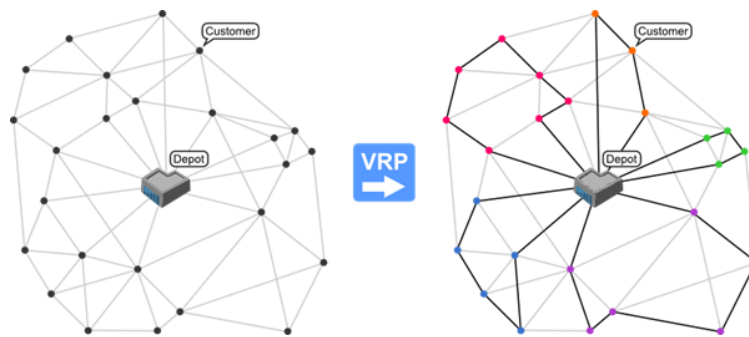


# Delivery Vehicle Routing System



## Group Q - Team Members

### Name/Roles:

**Henry Kavadias-Barnes**

*Backend - Solver implementation*

**William Burns**

*Frontend - GUI and file I/O*

**William Tat**

*Backend - Agent messaging system*

### Student ID:

102108294

101923920

100665166

## Contents

### Section:

### Page Range:

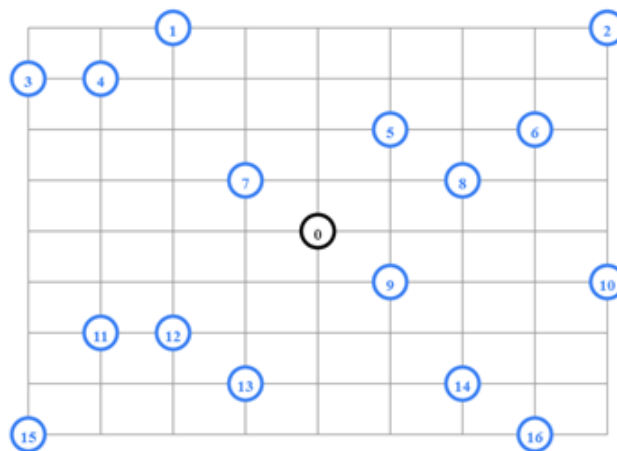
1. Introduction.....	2
2. Overall System Architecture.....	3
3. Implemented Interaction Protocols.....	4-5
4. Implemented Search/Optimisation Techniques.....	6-7
5. Scenarios/Examples of System.....	8-9
6. Critical Review of Implementation.....	10
7. Summary/Conclusion.....	11
8. Research Sources.....	12

# 1. INTRODUCTION

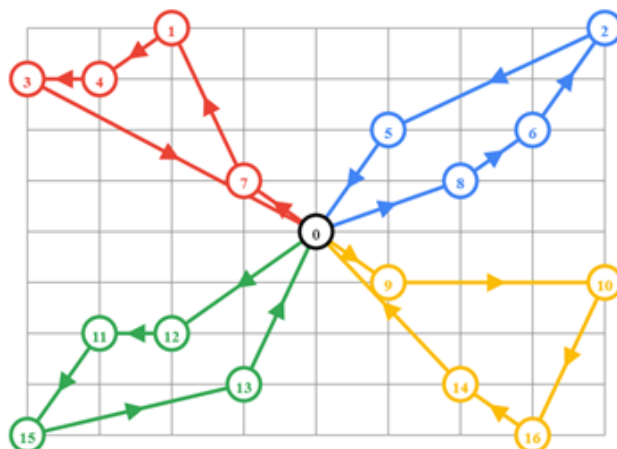
The Vehicle Routing Problem (VRP) is about finding the most optimal route for multiple agents collectively visiting all of a set number of locations. The most optimal solution is typically defined as to minimum total distance travelled by all the routes from each vehicle agent, but there are other constraints that are considered from the problem. One such hard constraint is vehicle capacity, where each location has a quantity or weight of items that need to be delivered but each agent has a limited carry capacity.

The problem is best visualised on a graph where the nodes are locations that the agents need to visit. The first node (0) is defined as the **Depot** where each agent must begin and end their route. Depot has no items to be collected since it is the “delivery location”.

