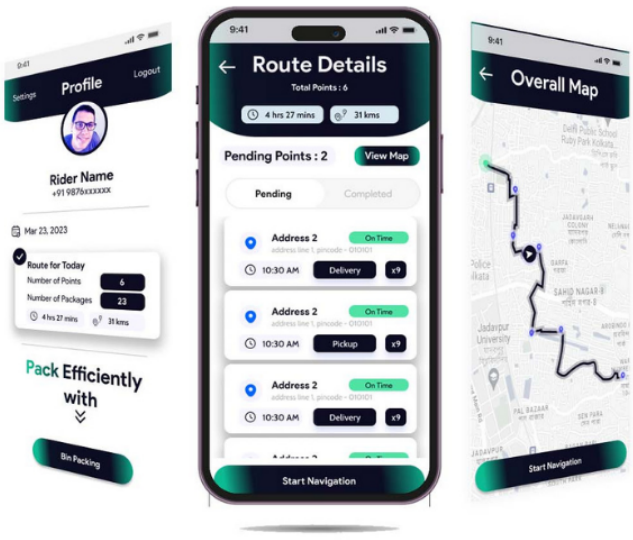
Grow Simplee



ROUTE PLANNING FOR OPTIMIZED ON TIME DELIVERY

Team 11

Dashboard Demo

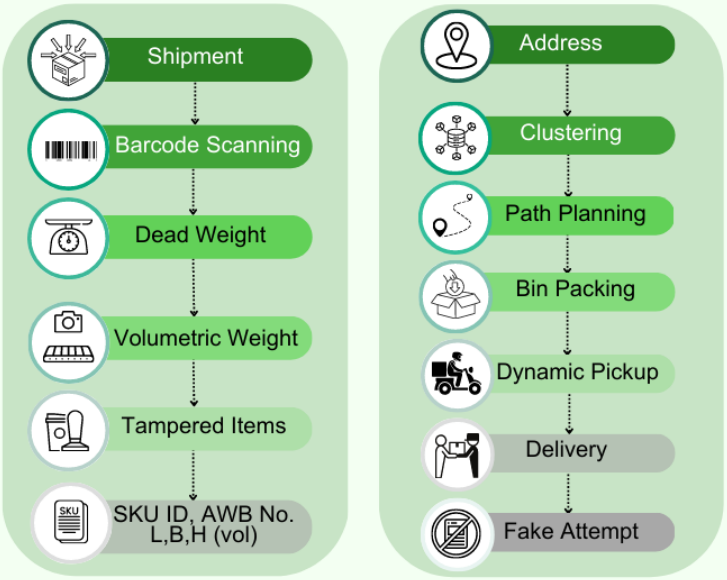
App Demo

Click here

Introduction

In the e-commerce industry, the last mile of the delivery process (wherein packages must be transported from the warehouse to their final destination) has become critical due to a boom in the customer base. To continue in the business, it has become vital for companies to ensure efficient, on-time delivery to the customer. Hence, for this problem statement provided by Grow Simplee, an optimized and affordable solution that skillfully automates processes for both the warehouse and the driver is being built.

Workflow



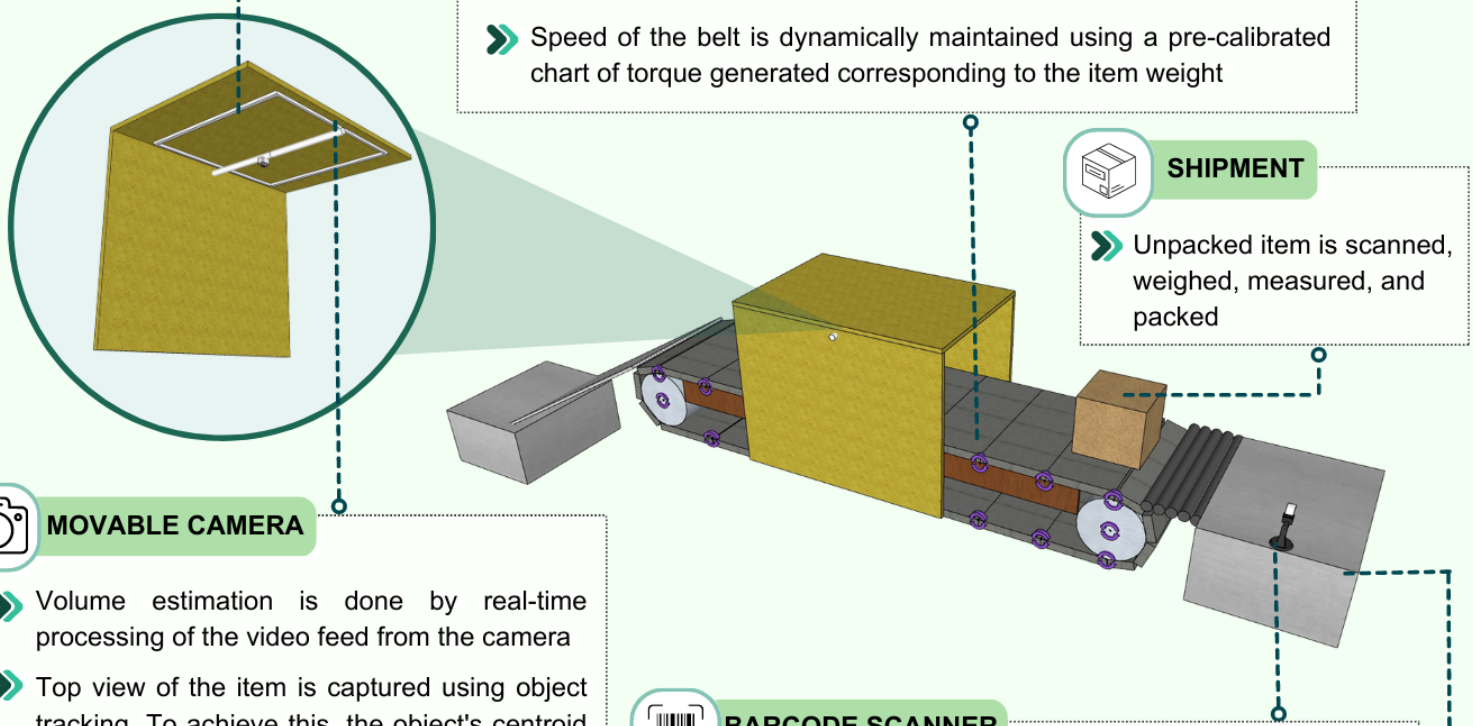
Computer Vision

LED LIGHTS

- Control lighting conditions

CONVEYOR BELT

- Auto-move the object to the camera setup for volume estimation
- Speed of the belt is dynamically maintained using a pre-calibrated chart of torque generated corresponding to the item weight



SHIPMENT

- Unpacked item is scanned, weighed, measured, and packed

MOVABLE CAMERA

- Volume estimation is done by real-time processing of the video feed from the camera
- Top view of the item is captured using object tracking. To achieve this, the object's centroid is aligned with the camera's focal axis

- We designed this single depth camera setup among other available two camera, 3D LiDAR or laser sensor mechanisms to optimize the processing time and cost of the volumetric measurement of the items
- Depth of the object is calculated from the z-component of the point cloud of the object
- Background removal on the center-aligned object ensures an accurate contour detection around the item. A minimum-area bounding box from the contour gives us the dimensions of the item

BARCODE SCANNER

- Identify the item using Laser 1D barcode reader through the decoded SKU ID and AWB No.
- Alternatively, the image can be processed from the camera. However, this involves inaccuracy due to noise, poor lighting and other unfavourable conditions

LOAD CELL

- Calculate the dead weight of the item with a strain gauge load cell, HX711 amplifier, and Arduino setup
- Convert the lateral strain force into an electrical signal which is captured by the Arduino board, then amplified by HX711, and finally processed in a digital format

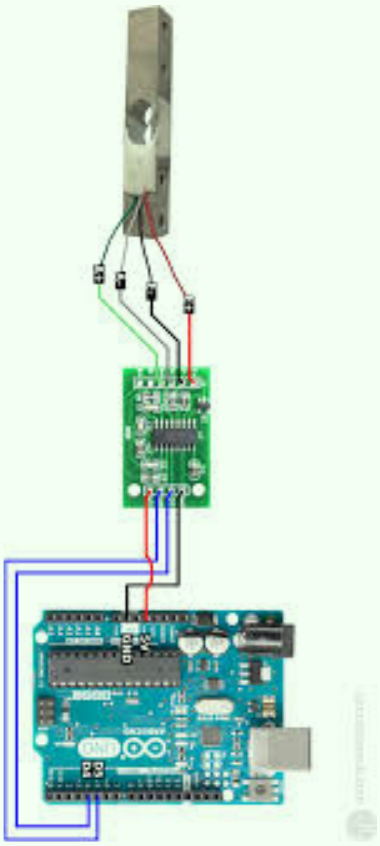
COMPUTER VISION

Barcode scanning

We start with identifying the items in the warehouse by scanning the barcode on the item and fetching the embedded AWB no. and SKU ID using a Laser 1D barcode reader. A low cost alternative to a laser scanner would be a camera where the identified barcode is decoded using single shot detection. However, this alternate technique involves inaccuracy due to noise, lighting conditions and other unfavorable elements of the image.

Dead Weight Estimation

Further for dead weight computation of the item, we use a strain gauge load cell along with an HX711 amplifier and arduino setup. To calculate the weight of the object the load cell converts the lateral strain force into an electrical signal. Since the changes in strain while weighing objects is small we use an HX711 amplifier to amplify the signals. An arduino board is used to perceive the amplifier's signals and process them in a digital format. For accuracy, we calculate the calibration factor of the load cell using a known weight and maximum error is reduced to 0.3%.



Volume Estimation

We are using a conveyor belt setup with a movable Intel RealsenseD455 depth camera in a controlled environment to have accurate volume estimation with minimal human intervention.
From the depth camera we retrieve the colour frame as well as the depth information. The colour frame contains the RGB image information which is equivalent to a feed from the normal camera. The depth frame is used to retrieve the depth at each point on the frame visible from the camera to estimate the height.

The colour frame is used to identify the position of the box and estimate its length and breadth. This is done in several steps for robustness.

Background Removal

Firstly a background removal process is applied to the whole frame so as to highlight the region of interest. Here the package box is the region of interest (ROI) for which we obtain a mask, where the background remover removes the entire background besides the object.

For background removal purpose we have used a deep learning based approach which uses a UNet architecture. U2Net is a machine learning model that allows us to crop objects in a single shot. Taking an image of a object it can compute an alpha value to separate the background from the panoramic view. We presently use U2NetP version of U2Net which is a lighter version. It has a two-level nested U-structure architecture that is designed for salient object detection (SOD). The architecture allows the network to go deeper, attain high resolution, without significantly increasing the memory and computation cost.

Identifying the region of interest

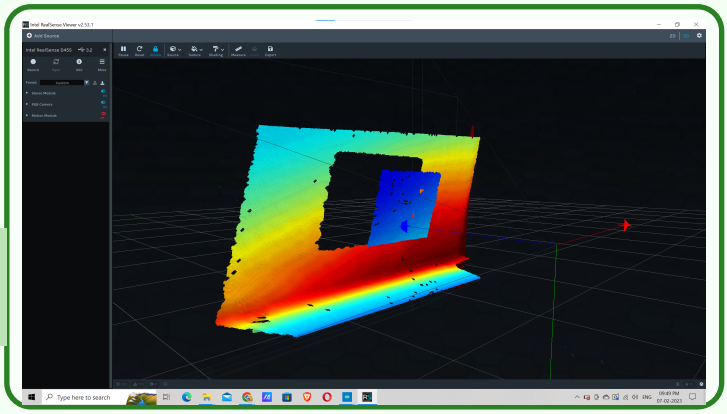
The camera has been fixed on the top, and since it has a large FOV (field of view) it includes objects in its view which are of no interest. External objects in the FOV often hinder the working of the formulated algorithms. So we crop out a fixed pre-set amount of image from the side-views of the image based on our physical setup. This gives a more centre specific and focused field of view.

In cases where the object is not found, the mask returned is noisy on the edges for which we crop out the middle 2.5% - 97.5% of the frame to check if any region of interest has been created. The frame is returned as a plain background when no ROI is found and no further processing takes place. When a ROI is detected, the mask is binary thresholded using a pre-set calibrated threshold value which enhances the ROI to a rigid mask of values 0 and 255 only.

Contour Detection

Then a contour detection algorithm is applied to identify the entire shape of the ROI which in our cases is representing the unpacked symmetric object. A contour here can be simply defined as a curve that joins a set of points enclosing an area having the same colour or intensity. The intensity change at the edges is detected and points are marked along the same group of points. Now over the detected contours we fit a rectangle with a minimum area so as to get the tightest fitting bounding box around each object detected.

Original Image



Point Cloud

Noise Removal

When there is no primary object in focus within the field of view of the camera, the background remover returns a noisy binary mask. It tries to find the image foreground but ends up returning noisy and small contours. The noise is removed based on area-thresholding technique, where detected small and noisy contours are filtered out based on a pre-set threshold area value, retaining only large and prominent contour.

