 3: Comparison of stack heights: Observing the efficacy of different algorithms.

8 Conclusion

This research set out to explore various algorithms to tackle the container packing problem, a complex problem that involves packing cuboid-shaped containers under a variety of constraints. Three algorithms – Shelfing with Rotation, Shelfing with Internal Search, and Guillotine Paper Cutting – were investigated in this study, their performance evaluated by packing containers of randomly generated dimensions into a container of fixed size. The results of the study demonstrate that the Shelfing with Internal Search algorithm performed the most effectively, resulting in the lowest average height of the packed containers. Shelfing with Rotation also showed commendable performance, while Guillotine Paper Cutting, despite its common use, had the highest average height and fell short when compared to the other two algorithms. The implications of this research extend to the shipping industry, where efficient use of container space has direct consequences for cost-effectiveness. By offering insights into the efficacy of various algorithms in solving the container packing problem, this study stands to assist shipping companies in optimizing their packing processes. However, certain limitations do apply to this study, notably the use of randomly generated package dimensions and a fixed container size, which may not accurately reflect the complexity of real-world scenarios. For future research, it would be beneficial to explore the performance of these algorithms under different constraints and with a broader spectrum of package sizes and shapes. Additionally, advancements in technology offer intriguing possibilities for further investigation. Specifically, the exploration of machine learning algorithms to solve the container packing problem may yield promising results and provide a more nuanced understanding of this problem.

Declaration of Competing Interests

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding Declaration

This research did not receive any grants from governmental, private, or nonprofit funding bodies.

Author Contribution

C. Raghavendra Kamath: Conceptualization, Investigation, Methodology, Formal analysis, Data curation, Software, Writing - original draft, Writing - review and editing

References

[1] G. Yaskov, T. Romanova, I. Litvinchev, and S. Shekhovtsov, "Optimal packing problems: from knapsack problem to open dimension problem," in *International Conference on Intelligent Computing & Optimization*, pp. 671–678, Springer, 2019.

[2] J.-L. Lin, C.-H. Chang, and J.-Y. Yang, "A study of optimal system for multiple-constraint multiple-container packing problems," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pp. 1200–1210, Springer, 2006.

[3] Y.-B. Shin and E. Kita, "Solving two-dimensional packing problem using particle swarm optimization," *Computer Assisted Methods in Engineering and Science*, vol. 19, no. 3, pp. 241–255, 2017.

[4] T. G. Crainic, G. Perboli, and R. Tadei, "Extreme point-based heuristics for three-dimensional bin packing," *Informs Journal on computing*, vol. 20, no. 3, pp. 368–384, 2008.

[5] S. Erbayrak, V. Özkır, and U. M. Yıldırım, "Multi-objective 3d bin packing problem with load balance and product family concerns," *Computers & Industrial Engineering*, vol. 159, p. 107518, 2021.

[6] H. Pålsson, *Packaging Logistics: Understanding and managing the economic and environmental impacts of packaging in supply chains*. Kogan Page Publishers, 2018.

[7] C. A. Vega-Mejía, J. R. Montoya-Torres, and S. M. Islam, "Consideration of triple bottom line objectives for sustainability in the optimization of vehicle routing and loading operations: a systematic literature review," *Annals of Operations Research*, vol. 273, pp. 311–375, 2019.

[8] C. Bierwirth and F. Meisel, "A follow-up survey of berth allocation and quay crane scheduling problems in container terminals," *European Journal of Operational Research*, vol. 244, no. 3, pp. 675–689, 2015.

[9] N. Tsolakis, D. Zissis, S. Papaefthimiou, and N. Korfiatis, "Towards ai driven environmental sustainability: an application of automated logistics in container port terminals," *International Journal of Production Research*, vol. 60, no. 14, pp. 4508–4528, 2022.

[10] E. Sdoukopoulos and M. Boile, "Port-hinterland concept evolution: A critical review," *Journal of Transport Geography*, vol. 86, p. 102775, 2020.

[11] S. Nurosidah, "The shift of containerisation influence: 50-year logistics innovation in international business," *The Business & Management Review*, vol. 8, no. 4, p. 93, 2017.

[12] L. Kroon and G. Vrijens, "Returnable containers: an example of reverse logistics," *International journal of physical distribution & logistics management*, vol. 25, no. 2, pp. 56–68, 1995.

[13] M. Hifi, R. M'hallah, *et al.*, "A literature review on circle and sphere packing problems: Models and methodologies," *Advances in Operations Research*, vol. 2009, 2009.

[14] S. Kumaraswamy and M. K. Nair, "Bin packing algorithms for virtual machine placement in cloud computing: a review," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 1, p. 512, 2019.

[15] A. Lodi, S. Martello, M. Monaci, and D. Vigo, "Two-dimensional bin packing problems," *Paradigms of combinatorial optimization: Problems and new approaches*, pp. 107–129, 2014.

[16] A. Lodi, S. Martello, and D. Vigo, "Recent advances on two-dimensional bin packing problems," *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 379–396, 2002.

[17] D. Pisinger, "Heuristics for the container loading problem," *European journal of operational research*, vol. 141, no. 2, pp. 382–392, 2002.

[18] S. Sweep, "Three dimensional bin-packing issues and solutions," 2003.

[19] P. Ramanan, "Average-case analysis of the smart next fit algorithm," *Information processing letters*, vol. 31, no. 5, pp. 221–225, 1989.

[20] A. Lodi, S. Martello, and D. Vigo, "Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems," *Informs journal on computing*, vol. 11, no. 4, pp. 345–357, 1999.

[21] A. Lodi, S. Martello, and D. Vigo, "Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem," *Meta-heuristics: advances and trends in local search paradigms for optimization*, pp. 125–139, 1999.

[22] B. S. Baker and J. S. Schwarz, "Shelf algorithms for two-dimensional packing problems," *SIAM Journal on Computing*, vol. 12, no. 3, pp. 508–525, 1983.

[23] K. A. Dowsland, "Some experiments with simulated annealing techniques for packing problems," *European Journal of Operational Research*, vol. 68, no. 3, pp. 389–399, 1993.