Graph map before solving:



Graph map after solving:

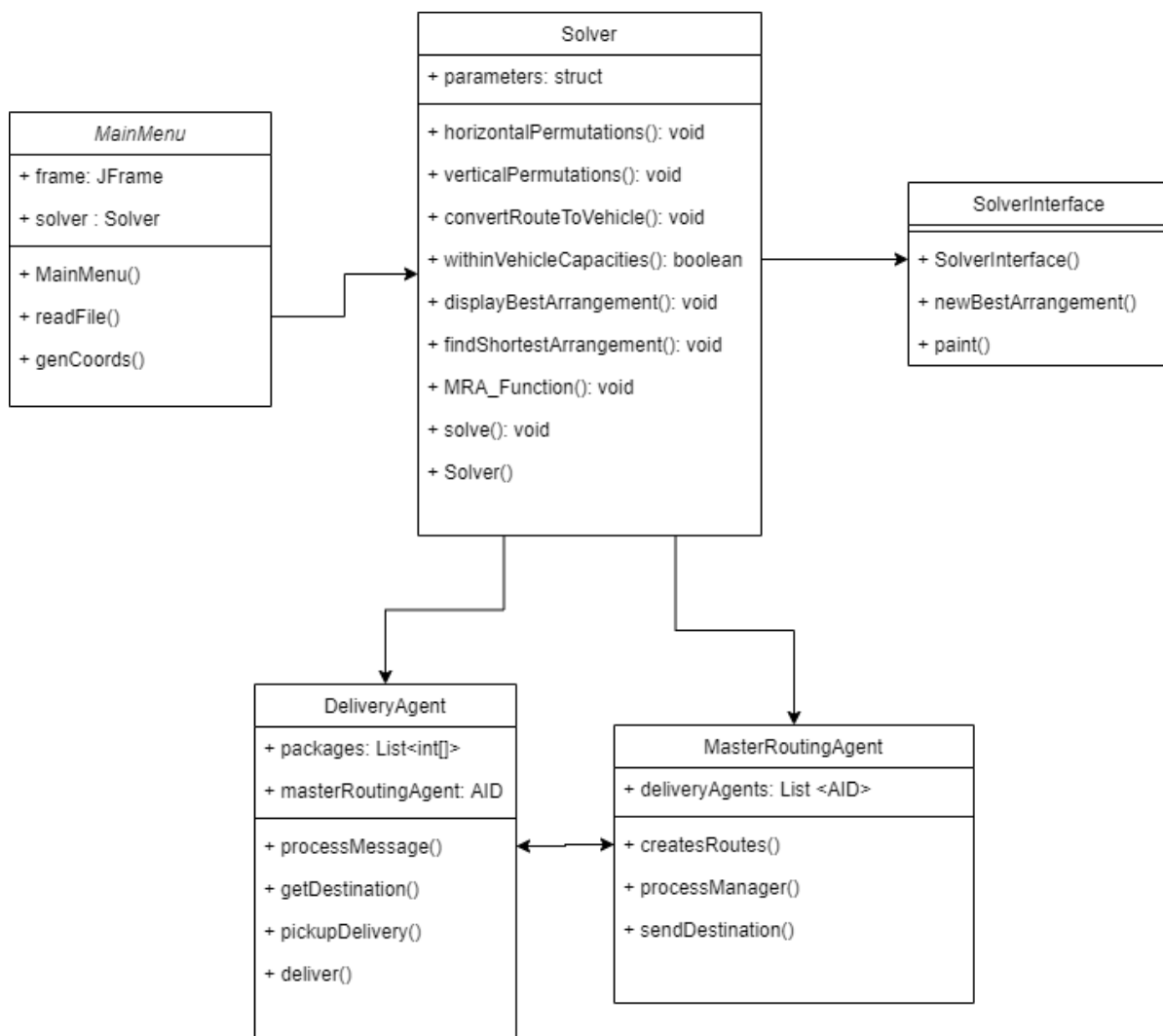


## 2. OVERALL SYSTEM ARCHITECTURE

The user interacts/initiates the solver through the **Main Menu** by generating a random map with the submitted number of locations and vehicles, or by selecting a file with the relevant data. Once data has been created/accessed it is then fed into the **Solver** which searches through all of the possible route combinations for each vehicle to find the most optimal solution. The current best solution that the solver has found is displayed through the **SolverInterface** UI. Once the solver has finished searching through all the combinations or when it's prompted to end early, the routes will be passed to the **Master Routing Agent** which assigns each of them to a **Delivery Agent**.