Camera Movement and Arduino

Among all the contours with bounding boxes, we have selected the one nearest to the center-line of the camera. Center-line here means the middle of the x-axis of the camera frame. This is particularly done to get the object in focus while there are other objects in the frame as well. Other objects in the frame correspond to the multiple objects moving on the conveyor belt at the same point of time. We maintain our focus on the object nearest to the center-line so as to maintain a particular order of processing. Now, when this center-most object is within a 10% range of the center-line, a command is sent to the conveyor to stop. This command is sent to the arduino using the serial communication framework.

Height Estimation

When the conveyor stops, this is the time we make use of the depth frame, which provides a depth estimate of each corresponding pixel in real-life. Taking information from this we separate out noisy components which consists of mainly high peaks and low dips. We applied simple filtering of high peaks and low dips based on a calibrated threshold factor. The depth is estimated using the simple fact that the lowest point visible is the ground itself and the highest point visible on the depth-map would correspond to the height of points on the top surface of the object. The absolute difference between the maximum point and minimum point in the depth-map multiplied by a pre-set calibrated depth factor gives the height of the object in real-life. This completes the height estimation process.

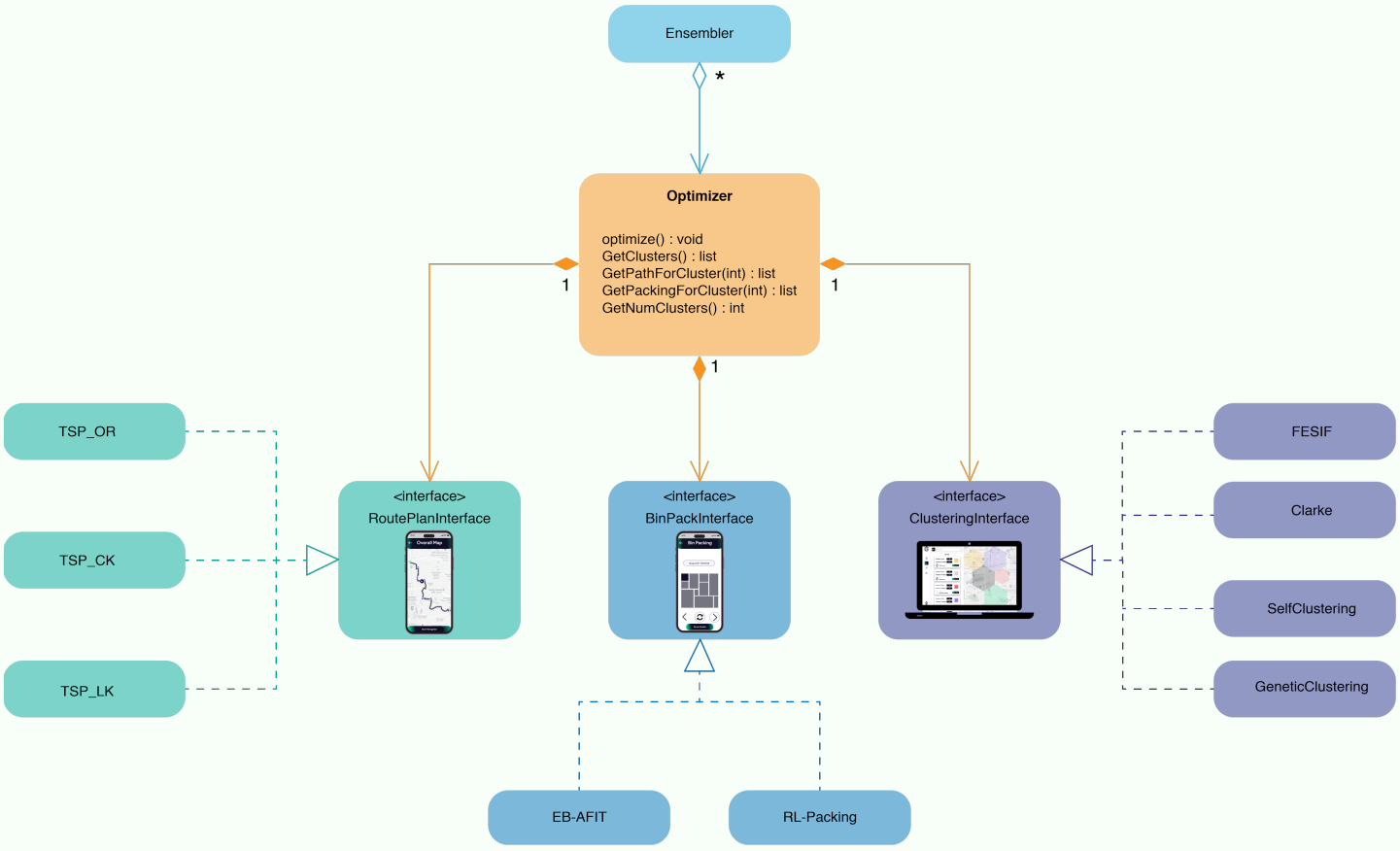
Length Breadth Estimation

So we move the camera in a perpendicular direction to the direction of the conveyor. We then use the same technique to find the bounding boxes as previously mentioned while identifying the position of the box. The camera moves and captures video for a fixed duration while travelling in a to-and-fro motion. The captured video frames are then used to identify the position of the object in each frame and get the particular frame having the least area of the bounding box created around the object. This gives us the best top-view image with the least side-view perspective coming into view. The error while measuring the dimensions would be the least from this top-view frame. Using the bounding box, we get the height and width of the object's tightest bounding box in terms of pixels. Using a pre-set calibrated length-per-pixel factor and a trigonometric formulation taking into account the height of the object measured previously, we get the length and width of the object in real length.

After computations of the dimensions, the camera would have reached back its original position and the conveyor is now commanded to move for a pre-set time of 1 sec to allow for the next object to be analyzed. Now the process continues for other objects .

Shape	Length(cm)	Error	Breadth(cm)	Error	Height(cm)	Error	Weight (g)	Error
Cuboid	16.5	1.10%	9.1	0.90%	5.2	1.70%	8975	0.30%
Cylinder	8.3	0.70%	8.4	0.80%	10.7	1.30%	2974	0.20%
Prism	18.4	1.90%	18.6	1.80%	16.2	1.90%	891	0.10%
Sphere	12.9	1.50%	12.6	0.80%	12.7	1.10%	754	0.20%

OPTIMIZER SYSTEM DESIGN



IDEATION

- We have designed our system keeping in mind that we were developing a solution for a fast-scaling startup working on a complex problem. Our design was motivated by three core principles:
  - Performant Scalability
  - Easy Extensibility
  - Rapid Testability.
- Route optimization for last-mile delivery has three primary subproblems:
  - Clustering of delivery packages
  - Routing in a cluster
  - Bin-packing of a route

SALIENT FEATURES

- With this architecture of the main optimiser class, we can better visualise the workflow of the algorithms. From a developer's perspective, this is very helpful, especially for adding newer features and debugging.
- We can easily add multiple solutions to our current codebase and efficiently combine the solutions to get the most optimal solution.
- A very strong advantage on our front is that we can run all the algorithms parallelly on multiple threads, making it much faster and enabling the exploratory algorithm to do more broader explorations.

IMPLEMENTATION

- Each of the subproblems is an established problem in computer science with a breadth of approaches and state-of-the-art algorithms to choose from. Hence to enable easy integration of new algorithms with no changes to the existing design, we have **designed an interface** for each. These have been represented in the above class diagram using the following interfaces:

  1. Route Planning Interface
  2. Bin Pack Interface
  3. Clustering Interface

- Fundamentally, Route Optimization is a sequential task, with clusters needing to be formed first and, subsequently, optimal routes and bin packing being computed. Instances of our **Optimizer Class** do these sequential calls to different algorithms.

PETAL SWEEP

OBSERVATION

From the outputs of previously applied algorithms, it was observed that the majority of the clusters tend to follow a flower petal pattern. This arises from the fact that the initial and final path of each route will be directed towards and away from the hub.

IDEATION

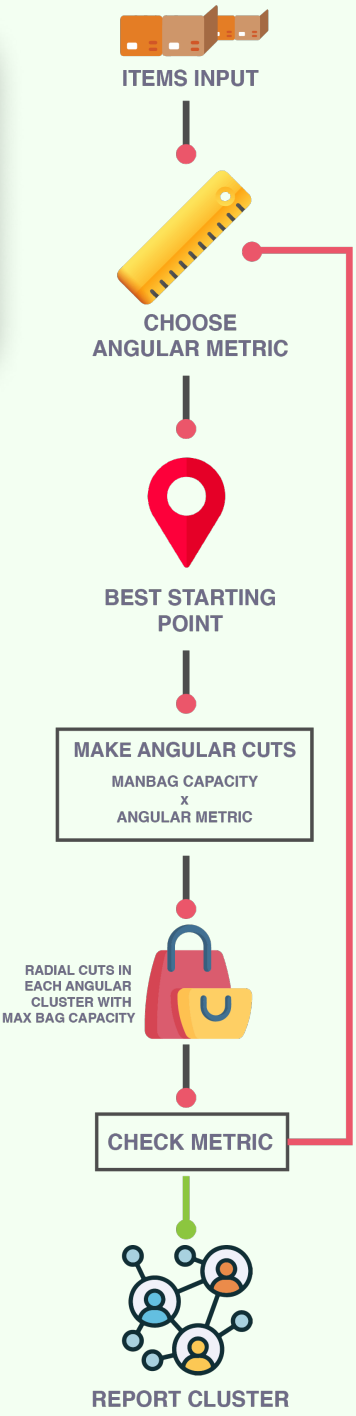
- Two types of cuts, the angular and the radial cut, are formalized to simulate the 'petal like' shape of clusters
- The region around the warehouse is divided into sectors using angular cuts, with the hub at the centre
- In each of these sectors, zero or more radial cuts are made depending on a metric
- This metric is based on the %on-time deliveries and the expected time and cost-effectiveness of the projected path for each cluster

IMPROVEMENTS

- By incorporating information about the shape of paths, our algorithm proves to be significantly better than existing algorithms. On average, it performs better than the ensemble of other algorithms by a factor of 1.15
- Our algorithm tries to z the number of on time deliveries, by smartly creating clusters to satisfy the maximum number of Estimated Delivery Times.

IDEATION

- The algorithm uses a stochastic initialization of state and uses the best possible clustering obtained
- The threat metric (from the Threat Intelligence Oracle project) takes into account the traffic and weather conditions into account while creating clusters
- For all clusters formed, the number of unserviceable points (in terms of EDD, bag capacity constraints etc.) is minimized



RESULTS

This algorithm provided similar and, in some cases, better results than existing state-of-the-art heuristic algorithms. This algorithm has time complexity* of *O(NlogN)* and space complexity* of *O(N)*

CONSTRAINTS

- Algorithm tries to minimize the number of riders by clustering points based on rider capacity.
- Increases on-time delivery by creating almost realistic clusters to have similar EDD's in a single cluster.

ENHANCED CLARKE-WRIGHT ALGORITHM

OBSERVATION

Our problem of creating clusters for delivery is similar to what we face in creating a Minimum Spanning Tree(MST). So going back to basics, we modified the basic Clarke-Wright algorithm by taking inspiration from the standard Kruskal's algorithm for finding MST. This helped us to make the algorithm efficient and fruitful, giving very real life results.

IDEATION

Define the savings of a location as the reduction in traveling costs on visiting that point. Firstly, the locations that give us maximum savings are selected and stored as disjoint sets. Disjoint sets are then iterated and merged if and only if the 2 points are "leaves" of their respective clusters until no more merges are feasible. Savings is defined so as to minimise distance while abiding with the weight, volume and time constraints. It is defined as the following -

$$S_{ij} = \alpha(T_i - T_j) + \beta(D_{io}+D_{jo} - D_{ij})$$

Where D is defined as the "cost" to travel between any 2 package drop locations (i,j)

CONSTRAINTS

With this, we have incorporated various constraints such as volume, weight, package count per cluster, and path length for ensuring cluster feasibility. Using our parameter-tuned heuristic along with consolidation strategy, we also constrain the number of riders and the estimated time for a cluster.

Our constraint incorporation in the algorithm has helped us to obtain optimised clusters which are resulting in better path planning for easy deliveries.

IMPROVEMENTS

We observed that the classical algorithm does not limit the number of clusters formed, which thus might result in clusters more than the number of available riders. We thus define a strategy to consolidate entire clusters further until we have the riders available to deliver them. This strategy ensures the merged clusters are not only close by but also can be bin packed together, thus resultant clusters are optimal as well as feasible.

SALIENT FEATURES

We define a comparator to score the merging - ability of clusters and sort the clusters according to this score. This score is calculated after normalisation to ensure the scores are scaled properly.

