Comparative Analysis of Algorithms for Cargo Container Space Optimization

1st Mrs. Lakshmi Gadhikar
Dept. of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Navi Mumbai, India
lakshmi.gadhikar@fcrit.ac.in

2nd Sanskar Bomble
Dept. of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Navi Mumbai, India
sanskarbomble013@gmail.com

3rd Swayam Chatur
Dept. of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Navi Mumbai, India
swayamchatur23@gmail.com

4th Sarvesh Karekar
Dept. of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Navi Mumbai, India
snkarekar6190@gmail.com

5th Sarvesh Khamkar
Dept. of Information Technology
Fr. Conceicao Rodrigues Institute of Technology
Navi Mumbai, India
sarveshkhamkar321@gmail.com

***Abstract*—This work delves into optimizing container space utilization in contemporary logistics and warehousing by addressing the intricate challenge of efficiently fitting various tire types into 2D and 3D containers. In our work we have used cutting-edge algorithms such as Next Fit, Best Fit Decreasing, Genetic Algorithm, Dynamic Programming, and Ant Colony Optimization, the study aims to maximize container space utilization while offering insights into tire arrangement and container occupancy percentages. Our evaluation showed significant differences in algorithm performance regarding time and space complexity. Next Fit is simple but can lead to suboptimal space usage. Best Fit Decreasing optimizes space better but demands more computation. Genetic Algorithm and Dynamic Programming are competitive but resource-intensive. Ant Colony Optimization excels in container space optimization with a unique, nature-inspired approach. This study serves as a valuable reference for professionals and researchers tackling complex packing challenges in logistics and warehousing, fostering advancements in efficiency.**

***Keywords:Tire Packing, Space Utilization, Container Optimization, Algorithm Comparison, Genetic Algorithm, Dynamic Programming, Ant Colony Optimization, Next Fit, Best Fit Decreasing*.**

I. INTRODUCTION

Loading tires efficiently within cargo containers is a critical aspect of modern logistics and supply chain management. The process involves strategically arranging tires of varying sizes and quantities to make optimal use of available container space. Key factors influencing this loading process include tire dimensions, loadability calculations, and order-specific requirements. However, the challenge lies in the natural variation of real-world loading scenarios, where discrepancies between theoretical loadability and actual loading often result in inefficient container space utilisation. Addressing these concerns is crucial to overcoming the drawbacks associated with suboptimal tire loading practices and achieving higher levels of container space efficiency.

Statistics from various studies emphasize the magnitude of the problem. An analysis of container loads imported into the United States in 2018 revealed an average container utilization rate of just 65% [1]. This means that, on average, containers were only slightly over half full. This underutilization translates into significant financial losses, with the potential for billions of dollars in saved freight fees if container capacity could be increased to 85% or 90% .

The causes of inefficient container loading are multifaceted. Inadequate container space optimization leads to higher shipping costs, longer lead times, and increased risk of damage to goods. Excessive packaging materials are required to secure improperly loaded cargo, leading to additional waste generation [2][3]. Inaccurate customs declarations and increased administrative tasks due to extra containers also inflate operational expenses [4][5]. To address these challenges and create a more sustainable and cost-effective solution, this work aims to develop an innovative optimization tool [6]. This work enables efficient loading, maximizes space utilization, reduces costs, and minimizes environmental impact, ultimately revolutionizing tire logistics and contributing to a more sustainable and efficient supply chain.

II. OVERVIEW

Loadability calculation for tire packing in a container is about efficiently placing tires of different sizes, where each size has a load factor determining how well it can be packed. This factor is multiplied by a constant to assess how many of a particular size can fit, and the sum of load factors for all tire sizes is compared to the container's predefined capacity. If it doesn't exceed the capacity, more tires can be accommodated.

979-8-3503-5981-7/23/$31.00 ©2023 IEEE

Real-world loading often leads to underfilled or overfilled containers, causing shipping cost increases or space inefficiencies. Manual loading decisions introduce operational risks, and dispatch planning ignores actual product assortments. Addressing these challenges is crucial for optimizing tire loading, improving space use, reducing costs, and enhancing operational efficiency in cargo container logistics.