The following software was used to create the program:

- **Java 11** and many of its internal libraries were used to create this program.
- The **Choco-Solver** model is used in the solver to calculate the shortest route for each vehicle.
- The **Java Agent DEvelopment** (JADE) framework is used as a basis for the agent interaction and messaging system.

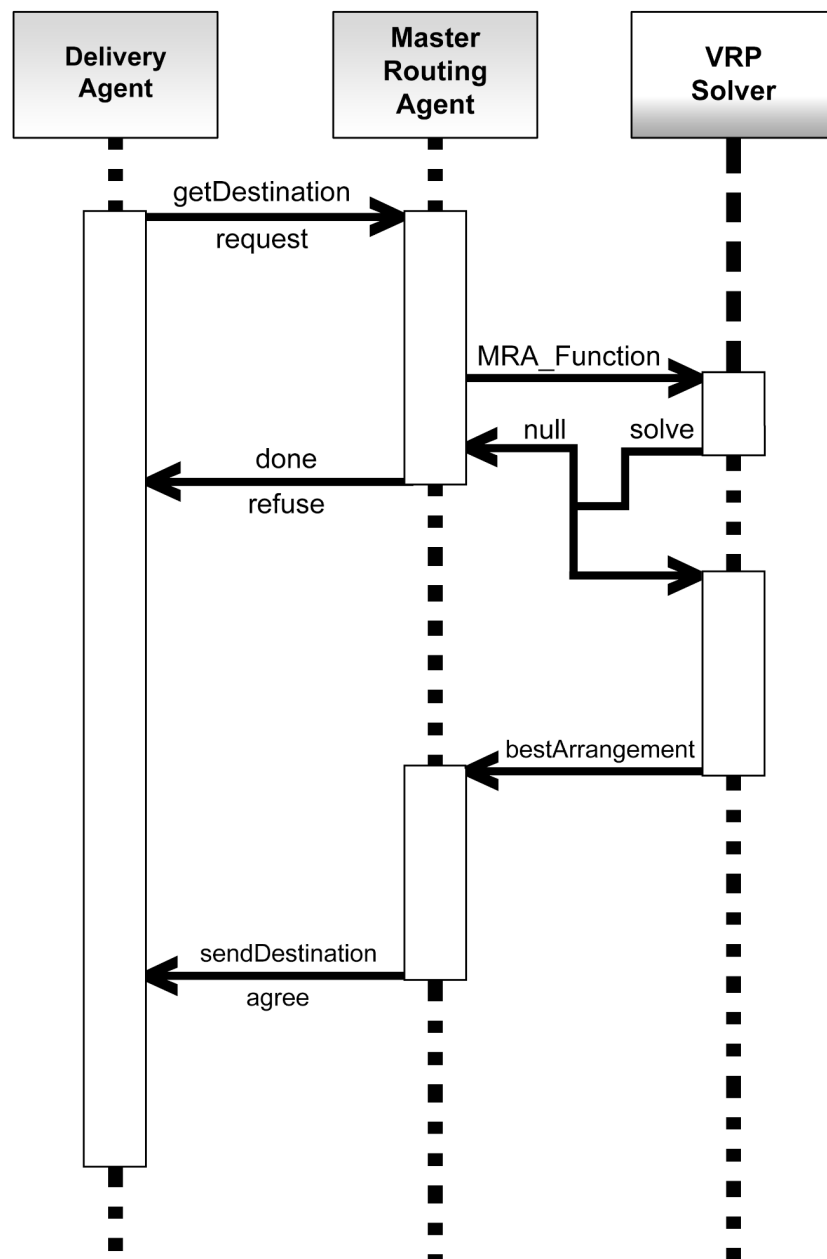


### 3. IMPLEMENTED INTERACTION PROTOCOLS

Interactions are based on the **FIPA Request Interaction** protocol for both communication and coordination between the Master Routing Agent and Delivery Agent(s).

The main performatives used are REQUEST and INFORM. Agents can either AGREE to or REFUSE incoming requests.

#### 3.1. Interaction flow



### 3.2. Utility functions

The system relies on two utility functions: **sendMessage** and **sendReply**. **sendMessage** is mainly used by the Delivery Agent(s) to initiate a REQUEST, while **sendReply** is used by the Master Routing Agent to respond with a reply message.

The content of a message always has a string identifier contained. These identifiers are stored within the **Strings** class, which the agents draw from to process incoming messages.

