# OPTIMAL CARGO MANAGEMENT FOR FLIGHTS

## FINAL REPORT

TEAM 34

## 1. SCOPE OF THE PROBLEM

The problem provides data for ULDs and packages of two categories: priority and non-priority. The goal is to generate a valid packing such that

- All the packages are completely inside the ULD and do not intersect with each other
- All the priority packages must be shipped in a ULD, and the spread of these packages across the ULDs must be minimized
- The weight of the packages shipped in an ULD must not exceed its weight capacity
- The cost of the spread of priority packages and unshipped economy packages is minimized

In addition to the above minimal scopes, our solution aims to provide the following additional features that can improve its generalisability and increase its adoption.

### 1.1 SUPPORT FOR ADDITIONAL CONSTRAINTS

The user should be able to specify the following constraints:

1. If certain pairs of packages cannot be shipped together in an ULD (useful for modelling situations where the transport of some items together can lead to safety hazards, like food and electronics)
2. If some packages cannot be shipped in particular ULDs (for example, situations like where some ULDs are not-refrigerated and some packages must be shipped in them)
3. Heavy Packages: If some packages are substantially hefty, they must be placed on the ground, and stacking must occur only on top of them
4. Fragile Packages*:* Some packages must be placed on the top level and cannot have packages stacked on top (partially or completely)

### 1.2 STABILITY OPTIMIZATION & CUSHIONING

The algorithm should account for various rotational and physical stability metrics and optimize the placement to maximize the same without compromising the score of the transportation. The optimization should be global (between ULDs) as well local (inside a ULD). The algorithm should also be capable of estimating the amount of cushioning material (for e.g. thermacol, packing peanuts etc.) that must be added to an ULD to ensure secure placement of the packages.

### 1.3 LOADING PLAN GENERATION

The solution should generate a loading plan that automated machines or human operators can use to fetch and load the ULD efficiently. This is important to ensure that the solution is physically realizable. Only one face of the ULD is assumed to be open and available for loading packages.

## 2. PROPOSED SOLUTION

### 2.1 LITERATURE REVIEW

The problem is an extension of a classical combinatorial optimization problem called $3D$ Bin Packing Problem (3D-BPP). The problem is NP hard but over the years various heuristic, metaheuristic, and exact methods have been proposed to address it. After testing, exact methods liked Mixed Integer and Constraint Linear Programming were found to be too computationally expensive to be applied to the given problem data (while being very accurate). The running time increased exponentially with respect to the input which made them non scalable.

Moreover, the existing methods could not incorporate the concept of priority and economy packages, along with the fact that they both have different placement requirements. These use a uniform spreading approach which does not allow for the minimisation of spread of priority packages.

Heuristics considered included Empty Maximal Spaces (EMS), Distance to the Front Top Right Corner (DFTRC), 3 Orientation Pivot Method, Peak Filling Slice Push (PFSP) and Fit Degree Algorithm.

Genetic algorithms were found to be the most appropriate and widely used meta heuristic algorithms used for solving 3D-BPP. Even though they have the capacity to traverse through search space intelligently, they are often very slow at convergence because of undirected randomisation. The proposed solution incorporates the same with biased selection to give the search an intelligent heuristic direction which leads to faster convergence with better results, while respecting all the provided and new additional stability and cost constraints.

## 2.2 GENETIC ALGORITHM

The solution uses a *Biased Random-Key Genetic Algorithm (BRKGA)* to encode, cross-over, and mutate the configurations.

