COS30018

Intelligent System

Project Title:

Vehicle Routing Problem

Team Members:

Tejass Sokhi – 104545000

Movindu Wijenayake 105016468

Leonald Emmanuel Ling Ji Zheng 102778969

Sudhish Raveendran - 104000460

Contents

[1.0 Introduction 3](#_Toc178950281)

[2.0 Overall system architecture 4](#_Toc178950282)

[3.0 Implemented interaction protocols 5](#_Toc178950283)

[4.0 Implemented search/optimization techniques 6](#_Toc178950284)

[5.0 Scenarios/examples to demonstrate how the system works 7](#_Toc178950285)

[6.0 Some critical analysis of the implementation 8](#_Toc178950286)

[Summary/Conclusion 9](#_Toc178950287)

# 1.0 Project Introduction

This project covers a version of the Vehicle Routing Problem (VRP), the Capacity constraint VRP (CVRP). The CVRP is described as a problem created when Delivery Vehicles are assigned to deliver parcels to certain locations. The vehicles leave from a home base/depot to deliver these parcels through the shortest path possible and return to the depot after deliveries are complete. The capacity constraint is implemented by setting a load that is lower than the total number of parcels, so that vehicles are required to return to the depot to restock parcels and go out again to complete their deliveries. The vehicles must take the shortest and efficient path possible to deliver these parcels whilst adhering and upholding the constraints.

We have implemented this problem using python and taken inspiration from Google OR-Tools to create the CVRP program. The program utilises a Master Routing Agent to create routes and assign them to Delivery Agents. Each Delivery Agent has a set capacity and max distance constraint that is assigned by the Master Routing Agent. The Master Routing Agent received an input text file containing parcels, their customer number and destination to be delivered to. The program is then presented to the end user as a Scatter plot, where the end user can input their desired number of locations, Delivery agents and distance range that each delivery agent can travel, this allows users to customize the settings and see how the routing algorithm works. The points on the plot are labelled by the customer number for easier viewing and tracking if the Delivery Agents are taking the correct route. A right-side log console is implemented to document the path each Delivery Agent takes and, this includes path cost, the total distance they travelled, where they travelled to and the number of packages they delivered.

# 2.0 Overall system architecture

The overall architecture of the system is loosely based off and inspired by the Google OR-Tools, the problem whilst it is VRP (Vehicle Routing Problem),m the more accurate term for it should be the CVRP problem (Capacity constraint). This problem and its solution was demonstrated in the OR-Tools examples, and our version of it implements ideas form OR-Tools but does not implement the module or API directly, it is merely an example and an inspiration.  
  
The main programming language we chose was Python due to readability and flexibility in terms of modules. The fundamentals of this project included creating a Master Routing Agent (MRA) which would create and optimize the routes given to the Delivery Agents (DA). The MRA also creates the DA’s themselves, the DA are defined as a class in the DA.py file. A visual was required to display the routes travelled by the DA’s, so we deemed that a cartesian plane was the most fitting visual representation. Below is a breakdown of each part of the program that creates the foundation of the CVRP Problem/Solution.

**CartesianPlane.py -** This file is kept separate from the rest of the program in a seperate file readability reasons and appropriateness of structure, a scatter plot is plotted onto a cartesian plane in the GUI file. Whilst it only contains a import random module and only creates the random points for the locations that are mapped later onto the GUI. The generate\_random\_points function determines the min max values of x and y depending on the num\_points created. This means a random coordinate is created for each point on the grapgh

**Parcels.py -** The parcels file is sectioned into two parts; the Package class and the write method, which writes to a txtx file which the MRA later reads. The Package class defines the necessities for a package to be exist, this includes the customer\_id (the customer that should receive the delivery), the destination (coordiante of delivery lcoation) and num\_parcels (number of parcels that need to be delivered to that particular location and customer). This is all defined in the \_\_init\_\_ method. The create\_parcels function assigns the necessary values to the variables created in the \_\_init\_\_ method and appends them to an array called ‘packages. The save\_packages method saves the necessary details for each parcel their respective variables, the write\_parcels\_to\_file methods is called to write to a parcels\_info.txt file that was named in the filename variable in the save\_packages method. To summarise, the Package class creates the definition of what each parcel should be and creates a txt file, writing to the file based on the number of locations and assigning each package with a location and customer number. This txt file can be saved, written to and later loaded by the MRA when generating routes.