Content always has a performative but may also contain additional information suffixed after the performative with a **SEPARATOR** (represented as a comma character “,”) included in between. To retrieve the suffixed content, the receiver splits the body of the content by using the **SEPARATOR** as a regex.

### 3.3. Delivery Agent (DA)

Upon instantiation, DAs obtain the Agent ID (AID) of the MRA and run a **OneShotBehaviour** to broadcast a REQUEST to the MRA. Additionally, it suffixes the body of the content with its own capacity constraint.

After receiving the message, the MRA will process its content and run the solver. If an optimal route has been found, the MRA will AGREE to its request, affix the **sendDestination** string, and attach the best path to its content.

If the solver is unable to find an optimised route, then the DA will be rejected with a REFUSE response, and default to INFORMing the MRA that it has “completed” its job.

### 3.4. Master Routing Agent (MRA)

The MRA keeps track of each DA by storing their AIDs in its own list. Responses will only be sent to DAs with a matching AID.

DAs always initiate interactions with the MRA first by sending a **getDestination** string with their own capacities suffixed to the content. They will either AGREE or REFUSE requests depending on whether the solver can find an optimal route.

## 4. IMPLEMENTED SEARCH/OPTIMISATION TECHNIQUE

The solver method that the Master Routing Agent (MRA) uses to find the best solution is Constraint Optimisation Problem (COP). All knowledge of the problem is handled by the MRA which uses brute force search to find the most optimal solution for the problem data it has been given. The solver uses brute force search since it is searching through all the possible route combinations to find the most optimal solution that is within the constraints.

### 4.1. How a Potential Solution is Formed

How the solver searches for a solution is by taking the list of supplier locations (this is excluding the depot) and breaks them up into groups/chunks/routes, where one route is made per vehicle. These routes are stored in a 2D array (length of largest route X number of routes) then fed into a pair of permutation functions.

To simplify, these functions shuffle the locations in the 2D array between routes. One function permutes through the 2D array horizontally along the array and the other permutes it vertically. The Horizontal permutation is always done before the vertical permutations of the current Horizontal permutation, which is to ensure that all solutions are searched.

With this method all arrangements of locations between routes are explored for possible solutions while avoiding making the search go through **NumberOfLocations!**, which takes far longer to process than just the combinations between routes.

Once an arrangement of locations has been formed it is then checked against the defined constraints to see if it's a potential solution.

### 4.2. Soft and Hard Constraints

The solver determines the quality and the validity of a solution by using two constraints:

- Capacity limit (hard constraint)
- Minimise distance (soft constraint)

For the first constraint a potential solution is tested against is the capacity constraint since there is no point in analysing the solution further if it doesn't pass this hard constraint. Locations have a number that represents the weight/quantity of items that need to be collected by a vehicle, and each vehicle has a number that represents the maximum weight/quantity that the vehicle can carry. The capacity constraint is applied to each route by summing up the route's total capacity and comparing it with the assigned vehicle. If the total sum of a route's capacity is higher than the assigned