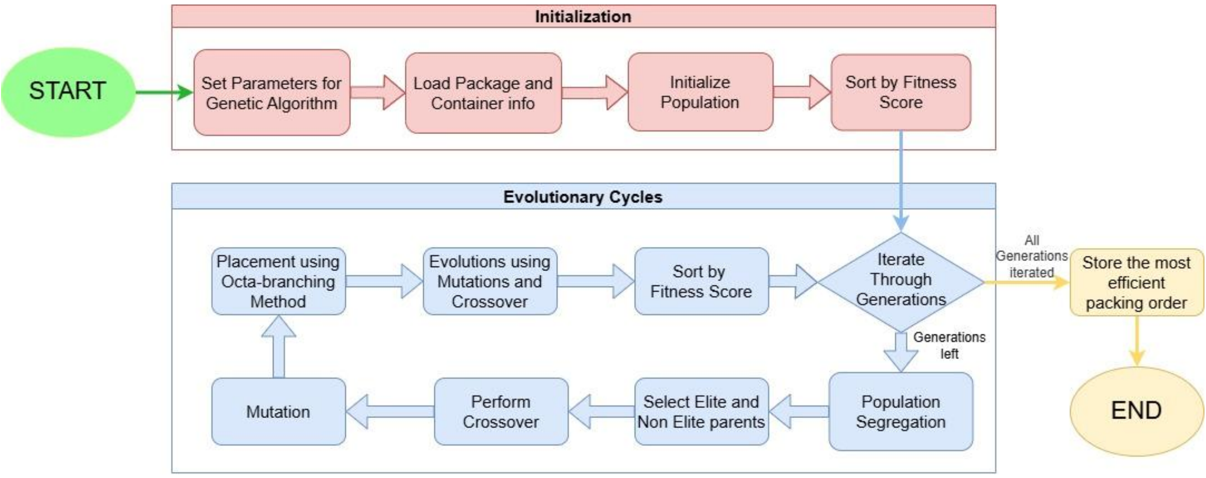


*Figure 1: Overview of Genetic Algorithm*

### 1. Encoding

The configuration chromosome is stored in two sequences of priority and economy packages. This sequence dictates the order in which the heuristic algorithm places these packages.

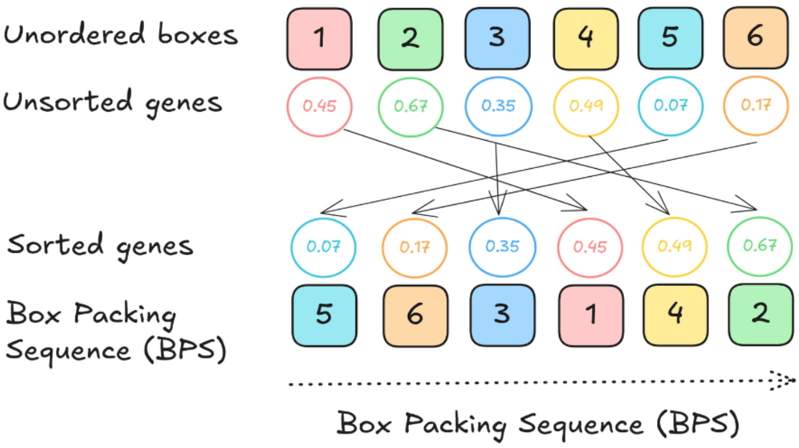


*Figure 2: Encoding of Box Packing Sequence (BPS) in a chromosome*

Random keys encode the chromosomes where the configurations are represented as vectors of real numbers within $[0,1]$. Each chromosome is represented by two vectors of sizes $n_p$ and $n_e$, where $n_p$ and $n_e$ are the number of priority and non-priority packages, respectively.

The *argsort* method is used to decode the chromosome into separate box packing orders for both types of packages, where the genes (packages) with the lowest values are packed first. We use *OctaBranching* (discussed later) to convert this packing order into a real placement of packages.

### 2. Population Segregation

Biased selection is used in GA for faster convergence. The existing population is segregated into *elite* and *non-elite* classes based on their chances of survival. A fraction of the fittest population (denoted by $f_{el}$) is chosen to be elite, and the rest is labelled to be non-elite.

### 3. Crossover and Mutation

*Parameterized uniform crossover* is used as the mutation engine. For each mating, one parent is chosen from the elite population and the other from non-elites. The selection of individuals is random. The $i^{th}$ gene of the spring will be either from the elite parent with a fixed hyperparameter $prob_{el}$. We typically set $prob_{el}$ in the range $[0.8, 0.95]$ to favor the elite parent.

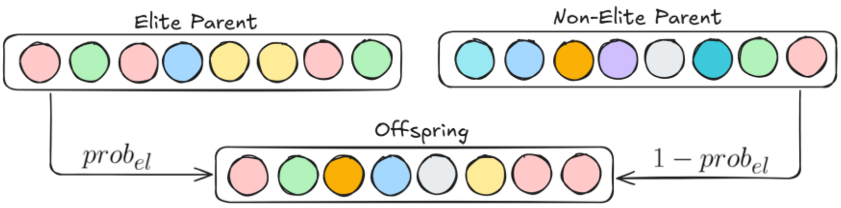


*Figure 3: Mutation & Crossover*

To further enhance the mating pool and to introduce new genes, in each generation, a fixed number of individuals $n_{gen}$ are generated randomly and added to the generation.

### 4. Fitness Score Evaluation

The algorithm passes this configuration to the Octa-Branching placement strategy, which returns a real-life packing scenario of the configuration, with the exact coordinates of the placement of each package into one of the provided ULDs. The fitness score $F$ is then calculated for each population member as follows:

$$F = C_s \cdot n_s + \sum_{i \in \mathcal{U}} c_i + st + P$$

where,

- $C_s$ is the spread penalty (cost associated with each new ULD for priority packages)
- $n_s$ is the number of ULDs with at least one priority package
- $\mathcal{U}$ is the set of unplaced packages
- $c_i$ is the cost associated with the $i^{th}$ package
- $st$ is the weighted sum of various stability metrics (discussed later)
- $P$ is the penalty associated with the packing (which might arise due to violating the packing constraints defined by the user in the package and ULD compatibility constraints).

