Optimal Cargo Management for Flights

Inter IIT Tech Meet 13.0

Final Submission Report

Author:

Team 67

Contents

1 Demonstrative Video **2**

2 Introduction **2**

3 Background Literature review **2**

4 Problem Statement **2**
4.1 Problem Formulation . . . 2
4.1.1 Input-Output Structure . . . 2
4.1.2 Necessary Constraints . . . 3
4.1.3 Cost Function . . . 3

5 Solution Methodology **3**
5.1 Heuristic Algorithms . . . 5
5.1.1 Extreme Point Insertion . . . 5
5.1.2 Axes Projection . . . 5
5.1.3 Space Defragmentation (SD) . . . 5
5.1.4 Priority and Cost based Package Assignment . . . 6
5.1.5 Refitting and Stability Improvement . . . 6
5.1.6 Overall Algorithm Efficiency . . . 6
5.2 Mixed Integer Programming (MIP) Solvers . . . 6
5.2.1 Iterative Insert MIP Solver . . . 7
5.2.2 Swap and Optimize MIP Solver . . . 7

6 Results **8**
6.1 Final Assignment Plot . . . 8
6.2 Metrics . . . 9
6.3 Further Analysis . . . 9
6.3.1 Center of Mass Heatmap . . . 9
6.3.2 Cost-Runtime Plot . . . 10
6.3.3 Stability of Packages . . . 10

7 Difficulties Faced **10**

8 Other Approaches Attempted **11**

9 Limitations **11**

10 Conclusion **11**

11 Web Application for ULD Loading **11**

1 Demonstrative Video

Kindly find the demonstrative video of our solution to the problem statement in the archived folder.

2 Introduction

Unit Load Devices (**ULDs**) are standardized air shipment containers with strict weight and volume limits, requiring efficient packing to **maximize space utilization**, **minimize costs of delay**, and ensure **timely delivery** according to priorities. The optimization of ULD packing offers significant advantages for both the companies and their customers because of its potential to streamline logistics operations, reduce operational costs, and enhance customer satisfaction.

The report focuses on developing and implementing a solution for efficient ULD loading subject to various different constraints which gives a close to optimal cost within reasonable time bounds.

3 Background Literature review

The problem is effectively an extension of the **3-Dimensional Bin Packing Problem** [1] and **3-Dimensional Knapsack Packing Problem** [2], in which items of different sizes must be packed into a finite number of bins or containers, each of a fixed given capacity with some additional weight and priority constraints, along with a custom objective function.

There have been numerous attempts to solve this problem, including formulating it as a **Mixed Integer Programming Model** [3], using **Greedy heuristics** [2], applying meta-heuristics like **Genetic Algorithms** [4] and **Simulated Annealing** [5], as well as the more recent **Reinforcement Learning** [6][7] based approach.

4 Problem Statement

4.1 Problem Formulation

The problem is structured with the following input-output specifications, constraints, and cost function:

4.1.1 Input-Output Structure

1. **Input:**
   (a) A **list of ULDs,** each with its ID, length, width, height, and maximum weight limit.
   (b) A **list of packages**, which are defined by its ID, dimensions (length, width, height), weight, type (Priority/Economy), and the cost of delay (for economy packages).
   (c) A **cost parameter** K, incurred for each ULD that contains at least one Priority Package in the solution.
2. **Output:**
   (a) The total cost, total number of packages packed, and the total number of ULDs containing Priority Packages.
   (b) A detailed list of all assigned packages, including their IDs, assigned ULD ID, and coordinates (Front-Left-Bottom and diagonally opposite corners) within the ULD.

4.1.2 Necessary Constraints

1. All Priority Packages must be shipped.
2. The total weight of packages assigned to a ULD must not exceed its weight limit.
3. No two packages may overlap, and all assigned packages must fit entirely within the ULD without any protrusion.

4.1.3 Cost Function

The cost function for the optimization problem is defined as:

1. The sum of the costs of all unassigned economy packages (S).
2. An additional penalty for spreading Priority Packages across too many ULDs, represented by $X \times K$, where X is the number of ULDs containing Priority Packages.

The total cost is given by:

$$\text{Total Cost} = S + X \times K$$

Among all feasible solutions that satisfy the constraints, the one with the lowest total cost is preferred.

