Optimization of Last Mile Deliveries

Introduction

The Capacitated Vehicle Routing Problem with Mixed Pickup and Delivery (CVRPMPD) is a optimization problem that involves finding the most effective routes while maintaining the various constraints such as capacity, route lengths and route times. It is an NP-Hard problem whose time complexity increases exponentially with increase in problem size and hence cannot be solved using exact methods in a reasonable amount of time.
To address this issue, the OR-Tools library is employed. OR-Tools is an open-source optimization library that provides tools which offers an efficient modelling framework to model the given problem while providing us with the flexibility for including contextual constraints and variables and gives results with great bounds in a reasonable time frame.

Modeling the problem

Step 1 : Problem Set-up

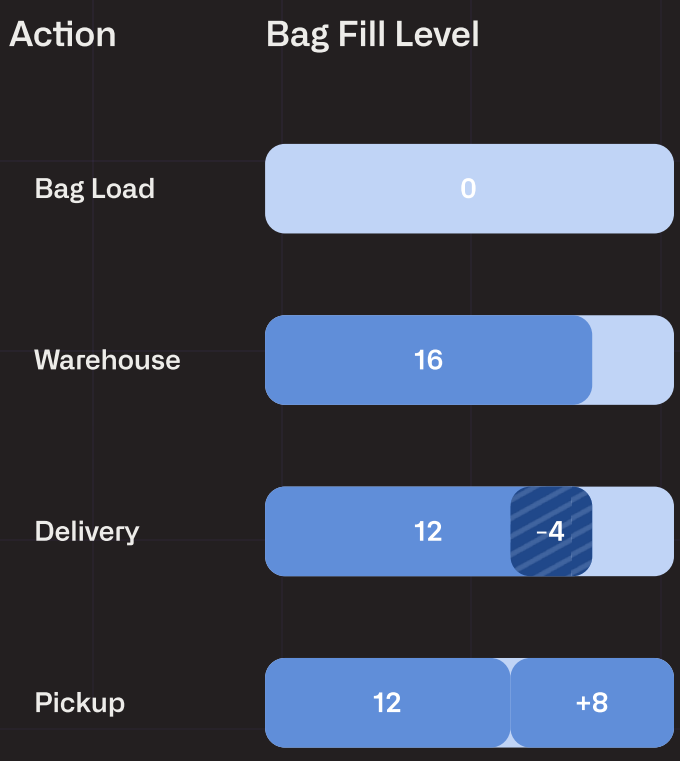
The decision variables in this problem are the routes that the vehicles take which minimized the total distance travelled while ensuring that all the customer demands are met without violating any constraints.

The distance matrix and the time matrix of the orders are calculated using OSRM and subsequently fed to the solver.

Step 2: Creating Constraints

Next, constraints must be defined that ensure the problem is solved correctly.

- Since both deliveries and pickups can be included in the same routes, we model this by giving **Positive** weights to Pickups and **Negative** weights to Deliveries. Then we set up constraints to ensure that the cumulative load at any customer location does not exceed the maximum capacity of a vehicle.
- The second set of constraints ensure that the time duration of a route does not exceed 5 hours. This is done by ensuring that the cumulative value of the time taken along a formulated route does not exceed the specified limit.



- The third set of constraints ensure that the total route length of a route does not exceed the given maximum limit. This is done by ensuring that the cumulative value of the route length along a formulated route does not exceed the specified limit.
- The final constraint is the start and end point of the driver, which is initially the depot for both the start and end points.

Step 3: Adding Disjunction Penalties

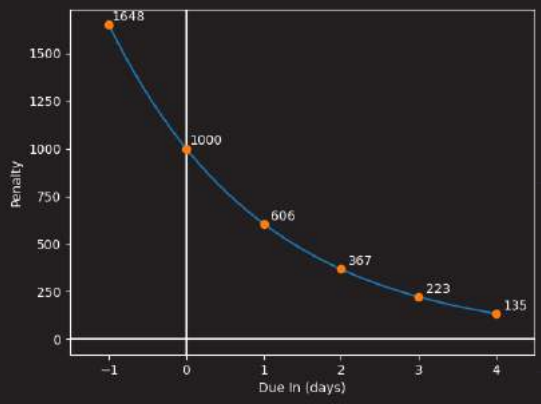
A disjunction is a variable that the solver uses to decide whether to include a given location in the solution. If the solver decides not to include a order in today's dispatch, the the objective function is penalized. Higher the drop penalty, less are the chances of the node to be dropped and carry forwarded to the next day.

Additionally, instead of solely scheduling deliveries based on their expected delivery date, we consider the order density as a factor in the solution.