The consolidation savings is defined as follows -

$$S = ((D_i+D_j-d_{ij}) - (V_i + V_j) - (W_i + W_j) - (R_i + R_j))/T$$

Where T is defined as ⇒

$$T = ((D_i+D_j +d_{ij})^2+(V_i+V_j)^2+(W_i+W_j)^2+(R_i+R_j)^2)^{0.5}$$

Where D represents total distance of cluster i

d_{ij} represents distance between i and j

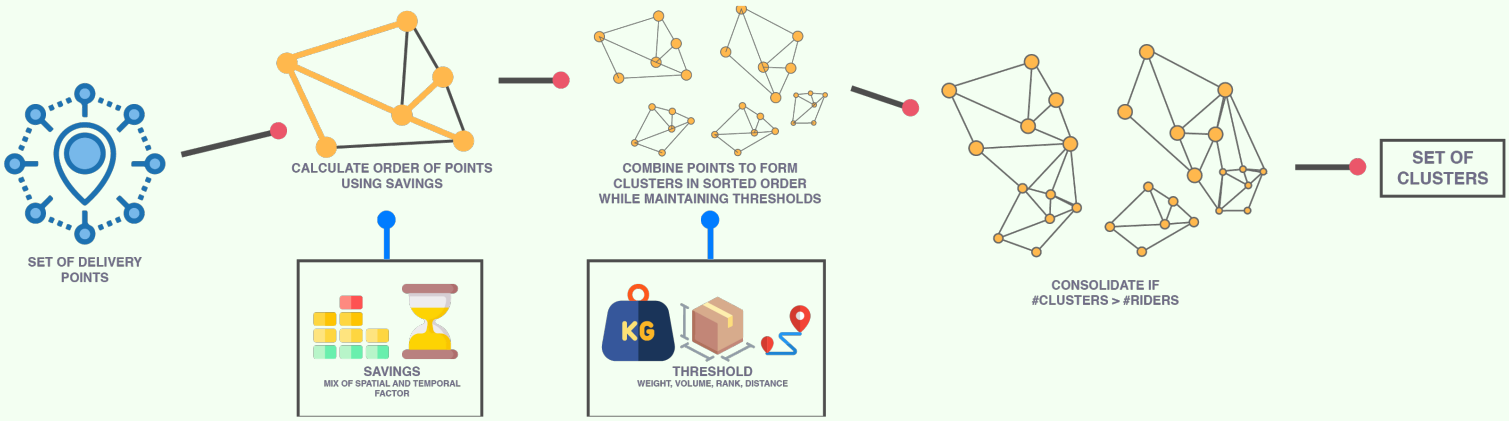
V represents total Volume of cluster i

W represents total Weight of cluster i

R represents Rank of cluster i

RESULTS

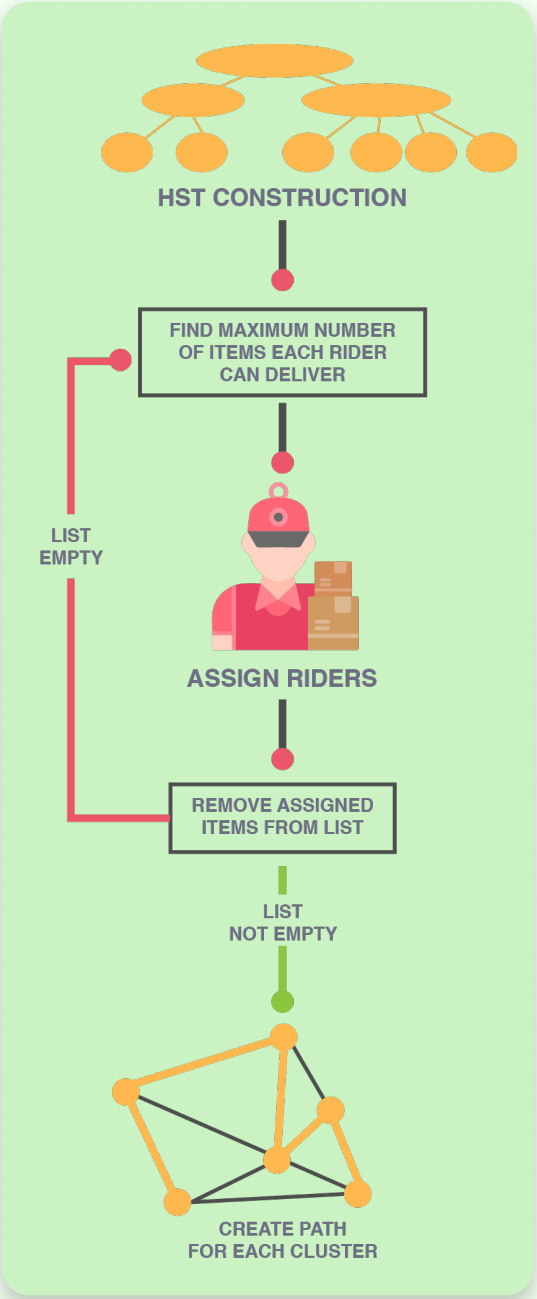
We experimented with various metaheuristics to calculate "savings" making our Enhanced Clarke-Wright Algorithm perfectly catered for Last Mile Delivery Problem. Our algorithm's time complexity is of the order $O(N^2 \log N)$ and space complexity is $O(N^2)$.



HIERARCHICAL EMBEDDED CLUSTERING

IDEATION

- This algorithm is an approximation algorithm that uses the fact that if an efficient approximation algorithm for the k-LMD problem exists, then we can use that algorithm to solve the CVRP problem using a binary search strategy to find an optimal package assignment.
- This is what motivates us to find the solution for the complete set of the locations among the drivers provided.
- We use the Hierarchiacally Separated Tree (HST) to create the spatial index which can be used for faster computation of paths among various clusters.



IMPROVEMENTS

- We have efficiently created and stored the HST for use while creating the cluster. In addition to construction of the HST, we have implemented services to join and split different clusters while planning to automatically get the most optimised set.
- Our algorithm is modular enough to add the feature of dynamic delivery and pickup in the future.
- The algorithm takes into account various constraints, such as delivery time windows, vehicle capacity, and road conditions, to ensure that the optimal solution is both efficient and feasible.

SALIENT FEATURES

- HST allows for efficient clustering of large datasets, as it reduces the number of computations required to find the optimal solution. At the same time it is highly scalable and helps us to accurately represent relationships between different locations.
- The algorithm for clustering in the FESI is executed multiple times so as to get the best clusters from the global space.

CONSTRAINTS

- In our algorithms we have ensured that the bag is optimally and comfortably packed for the rider, that is, the weight of the bag should be bearable and the items chosen should be packable.
- Further, we keep the nearby points in the same cluster thus saving time among different deliveries.
- Along with distance we have also considered the fact that we have to try to deliver the packages before their EDD, which is catered by estimating the time of delivery from the approximations made during the path planning.

RESULTS

- We have an algorithm with mathematical proofs that it'll be providing us with better clusters in comparison to various leading algorithms.
- The algorithms has:
  Time Complexity → $O(N^2MlogN)$
  Space Complexity → $O(NlogN + M)$
  where, N → Number of packages
  M → Number of available riders
- Mathematically proved with perfect distances we have the most optimised result on the given data.

OR-TOOLS TSP SOLVER

IDEATION

We make use of the approximate solver of the Travelling Salesperson Problem provided by Google OR-tools. We primarily make use of following algorithms to solve an instance of the given problem, with the help of the following strategies provided by the tolling utility:

1. Branch and Bound: This method involves exploring all possible solutions and bounding the search space by eliminating sub-optimal solutions. It is guaranteed to find the optimal solution, but is often better in practice for large TSP instances due to its ability to explore more search space in less time.
2. Dynamic Programming: This method involves breaking the problem down into smaller subproblems and using a table to store the solutions to these sub-problems. It can be faster than branch and bound for some TSP instances. Though not practical for large problems, it is highly optimal for smaller subproblems due to complete search.

CONSTRAINTS

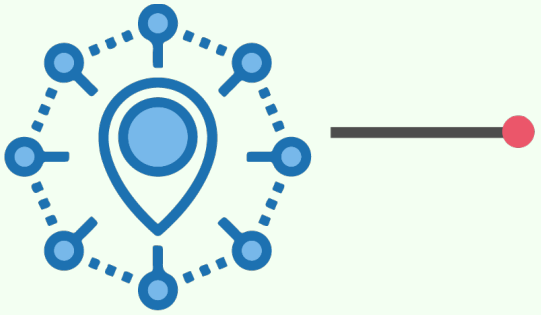
TSP Constraint →
Minimize the objective function:
$L=\sum^{E}_{(u,v)} W(u,v) * X(u,v)$
Where X(u,v) takes binary values for all (u,v) and W(u,v) represents the weight of the respective edge. To incorporate the EDD we had to add another constraint which can be described as follows: $t_i < T_i$ where T_i is the maximum serviceable time of the delivery location and t_i is the time at which we service the i^{th} delivery location.

IMPROVEMENTS

- This implementation of the TSP solver is very robust, especially maintained with the widely proliferating community of google. Our improvement to the implementation makes use of advanced caching mechanisms and improved search of the space with domain knowledge of last mile delivery using AUTOMATIC features.
- We have extended the above version of TSP solver by incorporating the ability to handle the Estimated Date of Delivery of the package as well, this allows us to ensure that the most packages are delivered on or before the EDD
- In addition to this the newer version of TSP OR is able to find the best route to drop off all the packages in the minimal time while ensuring all packages arrive before their respective EDD.

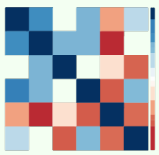
SALIENT FEATURES

We are able to choose the best local search strategy by adjusting special metaheuristics so as to give us the best results. This ensures that in both cases of Normal TSP and the Time Window variant we do not stop at just a local minima but at the absolute minima, In addition to this we are able to set the search time according to our needs to ensure the previous condition again.

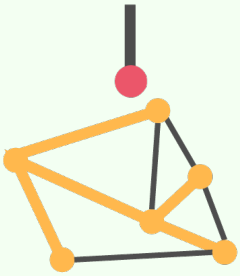


SET OF DELIVERY POINTS

PARAMETERS OF FIRST SEARCH SET



DISTANCE MATRIX AND DEPOT SET



START THE CHOSEN ALGORITHM FOR SET TIME

OPTIMIZED PATH

RESULTS

Normal TSP OR nearly always gives optimal solutions for clusters of less than 50 with our search heuristic and time limit, and this is carried forward to TSP OR EDD which also continues in this strong fashion. As we have set our clusters to have atmost 35 delivery locations we are able to ensure ideal paths.

LIN-KERNIGHAN ALGORITHM

ROUTING

IDEATION

Initial solution: The algorithm starts with an initial solution to the TSP, which can be generated using any simple heuristic such as the nearest-neighbor algorithm. Let $\mathbf{x = [x_1, x_2, ..., x_n]}$ be the initial solution, where x_i is the city visited at step *i* of the tour.

Lin-Kernighan heuristic: The Lin-Kernighan heuristic is the first stage of the algorithm. It works by selecting a subset of cities from the current solution and reversing the order of that subset. This generates a new candidate solution. The subset of cities selected is called the tour improvement cycle. The length of the tour improvement cycle is typically between 3 and 20 cities. The Lin-Kernighan heuristic generates a new candidate solution *y* by reversing a subset of the cities in *x*. The new solution *y* is given by: $\mathbf{y = [x_1, x_2, ..., x_{k-1}, x_{k'}, x_{k-1'}, ..., x_{2'}, x_{1'}]}$ where x_k to $x_{k'}$ is the subset of cities that are reversed in the tour.

2-opt and 3-opt local search: The second stage of the algorithm is the 2-opt local search. It works by swapping two edges in the current solution to generate a new candidate solution. Swapping two edges is repeated until a locally optimal solution is found. The 2-opt algorithm ensures that the candidate solution found using the Lin-Kernighan heuristic is improved further. Let *y* be the current solution and *y'* be the new candidate solution after swapping edges (i, j) and (k, l). Then, *y'* is given by:
$\mathbf{y' = [y_1, y_2, ..., y_{i-1}, y_k, y_i, ..., y_{j-1}, y_l, y_j, ..., y_n]}$

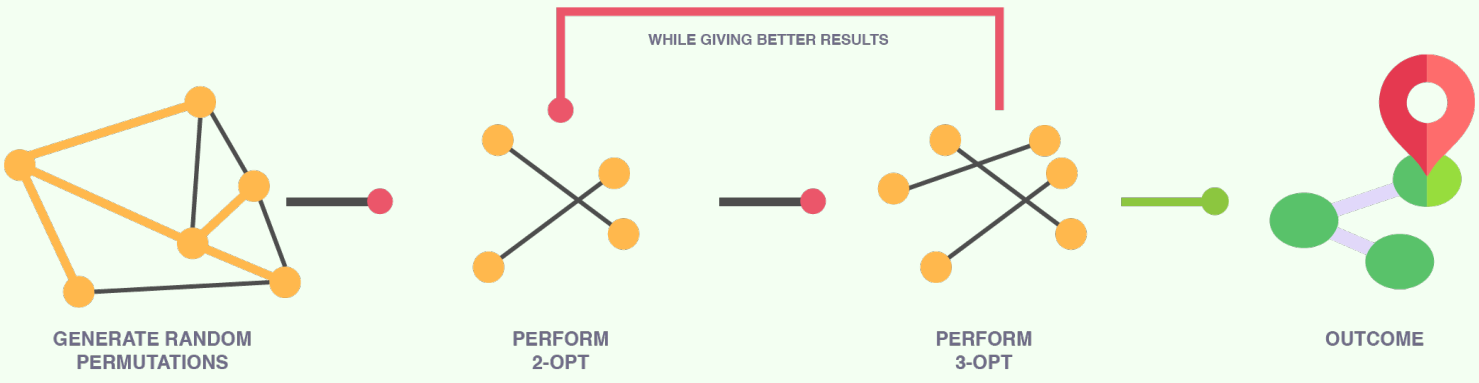
Repeat: The process of using the Lin-Kernighan heuristic and the 2-opt, 3-opt local search algorithm is repeated until a locally optimal solution is found or a stopping criterion is reached. These equations represent the basic steps of the Lin-Kernighan algorithm.