5 Solution Methodology

Bin packing is an **NP-hard problem**, making it infeasible to find an optimal solution in polynomial time. Methods like MIP (Mixed Integer Programming) can provide exact solutions, but they require exponential time. Heuristic approaches offer faster solutions, though often far from optimal. To address this, we smartly developed a **hybrid** approach that combines both methods. Our framework uses heuristics to quickly generate a **good initial solution** and **reduce the search space**, while MIP solvers **refine** this solution, aiming for optimal results within constrained time, without the computational expense of finding the perfect solution.

- **Heuristic Techniques**: ***Extreme Point Insertion*** [8], ***Space Defragmentation*** [9] and ***Axes Projection*** methods are used to generate an initial packing solution. This step efficiently assigns packages to available ULDs, minimizing penalty unassigned packages. The preliminary solution is then refined using the following two iterative MIP solvers.
- **Iterative Insert MIP Solver**: This solver aims to **add unassigned packages** to the filled ULDs, optimizing their placement within time constraints, taking advantage of the fact that an MIP with no objective and only feasibility constraints runs much **faster**.
- **Swap and Optimize MIP Solver**: After the first solver, the solution is further refined by **swapping placed packages** with unassigned ones within ULDs, where refitting is feasible, and cost reductions are possible.

Once the MIPs provide a solution, we again apply **Space Defragmentation** (swap in/out packages) and **Axes Projection** heuristics. The solution obtained is then sent back to the MIP solver to improve the cost, creating a **feedback loop** that adjusts based on the available time.

Both MIP solvers are designed to operate within **user-set configurable time limits**, by reducing their search space. Extended runtime can be used to progressively enhance solution quality.

This combination of heuristic and optimization-driven approach offers a **robust framework** for tackling the complexities of three-dimensional bin packing in **dynamic** and **cost-sensitive** environments.