The term $st$ is not essential to the correctness of the algorithm (as the placement algorithm guarantees the same), but it helps to choose the most stable configuration to carry out in the ULD. The penalty term $P$ is either zero or a large negative value, which helps to ensure that the user defined constraints of compatibility are always satisfied in the fittest individuals in any generation.

The new fittest offspring and the mutated members are then merged into the old population, and the process of evolution repeats until the fixed time quantum has elapsed or there is a stagnation in the fitness of the generations (saturation).

## 2.3 OCTA-BRANCH PLACEMENT STRATEGY

The OctaBranching strategy is a novel technique that lies makes use of novel ideas such as reference points and aggregation of multiple heuristics to search a large fraction of the exponential solution space in polynomial time. The strategy divides the space into 8 octants at each reference point to explore all the possible placements in the solution space.

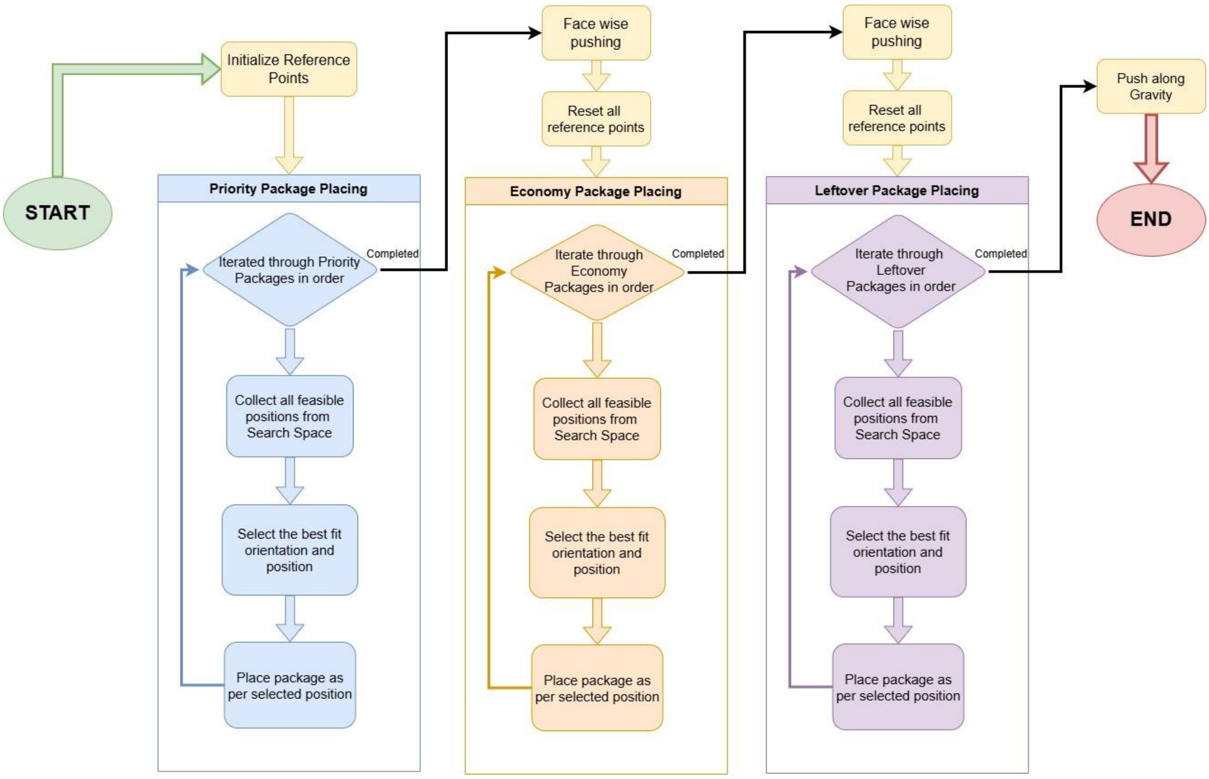


*Figure 4: General overview of OctaBranching strategy*

### 1. Reference Points

To tackle the exponential search space of this NP-hard problem, we use the intelligent searching strategy of reference points to significantly reduce the same while ensuring that the majority of the feasible search space is explored. This helps to reduce the algorithm's complexity to polynomial time (at any point in time, if the number of packages placed in a ULD is $n$, then the total number of reference points in the search space is bounded by $O(n)$).