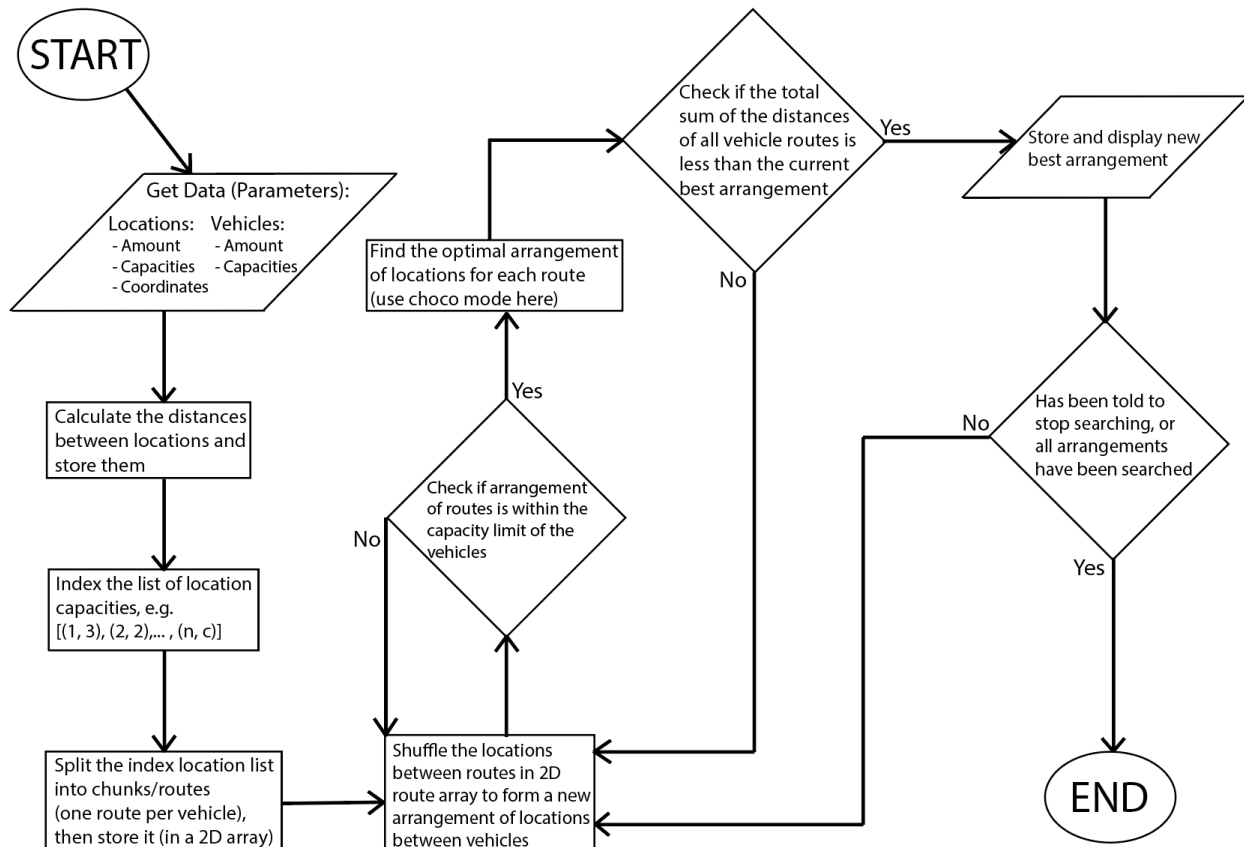
vehicle's capacity, then the potential solution is invalid. This also means that capacity can be defined as a weight or quantity, and each vehicle has its known unique max capacity which satisfies **Alternative 1** of the project.

If each route's capacity is within each vehicle's capacity limit, then the solution is valid and is processed to find the most optimal orientation for minimum distance. The solver measures the quality of the given solution by using the total distance of all the routes and comparing it to the current best solution.

Each route in the solution is put through a choco-solver model that finds the arrangement of locations in the route with the least total distance it a hamiltonian path starting and ending at the depot (in a similar vein to the Travelling Salesman problem). Once each route has been arranged for minimal distance, each route's distance is totalled and compared to the best current arrangements total distance. If the total distance of a new route arrangement is shorter than the current best arrangement, the solver updates it.

The solver then moves on to the next arrangement and continues until all arrangements have been checked or until the solver is told to stop searching.

### 4.3. Algorithmic Sequence Diagram of Solver

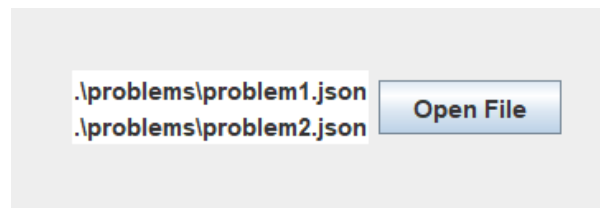


## 5. EXAMPLE SCENARIOS

Here are a couple of examples of the solver search for the best route combination.

### 5.1. Data Read from a File

Select a data file shown in the GUI:



Data is read from selected file which contains:

- The number of vehicles
- Capacity of each vehicle
- Number of locations (including depot)
- Capacity of each location (not including depot)
- Coordinations for each location (including depot)

In this example there are 3 vehicles and 11 locations

```
{
  "vehicle": 3,
  "vCapacity": [10,10,10],
  "location": 11,
  "lCapacity": [2,2,2,2,2,2,2,2,2,2,2],
  "lCoords": [[100,100], [196,84], [162,126], [132,184], [190,124], [128,84], [12,52], [74,90], [44,58], [144,34], [30,100]]
}
```

It then finds the best route orientations and displays them via a GUI.



```
Combination: 5249
Vehicle paths:
V1: [0, 7, 8, 6, 10, 0] ->
V3: [0, 5, 9, 0] ->
V2: [0, 3, 2, 4, 1, 0] ->
Total Distance: 705
Final Arrangement!
```