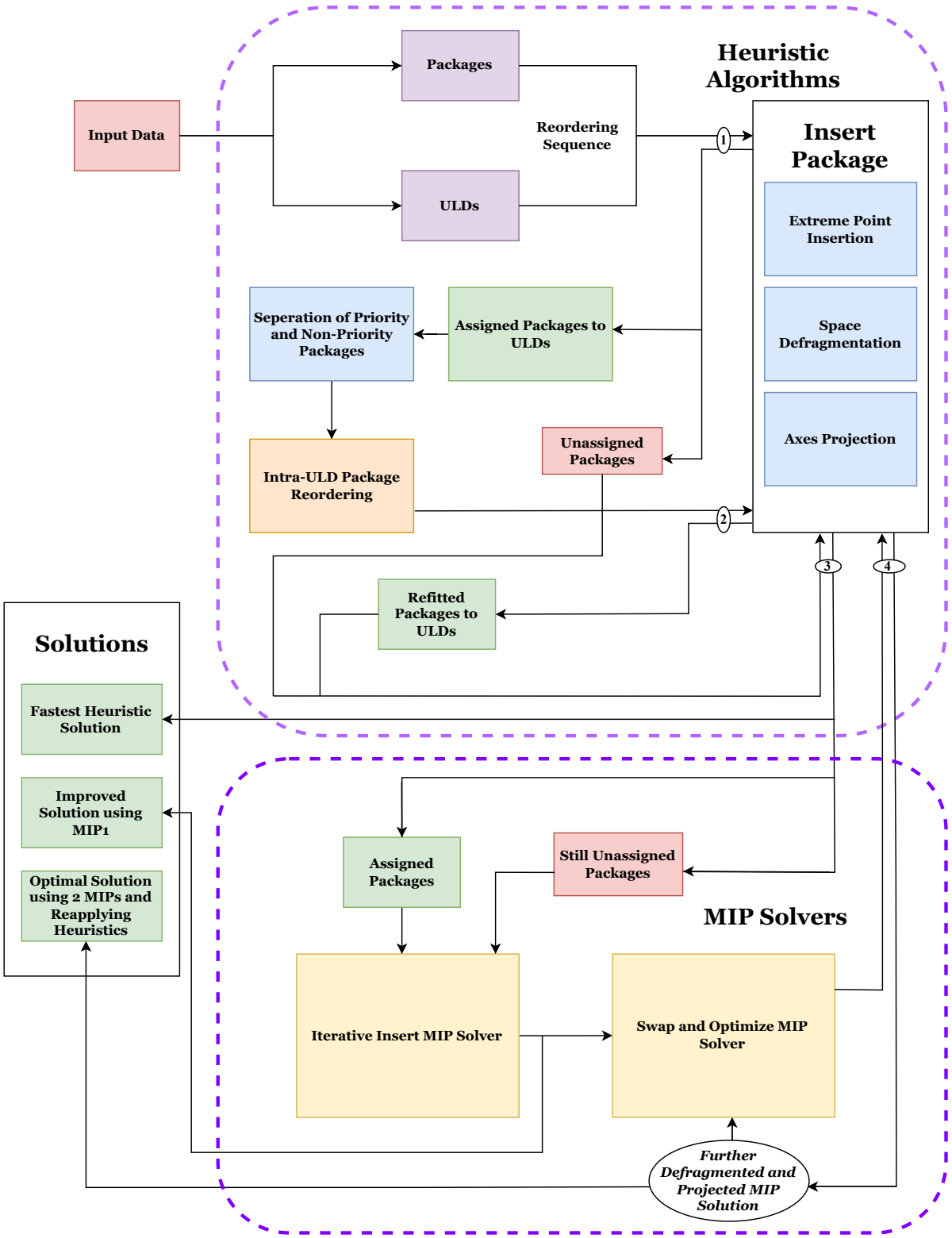


Figure 1: Pipeline of the proposed solution

5.1 Heuristic Algorithms

To achieve an efficient and quick initial packing solution, we employ a combination of heuristic algorithms, each designed to optimize space utilization and packing stability. The following sections describe the key heuristic techniques used in the process:

5.1.1 Extreme Point Insertion

- The Extreme Point Insertion algorithm is used to identify potential locations for placing packages within an ULD. The key objective is to place new packages adjacent to previously placed ones, maximizing space utilization while ensuring no overlap.
- **Package Placement:** The list of extreme points is iterated to identify feasible locations. The Front-Left-Bottom (FLB) corner of the package is placed at the first feasible extreme point. **All possible rotations** of the package are considered, and the best fit is selected.
- **Generation of New Extreme Points:** After placing a package, new extreme points are generated by projecting the corners of the package's bounding box along the container's axes. This step identifies potential locations for insertion of subsequent packages.
- **Maintaining Extreme Points:** The extreme points are maintained in **increasing order of their Euclidean distance** from the origin. This clustering of packages near the origin maximizes the **contiguous free space** and improves the chances of efficiently accommodating future packages.

5.1.2 Axes Projection

- The extreme point insertion along the corners need not place the package close to the other faces, leaving space to **push the package backwards**.
- To take advantage of that space, packages are shifted along the **negative direction** of the three axes (**x, y, z**) to minimize the free space near the origin.
- This ensures the direct utilization of **shiftable free space**, improving the **stability** and **compactness** of the packing.

5.1.3 Space Defragmentation (SD)

During the heuristic experiments, a significant challenge was identified: the presence of **fragmented spaces** between packed items. These voids reduced the overall volume utilization, negatively impacting the efficiency of the packing solution. To mitigate this issue, Space Defragmentation techniques were introduced to consolidate unused spaces and improve packing efficiency. The following outlines the key functions involved in this process:

1. **Calculating Push Limit:** The first step calculates the Push Limit for each packed item. This is the maximum distance a box can be moved along a specific axis without intersecting other already packed boxes. The process proceeds with the **Push Out** operation, which shifts all boxes located beyond the insertion point along the positive axes. This operation creates a void, allowing new packages to be placed in the available space.
2. **Inflating and Replacing Allocated Pieces:** In this step, the algorithm attempts to replace smaller, lower-cost packages with unassigned, higher-cost ones. This is achieved by inflating the space occupied by the previously placed packages, which pushes the remaining boxes away to create room for new items.

3. **Normalize and Recalculate Extreme Points:** After successfully inserting the packages, the solution is normalized by pushing the packed items back along the negative axes. Finally, the extreme points are recalculated to ensure that the newly packed items align with the most efficient spatial arrangement.