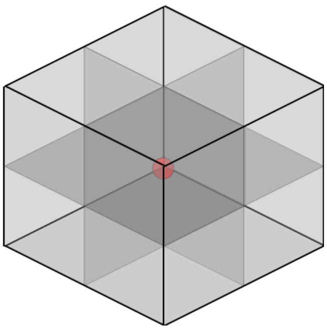

*Figure 6: Division of space into octants by reference points*

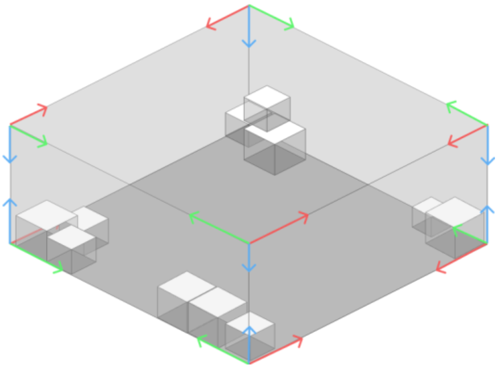

*Figure 5: Initialisation of reference points at the corners of an ULD*

A reference point is a corner of a package in $R^3$ space along with 1 out of 8 possible directions. As discussed later, 7 new reference points (with the same base point but different directions) are added to the search space whenever a reference point is consumed.

### 2. Initialization

Initially, 8 reference points are added to the search space (one for each corner of the ULD). These reference points are special as they can branch out to only $1$ direction.

### 3. Placement of Priority Packages

For each package in the BPS, each ULD's current reference points are scanned as possible candidates. For each point, all $6$ different orientations of the package are considered for physical feasibility (no intersections with the existing packages and the package should not exceed the dimensions of the container). Of all the pairs of reference points and orientations, the following tie-breaking criteria are used:

i. Prefer the container with most packages filled. This helps to minimize the spread of priority packages across ULDs.
ii. Prefer a point with a lower height (leaves more available $Z$ space)
iii. Prefer point with the least sum of distances from the sides (tighter packing in $2D$)
iv. Prefer orientation with maximum overlap between dimensions of package and box (more stable solution)

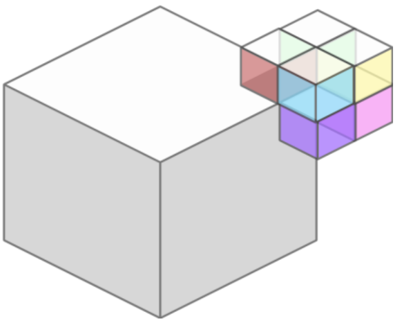

*Figure 7: Addition of 7 new reference points from a reference point*

The package is placed at the appropriate reference point. The solution space is updated accordingly (delete the reference point from the search space; for each corner of the placed box, generate all the reference points that do not intersect with any of the placed packages and add them to the solution space), before repeating the process for the next package in the BPS. The new reference points would become the candidates for addition of the next packages in the same generation.

### 4. Pushing packages along axes

After placing all priority packages, the entire configuration is pushed towards the sides (length and width of the ULD). This helps to create a more robust packing by increasing the area of lateral overlap between the packages. This also increases the space available for the other packages to be placed.

### 5. Placement of Economy Packages

The same strategy as the placement of economy packages is used, but with minor modifications in the selection strategy. The new tie-breaking criteria followed are:

i. Prefer ULD with the least packages filled. This ensures a uniform distribution across all the ULDs and, thus, a better volume utilization.
ii. Prefer a point with a lower height (leaves more available space)
iii. Prefer a reference point with the least sum of distances from the sides
iv. Prefer an orientation with maximum overlap between the package and boxes.

After choosing the best reference point and orientation, the reference points are updated, and the configuration is pushed along the lateral axes to make more space. After all the economy packages are placed, the algorithm again tries to place all the leftover packages with the same strategy in hope of utilizing the extra space created during the pushing. All packages are then pushed down (along gravity) until they come in contact with a resting surface. This ensures there are no floating packages and the packages are tightly packed.

## 2.4 STABILITY METRICS

During the calculation of fitness of each generation, the term $st$ is added to the fitness score to ensure that the overall configuration of the packages is stable to external forces. A weighted average of several metrics is combined from different papers so ensure rotational, translational as well as metastability of the packing.