CONSTRAINTS

Tsp Constraint →
Minimize the objective function:
$\mathbf{L=\sum(u, v) \in E\ w(u, v)x(u, v)}$
Where x(u,v) takes binary values for all (u,v) and w(u,v) represents the weight of the respective edge.

IMPROVEMENTS

Improved the calculations for distance so as to get more accurate results which were comparable to ideal values. We have also added a more robust heuristic to faster judge the places where 2-opt and 3-opt Optimization should be done.



SALIENT FEATURES

We combined the most efficient algorithms to merge both 2-opt and 3-opt local search to get output while ensuring the accuracy of the solver. In addition to this we were able to add the facility of working with multiple distance functions at the same time to increase our robustness and improve our results.

RESULTS

In addition to this we were able to get a more accurate real world distance with this algorithm, allowing us to improve our delivery system to ensure faster and cheaper deliveries, all while keeping the complexity the same. The algorithm works in the order $O(N^2 logN)$.

CHRISTOFIDES ALGORITHM

ROUTING

IDEATION

Minimum Spanning Tree (MST):
Given a weighted graph G = (V, E), where V is the set of vertices (Delivery Points), and E is the set of edges (distances between the points), the goal is to find a tree T that spans all vertices and has the minimum total weight. The weight of a tree $\mathbf{T = (V, E_T)}$, where E_T is the set of edges in the tree, can be represented mathematically as: $\mathbf{w(T) = \sum_{(u,v) \in E_T} w(u,v)}$ where w(u,v) is the weight of the edge (u,v).

Minimum Weight Perfect Matching:
Given a graph $\mathbf{G = (V, E)}$, where V is the set of vertices and E is the set of edges, a perfect matching is a set of edges M such that every vertex is incident to exactly one edge in M. The goal is to find a perfect matching M with the minimum total weight. The weight of a perfect matching M can be represented mathematically as: $\mathbf{w(M) = \sum_{(u,v) \in M} w(u,v)}$ where w(u,v) is the weight of the edge (u,v).

Eulerian Tour:
Given a graph $\mathbf{G = (V, E)}$, an Eulerian tour is a path that visits every edge exactly once. An Eulerian graph is a graph where all vertices have an even degree. To make a graph Eulerian, it is necessary to add edges to the graph so that all vertices have an even degree. The number of edges that must be added to make an Eulerian graph can be represented mathematically as: $\mathbf{\Delta(G) = (2 - d_1) + (2 - d_2) + \ldots + (2 - d_n)}$ where d_i is the degree of vertex *i* and *n* is the number of vertices in the graph.

Optimization:
Given a solution to the TSP represented as a tour $\mathbf{T = (v_1, v_2, \ldots, v_n)}$, the goal is to find a better solution by choosing a better start Index in T. The algorithm can simply repeat the Euler tour step with different starting cities and then choose the best Euler tour among all the tours generated from different starting points.
Let $\mathbf{G = (V, E)}$ be the graph representing the cities and distances between them. Let $\mathbf{C = (c_1, c_2, \ldots, c_n)}$ be an Euler tour of G starting from a vertex $\mathbf{v_0}$.
For each vertex v in V, repeat the following steps:

1. Compute an Euler tour $\mathbf{C' = (c'_1, c'_2, \ldots, c'_n)}$ starting from vertex v.
2. Compute the length of the tour $\mathbf{L' = \sum_{(c'_i, c'_{\{i+1\}}) \in E} w(c'_i, c'_{\{i+1\}})}$
3. Choose the Euler tour C with the minimum length L among all the tours C'.

CONSTRAINTS

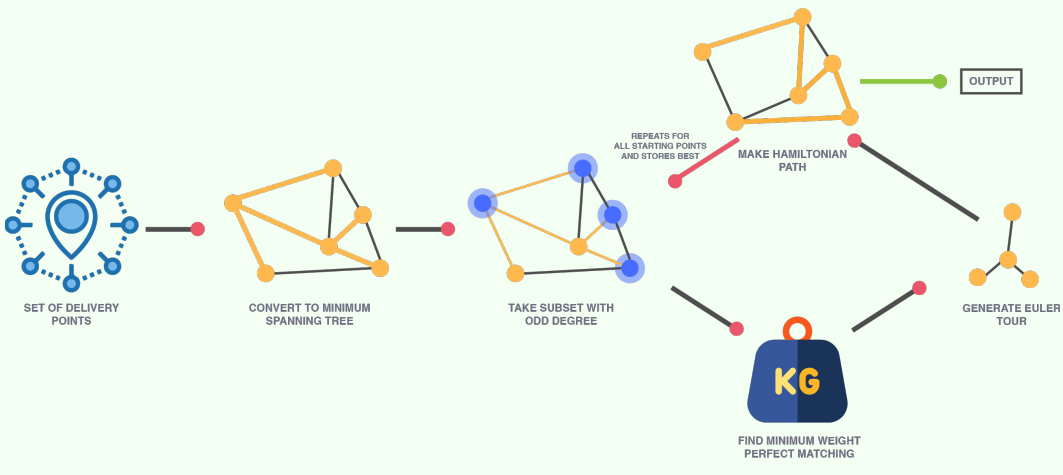
TSP Constraint →
Minimize the objective function:
$L = \sum_{(u,v) \in E} w(u,v)x(u,v)$
Where x(u,v) takes binary values for all (u,v) and w(u,v) represents the weight of the respective edge.

SALIENT FEATURES

We combined the most efficient algorithms in all the phases of the Christofides implementation to get the fastest possible output while ensuring the accuracy of the solver.

IMPROVEMENTS

Improved Efficiency for faster running times, allowing us to process and give results even faster, also improved the heuristics to allow for better results.



RESULTS

The algorithm works in the order $O(n^4)$ with a proved approximation of 1.5, but for most tested real-world applications, the approximations were under 1.38.

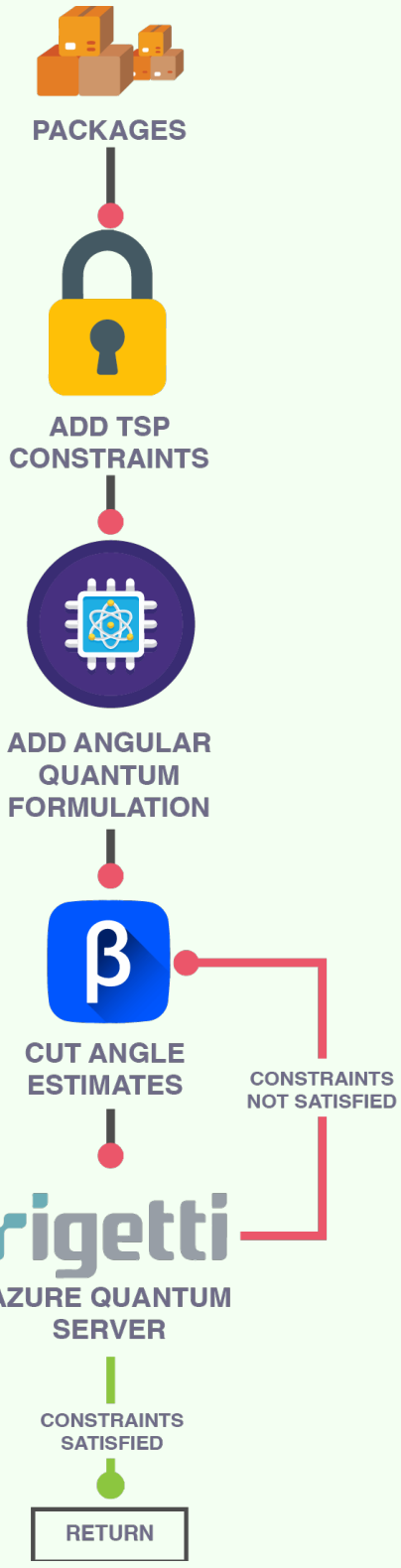
QUANTUM APPROXIMATION

IDEATION

The Traveling Salesman Problem has a non-convex search space in which it can be very difficult, sometimes impossible, to find the global minima for large networks. Therefore, we use Quantum Inspired Optimization to efficiently search through this rugged space. This attempt is also inspired by theoretical foundations which show that it is possible to solve TSP using quantum algorithms such as quantum annealing and quantum adiabatic search.
Our algorithm consists of two steps. Firstly we model the cost function, which penalizes the distance travelled by the rider in the assigned path and incorrect solutions, like those starting or ending at a random city. States where the rider is present at multiple locations at once are also penalized. The mathematical model is then passed to the Quantum Optimizer provided by Azure Quantum services.

IMPLEMENTATION

Through experimentation of our algorithms on different datasets, we found that due to the timeout constraints of the Azure Quantum server, the path found by the solver was sub-optimal. Therefore, we improved our formulation of TSP using academic research on Quantum Approximation algorithms. The algorithm starts with an initial quantum state that is easy to prepare and evolves the state over time according to a schedule determined by the problem instance. The final state of the algorithm encodes the solution to the optimization problem, which can be measured to obtain an estimate of the optimal solution.



RESULTS

The algorithm tries to find the best-effort solution in the minimal possible time. The complexity of the proposed approach is $O(n^3)$, where n denotes the number of stops.

CONSTRAINTS

In the formulation of TSP, we introduced all the constraints of TSP, which include:

- Completeness: The solution must visit all the cities exactly once, without any repetitions
- Optimality: The solution must be the shortest possible route that visits all the cities
- Degree Constraint: Each city must have exactly two incident edges, one entering and one leaving
- Triangle Inequality: The length of a path between two cities must be less than or equal to the sum of the lengths of the two paths connecting the same two cities through a third city
- Time Window Constraints: Some cities may have a time window during which they must be visited, for example, for delivery purposes
- Capacity Constraints: The salesperson may have limited carrying capacity and may need to visit depots to restock

Other than these, we also incorporated constraints imposed by Quantum solver due to hardware limitations

BIN-PACKING

IDEATION

The human intuition of packing bags in layers inspires the idea of packing bags using a bottom-up approach. We propose an out-of-the-box algorithm to get the best-fit bin packing. Our algorithm inserts each item sequentially inside the bag by rotating and rearranging them in each iteration while maximizing efficiency and availability. It achieves bin packing by building it layer-by-layer. This algorithm uses human intuition-based heuristics, like building the layer in a bottom-up approach, where packaging is done against the walls.

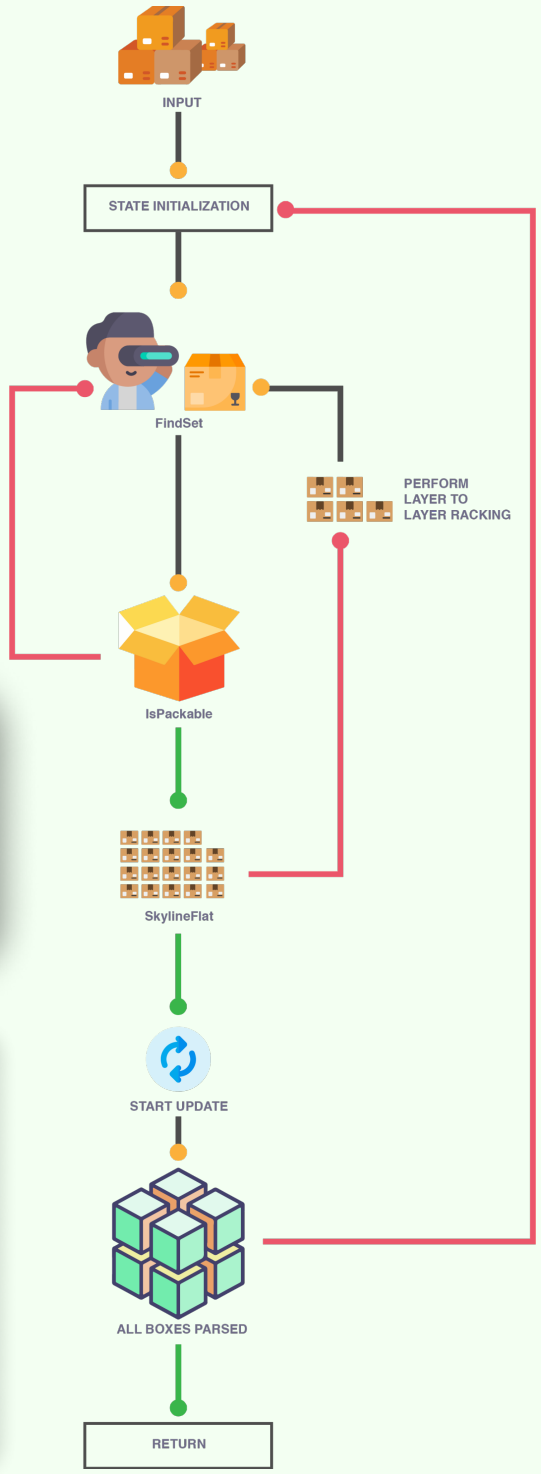
The layers are built by deciding the layer's height and then searching for all the items that can be accommodated in a given layer. The layer's height is then increased by a minimal amount. If any additional space is empty, maintain a flat forward packing face to fit more items in that layer. The solutions' starting layer thickness value and volume utilization are stored to optimize the solution further. At the same time, in our algorithm, we create vertical partitions in each layer and then fill these partitions by pushing the best-fitting items together, further optimizing the space. The Bin Packing of the items based on the path of the delivery person is an NP-hard problem.