5.1.4 Priority and Cost based Package Assignment

- **Priority Packages:** Packages with higher priority are assigned an infinite cost, ensuring that they are selected and packed first.
- **Cost Density Sorting:** Packages are sorted in decreasing order of cost density, with a preference for smaller in volume, higher-cost items.
- **Iterative Fitting:** Once the packages are sorted, they are iteratively fitted into ULDs using the packing rules described in sections 5.1.1 to 5.1.3.

5.1.5 Refitting and Stability Improvement

- **Re-sorting Assigned Packages:** While higher cost density packages should generally be prioritized, packing them in this order often results in large empty spaces and instability. To address this, assigned packages are re-sorted based on a **clustered height area** heuristic. This method groups packages by height, with intra-cluster sorting done by base area.
- **Refitting:** After re-sorting, the packages are repacked into the ULDs. Packages that were not initially placed are then attempted to be fitted into the available space.

Refitting not only enhances volume utilization but also significantly improves stability within the ULDs. This is achieved by ensuring that heavier, larger packages are placed before smaller ones, thereby reducing the risk of instability caused by uneven weight distribution.

5.1.6 Overall Algorithm Efficiency

- The hybrid approach combining heuristic techniques terminates in less than a minute, offering a **fast, preliminary solution** to the problem.
- The time complexity of this part of the solution is $O(n \cdot N^2)$ where n is the maximum number of packages in a single ULD and N is the total number of packages.

5.2 Mixed Integer Programming (MIP) Solvers

The 3D Bin Packing problem can be formulated as a **Mixed Integer Programming (MIP)** problem. Constraints can be made by making binary variables and their linear inequalities to ensure that no two packages overlap, all assigned packages fit within the respective ULDs, the assignment doesn't exceed ULDs' weight limits and that some degree of stability is ensured. Further, cost function can be added to the MIP which takes into account the spread of Priority Packages and penalties of unassigned packages. However, solving the problem is **NP-hard**. If n packages are considered for assignment to m ULDs, the number of variables to represent the MIP can grow up to $O(m \cdot n^2)$. Given this exponential growth, solving the MIP becomes computationally expensive, and it is impractical to consider all constraints simultaneously due to time limitations. Therefore, careful formulation of the MIP is essential to balance solution quality and computational efficiency within reasonable time limits.

We use two MIP solvers, each strategically designed to break the problem into smaller sub-parts. By modifying the constraints and objective for each solver, we can manage the

complexity more effectively, rather than feeding the entire problem into a single solver. Gurobi's MIP Solver is employed to handle this computationally intensive task, with the goal of achieving a near-optimal solution within an acceptable time frame.

5.2.1 Iterative Insert MIP Solver

- **Objective:**
  - This solver improves the greedy heuristics-based solution by focusing solely on checking the **feasibility of constraints** making clever use of the fact that formulating and solving a complex cost function is much more time-consuming.
- **Algorithm Overview:**
  - For each ULD, check if there is **any unassigned package** that can be added to it without affecting the already assigned packages.
  - If a feasible solution is found, the new package is assigned to the ULD along with the previously assigned packages and a refit is made. This **guarantees** a **reduction** in the overall cost as some unassigned packages are now assigned.
- **Optimizations for Enhancing Speed:**
  - Instead of processing all leftover packages, only the top 10–15 unassigned packages (based on a **priority metric** i.e. creating bins of volume/100 and reverse sorting by cost within each bin) are selected.
  - This ensures a faster solution by only checking for packages that have the **highest chance** of being placed.