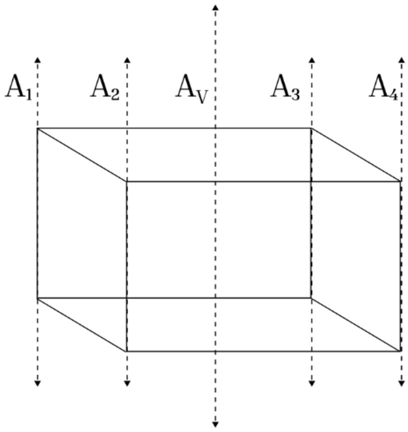


*Figure 8: Axes used in moment stabilization*

### 1. Moment Stabilization

This metric calculates how even the weight distribution is in the ULD, as a more even distribution leads to a less agile ULD and minimizes the chances of damage to the ULD in flight.

$$\vec{V_c} = \sum_{i=1}^{n} \vec{r_i} \cdot V_i$$

$$MOI_{\vec{A}} = \sum_{i=1}^{n} dis^2(\vec{r_i}, \vec{A}) \cdot w_i$$

$$M = \frac{MOI_{\vec{A_v}}}{mean(\sum \vec{A_i}) + std(\sum \vec{A_i})}$$

Moment stabilization is defined as the ratio of the minimum moment of inertia of the packing (which is obtained at the center of volume) with the standard moment of inertia at all four corners of the ULD.

### 2. Structural & Physical Stability

To ensure that the placement is physically stable concerning the center of gravity of the individual packages and is resistant to lateral turbulence and disturbing forces, we calculate a weighted sum of several factors as the stability score:

1. Base Support Area Check ($BSA$): Stability of the package is proportional to its base area.
2. Center of Gravity Height ($CGH$): Stability decreases, and the chances of toppling increase as the center of gravity height increases relative to the package height or the ULD height.
3. Stacking Factor ($SF$): This is the ratio of the packages stacked above a package to the package's weight. The greater the stacking factor, the lesser the stability.
4. Placement Distribution ($PD$): Uniformly distributed loads across the ULD's base improve stability. This metric calculates the deviation of the load from the center of the ULD and measures its effect on tilting moments.

$$CGH = \textstyle\sum_i (y_i \cdot w_i) \qquad PD = dis_{XY}(\overrightarrow{c_{ULD}} \cdot \vec{r_c}) \text{ where } \vec{r_c} = \textstyle\sum_i (\vec{r_i} \cdot w_i)$$

$$BSA = \frac{lb}{\max(lb, bh, hl)} \qquad SF = \frac{stacked_i}{w_i}$$

The final metric is calculated as a weighted sum of all the components:

$$M = \alpha_1 \cdot BSA + \alpha_2 \cdot (1 - CGH) + \alpha_3 \cdot (1 - EFS) + \alpha_4 PD$$

## 2.5 ESTIMATION OF CUSHIONING MATERIAL

Despite pushing along the axes, there exists some spaces between packages by virtue of geometry. These spaces can be efficiently filled by cushioning material to avoid any package from sliding due to phenomena like turbulence and disturbances during transportation. The volume of this cushioning material should be minimised while maintaining stability.

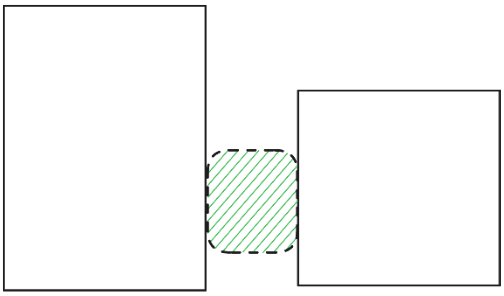

*Figure 9: Placement of cushioning material*

For computing the same, the adopted method tries to laterally search through the space of ULD and find the next nearest placed packages. Method then places a cushion between the intersecting areas in a way to minimize the volume used.

## 2.6 LOADING PLAN GENERATION

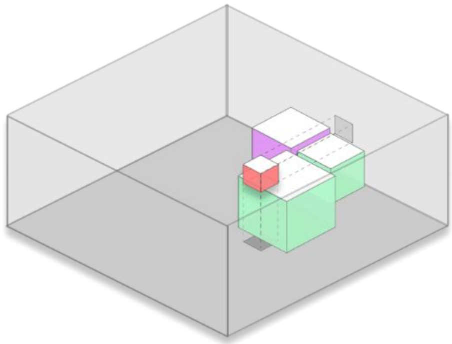

*Figure 10: Finding the dependencies of a package in loading*

To generate the loading plan for all the packages, we first construct a dependency graph of the packages, where a directed edge from $P_i$ to $P_j$ denotes that package $P_i$ must be placed before the package $P_j$.

To identify the dependencies for any package, we construct rays along the edges of the package that travel vertically downwards (toward the ground) and backward (towards the closed face of the ULD). Then, the first package that intersects with any of these rays is a direct dependency of this package (as it can be realized that the current package would either be vertically or laterally placed on this package).