IMPROVEMENTS

The initial algorithm is iterated through all the items to search for the best fitting box in available gaps, but this hampered the ordering of boxes which created problems for riders in taking items out of the bag while delivering. To measure this, we introduced an inversion score metric based on the number of inversions in the original ordering of boxes and the final ordering. Through exhaustive experimentation, we found that the number of inversions was of order $O(N^2)$.

$$\text{Inversion Score} = 1 - ((2*n_1)/(n * (n-1)))$$

Where, n_1 is number of inversions in order of boxes and n is total number of boxes to be packed. Therefore, we improved the initial algorithm by reducing the search scope of the initial algorithm. We start by selecting a set of boxes to find the largest set of boxes from the end which can be packed in one layer built over existing skyline which is stored in the state of algorithm. This layering in reversed order reduced the number of inversions to the order of O(N).



SALIENT FEATURES

- Reduced the number of inversions in the bag packing from $O(N^2)$ to $O(N)$ by introducing reduced set of boxes
- The algorithm tries to achieve the maximum number of boxes which can be fitted into the box
- The flat forward face is also maintained at all times by the algorithm

RESULTS

The time complexity is $O(N^2)$, and space complexity is $O(N)$. An important property of this algorithm is that the order of packages is maintained, making it easier for riders to take packages out from their bags.

CONSTRAINTS

The proposed algorithm takes into account the constraints in the rotation of the boxes and also tries to maximize the volume utilization of the bag using the layer-in-layer method.

ENSEMBLE

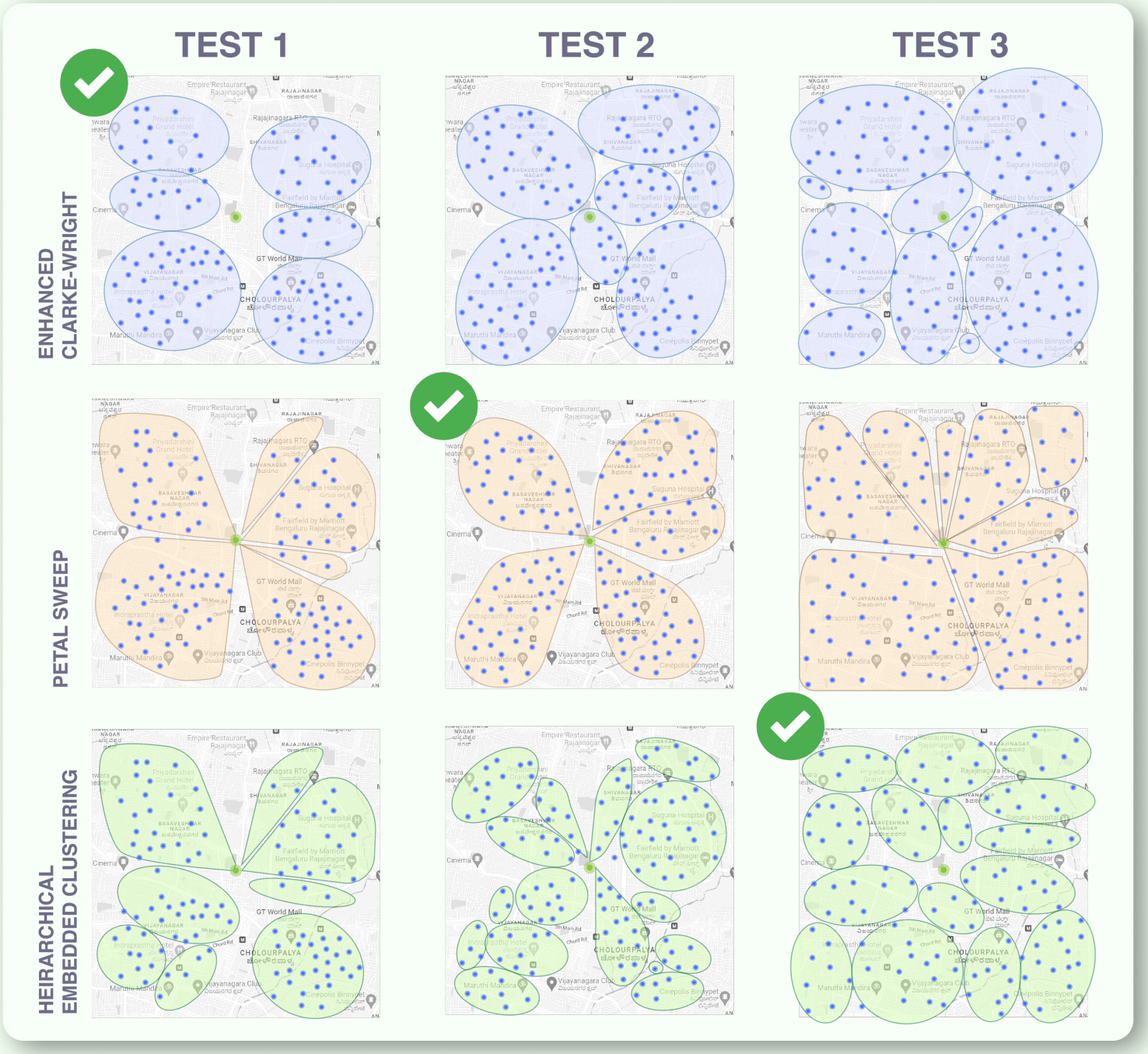
IDEATION

We solve the 2nd challenge by providing our approach in three parts, namely Clustering, Path Routing and Bin Packing. To approach each segment, we have a broad spectrum of algorithms which perform differently based on provided datasets. Some of the cases where one algorithm outperforms others are shown below.

To leverage this property of algorithms, we have performed ensembling of combinations of algorithms and finally chose the best-performing combination aimed to minimize the following cost function.

Cost = Routing Cost + λ * Packing Cost

Where, RoutingCost is the cost of performing routing on all clusters, which is the total distance travelled by all riders and PackingCost is the cost of performing bin packing, which is defined as the fraction of packages unpacked and λ is a relative weighting factor.



IMPLEMENTATION

- **Ensembling** all the algorithms in our codebase requires running many combinations of algorithms that could have been executed sequentially. We identified that sequential execution of combination took a lot of time and involved a lot of repeated computations.
- To avoid this, we introduced concurrency in our program using multi-threading. The implementation of threads is done using the “p-threads” library to improve the latency of our execution.
- Since global variables are shared among all threads, we identified the potential synchronization issues due to the presence of global variables with safe uses of locks.
- We used mutex locks on threads which involve a trade-off between the correctness of outputs and speed of execution. To tackle the slow-down due to repeated computations, we cached the output of some computations to introduce further speed-up in our execution.

SALIENT FEATURES

SPEED

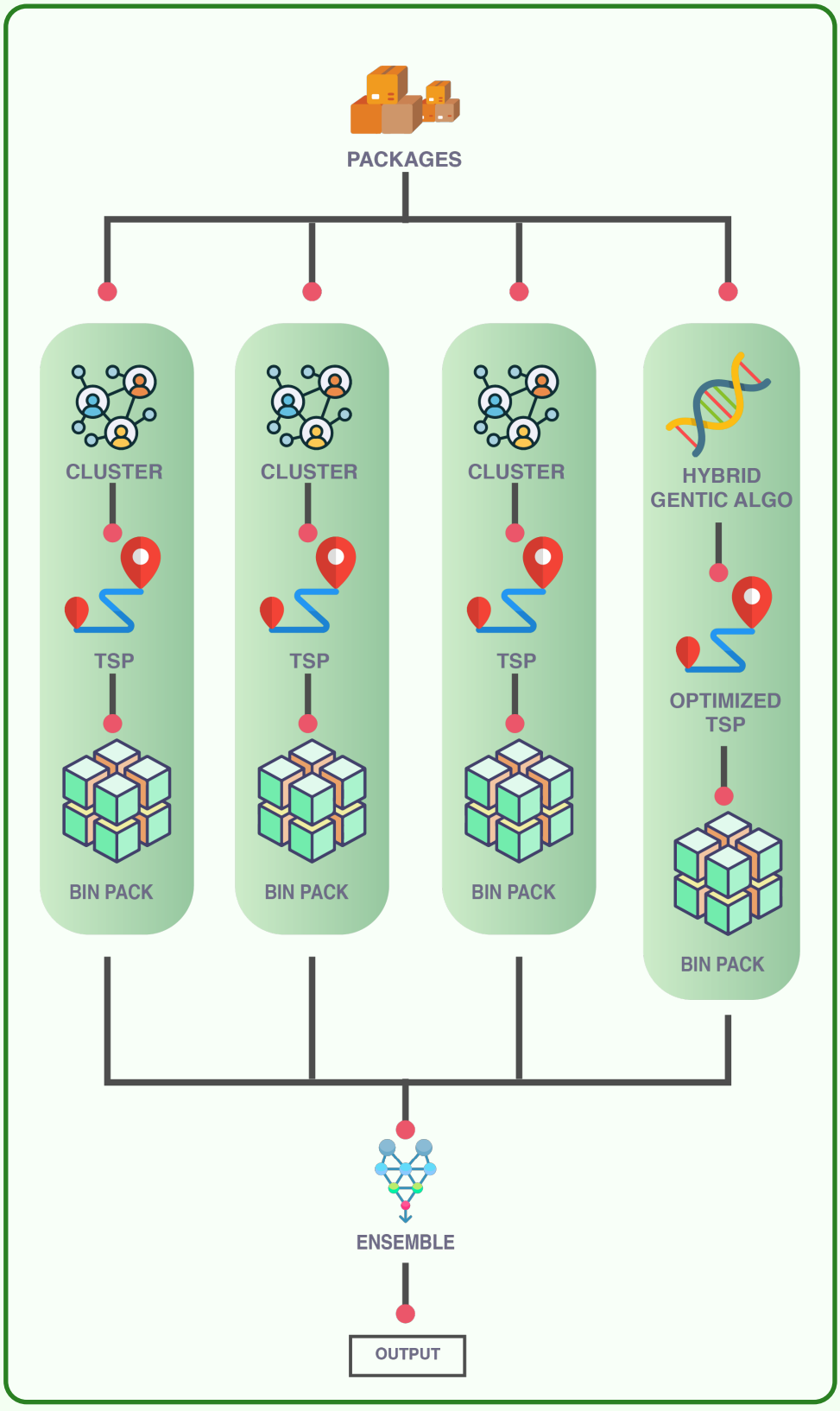
We have a broad spectrum of algorithms for each segment of the problem statement, and hence, to speed up running all combinations, we implemented parallelised thread-based execution.

CORRECTNESS

To ensure that no error is introduced in the execution due to synchronisation issues in modifying global variables, we have safely utilised mutex locks on threads, thus avoiding the race condition.

EFFICIENCY

Since we are running all combinations of implemented algorithms, we needed a lot of repeated computations which we avoided through efficiently caching of outputs in steps where a bunch of repeated computations can occur without exponentially harming the memory available.



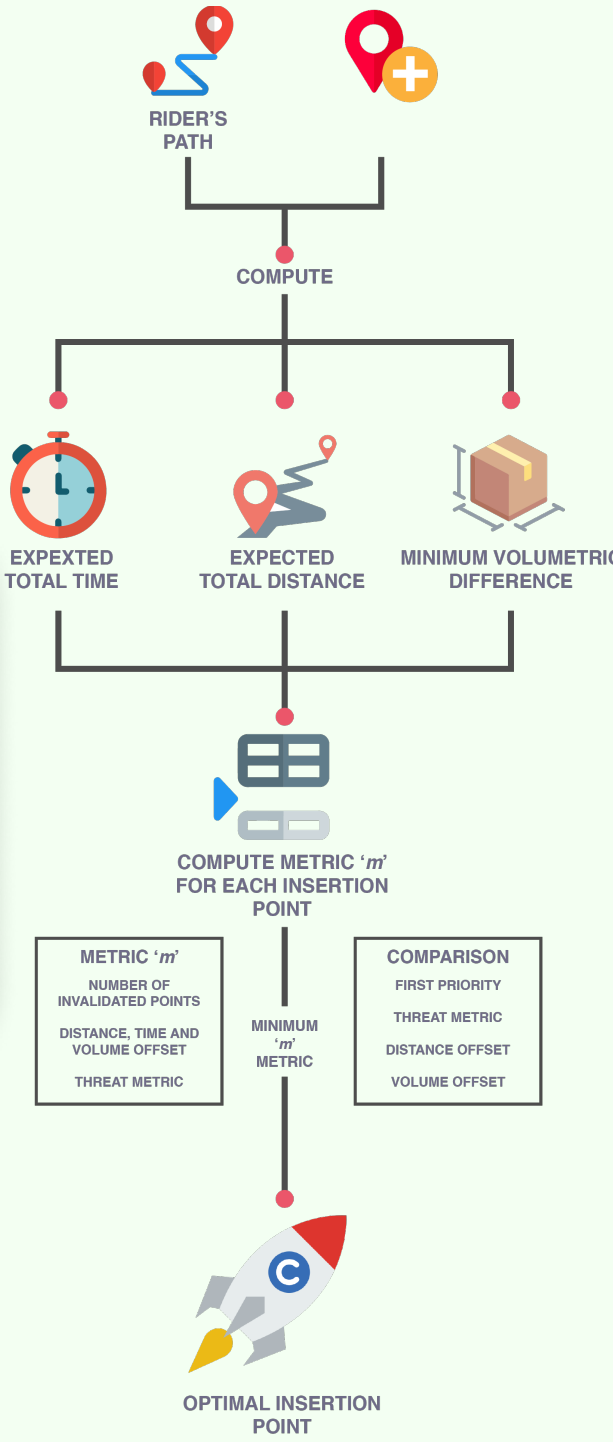
DYNAMIC PICKUP

IDEATION

- To cater to dynamic pickup requests, a subset of nearby riders are selected as potential candidates to service the requests and metrics are used to select the best possible one
- Selecting riders nearby allows us to reduce pickup request service time and reduce cost computations while also maintaining close to optimal solutions.
- For each rider, we check if the pickup can be added before any location in a way that minimizes a self-designed metric that maximizes %on-time delivery, and ensures the time and cost-effectiveness of addition
- Finally, we select the rider that minimizes the said metric and assign the request to them.