The main objective of the optimization problem would thus become minimizing the distance of the routes and at the same time minimizing the dropping penalties of orders which were not scheduled. The dropping penalty has two factors:

1. EDD

Including this factor in dropping penalty will enable us to prioritize orders with imminent delivery dates. Thus, an exponentially decreasing EDD penalty function will allow us to penalise the algorithm for missing out on deliveries scheduled immediately while, at the same time, allowing it to drop the orders scheduled a few days from now without major retribution. Mathematically,



$$P_{EDD} = 1000 * e^{-\frac{\text{Due In (days)}}{2}}$$

2. Location

An order in nearby areas with a high order density has a high priority value, as drivers delivering in nearby areas can piggyback this order, saving fuel and time and reducing trips. To include this factor, the city is divided into tiles based on Bangalore's ward data.

But, there might be cases where an order lies in a ward with a low order density but the neighbouring tiles have a very high order density. To overcome this problem each tile's priority is calculated by taking into consideration the priority of its neighbours' as well.

Suppose we have n tiles. We define the priority vector $\mathbf{v}_{\text{initial (nx1)}}$ where

$$v_{\text{initial},i} = \text{fraction of orders in tile i}$$

Then, we set **A** as the n by n weighted adjacency matrix, considering tiles as nodes and weights as:

$$A_{ij} = \begin{cases} 1 & i = j \\ \frac{c}{\text{time}(i,j)} & i \neq j \end{cases}$$

Here, c is a constant derived experimentally, such that $\mathbf{A_{ij} \leq 1}$ and **time(i, j)** returns the average time of traversing from tile i to tile j.

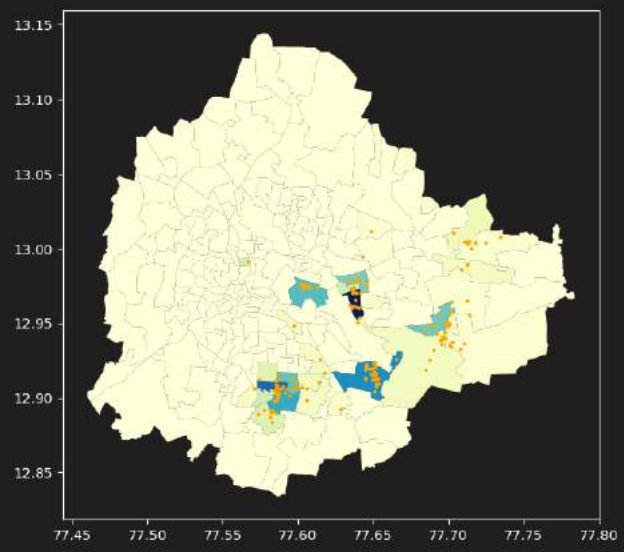
We then calculate the final priority vector of tiles, by :

$$v_{\text{final}} = A \cdot v_{\text{initial}}$$

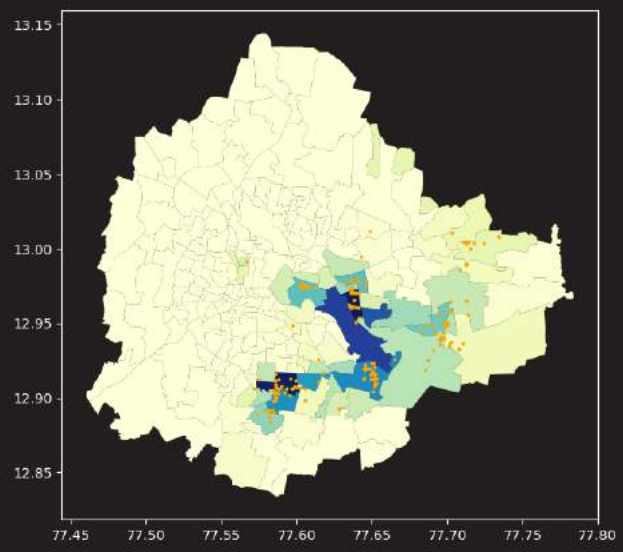
This will allow the order densities to propagate from tiles with significant order densities to nearby tiles thus allowing us to get a likeliness index of a tile being visited by a rider.

The location priority of an order: **p,** thus can be defined as $\mathbf{v_{final,\,i}}$ where i is the tile corresponding to the order location. Then we define the penalty for location as

$$P_{\text{Location}} = e^{7p}$$



Initial Priority



Final Priority

Combining the penalty values obtained

$$P_{Dropping} = P_{EDD} + P_{Location}$$

Example

Example	Case 1		Case 2	
	Order 1	Order 2	Order 1	**Order 2**
EDD - TODAY	2	3	2	3
Location Priority	0.4	0.6	0.2	0.8
Dropping Penalty	**383**	290	371	**494**

Step 4: Setting up the parameters

In OR-Tools, the parameters of the optimization model can be set to control the behavior of the solver and influence the solution obtained. The parameters include various elements such as the method used to find the initial solution, local search options (metaheuristics), the time limit for finding a solution and the limit to the number of solutions searched.

Step 5: Running the solver

The solver is then run to solve the problem. Upon completion the solver returns a list of nodes for each vehicle in the order in which they are supposed to visit them.