**DA.py -** The Delivery Agent (DA) program primarily focuses on defining what a DA is and does not particularly focus on creating the DA objects themselves. The DevliveryAgent Class defines what each DA needs to exist and fulfil its purpose. Each DA has a da\_id to (identify) itself, a constraint (as per the CVRP requirement), a maximum distance it can travel (minimum distance is not set as per logic a vehicle can choose to travel or not travel depending on the locations and max distance, thus minimum distance can be 0, so it is not set as a necessary value each DA must have), a (load) representing its current number of parcels being carried (initially set to 0), and a route it needs to follow (received from the MRA). This is all defined in the \_\_init\_\_ methods. The (capacity constraint) is defined by a random integer between the range of 20 – 50, whilst this may seem like a large amount and not adhering to the requirements of the CVRP problem, the number of packages generated are random too, furthermore the number of locations can always be increased to further push the vehicles to continue to revisit the depot to refill on packages. The (maximum distance) is set to self as the user determines the maximum distance when the program runs, respective get and set maximum distance functions are created to get the integer from user input and set it appropriately. The (recieve\_route) sets the route to self, as the (MRA performs the route assignment) later, a confirmation message is generated if a vehicle is at full capacity, which is mainly for logging and troubleshooting reasons, used if necessary. A (get\_status) method simply returns the da\_id, capacity and current load, just used as a progress check. The (create\_delivery\_agents) function utilises the DeliveryAgent class to create the delivery agents, although this method is defined here, it is utlised by the MRA when creating the DA’s. Overall, the DA defines a DeliveryAgent class which determines the attributes a DA requires to be defined, the DA’s creation method is defined in the DA program however is called by the MRA when needed.

**MRA.py -** The MRA is a large program mainly focusing in the creattion and optimization of routes for DA’s. The MRA also creates the DA’s through the create\_delivery\_agents function defined in the DA.py, The MRA receives the list of parcels, assigns each respective DA with the parcels to deliver to their locations. The MRA ensures that if a delivery cannot be made, whether due to low number of DA’s or Max distance being insufficient, the user is notified through the GUI porgram. The MRA also handles printing the vehicle logs (where the vehicle has travelled to, the total cost and the number of packages delivered).

The load\_parcels\_from\_file method reads the file set as the filename parameter, stripping it and skipping any header rows. The contents are then placed into a parcels array and returned to later be assigned to DA’s.

The MasterRoutingAgent Class defines the necessary methods to calculate, compute and optimize the routes for the DA’s, and it also creates the DA’s. The \_\_init\_\_ method is catered towards defining the number of agents, capacity per agent and depot location, other variables are set to none as they are no necessary yet or are to be determined under certain circumstances and change by the user (such as distance changes depending on the number if points or their coordinates, whilst delivery\_agents can change depending on how many the user creates).

The set\_parcels functions is made up of two main functions that are involved in the capacity constraint and the distance calculation:

The first is the \_adjust\_capacity function, which compares the total number of parcels and the total capacity of the agents altogether. If the total capcaity is greater than or equal to the nmber of parcels, this means that the CVRP problem is not created, as there is no problem if all parcels are delivered in on trip due to the load of each DA satisfying the deliveries without returning to the Depot. Thus, if this condition is met, the capacity is reduced by 30% and the new capacity is set. The create\_delivery\_agents function is then called passing in \_adjust\_capacity to ensure each DA has a capcaity less than the number of packages being delivered

The second function, \_precompute\_distances, uses the calculate\_distances function (a function which executes the distance formula) to create a distance matrix usig the total number of locations (the depot + every destination assigned to each parcel). The destinations that the DA’s need to visit are retrieved from each parcel, as each parcel is assigned a destination already.