III. Related Work

The majority of researchers have primarily directed their attention towards the management of containers within port terminals and warehouses. To the best of our knowledge, only a few people have ventured into the challenges of efficiently packing goods into containers. This section provides a concise overview of previous related works concerning the problem of allocating space within containers, specifically in the context of the loading process.

A. Solution Approaches for space allocation techniques:

The evolution of container space allocation techniques has been significantly influenced by a series of seminal research papers. The foundational work laid down by (Chien et al., 2009) introduced an approach employing three-dimensional cutting techniques and knapsack problem solutions to efficiently determine container-loading patterns, focusing on maximizing space utilization for rectangular boxes within a single container [7]. Then (Bortfeldt & Wäscher, 2013) contributed a comprehensive review that critically assessed the practical relevance of container loading research, identifying key factors and evaluating their representation in solution methods [8]. Furthermore (Patil & Patil, 2016) introduced optimization algorithms, including LAFF and LAFF with Weight consideration, optimized to maximize cargo space utilization [9]. Later (De et al., 2018) presented a mixed-integer linear programming model incorporating vessel waiting times, fuel costs, and operational durations for enhanced port operations [10]. Addressing the multi container loading problem in (Alonso et al., 2019) proposed mathematical models to solve complex loading problems with practical constraints [11]. Transitioning further (Zhou et al., 2020) introduced rule-based optimization and Particle Swarm Optimization for space allocation, emphasizing the importance of reshuffling activities [12]. Finally, (Czerniachowska & Lutosławski, 2021) proposed a dynamic programming approach for solving the retail shelf-space allocation problem, demonstrating dynamic programming's versatility in addressing practical allocation challenges [13]. These papers collectively illustrate the progressive advancement of container space allocation methodologies, making significant contributions to logistics efficiency, resource utilization, and overall operational efficacy.

B. Research Gaps

The reviewed papers shed light on significant advancements in container space allocation methodologies, yet each study also unveils potential research gaps that warrant attention. In (Chien et al., 2009), while their algorithm offers efficient container-loading patterns, its mixed integer programming formulation may limit scalability and efficiency. Moreover, the algorithm's reliance on a predefined utilization threshold could lead to suboptimal solutions. Similarly, (Patil & Patil, 2016) provide optimization algorithms for cargo space utilization, yet the absence of comprehensive performance comparison and limited experimental evaluation hinders a broader understanding of their effectiveness. The (De et al.,2018) paper introduces a chemical reaction optimization algorithm for berth allocation, but the lack of detailed limitations and performance comparisons with other algorithms restricts its applicability. (Wang et al., 2019) offer an integer linear programming model, yet its limitations in handling varying quantities of goods and potential incompleteness in considering container types could restrict its applicability [14]. Meanwhile, (Olsson et al. 2020) acknowledge potential constraints not accounted for in their algorithm and the absence of empirical data supporting their claims of benefits [15]. Addressing these gaps is crucial for refining container space allocation strategies.

IV. Implementation Details

Our work focuses on efficiently packing tires into 2D containers to maximize space usage. We employ a variety of algorithms, starting with preprocessing tire data for dimension considerations, then using Next Fit for simplicity, Best Fit Decreasing for optimization, and more advanced methods like Genetic Algorithms, Dynamic Programming, and Ant Colony Optimization. We provide detailed tire arrangements in containers and calculate space utilization percentages. Through experimentation, we aim to assess algorithm performance and their applicability in practical tire storage and inventory management scenarios.

A. BFD(Best Fit Decreasing)

The "Best Fit Decreasing" algorithm is a bin packing algorithm used in space optimization problems, where the goal is to efficiently allocate items of different sizes into containers or bins. The Best Fit Decreasing (BFD) algorithm can be applied to optimize the arrangement of tires in a container while maximizing space utilization. In the context of tire loading in cargo containers, the algorithm works as follows:

- Sort the Tires: Begin by sorting the tires in decreasing order of size, typically based on a measure of size like outer diameter or width.
- Initialize Bins: Create an empty list of bins, where each bin represents a potential arrangement of tires within the container. Initially, the list is empty, and you'll place tires into bins as needed.
- Packing Process: For each tire in the sorted list, attempt to fit it into one of the existing bins. Iterate through the list of bins and find the bin where the tire can fit with the least leftover space. This involves finding the bin that can accommodate the tire while minimizing the unused space inside the bin. Place the tire in the chosen bin.