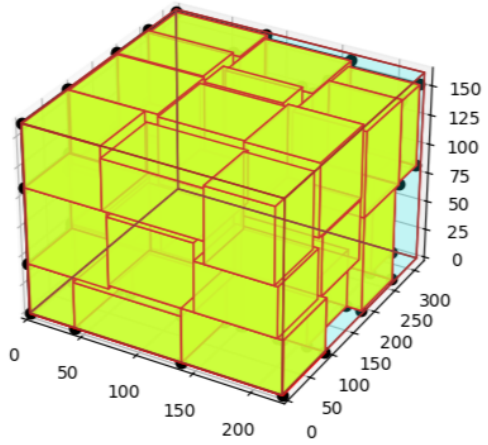
5.2.2 Swap and Optimize MIP Solver

- **Objective:**
  - This solver is designed to **fine-tune** the refined solution obtained from the heuristic algorithms and the previous solver.
  - Unlike the iterative insert MIP, this solver incorporates the **cost function** for optimization and provides much more freedom for **swapping of packages among ULDs** at the expense of runtime.
- **Algorithm Overview:**
  - This algorithm takes a subset of ULDs of and runs an MIP with all the already assigned packages to the respective ULDs and some unassigned packages which are chosen by sorting them on a metric that maximizes cost to volume ratio.
  - For each ULD, the solver swaps already assigned packages with assigned packages of other containers or the unassigned ones if the cost is being optimized further.

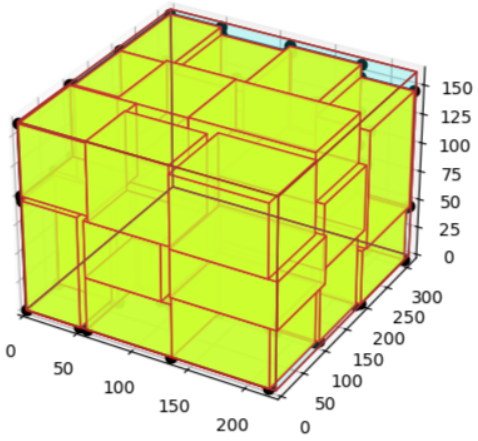
Once the user-specified time limit for the second MIP Solver expires, the solver is terminated, and the Add Package Heuristic Algorithms are re-applied to handle the unassigned packages. This step ensures that any instability or fragmentation introduced by the MIP within ULDs is addressed. Following this, the updated solution is fed back into the MIP solvers. This creates an iterative feedback loop, where both the MIP solvers and heuristic algorithms continually refine the solution. The process continues until the specified time limit is reached, at which point the solution is considered near-optimal.