The optimize\_deliveries function is a key part of the MRA, asi it intializes the routing process for all DA’s and maintains a list of unassigned parcels, routes for DA’s, parcels delivered by each DA, current loads and distances travelled. For each DA, it ensures that the DA starts from the depot (whether it is beginning its route or has returned to pick up more parcels), it also looks for the most optimal route based on thr DA’s remaining capacity, distance to customers, the maximum distance constraint and number of parcels left to be delivered. The MRA prioritises delivering more parcels over minimising distance travelled (as the customer takes priority. If the optimal conditions cannot be met, the DA returns to the depot, and always ensures that every DA always returns to the depot. Overall, the MRA ensures an efficient route is created for each DA to take, minimizing any failure

The last three functions (calculate\_route\_costs, calculate\_detailed\_route costs and print\_routes) focus on calculating the distance travelled by each DA and prinitng it onto the left pan/console when the porgram runs. It focuses on point-to-point calculation and total cost of the travel

**GUI.py -** The GUI program mainly create the user interface for user interaction, little logical calculation is done in this part of the program (mainly done in generating\_locations). The \_\_init\_\_ method sets the default values for the essential variables, such as number of DA’s, locations and capacity per agent, these are just starting/placeholder values that can be changed by users later. This also creates the canvas for the cartesian plane and scatter plot to be plotted to. The create\_widgets function creates the creates the labels for the the x and y axis and responsive texts meant for the end user to know the state of the route generation. The remaining functions all use imports from the MRA, CartesianPlane and parcels python files to create the scatter plot, randomnly generate points and ensure the routing is done optimially. Several status texts ae displayed to the end user, such as path costs, such as if the maximum distance is exceeded, if the route genration has failed, and printing the total cost, distances travelled and max distance from each DA. The actual scatter plot and the points on the plot are color coded, blue for the delivery locations/customers and red for depot, each location is labled as C(n) (n being the number of the customer) so is easier to visually identify the route each DA takes. Each DA has its own colored line to follwo with directional arrows and a dotted line that signifies a DA returning to the depot (whether it's to restock or finish its route). All main movements by the DA are printed onto the left log/console.

# 3.0 Implemented interaction protocols

There is no predetermined or structured method of communication we have set for out interaction protocol (for example the contract net protocol (CNP) is a structured method of communication). So, whilst we may not have a named/formal interaction protocol, the Master Routing Agent (MRA) and the Delivery Agent(s) (DA) implement a centralized coordination protocol, where the MRA acts as the central coordinator for the multiple DA's.

The MRA initializes the problem by loading in the parcel information and creating the DA’s. Route Optimization is then executed, as the MRA uses a heuristic algorithm to optimize routes for each DA, and iteratively assigns parcels to each DA based on their capacity and maximum travelling distance. The MRA then continuously updates the DA states as parcels are assigned. The DA’s execute their assigned routes independently and report their status to the end user. This proves the interaction between a human user, and two intelligent systems, the MRA and the DA, where the MRA acts as the coordinator and the DA executed any tasks provided by the MRA, reporting back to the end user.

This communication from the MRA is sent out to all DA’s at once, the protocol ensure efficiency and coordination between the MRA and DA’s without requiring direct communication between individuals (similar to a broadcast from CNP, but not a request for assistance but rather assignment to all capable agents), this simplifies the overall architecture while still upholding route optimization standards.

4.0 Implemented search/optimization techniques

The search technique implemented in the CVRP program is the Nearest Neighbour Search (NNS). This technique is the optimization problem of finding the point in a given set of data that is closes to a given point. In combination with constraints, the idea of what the “closest” or most optimal solution changes depending on the constraint. In context of a standard VRP problem, each DA is to follow a route given by the MRA that follow this exact search technique, where the shortest path possible is taken (meaning the closest point is visited by the DA) until the route is complete and all parcels are delivered.

However, in a CVRP program, constraints can change the meaning of what the closest point is considered, it rather changes from literal distance between two points being given priority to what satisfies the constraint and fulfils the objective (being delivering all parcels). This is demonstrated through our CVRP Program, as despite the distance between two points, if the max distance is less that the total distance needed to travel, and the load has reached 0, the DA will prioritize and consider the depot as the closest point, this is because the MRA ensures that the constraint conditions are met by all DA’s, otherwise the current instance of the solution is considered unfeasible.

In summary, due to our CVRP program, the NNS technique focuses on prioritizing the constraints, which is viewed as the most similar to a closest point, as without the constraint being satisfied, the solution would be unfeasible.