- Repeat: Continue this process for all tires in the sorted list.
- Output: The final arrangement of tires in the bins represents an optimized packing arrangement. To calculate space utilization, compare the used space to the total space available in the container. This can be expressed as a percentage. Pseudo Code for the algorithm is shown in Table 1.

TABLE I. PSEUDOCODE OF BFD

```
Algorithm BFD
FUNCTION bestFitDecreasing(tires, containerSize):
  sortedTires = Sort tires in decreasing order
  bins = Initialize a list of bins, each with a remaining space of containerSize
  FOR EACH tire IN sortedTires:
    bestFitBin = -1
    minRemainingSpace = containerSize + 1
    FOR i FROM 0 TO LENGTH(bins) - 1:
      remainingSpace = Remaining space in bins[i]
      IF tire <= remainingSpace AND remainingSpace < minRemainingSpace:
        bestFitBin = i
        minRemainingSpace = remainingSpace
    IF bestFitBin != -1:
      Place tire in bins[bestFitBin]
      Reduce bins[bestFitBin]'s remaining space by tire
    ELSE:
      Create a new bin with remaining space (containerSize - tire)
      Add tire to the new bin
  packingArrangement = Create an empty list
  FOR EACH bin IN bins:
    IF bin is not empty:
      Add bin contents to packingArrangement
  RETURN packingArrangement
```

B. NextFit Algorithm

The Next-Fit algorithm is a simple and intuitive bin packing algorithm used in space optimization problems. It is particularly useful when items are arriving one at a time and need to be placed into containers. The algorithm is based on two one-dimensional bin-packing algorithms. In the context of tire loading in cargo containers. Pseudo Code for the algorithm is shown in Table 2.

- Create empty container with size.
- Sort tires by size.
- Initiate current bin.
- Check, place tires, update space.
- Continue until all tires placed.
- Calculate space utilization.
- Optimize tire arrangement if needed

TABLE II. PSEUDOCODE FOR NEXTFIT ALGORITHM

```
Algorithm NextFit
FUNCTION nextFit(tires, containerSize):
bins = List of bins with one bin of size containerSize
FOR EACH tire IN tires:
placed = false
FOR EACH bin IN bins:
remainingSpace = Remaining space in bin
binContents = Contents of bin
IF tire <= remainingSpace:

    Place tire in bin
    Reduce remainingSpace by tire
    placed = true
    BREAK

IF NOT placed:
Create a new bin for tire with remaining space (containerSize - tire)
packingArrangement = List of non-empty bins
RETURN packingArrangement
```

C. Genetic Algorithms

Genetic Algorithm are heuristic search algorithms inspired by the principles of natural selection and genetics. In the context of tire loading in cargo containers, the algorithm works as follows:

- Chromosome Representation: Represent tires as genes in a chromosome for tire arrangement.
- **Initialization:** Generate an initial population of chromosomes randomly or using a helper function for tire placement.
- **Fitness Function:** Define a fitness function to measure space utilization, e.g., container space occupancy percentage.
- **Selection:** Choose parent chromosomes for the next generation, favoring those with higher fitness values using rank-based selection.
- **Crossover (Recombination):** Combine genes from two parents to create offspring chromosomes, inheriting genes from both parents.
- **Mutation:** Introduce small random changes in the offspring's genes to explore the solution space more effectively.
- **Replacement:** Form a new population by combining parent and offspring chromosomes using a strategy like generational replacement.
- **Termination Criteria:** Define stopping conditions, such as a maximum number of generations or achieving a satisfactory solution.