Flexibility

Balance Vehicle Utilization

Economically, it makes sense to minimize the number of vehicles used, but there are cases where timely delivery of the orders has to be prioritized in which case our solution has the ability to balance the vehicle loads among all the available vehicles to minimize the time of delivery.

A two step approach was employed to achieve this:

1. Minimize the maximum number of orders carried out by a vehicle.
2. Penalize the solver if the number of orders for a vehicle exceed

Time Windows

Incorporating time windows in a Capacitated Vehicle Routing Problem (CVRP) can add a temporal dimension to the routing problem, making it more realistic and allowing us to incorporate EDT (Estimated Delivery Time) for the orders. The solver is encouraged to prioritize making routes which enable the deliveries to be completed in the time window to avoid the penalty.

Incorporate Weather Conditions

Incorporating area-wise weather data into a Vehicle Routing Problem (VRP) can help make the routing solution more robust and efficient. The idea is to incorporate the hyper-local area effects of weather conditions on the routing distances, so that the VRP model considers the weather data when optimizing the routes. One way to do this is to increase the distance matrix cost for points that lie in areas with bad weather conditions. In this way, incorporating weather data into a VRP can help ensure that the routes are optimized for both distance and safety, taking into account the impact of weather on the road conditions.

Service Time

In addition to travelling time, we can include the time of servicing orders as well. Orders may take more time in certain locations such as societies due to time taken to do entry which may be included in the solver while forming routes which will maximise the percentage of successful deliveries in the predicted time.

Maximum Route Distance and Travel Time

While calculating the routes, the solutions allows us to restrict the maximum travel length of a particular route as well as the time of travel of the route. This is achieved by putting a constraint on the cumulative distance and cumulative travel time for routes to be less than the maximum limit.

Dynamic Pickups

Initial Condition

After the initial routes are created, the drivers set-off on their routes. As the driver goes on completing the orders, the status of the deliveries are constantly being updated in the solver.

When dynamic points are added and a rerouting request is created, we have the following information:

- Remaining orders
- Initial routes for each vehicle
- Current position of each vehicle
- Current capacity of each vehicle

Additional Constraints

- We add constraints on the start point of each driver as the current location and the end point being the depot.
- Another constraint that has to be applied is that since the rider already has the delivery orders in his bag, these orders cannot be assigned to different vehicles. Hence we add a constraint that delivery orders have to be assigned to the previously assigned vehicles.
- Since we already have the routes from the initial assignment, we provide the initial solution to the solver as the initial route minus the orders already delivered by the rider to improve performance.

Note: Pickup orders that have not been completed can now be assigned to a different driver to give more optimal results.

Volume Edge Cases

1. Imagine a scenario where the rider has attempted the delivery of an order and it failed, since the delivery can't be reattempted today, the space occupied by the order is technically 'blocked' until the orders are unloaded at the depot. Similarly, once a pickup order is completed, the space in the bag is 'blocked' until the orders are unloaded at the depot.
2. To overcome this challenge, two variables are maintained for each vehicle, namely *maximum_capacity* and *actual_capacity*.
3. To get the initial routes, we consider the *maximum_capacity* of the vehicles. But during rerouting, we directly consider the *actual_capacity* of the vehicle, hence simplifying the problem. Once the rider reaches the depot, the *actual_capacity* is reset to *maximum_capacity*.

Optimization Model Performance

Testing Against Benchmark Datasets

Benchmark Results

The CVRP solver was run on the various problem sets from the Uchoa et al. (2014) benchmark. The time taken to obtain a solution ranged from 4 seconds to 200 seconds with the optimality gap ranging from 4% to 12%.

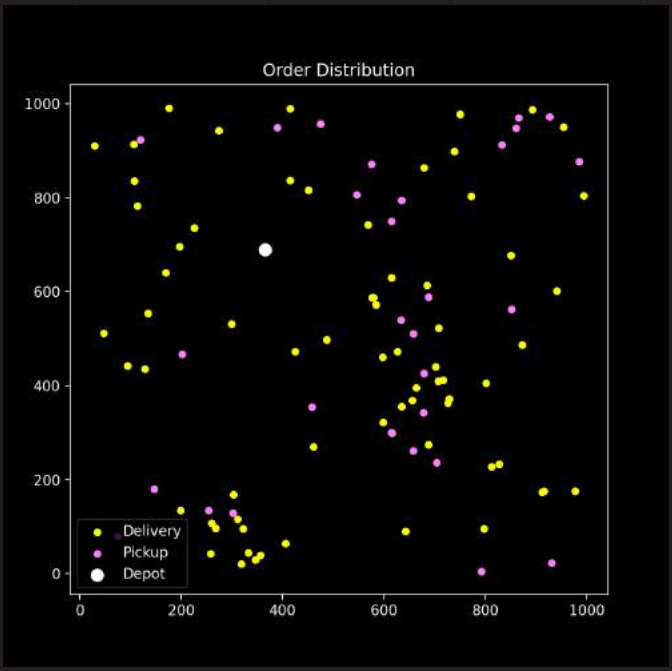
Benchmark Instance	Nodes	Best Known	Result	Gap (%)	Time Taken(s)
X-n125-k30	125	55539.0	58198.4	4.78	4.45
X-n266-k58	266	75478.0	80286.0	6.37	30.25
X-n313-k71	313	94043	99075.7	5.35	66.05
X-n536-k96	536	94846	105773.5	11.52	59.78
X-n749-k98	749	77269	83716.2	8.34	102.81
X-n979-k58	979	118976	124390.9	4.55	197.98