6 Results

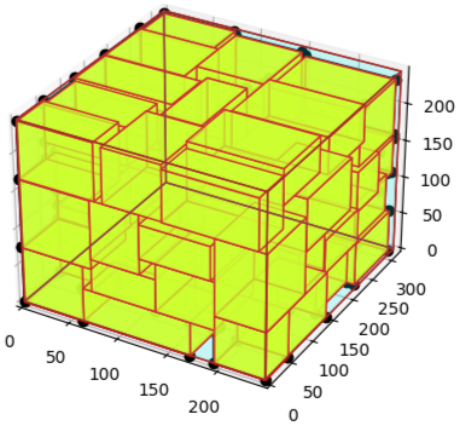
6.1 Final Assignment Plot



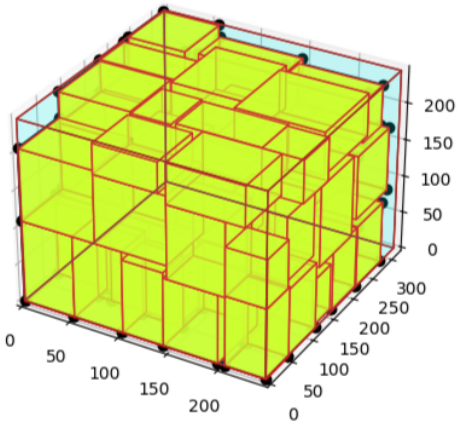
(a) ULD 1



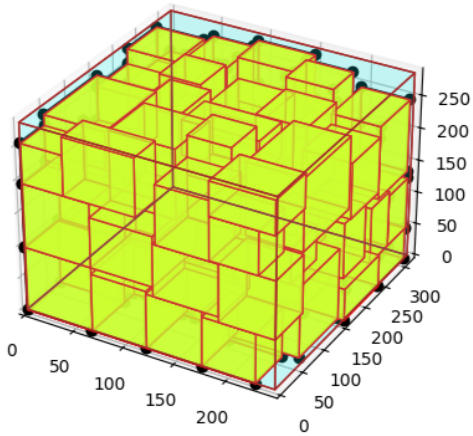
(b) ULD 2



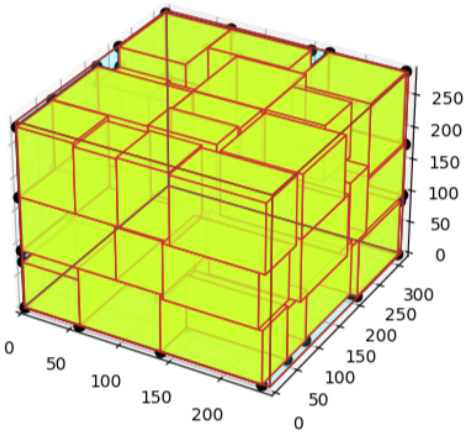
(c) ULD 3



(d) ULD 4



(e) ULD 5



(f) ULD 6

Figure 2: Plot of the ULDs

6.2 Metrics

Total number of packages assigned: **255**

Number of ULDs containing priority packages: **3**

Final Cost of our Solution: **27940**

ULD ID	# Priority Packages	# Economy Packages	% Free Volume	% Free Weight
ULD1	0	31	11.2%	25.2%
ULD2	0	27	10.8%	17.9%
ULD3	0	50	16.0%	0.5%
ULD4	33	12	14.8%	2.4%
ULD5	39	24	20.7%	11.3%
ULD6	31	8	12.0%	0.5%
Total	103	152	14.8%	9.0%

Table 1: ULD Metrics

6.3 Further Analysis

6.3.1 Center of Mass Heatmap

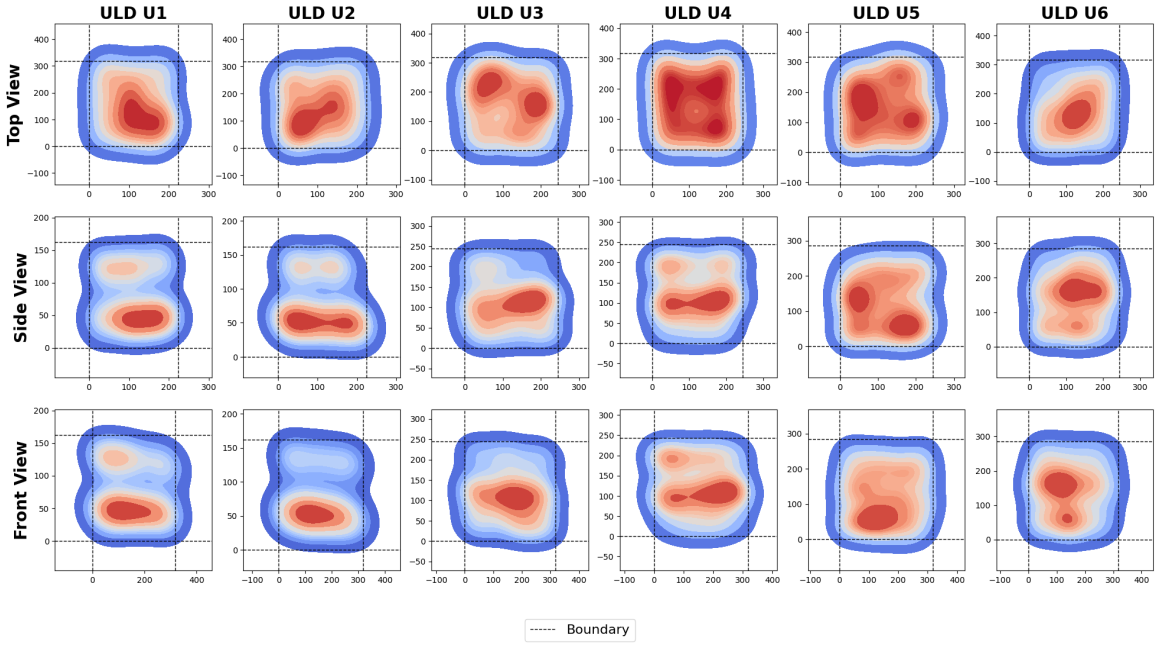


Figure 3: Weight distribution of 6 ULDs and 3 planes with uniformity on XY **base** plane

The heatmap uses kernel density estimation (KDE) to show the density distribution of packages (weighted by weights) in 2D projections. Color coding is

- **Red zones:** Higher density areas
- **Blue zones:** Lower density or sparse areas.