- **Result Extraction:** Extract the best chromosome found as the solution after the algorithm terminates.
- **Parameter Tuning:** Experiment with different settings like population size, crossover rate, mutation rate, and selection mechanisms to optimize the algorithm's performance.

D. Ant Colony Optimization

This approach is particularly well- suited for diving optimization problems. ACO algorithms emulate specific actions and evolutionary traits set up in the natural world. In the context of tire loading in cargo containers, the algorithm works as follows:

- Problem Representation: Nodes (States): Represent tires. Edges (Transitions): Indicate the feasibility of placing one tire after another based on dimensions.
- Objective Function: Define fitness function to maximize space utilization.
- Pheromone Update: In it pheromone matrix with positive values. Ants construct solutions using pheromones and heuristics. Update pheromones based on solution quality.
- Ant Construction: Ants iteratively place tires using pheromones and heuristics.
- Termination: Define stopping criteria, e.g., iterations or quality threshold.
- Best Solution: Track the best-found solution.
- Output: Display the best arrangement maximizing space utilization. Pseudo Code for the algorithm is shown in Table 4.

TABLE III. PSEUDOCODE FOR ANT HILL OPTIMIZATION

Algorithm Ant Hill Optimization

```
FUNCTION initializePheromone(num_tires):
   Initialize pheromone values between tires
FUNCTION computeProbability(current_tire, next_tire, heuristic):
   Calculate probability for moving from current_tire to next_tire
FUNCTION computeFitness(arrangement, tire_widths, container_width):
   Calculate fitness of an arrangement based on used space
FUNCTION main():
   Set container_width and num_tires
   Initialize pheromone values
   Initialize tire_widths randomly
   FOR each iteration:
      Initialize ant_arrangements
      FOR each ant:
         Initialize tire_used
         FOR each tire:
            Select the next tire based on probability
         Record the arrangement
      # Update pheromone levels and evaporation (TODO)
   # Find the best arrangement (TODO)
   PRINT "Best arrangement:", best_arrangement
   PRINT "Fitness:", computeFitness(best_arrangement, tire_widths, container_width)
   RETURN 0
```

E. Dynamic Programming:

The dynamic programming approach can lead to efficient utilization of space within the container. By considering various combinations and sub problems, it helps in identifying the arrangement that minimizes empty spaces and maximizes cargo capacity. In the context of tire loading in cargo containers, the algorithm works as follows:

- **Problem Formulation**: Formulate the problem concisely by representing states as available container space and decisions as tire placement. The goal is to maximize container space utilization.
- **State Representation**: Condense state information to capture relevant details, like remaining container space.
- **Recurrence Relation**: Express the relationship between optimal space utilization for larger and smaller sub problems.
- **Initialization**: Set base case values in the DP table, e.g., for an empty container.
- **Bottom-Up Approach**: Employ a bottom-up strategy to populate the DP table with optimal solutions, starting from smaller sub problems and progressing to the original problem.
- **Backtracking**: Use backtracking to reconstruct the optimal tire arrangement once the DP table is complete. Pseudo Code for the algorithm is shown in Table 5.

TABLE IV. PSEUDOCODE FOR DYNAMIC PROGRAMMING

Algorithm Dynamic Programming

```
FUNCTION placeTire(tire):
    FOR i FROM 0 TO rows - 1:
        FOR j FROM 0 TO cols - 1:
            IF canPlaceTire(tire, i, j):
                placeTireAtPosition(tire, i, j)
                RETURN TRUE
    RETURN FALSE
  FUNCTION canPlaceTire(tire, row, col):
    FOR i FROM 0 TO tire.length - 1:
        FOR j FROM 0 TO tire.length - 1:
            r = row + i
            c = col + j
            IF r >= rows OR c >= cols OR container[r][c] == 1:
                RETURN FALSE
    RETURN TRUE
  FUNCTION placeTireAtPosition(tire, row, col):
    FOR i FROM 0 TO tire.length - 1:
        FOR j FROM 0 TO tire.length - 1:
            container[row + i][col + j] = 1
    filledSpace += tire.length * tire.length
  FUNCTION isOccupied(row, col):
    RETURN container[row][col] == 1
  FUNCTION getPercentageFilled():
    RETURN (filledSpace / (rows * cols)) * 100.0
FUNCTION compareTires(a, b):
  RETURN a.length > b.length
```

In this dynamic programming algorithm, we observed various scenarios to test a code's ability to strategically arrange tires within containers of varying dimensions. Each case offers insights into how well the code performs in handling both common and exceptional packing challenges, shedding light on the art and science of optimizing cargo loading operations.