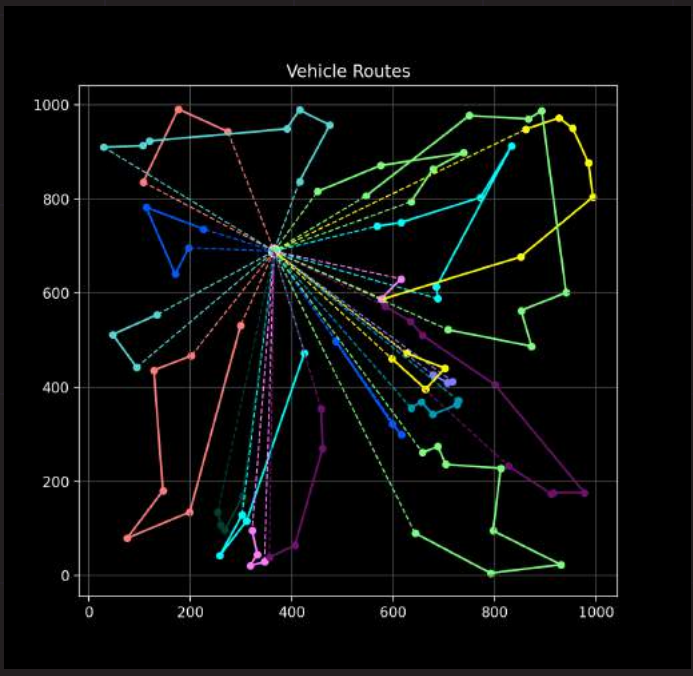
The results are expected to approach the Best Known value if allowed to run for more time.

Result Plots

Various distributions were simulated and the routes were calculated as follows:

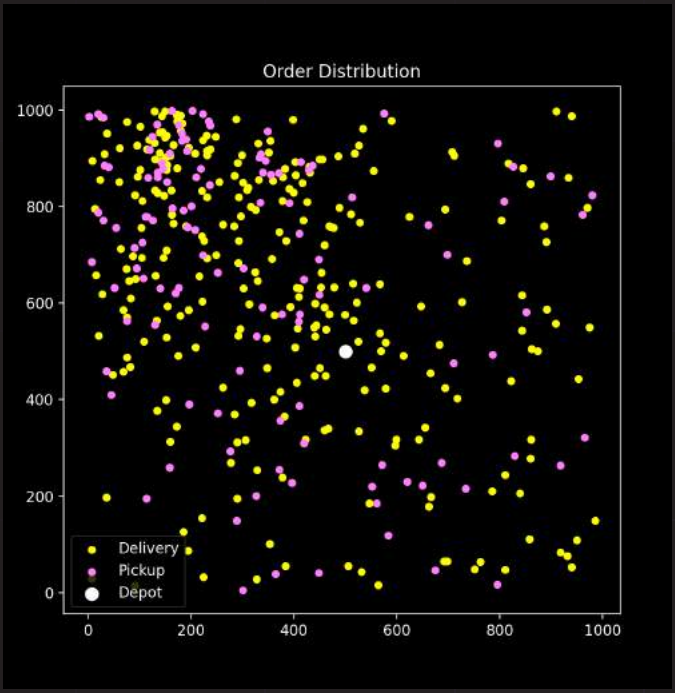
1. Uniform Distribution

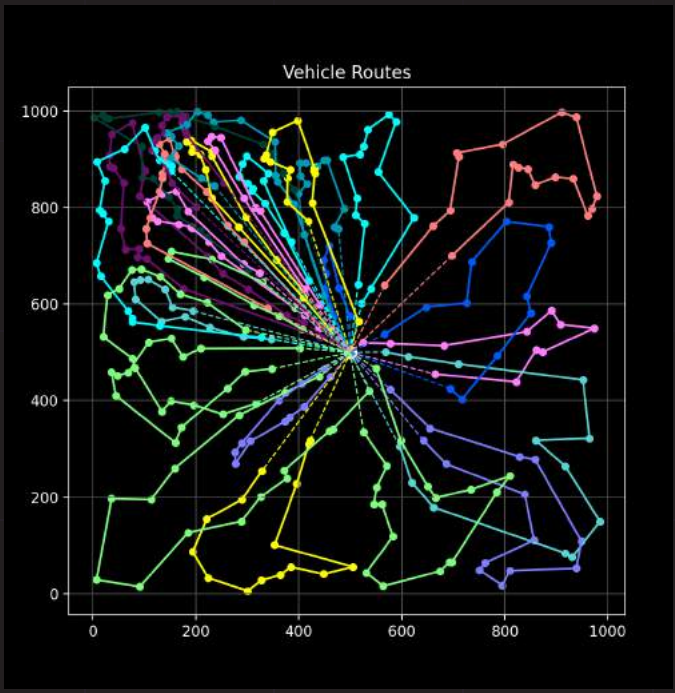




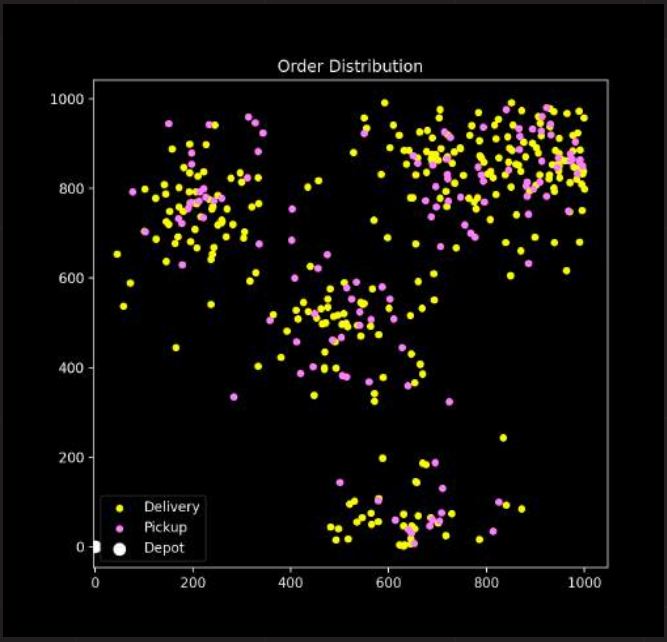
Optimization Model Performance

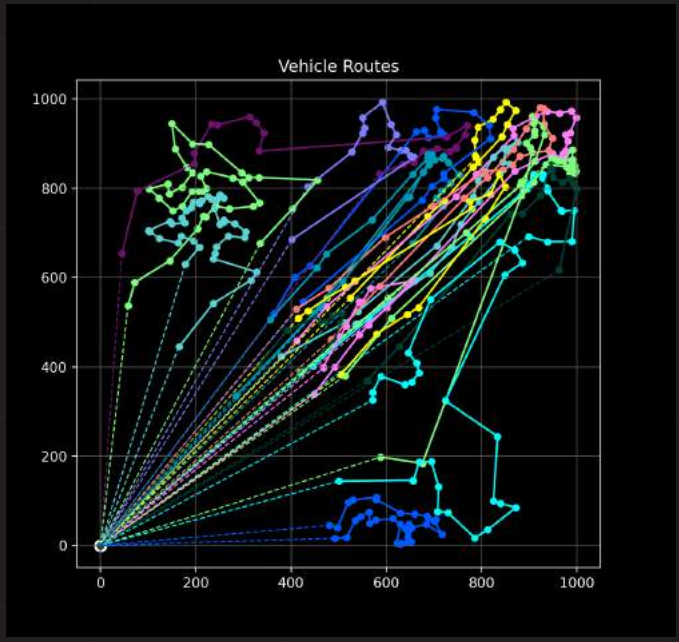
2. Dense and Sparse Distribution





3. Clustered & Corner Depot Distribution





Scalability

Introduction

The CVRPMPD is an NP-hard problem, where even state-of-the-art solvers like OR tools struggle with large-scale datasets, around 5000 nodes or so. As the company scales up in cities, 5000+ orders are common, which will hamper scalability significantly. Clustering the nodes to represent virtual customers reduces the complexity by a large amount and allows the CVRP solvers to handle very large instances with low optimality gaps.

Advantages of Clustering

- Reduced complexity: Clustering allows for dividing a large problem into smaller, more manageable sub-problems, making it easier to solve the VRP.
- Improved scalability: As the size of the VRP increases, clustering can be used to efficiently handle larger problem instances by dividing the problem into smaller sub-problems.
- Flexibility: Clustering algorithms can be adapted and modified to incorporate specific constraints and requirements of the VRP, making it a highly flexible solution approach.