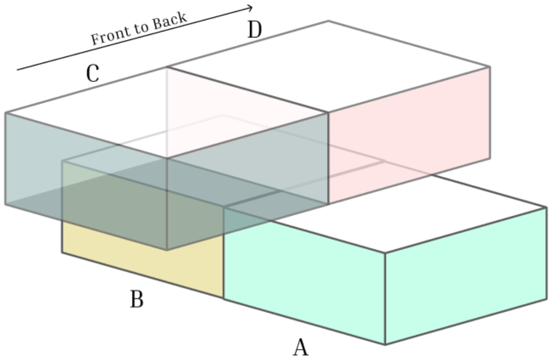


*Figure 12: Sample scenario while packing in an ULD*

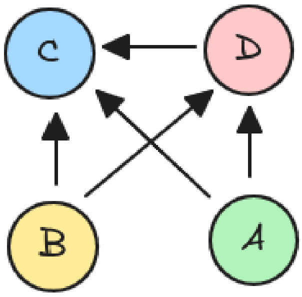


*Figure 11: Sample dependency graph generated*

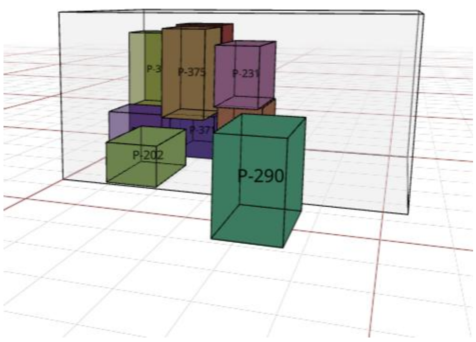


*Figure 13: Sample loading animation*

Finally, we perform a topological sorting on the generated graph to create an ordering plan. The placement strategy guarantees that the generated dependency graph is acyclic; thus, we are assured that a valid loading plan always exists in the solution. Even during the topological sorting, we try to prioritize the packages that are towards the back of the container and are at a lower height, so that the net movement of the operator while loading the packages if as minimal as possible. The UI dashboard developed is accompanied by a visualizer for the loading process and the ability to export the loading plan.

Loading Plan for ULD U3

| S. No. | Package ID | R1 | R2 | Orientation |
|---|---|---|---|---|
| 1 | P-14 | (64, 237, 146) | (109, 318, 192) | XZY |
| 2 | P-15 | (47, 103, 61) | (131, 152, 121) | XYZ |
| 3 | P-16 | (64, 255, 0) | (112, 318, 93) | XZY |
| 4 | P-17 | (0, 150, 0) | (63, 233, 57) | YXZ |
| 5 | P-23 | (134, 109, 0) | (244, 159, 51) | ZYX |
| 6 | P-25 | (167, 63, 129) | (244, 107, 182) | YXZ |

*Figure 14: Sample exported loading plan*

# 3. DELIVERABLES

The solution is accompanied by a Vite and React-powered dashboard that can be used to interact and generate the solutions. The backend is hosted on a Railway server with 4GB RAM. We can use the dashboard to upload the data related to the packages and the ULDs as a CSV. After uploading the same, you can use the compatibility screen to add additional constraints to the generated packages. The constraints can be added with the help of the provided UI or as a CSV in a pre-defined format.

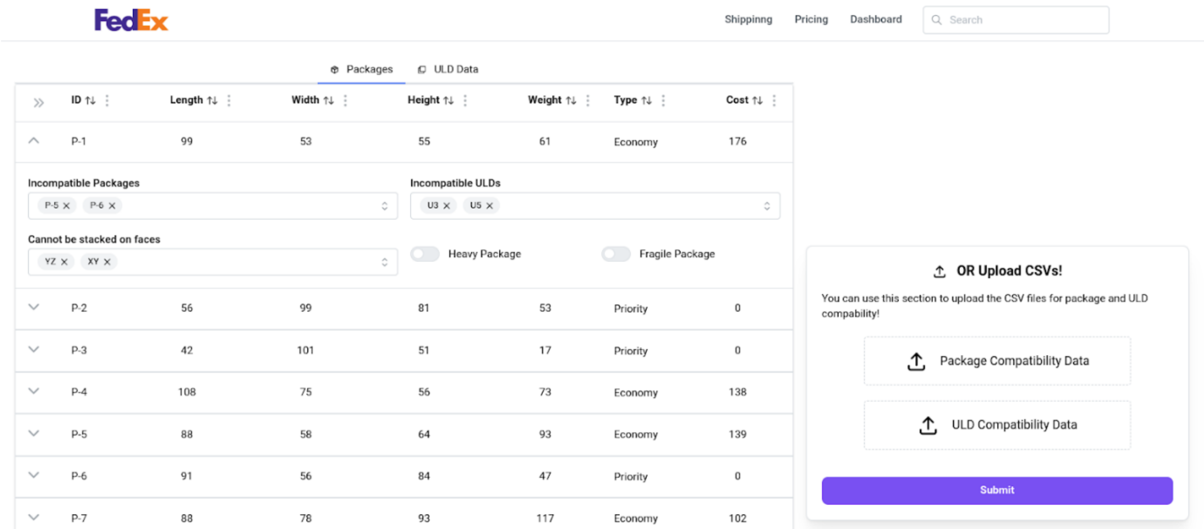


*Figure 15: Compatibility constraints definition screen*

The various constraints that can be defined include:

- Incompatibility between packages
- Incompatibility between packages and ULDs
- Faces of the packages which cannot be placed on the ground
- Heavy and fragile package constraints

After the constraints are added, a 3D arena is created where the users can manually see the packages placed in the ULDs, and as well as see the various stability and packing metrics associated with each ULD. The user can also view an animation of the loading of the packages, as well as export the loading plan as an PDF. The UI also supports viewing only a certain kind of packages.

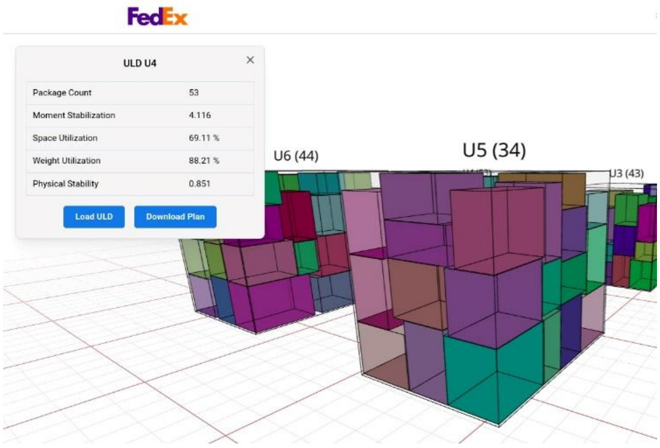


*Figure 16: Metrics for packing of ULD U4*

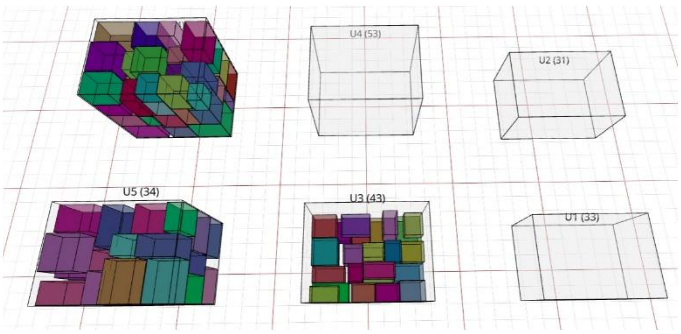


*Figure 17: Top View of packed priority packages across the ULDs*

The backend is written in Rust, which helps to ensure that the core genetic algorithm pipeline manages the memory directly, and thus is performant under large loads. The genetic algorithm as well as the placement strategy is written without the help of any external libraries to ensure that each atomic operation is as optimised as possible. The web server was created with the framework Actix.

## 4. INCORPORATION OF LITERATURE

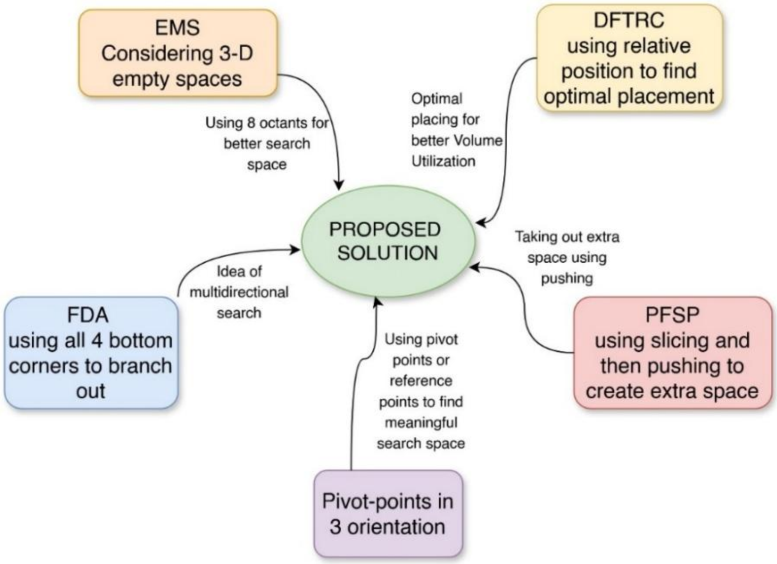


*Figure 16: Incorporation of various techniques from literature into the proposed solution*

Approaches from several papers were reviewed and inspirations were taken to create the solution:

1. From the 3-orientation pivot point strategy, the concept of pivot points was introduced which transformed into reference points. This resulted in major runtime optimization when compared to the brute approach of iteration over all integral points inside a ULD as candidate points.
2. The idea of initialization of search space from all corners was taken from the fit degree algorithm (FDA) for more holistic utilization of containers.
3. Empty maximal space (EMS) used a 3D space utilization strategy, which inspired the 8-octant search algorithm. The 8 octants provide more robust searching orientation.
4. Strategies like closest to origin and distance to the front top-right corner (DFTRC) were used to find optimal placement points. The proposed solution also uses similar criteria like height, overlapping fraction and distance from sides as tie breakers for choosing the optimal placement point and orientation.
5. Peak Filling Slice Push (PFSP) introduced the concept of slicing and pushing, where the pushing operation creates extra space.
6. We modified the encoding scheme used in BRKGA to fit the robustness of heuristic octa-branching algorithm. Proposed encoding encodes 2 different chromosomes in the form of packing order of priority and economy packages to randomise the order of the packages, rather than using a single chromosome for all the items. This leads to faster convergence of algorithm (has been validated by running multiple simulations of the contrasted variants of encoding)

Table 1: Comparison of Algorithms

| Algorithm | Approach | Packages Shipped | Cost of Shipping |
|---|---|---|---|
| Heuristics | Slicing | 185 | 41253 |
| | 4 Directional Greedy | 230 | 31432 |
| | 3 Point Orientation | 206 | 36832 |
| | 3 Point Orientation with Separate Strategy for Priority Packages | 210 | 38734 |
| Genetic Algorithm | 3 Point Orientation | 215 | 42232 |
| | EMS | 170 | 48181 |
| | 8 Directional without Pushing | 228 | 30534 |
| | 8 Directional with Pushing | 234 | 30393 |

# 5. RUNTIME ANALYSIS & SCALABILITY

## 5.1 TIME COMPLEXITY ANALYSIS

During a cycle of placement, the time complexity for each item to be placed is (where $n$ is the number of packages):

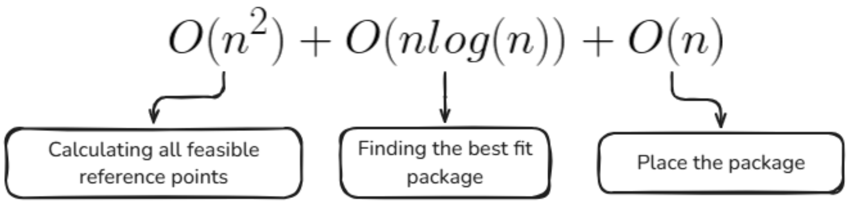


*Figure 17: Time complexity analysis*

Considering the entire population, the elite population is withheld while during the evolution of a generation. Thus, the final complexity is $O\left(pop\_size \cdot (1 - prob_{el}) \cdot (n^3 + n^2 \cdot log\ n)\right)$ for each generation. Considering a population size of about $256$, the total number of operations needed to be performed for one generation is of the order $10^8$, but due to high constant factor it takes about $7$ seconds to run. Thus, one can run about $500 - 600$ generations conveniently under an hour on standard hardware. As per the experiments, this time frame is usually enough for the algorithm to reach stagnation (indicating global optimum).

## 5.2 SCALABILITY & FUTURE SCOPES

- The calculation of fitness and evaluation of packing order for each member of the population are independent of each other. This makes the algorithm inherently parallelizable when run in multi-threaded environments. This would allow the algorithm to be scaled to a great population size and generations, and thus arrive at even better results.
- The stability of packages can also be increased by incorporating the use of rods and cables. These stabilizing mechanisms will be strategically placed to secure packages, preventing shifting, or toppling during transit by enabling better weight distribution and minimizing the risk of damage.

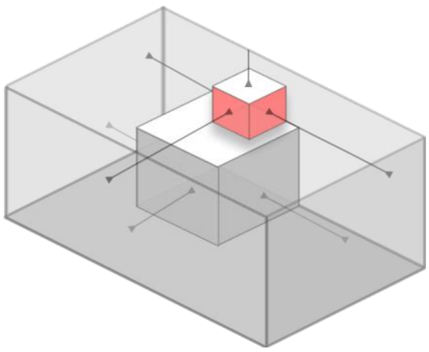

*Figure 18: Use of cables and rods in securing packages*

- This approach builds an offline loading strategy to plan the arrangement of packages. However, in real-life scenarios, packages often arrive dynamically, such as on a conveyor belt, requiring real-time decision-making for efficient loading. The incorporation of genetic algorithms in such online, adaptive, and real-time environments can be of great value.