The plot indicates that packages are uniformly spread across the X-Y plane, creating a balanced base, with heavier ones naturally settling lower, enhancing overall stability.

6.3.2 Cost-Runtime Plot

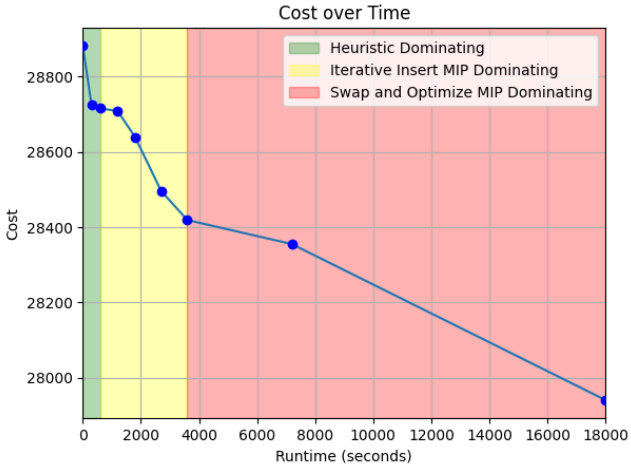


Figure 4: Trade-off between cost optimality and runtime (in seconds)

This plot illustrates the trade-off between cost and running time. Heuristic methods provide a good solution in a short time. The Iterative Insert MIP delivers significant improvements for intermediate run times. Finally, the Swap and Optimize MIP yields steady gains in solution quality over longer run times, though at a slower pace.

6.3.3 Stability of Packages

1. Defining the stability of packages in a packing arrangement is a complex task, as no single metric universally captures all the factors that contribute to stability. One approach is to define stability by ensuring that the package is supported at its **center of mass**. However, this criterion alone is insufficient. For example, a package may appear stable if supported at its center of mass, but it could tip over when subjected to turbulence.
2. To address this, we introduce a different metric: a package is considered stable if more than **50% of its base area** is supported by another package or the ground. This condition guarantees a certain level of stability but is not comprehensive, as it may exclude some stable packages that are supported by the sides of other packages.
3. Using the base area support criterion, we find that **more than 85%** of the packages in our best solution meet the stability condition. However, this is likely an underestimation, as it does not account for packages that are supported laterally by adjacent packages or by the ULD structure itself. As such, the true stability rate is higher than our estimate.

7 Difficulties Faced

1. **Resource Limitations**: Running an MIP demands **huge computational power** and memory, thus leading to the need for effective **approximation methods** that can provide near-optimal solutions.
2. **Formulating effective heuristics**: The **dynamic nature** of the problem introduces complexity in developing heuristics that perform consistently well across diverse datasets.

8 Other Approaches Attempted

1. **Reinforcement Learning**: We trained a **Pointer Network** [10][11] using the Policy-Based Reinforcement Algorithm to generate an initial packing sequence for the greedy algorithm. However, the pre-trained model **underperformed** compared to our sorting heuristics. Fine-tuning improved results but failed to meet time constraints.
2. **Meta-heuristics like Genetic Algorithms, Simulated Annealing and Tabu Search**: In these techniques, the solution depends on the randomly generated values and did not offer any **consistent improvement** over the cost determined by the greedy algorithm.
3. **Fully Mixed Integer Programming Based Approach**: Due to the large number of constraints of the problem, it is not viable to run the entire problem as a single MIP since even for an input as small as 3 ULDs and 6 packages, it takes **15 minutes** to reach an optimal solution.
4. **Merit Functions for Package Sorting in Greedy**: Various functions were tried to sort packages both during assignment and during refitting. These include Decreasing Volume, Decreasing Volume-Height, Clustered Area-Volume, Density + Cost Density

9 Limitations

1. **Non-Deterministic MIP Solver:** Gurobi, being hardware dependent, does not guarantee the same solution when executed on different devices, especially when a time limit is imposed. This can also lead to **different results** even for the same test cases on **repeated runs**, introducing variability in outcomes.
2. **Existence of Adversarial Inputs**: The heuristics will perform well on average, but there can always exist a few adversarial inputs that result in poor performance.