- Case 1: Basic Case - Square Tires Fit Perfectly

Container dimensions: Rows = 5, Columns = 5
Number of square tires: 3,Tire lengths: 2, 3, 2
We can observe in Fig.1 that all square tires fit perfectly into the container.

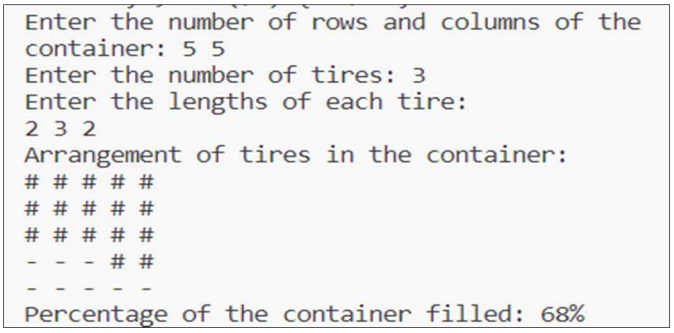

Fig. 1. Square tires Fit perfectly in container using Dynamic Programming.

- Case 2: Insufficient Container Space for Square Tires

Container dimensions: Rows = 4, Columns = 4
Number of square tires: 3,Tire lengths: 3, 2, 3
We observe in Fig.2 that the code detects all square tires cannot fit into the container.

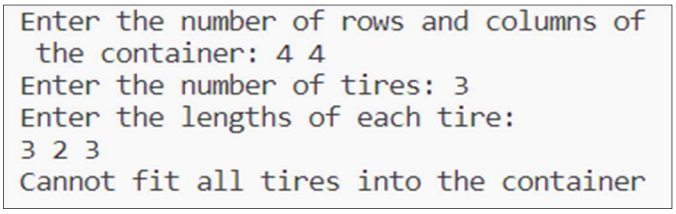

Fig. 2. Insufficient ContainerSpace for Square Tires in Container Using Dynamic Programming.

- Case 3: Small Square Tires in a Large Container

Container dimensions: Rows = 8, Columns = 8
Number of square tires: 4, Tire lengths: 2, 2, 2, 2
We Observe in Fig.3, the code demonstrating efficient packing for small square tires.

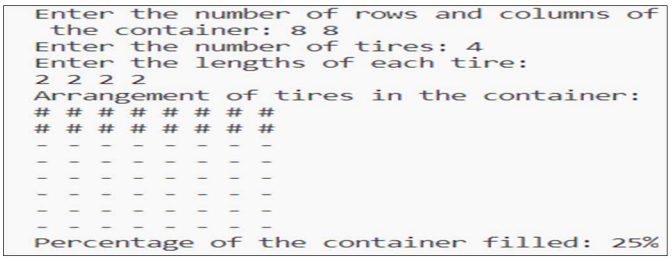

Fig. 3. Small Square Tires in a Large Container using Dynamic Programming.

V. RESULTS

In our extensive research on optimizing the arrangement of various tire types within 2D containers, we implemented and rigorously compared several state-of-the-art algorithms, including Next Fit, Best Fit Decreasing, Genetic Algorithm, Dynamic Programming, and Ant Colony Optimization. The results of our comprehensive evaluation reveal notable differences in the performance of these algorithms with respect to both time and space complexity. Each algorithm exhibited distinct strengths and trade-offs. While Next Fit demonstrated simplicity and ease of implementation, it occasionally resulted in suboptimal space utilization. Conversely, Best Fit showcased improved space optimization at the cost of increased computational effort. Genetic Algorithm and Dynamic Programming, leveraging sophisticated optimization strategies, exhibited competitive performance with respect to container space utilization but required more computational resources. Ant Colony Optimization, inspired by nature's swarm intelligence, demonstrated the ability to discover efficient tire placement solutions while offering a unique approach to space optimization. Our research findings provide valuable insights for practitioners and researchers in the field, offering a comprehensive comparison of these algorithms' performance metrics to inform informed decision-making in tire storage and container packing scenarios. Fig. 6 shows the comparison between time complexity and space complexity between the algorithms.