IMPROVEMENTS

- Our customized metric incorporates the time offset created due to re-routing, leading to the possibility of some delivery/pickup requests in subsequent paths becoming unserviceable. This metric minimizes the invalidation of such points while also considering cost and time constraints along with the traffic and weather conditions using the threat metric from the Threat Intelligence Oracle project.
- Real world distance-time data has been used to judge the insertion of points, with a fallback condition to haversine distance.
- To estimate the distance and time offsets, we use Google/Bing distance-matrix API to adjust for real-world conditions while performing multi-threading to accommodate the bottleneck created by the API requests.
- Since the addition of a point is done greedily, this will cause a non-optimal insertion of a point in a capacitated VRP. Hence, we run a few route optimization steps to improve costs.



SALIENT FEATURES

- **Selection of a subset of nearby riders using geohashing essentially reduces the search space leading to faster convergence to a near-optimal solution**
- **Multi-threading is incorporated to cope with the bottleneck created by the API request for *distance_matrix* data**
- **The threat metric (from the Threat Intelligence Oracle project) takes into account the traffic and weather conditions into account while creating clusters**
- **For each insertion, the use of the insertion metric ensures minimization of unserviceable points, with the solution being a cost and time effective one**

RESULTS

The time complexity for the algorithm is *O(M + NlogN)*, where M is the number of points returned, and N is the total points stored in the database

CONSTRAINTS

- Number of unserviceable points
- Volume capacity constraint
- Cost effectiveness
- Time effectiveness
- Threat Metric

HYBRID GENETIC SEARCH

OBSERVATION

Running a guided randomization algorithm for sufficient time can help us do an exhaustive search over preferable solutions. This is expected to generate an almost optimal set of clusters via iterative improvement over generations.

IDEATION

The main inspiration behind the Hybrid Genetic Search algorithm is to model an individual of the population as the set of all ordered routes to be traversed by all the riders. Thus, using natural selection, successive generations of the population will provide more optimal routes.

We summarize the implementation of the genetic algorithm below -

- **Population Generation** - We initially start off with a randomly generated population (a bunch of individual solutions). The two most optimal parents are selected for crossover using binary tournament selection. We then generate a new 'individual' / offspring from these 2 parents. This offspring is now refined through 'guided local search' which not only optimizes distance but also tries to make an infeasible offspring into a feasible one.
- **Sub-Population Division** - Unlike the basic genetic algorithm where infeasible solutions are thrown away and the search proceeds only by considering the feasible solutions, our algorithm maintains the infeasible sub-populations also throughout the entire search process and uses these to provide better solutions. **Since optimal solutions are expected to lie in the borders of the feasible region, crossing over feasible and infeasible solutions are observed to produce better outcomes**. Here, infeasible solutions are the ones that violate the limits on vehicle capacities and maximum route travel time specified.
- **Guided Local Search** - Local search is performed through a set of 9 route improvement strategies, which are applied to the individual at random till all strategies have been implemented without solution improvement. These strategies include inserting a delivery location from 1 route to another, swapping locations between two routes and reordering an existing route.
- **Diversity Contribution** - the proposed algorithm also takes the diversity contribution of each solution to the population into account and defines a biased fitness function. This enables the algorithm to move towards a highly diverse population. This biased fitness (BF) of an individual is defined as shown below:

$$BF(P) = fit(P) + \left(1 - \frac{nbElit}{nbIndiv}\right) dc(P),$$

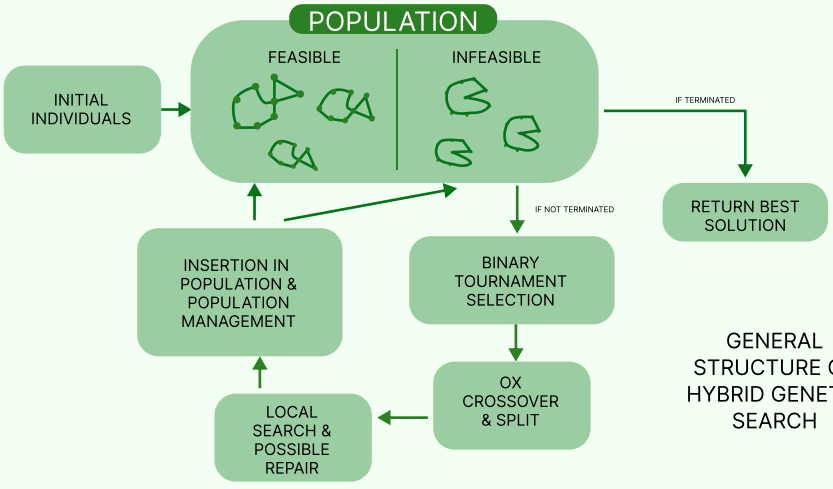
  fitness(P) is the rank of the individual evaluated based on its penalized cost.
- **Inter and Intra Route Optimization** - We have also implemented a SWAP* algorithm that exchanges locations in two routes, or within the same route if the polar sectors of the routes with the depot as the centre overlap to optimize overall distance. This is implemented in sub-quadratic time.

CONSTRAINTS

We have incorporated the following constraints into HGS using our improved metric to evaluate individuals -

- Distance, Duration and Capacity Constraints
- Constraints on packages to be delivered before the estimated time of delivery even after considering the 'service time' spent by the rider.

Symbol	Parameter	Value
μ	Population Size	25
λ	Generation Size	40
η_{ELITE}	Elite solutions	4
$\eta_{CLOSEST}$	Diversity solutions	5
Γ	Search Parameter	20
ξ^{REF}	Penalty adaptation	0.2



GENERAL STRUCTURE OF HYBRID GENETIC SEARCH

HYBRID GENETIC SEARCH

OBSERVATION

Running a guided randomization algorithm for sufficient time can help us do an exhaustive search over preferable solutions. This is expected to generate an almost optimal set of clusters via iterative improvement over generations.

IMPROVEMENTS

- **Mimics Real World Delivery** - Our algorithm takes into account that a delivery guy has to spend some time even after arriving at the location, thus we have kept a parameter called 'Service Time' which describes the average time spent by a delivery agent after arriving at the location. Along with this, we limit the entire duration of a route to 5 hrs. This is done by taking all optimization metrics in terms of 'time' instead of 'distance'. Additionally, to ensure our algorithm works well in the streets of Bangalore, we estimate the speed of a rider to be 18.7 kmph, the actual average statistic of average vehicle speed in the city.
- **Parallelized generation of individuals** - Implemented Multi-threading to support parallel generation of individuals, thus adding multiple individuals to the population at each iteration, effectively speeding the up the algorithm by a factor of 10x.
- **Improved Evaluation Metric** -The penalty function to evaluate individuals is as follows -

$$\varphi(r) = t(r) + \alpha * \sum_{p} max\{0, add(p_i) - edd(p_i)\} + \beta * max\{0, t(r) + S(r) - T)\} + \Upsilon * max\{0, q(r) - Q\}$$

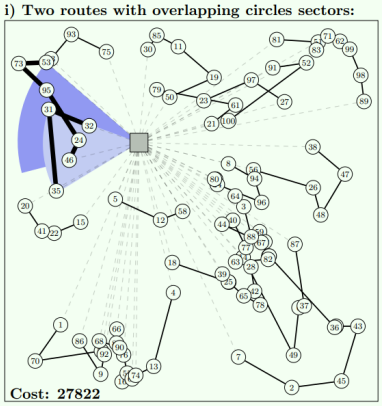
Where *t(r), S(r), q(r)* are the total time spent for travel, total service time and total volume to deliver packages in a route, *α, β, γ* are the penalty weights for exceeding the estimated delivery time, maximum duration and vehicle capacity constraints respectively. *T,Q* are the maximum allowed duration of a route and the maximum allowed vehicle capacity. The penalized cost is then evaluated as the sum of the penalties of all member routes. $edd(p_i)$, $add(p_i)$ are the estimated delivery time and actual delivery time respectively.
This method of evaluating individuals not only optimizes distance length while keeping the time and capacity constraints in check, but also handles 'service time' as well as keeps track on the delivery time of packages, and penalizes any package being delivered after its estimated delivery time.

- **Splitting Individuals** - The algorithms start as one giant tour that starts and ends at the depot/ warehouse as the individual chromosome. This giant tour chromosome is then split into shorter routes using multi-dimensional dynamic programming, being viewed as the shortest path problem on an auxiliary acyclic graph.
- **Online Learning of Constraint Parameters** - The constraint parameters *α, β, γ*, are first initialized to some value at the start of the algorithm, but then are scaled up or down at the time of execution of iterations, i.e. the constraint parameters are learned live while the algorithm executes to find the most optimal values of constraint hyperparameters which minimizes distance while maintaining feasibility.

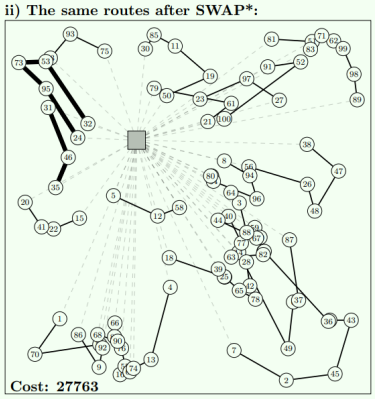
Algorithm 1 HGSADC

1: Initialize population
2: **while** *number of iterations without improvement* $< It_{NI}$, and *time* $< T_{max}$ **do**
3: Select parent solutions P_1 and P_2
4: Generate offspring C from P_1 and P_2 (crossover)
5: Educate offspring C (local search procedure)
6: **if** C infeasible **then** insert C into infeasible sub-population; repair with probability P_{rep}
7: **if** C feasible **then** insert C into feasible sub-population
8: **if** *maximum sub-population size reached* **then** select survivors
9: Adjust penalty parameters for violating feasibility conditions
10: **if** *best solution not improved for* It_{div} *iterations* **then** diversify population
11: **end while**
12: Return best feasible solution

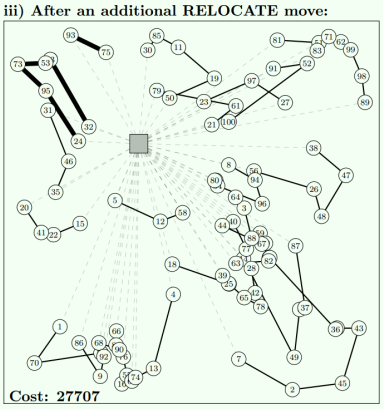
i) Two routes with overlapping circles sectors:



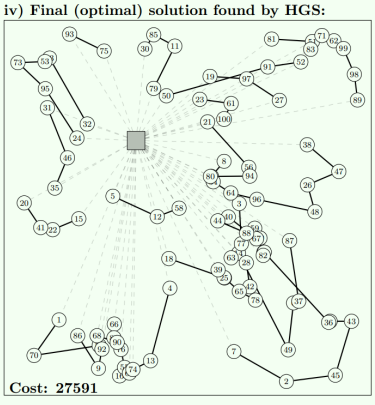
ii) The same routes after SWAP*:



iii) After an additional RELOCATE move:



iv) Final (optimal) solution found by HGS:



Threat Intelligence Oracle (TIO)

IDEATION AND IMPLEMENTATION

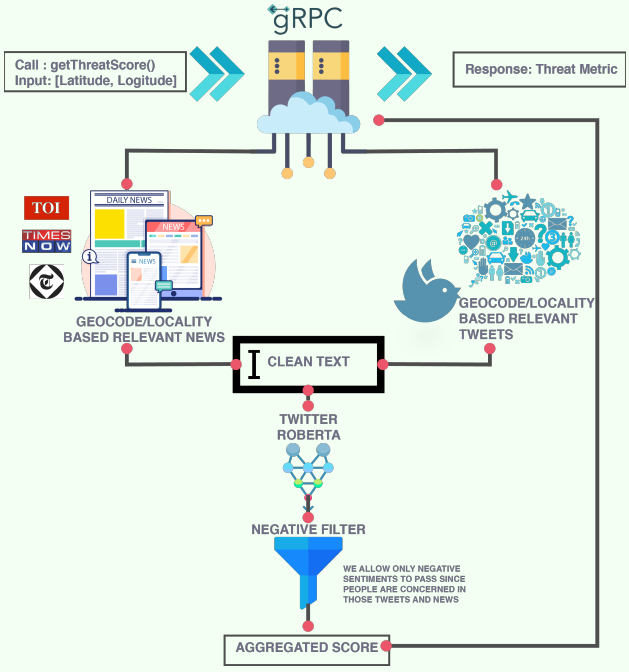
The motivation behind modelling the Threat Intelligence Oracle is that social media is a fast-moving and real-time source of information, especially when combined with real-time news. Suppose we have an analyzer of the data that is constantly flowing in. In that case, we can pose a real-time estimation of the threat that particular location of the road poses due to weather, traffic, rallies or lockdowns. We make an efficient model first by analyzing the Tweets, News, Traffic status data and real-time weather data in consensus with the Meteorological Department.