Proposed Method

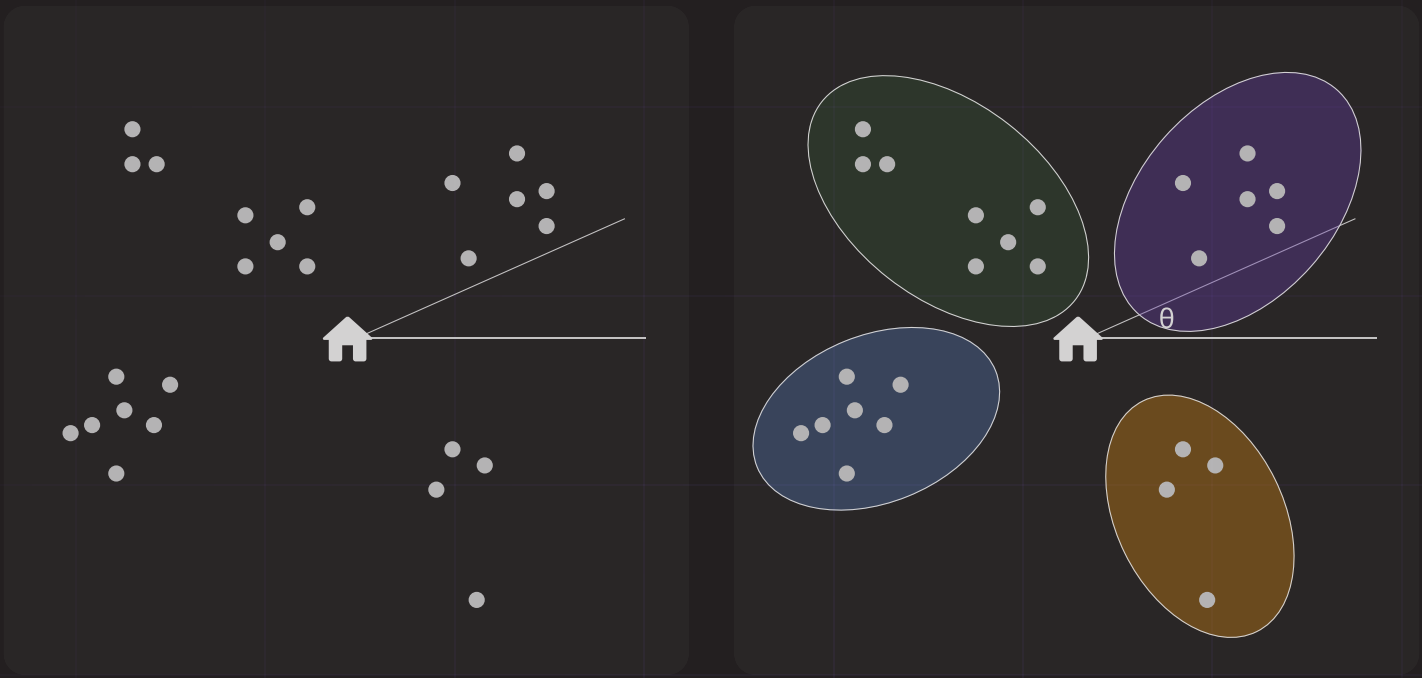
Each order in a cluster can be represented at the centroid of the cluster, and the set of clusters can be used as a warm-up solution for solvers to route. Within the cluster, each cluster can further be solved using VRP.

We use a divide-and-conquer strategy to solve the routing problem and account for future scalability.

We have considered the sweep algorithm, a standard for vehicle routing problems, due to its robustness.

Sweep Algorithm

The Sweep Algorithm is a clustering method used for grouping geographical coordinates based on their cosine similarity to each other with respect to the depot. The algorithm works by sorting the orders according to the polar angles and iteratively sweeping a line in an angle and grouping orders together into clusters until the capacity is satisfied. The Sweep Algorithm is efficient and scalable, making it suitable for large datasets, and it can be easily adapted to incorporate specific constraints and requirements.



Suppose we have a total of 2000 orders and we have 80 riders. Let us cluster the orders into 4 clusters of 500 orders each with 20 riders assigned to every cluster.

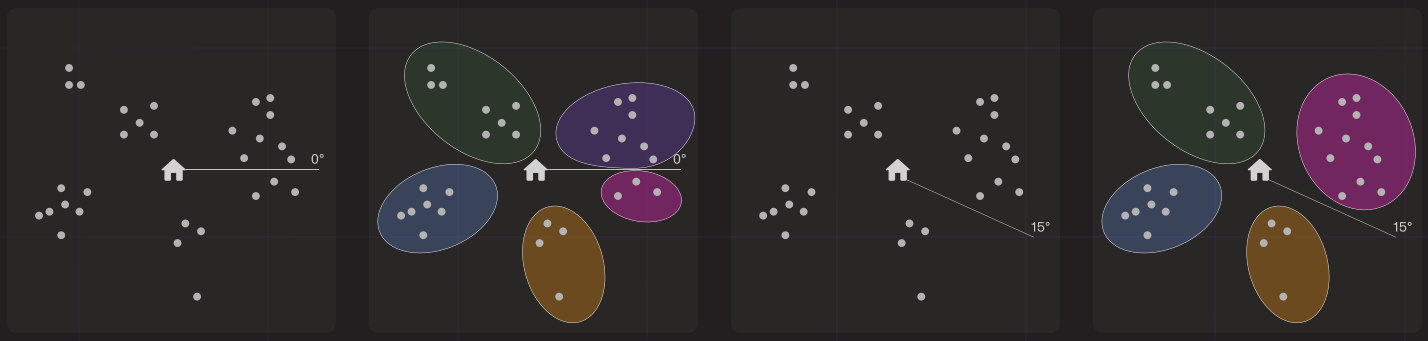
```
coords[Depo] = [0, 0]
θ_i (Polar Angle) of i^th order = tan^-1 (lat(i) / long(i))
Sort customers according to θ in ascending order
Set cluster_id = 1
Set θ_c = 0°
Sweep customer by increasing θ_c , add customer to cluster.
Stop the sweep when we reach 500 nodes in the cluster.
cluster_id = cluster_id + 1
Repeat above 3 steps till we have completed allotment of all customers
```