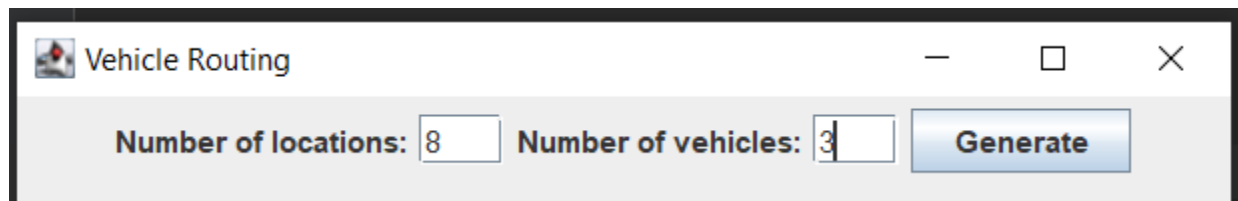


Final routes are assigned from the MRA to each of the DAs.

```
MRA: Route for DA#0 is -> 0 7 8 6 10 0
MRA: Calculating route for DA#1...
MRA: Route for DA#1 is -> 0 5 9 0
DA#0: Received message from MRA, processing...
DA#1: Received message from MRA, processing...
DA#1: Obtained route from MRA. Beginning dispatch along route -> 0 5 9 0
DA#0: Obtained route from MRA. Beginning dispatch along route -> 0 7 8 6 10 0
```

## 5.2. Generate Random Data

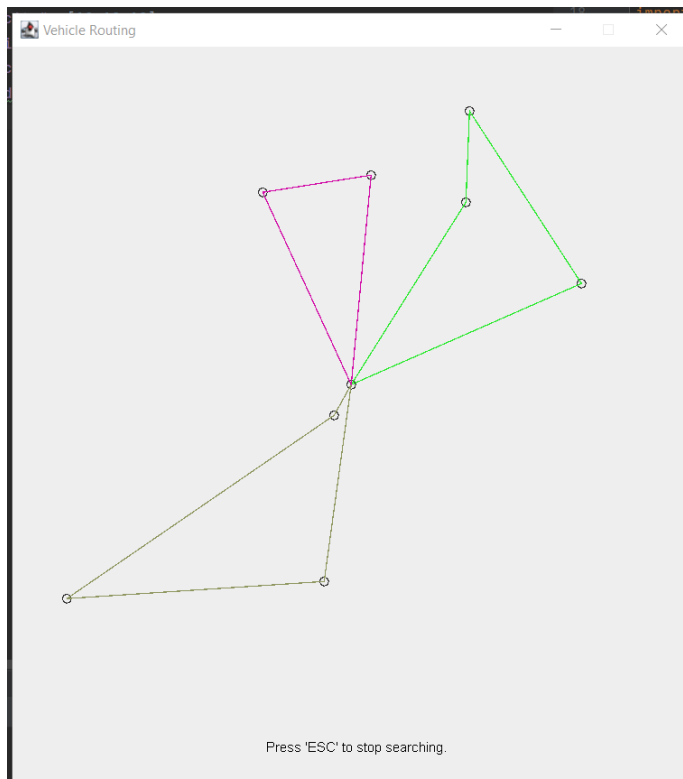
Enter the number of locations and vehicles for the random data set then press “Generate”.



Vehicle Routing

Number of locations:  Number of vehicles:

Generate a map, locations have randomized coordinates. Then find the most optimal arrangement.



```
Combination: 226
Vehicle paths:
V1: [0, 5, 6, 4, 0] ->
V2: [0, 7, 1, 8, 0] ->
V3: [0, 2, 3, 0] ->
Total Distance: 619
```

## 6. CRITICAL REVIEW OF IMPLEMENTATION

This section will detail some of the flaws with the program.

### 6.1. Processing can be slow

The solver uses a brute force algorithm approach since it searches through all possible combinations of locations between routes with one route per vehicle. Depending on the size of data that it is fed (number of locations and number of vehicles) the solver can take a long time to find the most optimal solution to the problem, and will only know that it's the most optimal when all the data has been searched.