10 Conclusion

In this report, we addressed the ULD Optimization Problem by implementing our solution that effectively combines **Greedy Heuristic** Methods and **Mixed Integer Programming** (MIP) solvers for making **practical** and **intuitive** decisions. We offer **3 different solutions**, all of which are useful for different conditions. The first solution, using only the greedy heuristic part of the pipeline, is obtained in **less than a minute**. The second solution further optimizes it and improves the packing after running for a few minutes. The third solution, while taking the longest to run, offers the **best cost** out of the three. Having multiple solutions makes our solution well-suited for **dynamic and cost-sensitive** logistics environments. Any one of the three solutions can be used depending on the amount of prior knowledge and time available for loading preparation, thus offering a large **degree of control to the user**.

11 Web Application for ULD Loading

At the core of our solution is a user-friendly **Streamlit web application** that accepts both file-based and manual inputs for package and ULD specifications. The application runs the packing algorithm and generates a **step-by-step 3D visualization** of the **loading sequence** for each ULD, allowing users to visually track the placement of packages, in order for efficient logistics operations. Additionally, it offers configurable time limits to tailor the algorithm's performance to specific needs.

References

[1] Silvano Martello and Paolo Toth. "Bin-packing problem". In: *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, UK: John Wiley and Sons, 1990. ISBN: 0471924202.

[2] Jens Egeblad and David Pisinger. "Heuristic approaches for the two- and three-dimensional knapsack packing problem". In: *Computers Operations Research* 36.4 (2009), pp. 1026–1049. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2007.12.004. URL: https://www.sciencedirect.com/science/article/pii/S030505480700264X.

[3] C.S. Chen, S.M. Lee, and Q.S. Shen. "An analytical model for the container loading problem". In: *European Journal of Operational Research* 80.1 (1995), pp. 68–76. ISSN: 0377-2217. DOI: https://doi.org/10.1016/0377-2217(94)00002-T. URL: https://www.sciencedirect.com/science/article/pii/037722179400002T.

[4] José Gonçalves. "A parallel multi-population biased random-key genetic algorithm for a container loading problem". In: *Computers OR* 39 (Feb. 2012), pp. 179–190. DOI: 10.1016/j.cor.2011.03.009.

[5] Turkay Dereli and Gülesin Daş. "A Hybrid Simulated Annealing Algorithm for Solving Multi-Objective Container-Loading Problems." In: *Applied Artificial Intelligence* 24 (May 2010), pp. 463–486. DOI: 10.1080/08839514.2010.481488.

[6] Hang Zhao, Yang Yu, and Kai Xu. "Learning Efficient Online 3D Bin Packing on Packing Configuration Trees". In: *International Conference on Learning Representations*. 2022. URL: https://api.semanticscholar.org/CorpusID:251649027.

[7] Anan Ananno and Luis Ribeiro. "A Multi-Heuristic Algorithm for Multi-Container 3-D Bin Packing Problem Optimization Using Real World Constraints". In: *IEEE Access* PP (Mar. 2024). DOI: 10.1109/ACCESS.2024.3378063.

[8] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. "Extreme Point-Based Heuristics for Three-Dimensional Bin Packing". In: *INFORMS Journal on Computing* 20.3 (2008), pp. 368–384. DOI: 10.1287/ijoc.1070.0250. URL: https://doi.org/10.1287/ijoc.1070.0250.

[9] Wenbin Zhu et al. "Space defragmentation for packing problems". In: *European Journal of Operational Research* 222.3 (2012), pp. 452–463. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2012.05.031. URL: https://www.sciencedirect.com/science/article/pii/S037722171200389X.

[10] Irwan Bello et al. *Neural Combinatorial Optimization with Reinforcement Learning*. 2017. arXiv: 1611.09940 [cs.AI]. URL: https://arxiv.org/abs/1611.09940.

[11] Haoyuan Hu et al. "Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method". In: *CoRR* abs/1708.05930 (2017). arXiv: 1708.05930. URL: http://arxiv.org/abs/1708.05930.