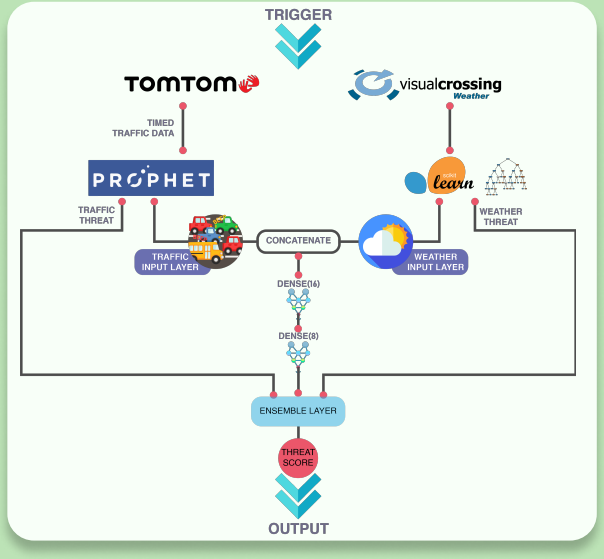
Real-Time Twitter and News Oracle

- Tweets are tagged with geolocation only when the user turns on the location service. Hence to widen our search, we use reverse geocoding to find the locality, sub-locality, and administrative locality to search particular tweets that mention the target location. In addition, we use the latest news from the most widely used news sources, including The Times of India, to know the real happenings near that area, which also acts as a validation for the tweets.

- Social media-based methods are used because no API can provide details of rallies or lockdowns. Still, many people are usually informed about these on social media. Data gathered from these sources are cleaned using standard NLP text cleaning to extract the items that concern us and discard useless tags and hyperlinks.

- Most efficient Twitter Roberta Model provided by Cardiff university is fine-tuned on News Headline Sentiment Dataset to handle news sentiments. Finally, we pass the scores generated through a negative filter because we are worse affected by heavy traffic roads.





Real-Time Traffic and Weather Feeds

- The traffic and weather data is collected from the TomTom and Visual Crossing API endpoints. For weather, we collect the data from the humidity, precipitation and windspeed as features. On the traffic front, we are using the current average speed and the free-flowing speed of the road. The ratio of the current average to the free-flowing speed of the road gives an estimate proportional to the safety and time of travelling on the road.

- We use a Facebook prophet to learn about the relation between the delay in Bangalore with the ratio. Since the delay is a marker of threat on the road, thus the trained prophet model predicts the threat score based on traffic given the input at a particular time. A similar analysis is done with the help of a Random-Forest Regressor to predict the amount of rainfall in Bangalore in June that heavily affected the traffic.

- Features extracted from the prophet and the Random Forest model to train a DNN, forming the encoder network. This network is connected with a decoder to train on a downstream task of classification of the type of climate into “SUNNY”, “WINDY”, “RAINY”, “THUNDERSTORM”, etc. For generating the real-time threat score, we disconnect the decoder layer and use the value generated by the encoder to ensemble it with the individual traffic and weather threat.

SALIENT FEATURES

Fetch Tweets

Fetches tweets from Twitter API and stores them in a Redis database. The search is efficiently fine-tuned and targeted using the "#" hashtags and boolean operators with a rolling combination of curated threats and geolocation of the tweets. The tweets are then processed using NLP and Sentiment Analysis to determine the threat level of the tweets.

Fetch News

Fetches news from News API and processes using NLP and Sentiment Analysis to determine the threat level of the news. The threat level is then stored in a Redis database. The search is fine-tuned and targeted using the logical OR and AND operators to fetch the latest news targeted to the location and tagging threats.

Fetch Traffic

Fetches traffic data from Tom Tom API and stores them in a Redis database. The traffic data is then processed using Facebook Prophet to determine the threat level due to the current traffic flow. The traffic flow is then stored in a Redis database with timed updates to reduce the service query time. The search is fine-tuned and targeted using the geolocation of the traffic data.

Fetch Weather

Fetches weather data from Open Visual Crossing Weather API and stores them in a Redis database. The weather data is then processed using Random Forest Regressor to determine the threat level due to weather conditions. The response is then stored in a Redis database. The search is fine-tuned and targeted using the geolocation of the weather data to get a targeted response.

Geocoding

The geocoding is done using the Google Maps API and the Geopy python package. A reverse geocoding map is run to determine the location of the tweets and news that needs to be targeted. The place is broken down into several levels of granularity to run a wide-span targeted search to determine the threat level of the en-route traffic, including locality, sub-locality and administrative_area_level_1 (state).

Efficient Search Queries

The search is done using Twitter's distance fine-tuning using geohash radius and location granularity levels to determine the focus radius of the tweets. The search is done using the News API using the logical OR and AND operators to access the latest news to assess the threat level of the en-route traffic along with the geohash radius and levels of location granularity.

Hand-curated Threat List

The threat list is hand-curated and is used to determine the threat level of the en-route traffic. The threat list determines the events we should be interested in and the specific radius around the location in very efficiently tuned order. AI works well, but we can make it more efficient and accurate with manual intervention. Hence, the order of the threat events curated in the list makes it very important and practical to use.

Real-Time Data Handling

The real-time data is handled using the Redis database and data from real-time Traffic, Twitter, News and Weather APIs to get the updated road or location status. We efficiently manage the API limits and latency of the system using the Redis database and rolling API access. We make the systems scalable for a higher number of API calls while still staying within the provider's limit and incorporating more sources of information.

Ability to extend

This set-up is efficiently made so that it can be extended to incorporate threats once analysed from other sources like Facebook news or Instagram pages of news. Instagram is among the widest spreading source of information with the onset of the influencers in the market and thus is a very strong source of information. We can also include other sources of news very easily if we build a content crawler for respective sources.

RESULTS

		THREAT SCORE	
HIGH TRAFFIC (INDIA) (Nelamangala, Bangalore)	2 hours before the data fetching, people have tweeted, tagging the Bangalore police and informing heavy traffic near the region There have been recent updates about the hikes in the toll values and increased number of accidents near the Nelamangala region	0.75419	
LOW TRAFFIC (INDIA) (NH-48, Delhi)	Newer road maintenance brings a more consistent traffic flow Free flowing speed - 80 Km/h Current Average Speed - 68 Km/h We get a ratio of nearly 1, which means there is nearly free-flowing traffic.	0.12108	
HEAVY SNOWFALL (CANADA) (Perth County, Ontario)	Roads are heavily covered with traffic,and cars face difficulty moving through Due to heavy snowfall, Environment Canada issues snow squall warning	0.88523	

NH-48

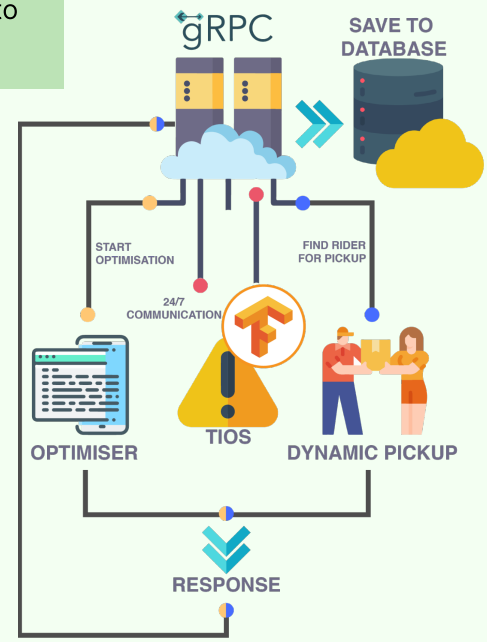
BACK-END SYSTEM

ARCHITECTURE AND IMPLEMENTATION

- Among the multiple components working together to provide an efficient, robust and reliable solution for a backend to handle thousands of requests, Node.js is the primary technology used for the monolithic backend architecture. This single codebase contains both the server-side and client-side components of the system. Node.js is a popular choice for server-side development due to its fast, efficient performance and a large and active community.
- The system's primary database is MongoDB, a document-based NoSQL database. This allows for a flexible and scalable solution for storing and managing the system's data. MongoDB supports many data types, making it an ideal choice for the backend system. With the support of the vast distributed system of MongoDB, the latency is in mere milliseconds.
- In addition to MongoDB, the system also utilises Redis as a caching database. Redis is an in-memory data store optimised for storing frequently accessed data, allowing fast and efficient retrieval. This helps improve the backend system's performance by reducing the number of calls to the primary database.
- We use GRPC-based messages passing through the system due to their robustness and speed. On average, we get an improved speed of about 6-7 times over REST API. GRPC is proliferating and is a very scalable mechanism to implement new generation API.

EFFICIENT ASYNCHRONOUS SYSTEM

- The system also uses RabbitMQ, a message broker that facilitates communication between the backend and the GRPC handler client. RabbitMQ is a reliable and scalable solution for communication between components and helps to ensure that requests are processed efficiently and effectively.
- The GRPC handler client is a Node.js GRPC client responsible for processing requests sent from the backend. The requests are processed asynchronously, allowing the system to handle multiple requests simultaneously. This helps improve the backend system's performance and efficiency, as it can process multiple requests simultaneously instead of waiting for each request to complete before processing the next one.
- The asynchronous processing of GRPC requests ensures efficient handling of incoming requests, and the integration of RabbitMQ ensures reliable communication between the backend and the GRPC handler client.



SALIENT FEATURES

- Async handling of requests for serving multiple requests at a time. This enables the client to continue interacting with the rest of the service without waiting for the result of the process.
- Every Service runs on an EC2 server as a dockerized container, enabling the setup to reproduce on any machine. Thus, the system is even robust enough to integrate with Kubernetes if required.
- GRPC requests are handled in the best possible time since it uses HTTP/2.0 under the hood for managing connections, which are much faster than traditional REST API calls.
- Coordinates of packages are Geohashed in Redis, which is later used to find packages in increasing order of distance using GeoSearching quickly.

SCALABILITY ASSETS

MongoDB is a highly scalable database that handles an enormous amount of data. Redis is a fast and efficient cache database that serves data under load conditions. The use of RabbitMQ also greatly relieves the scalability bottleneck since the optimisations might take time, and the waiting process is asynchronous.