TABLE V. COMPARISON TABLE OF TIME COMPLEXITY AND SPACE COMPLEXITY

Algorithms	Time Complexity	Space Complexity
Best Fit Decreasing(BFD)	• $O(t^2)$ • $O(n^2)$	• O(t) • $O(n)$
Next Fit Algorithm	• O(t * m) • $O(n^2)$	• O(t * (1 + k)) • $O(n^2)$
Dynamic Programming Algorithm	• O(r * m * t + t * log(t)) • $O(n^3)$	• O(r * m + t) • $O(n^2)$
Ant Colony Optimization	• $O(i * a * t^2)$ • $O(n^4)$	• $O(t^2 + a * t)$ • $O(n^2)$
Genetic Algorithm	• O(g * p * log(p)) • $O(n^2 log(n))$	• O(p * t) • $O(n^2)$

Where, t=number of tires, b=number of bins, k=number of tires that can be fitted in the bin, r=rows, m=columns, i=number of iterations, a=number of ants, g=number of generations, p=Population size, To understand and compare time complexity more effectively, a generalized time and space complexity is being derived and written below the actual time and space complexity for each algorithms n= worst case input size.

The time complexity of these different algorithms could be graphed as shown in the Fig. 4 below.

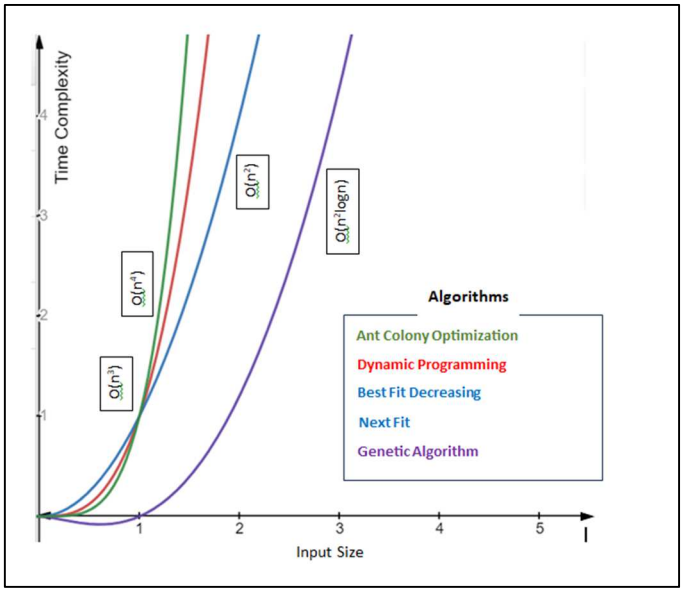

Fig. 4. Time Complexity Comparison Graph of Algorithms

In the graph above, the input size (x-axis) is plotted against the corresponding time complexity (y-axis) for various algorithms. The graph visually illustrates how each algorithm's performance scales with increasing input size (n), providing valuable insights into their computational efficiency and helping to identify the most suitable algorithm for different problem sizes.

CONCLUSION

In this work, we have presented the optimization of tire arrangement within 2D containers and have yielded valuable insights into the comparative performance of various algorithms, shedding light on their strengths and limitations. Through methodical experimentation and detailed scrutiny, it is understood that Dynamic Programming (DP) emerges as the most efficient algorithm among those studied, consistently demonstrating superior container space utilization while offering a reasonable compromise in terms of computational resources. However, it is essential to acknowledge the limitations of other algorithms examined in this research. Genetic Algorithms, although capable of finding competitive solutions, often require extensive computational resources. Best Fit Decreasing, while enhancing space optimization, demands increased computational effort. Next Fit, though straightforward, occasionally results in suboptimal space utilization. Ant Colony Optimization, despite its promise, exhibits sensitivity to parameter tuning and may require intricate fine-tuning for optimal performance. By highlighting these nuances, our research equips practitioners and researchers with critical information for selecting the most suitable algorithm for specific tire storage and container packing scenarios, with Dynamic Programming emerging as the preferred choice for achieving the best balance between efficiency and effectiveness.