This is partially an issue with large sets of data where search times can take a while to finish. A work around for this is to stop the solver mid-way through the search whether it be manually (button press) or pre-set (after a number of searches or set time limit). While this does reduce the search time it eliminates the certainty that the solver will find the most optimal solution. If the most optimal solution is further done the search then the solver might not find it. Using this fix will most likely give the user the most optimal solution that the solver has searched, not the most optimal in the data.

### 6.2. MRA does not respond in a timely manner to multiple DA requests

Depending on the amount of DAs that are instantiated, the MRA may not be able to respond to more than two DA REQUESTs at a time. A prime example of this occurring is with the dataset stored in **problem1.json** which has a total of 5 DAs involved.

Once the MRA receives a REQUEST from all 5 simultaneously, it will often ignore the first, third and fifth DAs, and only respond to the second and fourth DAs. When the solver has calculated the routes, the MRA will only reply to those 2 DAs whilst the remaining 3 either do not receive a response, or will be replied to only after several minutes have passed.

One possible solution to this is to stagger the initialisation of the DAs using a one-shot **DelayBehaviour** so that the MRA is not overwhelmed by the number of concurrent REQUESTs. Alternatively, a **TickerBehaviour** could be used in place of the current one-shot method. This would act as a failsafe so that if the DAs did not get a response initially, it would be able to REQUEST again until the MRA responds -- at which point, the TickerBehaviour would then call **stop()** to prevent further REQUESTs.

## 7. SUMMARY/CONCLUSION

The program works with any set of data it is given, provided that the data is valid (i.e. can't have 11 locations with coordinates of only 5 of them).

The Solver uses a COP brute force algorithm to find the most optimal solution between the hard (capacity) and soft (total distance) constraints. The solver will keep searching until all route arrangements have been searched or if the user prompts it to stop early. Capacity can be defined as a weight / quantity for locations and vehicles which satisfies **Alternative 1**. When a better solution is found the GUI display of the route arrangement is updated to show the new best arrangement.

Once the solver has concluded its search the Master Routing Agent will send each route to a Delivery Agent. Each DA obtains its capacity limit from the provided dataset. The MRA is then able to individually obtain this capacity from each of the DAs. Through the Solver, the MRA finally passes on optimised routes back to each DA.

The program does have some faults. First, since the solver uses a brute force algorithm, the program takes longer to search for the optimal solution the larger the data set is. The workaround of giving the user the ability to end the search early does reduce the search time, but removes the certainty that the solver will find the optimal solution.

Second, the MRA is unable to consistently respond to all DAs simultaneously depending on how many vehicles are included in a dataset. Replies to some of the agents may either be delayed or outright ignored if there are too many requests happening at the same time. Therefore, the suggested solution of staggering DA requests would be needed to help rectify this limitation.

Overall, the program demonstrates a functioning vehicle routing system. It is capable of providing a solution given a set of vehicles and locations. While it only offers basic routing with some limitations in its current form, it can be further improved upon to allow for less resource-intensive optimisations, or to accommodate more complex sets of problems.

## 8. RESEARCH SOURCES

### Index:   Details:

- 1     Google. 2019. *Vehicle Routing Problem*. Software Tool Application. Google OR-Tools. Viewed: 6/09/2021.  
<<https://developers.google.com/optimisation/routing/vrp>>
- 2     Sebastian Sanchez. 13/12/2016. *The Vehicle Routing Problem, Example*. YouTube. Viewed: 7/09/2021.  
<<https://www.youtube.com/watch?v=kserookKILM>>
- 3     OptaPlanner. 2020. *Vehicle Routing Problem*. Software Tool. Viewed: 4/09/2021.  
<<https://www.optaplanner.org/learn/useCases/vehicleRoutingProblem.htm>>
- 4     Altexsoft: software r&d engineering. 22/08/2019. *How to Solve Vehicle Routing Problems: Route Optimization Software and Their APIs*. Software Problem Discussion. Viewed: 8/09/2021.  
<<https://www.altexsoft.com/blog/business/how-to-solve-vehicle-routing-problems-route-optimization-software-and-their-apis/>>
- 5     Stack overflow. 08/2020. *Order array in every possible sequence*. Public Software Discussion Board. Viewed: 4/10/2021.  
<<https://stackoverflow.com/questions/14132877/order-array-in-every-possible-sequence/14133501>>
- 6     Choco-Solver. 03/06/2020. *Traveling Salesman Problem*. Tutorial webpage. Viewed: 23/09/2021.  
<<https://choco-solver.org/tutos/traveling-salesman-problem/>>