GROW SIMPLEE AGENT MOBILE APP

Development Framework

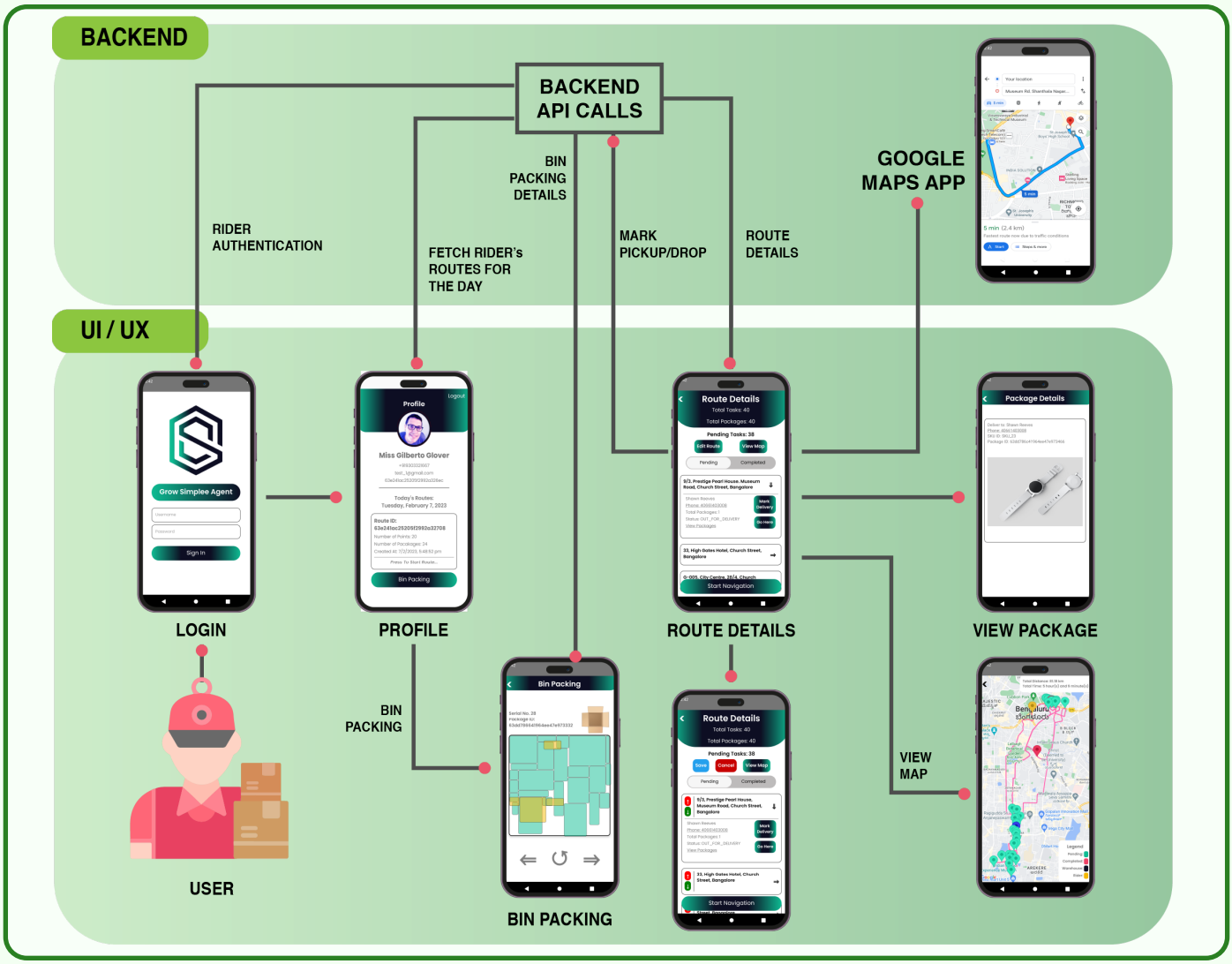
1. The app is developed using React Native (RN) with JavaScript XML (JSX). React Native is a developer friendly low-cost highly scalable cross-platform mobile development environment which allows us to build apps for Android and iOS devices simultaneously. The app is lightweight, with a 12 MB download size from Google Play Store. The app pages receive information using API calls on page load and store it in the React Native States for fast client-side processing

Salient Features

1. We used React Native Navigation library to provide intuitive page-based app navigation. React Native Navigation library provides components to integrate custom touch and gesture-based navigation controls.
2. For driving navigations, we use React Native Google Maps Directions library to provide navigation through Google Maps to pickup/delivery. The app updates the rider's location to the backend at regular intervals.
3. Rider is provided with interface to reorder the route locations and upload it to the backend. Both the rider and admin are notified in case the route is updated
4. React Native Maps package is used to display the rider's complete route in an interactable map window, along with the total distance, time required to cover the route and color coded markers. Time is calculated considering the present traffic conditions along the route
5. The app provides a sequential step-by-step bin packing guide to assist riders in efficiently packing packages for delivery and pickup

Bin Packing Visualization Algorithm

The packages are added to the bin sequentially based on their distance from the container base, from deepest to shallowest. Packages are rendered as React Native components and colored based on the number of overlapping packages below it. The bottommost packages have the same color, and each subsequent layer is assigned colors different than the colors of the packages below and overlapping it



GROW SIMPLEE WAREHOUSE PANEL

Development Framework

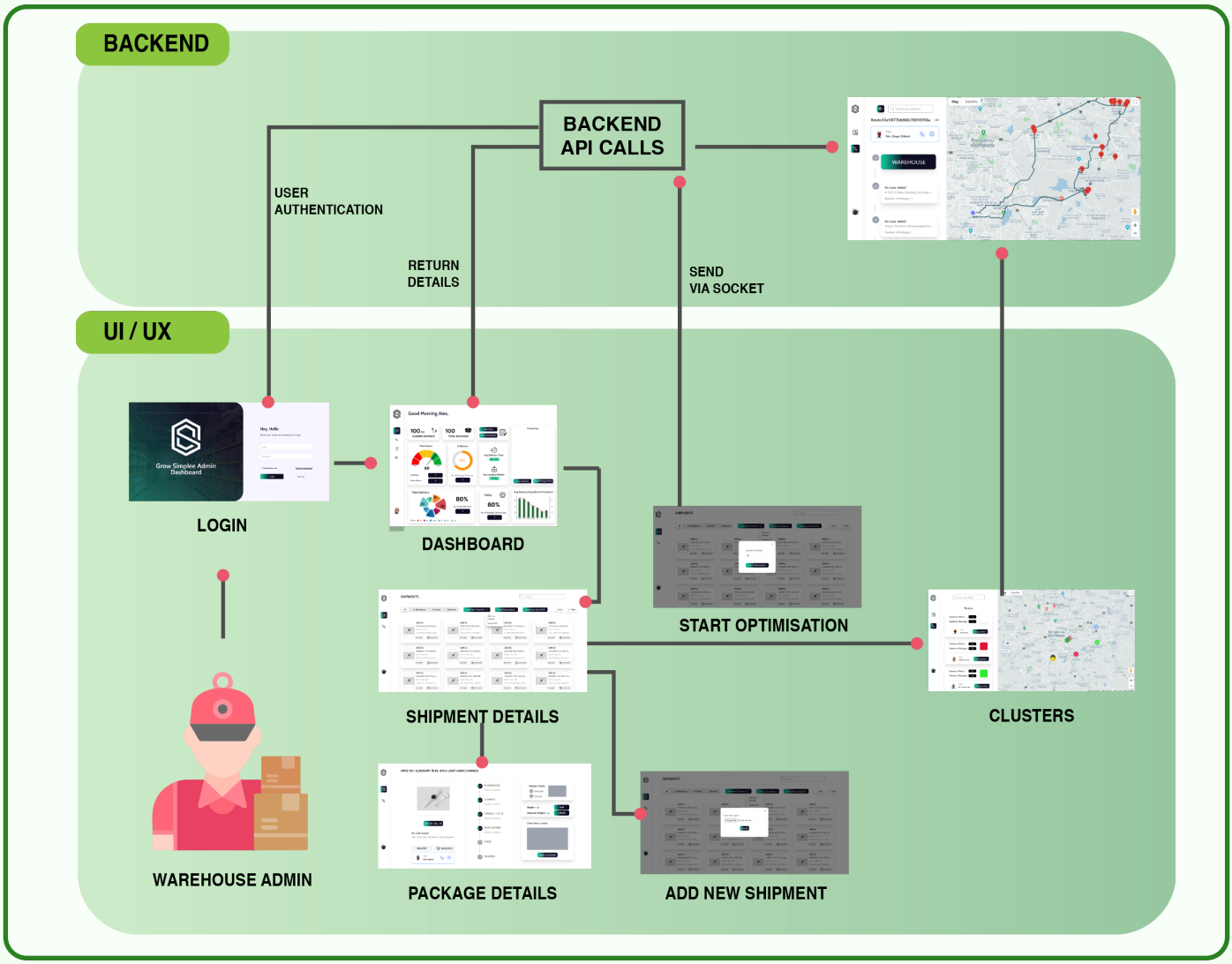
1. The app is developed using React Native (RN) with JavaScript XML (JSX). React Native is a developer friendly low-cost highly scalable cross-platform mobile development environment which allows us to build apps for Android and iOS devices simultaneously. The app is lightweight, with a 12 MB download size from Google Play Store. The app pages receive information using API calls on page load and store it in the React Native States for fast client-side processing

Salient Features

1. We used React Native Navigation library to provide intuitive page-based app navigation. React Native Navigation library provides components to integrate custom touch and gesture-based navigation controls.
2. For driving navigations, we use React Native Google Maps Directions library to provide navigation through Google Maps to pickup/delivery. The app updates the rider's location to the backend at regular intervals.
3. Rider is provided with interface to reorder the route locations and upload it to the backend. Both the rider and admin are notified in case the route is updated
4. React Native Maps package is used to display the rider's complete route in an interactable map window, along with the total distance, time required to cover the route and color coded markers. Time is calculated considering the present traffic conditions along the route
5. The app provides a sequential step-by-step bin packing guide to assist riders in efficiently packing packages for delivery and pickup

Bin Packing Visualization Algorithm

The packages are added to the bin sequentially based on their distance from the container base, from deepest to shallowest. Packages are rendered as React Native components and colored based on the number of overlapping packages below it. The Bottommost packages have the same colorCode 0, and each subsequent layer is assigned colors disserent than the colors below and overlapping it

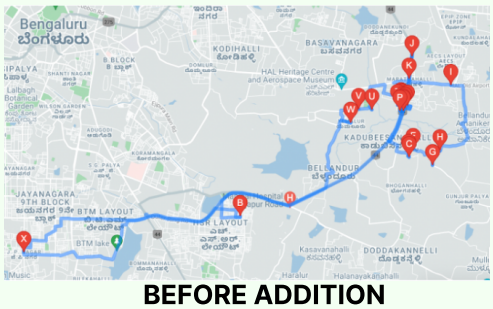


RESULTS

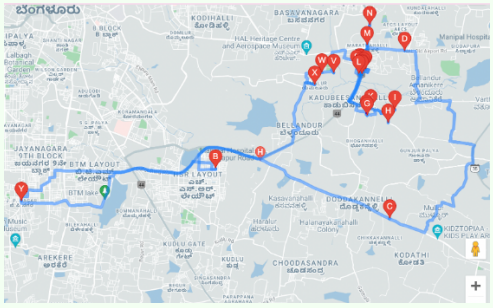
ON MOCK DATASET - DYNAMIC PICKUP

Trial 1 (150 pts)		Optimal	Our output after addition
1		752.841	772.778
3		706.316	783.814
7		652.332	823.309

This table represents the total travelling distance before and after addition of dynamic points



BEFORE ADDITION



AFTER ADDITION

ON CHRISTOFIDES DATASET

We have chosen algorithms of contrasting natures for every sub-part of the problem statement. Hence to get the best solution for various cases, we do an ensemble of these algorithms. The ensemble is primarily aggregated on the combined metric of minimizing the total distance and maximizing the number of deliveries. The secondary metrics involve equally distributing the workload among the riders.

In Figure (1): We plot the total routing cost of the ensemble of our algorithms against the number of drop locations while keeping the number of drivers constant. We observe that our algorithms follow the optimal results closely.
In Figure (2): We again plot the total routing cost of the ensemble of our algorithms against the number of drivers while keeping the number of drop locations constant. We again observe that our algorithms follow the optimal results closely.

Table 1 represents the performance of our solution on the Christofides 1979 - CMT dataset. The table shows that our ensemble results are very close, and a couple are even better than the current state-of-the-art-solutions. Additionally, the average packages per rider are maintained below 25 for most cases, thus distributing the workload assigned to each driver. With an optimal packing of more than 85%, we also validate the efficiency of our packing algorithm for real-world scenarios.

Figure (1)

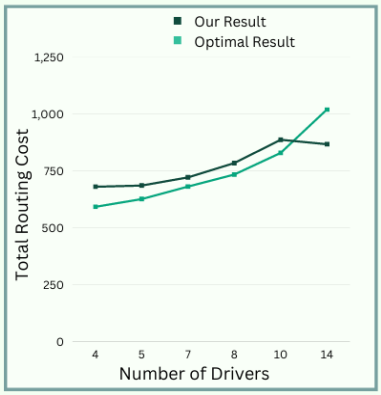


Figure (2)

N	M	Optimal Distance	Ensemble Distance	Average Cluster	Optimality Ratio*	Average % Utilization
20	2	216	233.92	9	1.08	89.5
101	4	681	759.56	25	1.12	88.3
121	7	1034	1045.64	17	1.01	85.1
262	25	6119	5880.62	10	0.96	90.5
16	8	450	423.5	1	0.94	91.2
45	7	1146	1593.099	5	1.39	81.5

Figure (2)

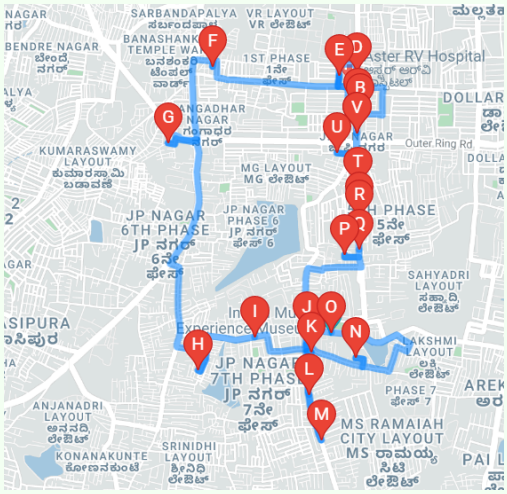
RESULTS

ON MOCK DATASET

	Num of Packages	Dist Travelled	Time of Each Cluster (Min)	Number Inversions per box
	24	15.64	44.68571429	0.87
	29	39.68	113.3714286	0.96
	30	43.5	124.2857143	0
	22	67.05	191.5714286	1.75
	27	55.18	157.6571429	0.71
	29	69.65	199	1.13
	33	23.2	66.28571429	1.01
	19	13.13	37.51428571	1.23
Average	**26.625**	**40.87875**	**116.7964286**	**0.9575**

This Table shows the total distance travelled in each cluster, the time required for each cluster and the number of inversions for bin packing

An actual cluster formed after route planning and bin packing



A snapshot of all the clusters formed on the mock dataset