REFERENCES

[1] Gamal Abd El-Nasser A. Said, undefined. El-Sayed M. El-Horbaty. "A Simulation Modeling Approach for Optimization of Storage Space Allocation in Container Terminal," in CoRR, vol. abs/1501.06802, 2015.

[2] X. Xiang, C. Yu, H. Xu, and S. X. Zhu, "Optimization of Heterogeneous Container Loading Problem with Adaptive Genetic Algorithm," in Journal Name, vol. 2018, Article ID 2024184, pp. 1-12, Nov. 2018. DOI: 10.1155/2018/2024184.

[3] Gerhard Wäscher, Heike Haußner, Holger Schumann. "An improved typology of cutting and packing problems," in European Journal of Operational Research, vol. 183, no. 3, pp. 1109-1130, 2007.

[4] H. C. W. Lau, W. T. Tsui, C. K. M. Lee, G. T. S. Ho and A. Ning, "Development of a Profit-Based Air Cargo Loading Information System," in IEEE Transactions on Industrial Informatics, vol. 2, no. 4, pp. 303-312, Nov. 2006, doi: 10.1109/TII.2006.885193.

[5] Semih Önüt, undefined. Umut R. Tuzkaya, undefined. Bilgehan Doğaç. "A particle swarm optimization algorithm for the multiple-level warehouse layout design problem," in Computers & Industrial Engineering, vol. 54, no. 4, pp. 783-799, 2008.

[6] Andreas Bortfeldt. "A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces," in European Journal of Operational Research, vol. 172, no. 3, pp. 814-837, 2006.

[7] C.-F. Chien, C.-Y. Lee, Y.-C. Huang, and W.-T. Wu, "An efficient computational procedure for determining the container-loading pattern," Computers & Industrial Engineering, vol. 56, pp. 965-978, 2009.

[8] Bortfeldt, A., & Wäscher, G.. (2013). Constraints in container loading – A state-of-the-art review. 229(1). https://doi.org/10.1016/J.EJOR.2012.12.006

[9] J. T. Patil and M. E. Patil, "Cargo space optimization for container," 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Jalgaon, India, 2016, pp. 68-73, doi: 10.1109/ICGTSPICC.2016.7955271.

[10] De, A., Pratap, S., Kumar, A., & Tiwari, M.. (2018). A hybrid dynamic berth allocation planning problem with fuel costs considerations for container terminal port using chemical reaction optimization approach. 290(1). https://doi.org/10.1007/S10479-018-3070-1

[11] Alonso, M. T., Alvarez-Valdés, R., Iori, M., & Parreño, F.. (2019). Mathematical models for Multi Container Loading Problems with practical constraints. 127. https://doi.org/10.1016/J.CIE.2018.11.012.

[12] Zhou, C., Wang, W., & Li, H.. (2020). Container reshuffling considered space allocation problem in container terminals. 136. https://doi.org/10.1016/J.TRE.2020.101869

[13] Czerniachowska, K., & Lutosławski, K.. (2021). Dynamic programming approach for solving the retail shelf-space allocation problem. 192, 4320–4329.

[14] L. Wang, M. Ni, J. Gao, Q. Shen, Y. Jia, and C. Yao, "The Loading Optimization: A Novel Integer Linear Programming Model," in 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Macao, China, Dec. 2019, pp. 1234-1239.

[15] Olsson, J., Larsson, T., & Quttineh, N.-H.. (2020). Automating the planning of container loading for Atlas Copco: Coping with real-life stacking and stability constraints. 280(3). https://doi.org/10.1016/J.EJOR.2019.07.057