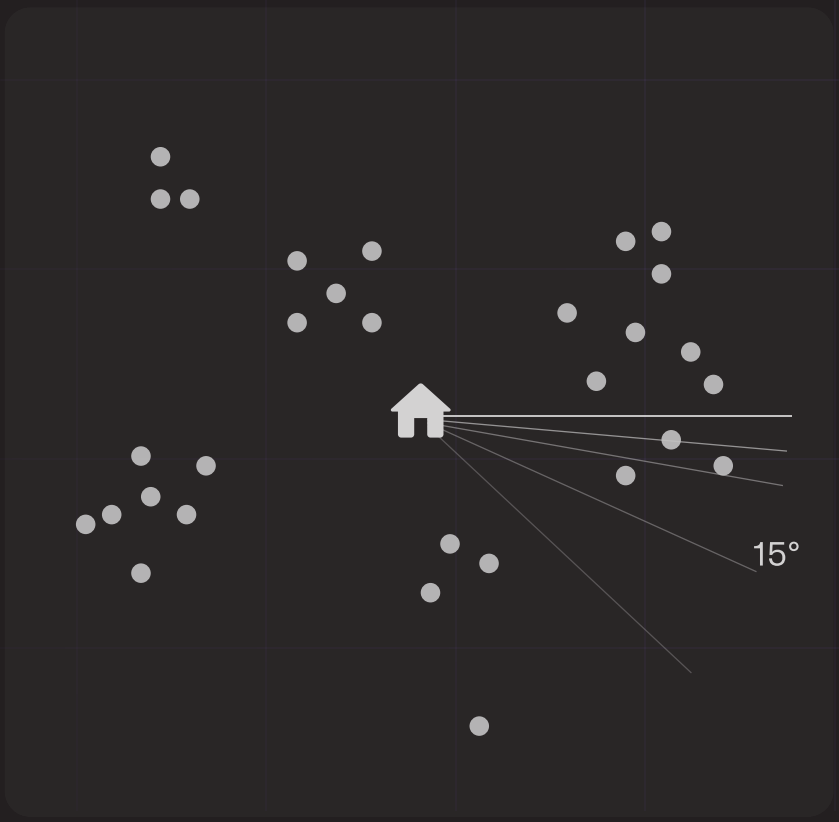
Potential Pitfalls

Choosing the starting angle θ_c = 0° is not always efficient.

Consider the following example where if we start from 0°, we need to have 5 clusters instead of the optimal 4 clusters. Also, a clustering is considered good if each cluster has a small overall spread. Hence, to overcome this issue, we check the clustering for θ_c in increments of 5°.

The overall complexity of the algorithm will be O(n*logn) where n is the number of orders.





To address this we can calculate clusters by varying the start point by 5° every time and calculating the number of clusters and spread of points we can evaluate how good the starting point is.

Bin Packing

Our algorithm uses a predefined order of box insertion in the container. Efficiency is improved by sorting based on volume, dimension or area of the largest face.

In our case priority of orders place a very important role so we are sorting the items based on priority of orders considering the one that should be delivered last should be placed on top.

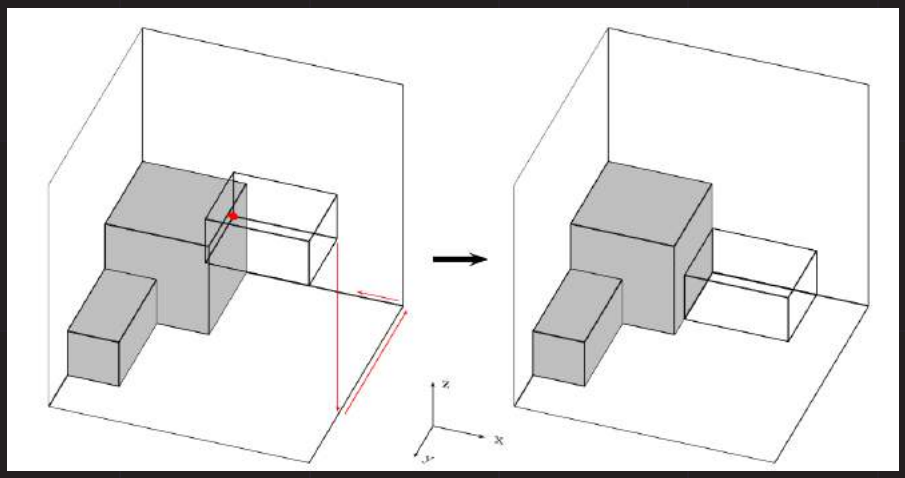
Methods Used:

Extreme Point Insertion:

The algorithm starts by setting a default IP of (0, 0, 0) and adding it to a set of available IPs. For each piece, the set is used to find possible insertion points in the container. The algorithm loops through all pieces in a set order and tries to insert them, rotating the piece in all allowed rotations. It checks if the piece can be loaded at each insertion point and calculates a score based on specific metrics after each packing. The process repeats until a better score is obtained or a timeout occurs.

Push Insertion:

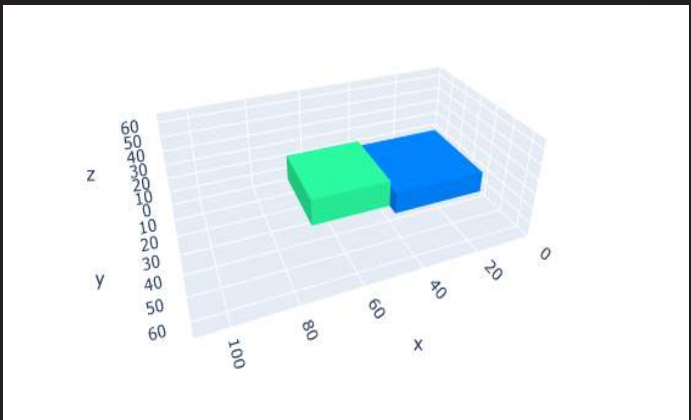
The concept relies on normalizing by pushing pieces towards the container's origin along defined axes. The order of the axes can be set as a parameter and must include all axes. The algorithm loops through the given pieces in order, inserting each one without a set of available insertion points, but generating them on-the-fly.

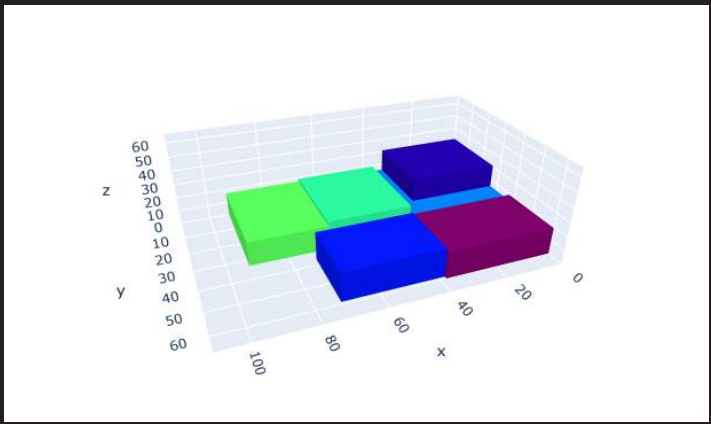


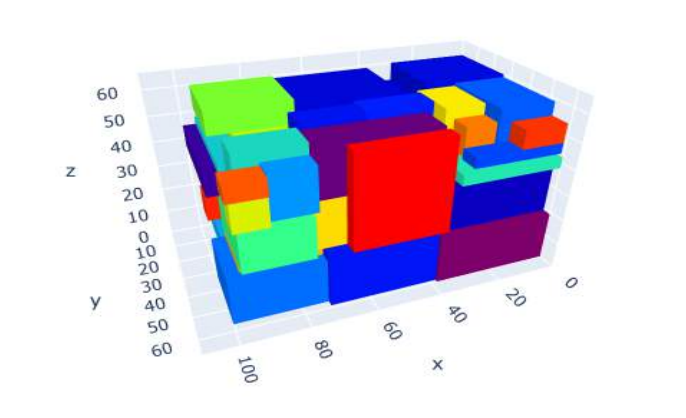
Initially the piece is inserted at a corner of the container and the next step is normalizing it if the insertion was successful.

Greedy Adaptive Search Procedure:

The solution generated from above heuristics is optimized by trying different permutations of the order of blocks and orientation. Pieces are assigned a score based on the initial order, which is modified and re-sorted after each iteration. If no improvement is made in a set number of iterations, the score is reset to the best solution found so far.







Evaluation Metrics:

- Minimize residual volume of container after packing.
- Minimize the base area for the items to allow for easy pickups.
- Maximize the residual space's utilization of the resulting packing: The method aims to fit new pieces best by tracking residual space at each extreme point by projecting three axes until hitting another piece in the container.
- Minimize the Euclidean distance to the origin to increase the density of blocks.

Reference paper: https://github.com/merschformann/sardine-can/tree/main/Material/MasterThesis