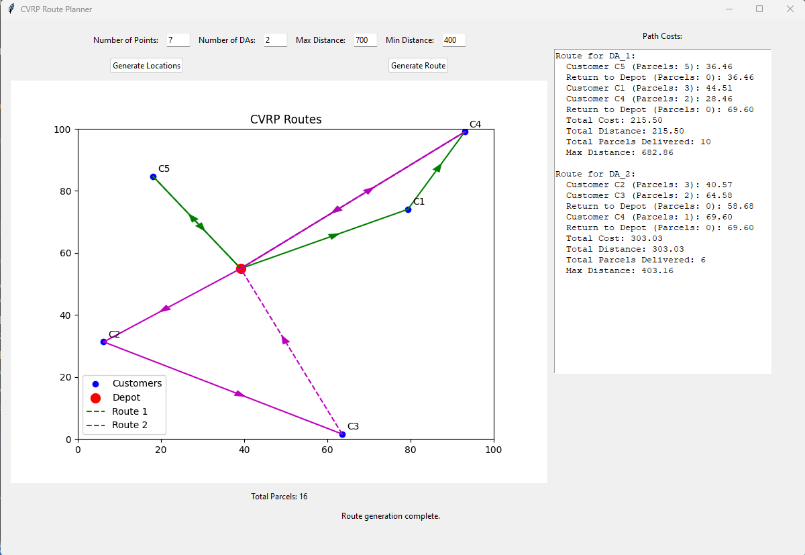
# 5.0 Scenarios/examples to demonstrate how the system works

To display the full functionality of the CVRP Program and its optimization/search technique, we have to consider several scenarios:

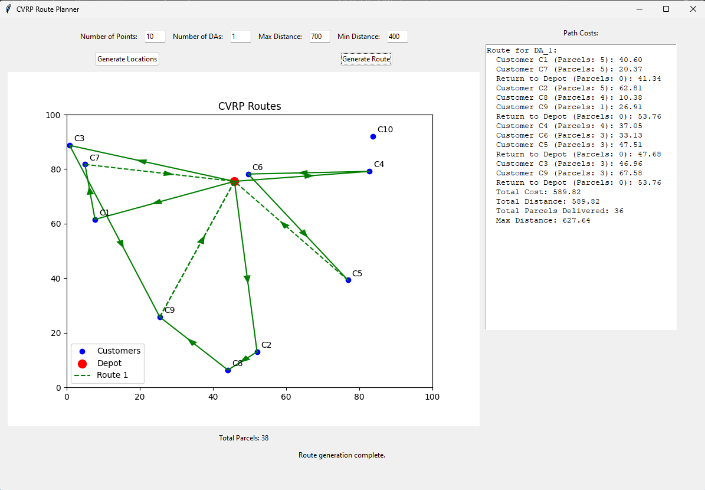
1. The Number of DA’s and the Distance Range is at enough
2. The Number of DA’s are insufficient
3. The Distance range is too small
4. The Number of DA’s and the Distance Range is insufficient

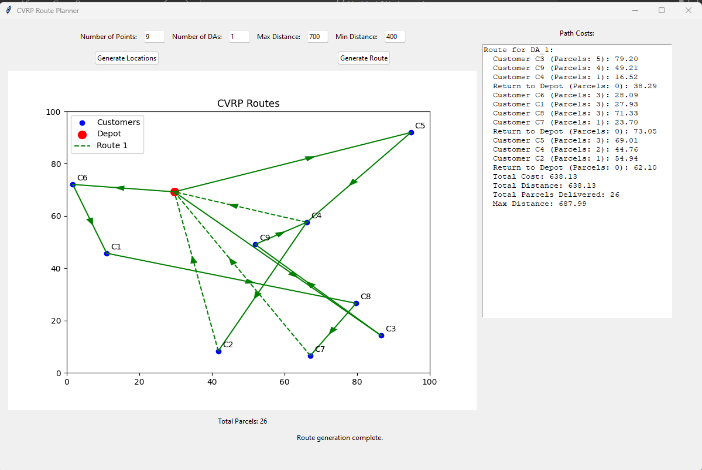
Note: Despite any sufficiency or insufficiency in the constraints, the DA’s will always have to return to the Depot to ensure that there is an actual problem (as it is constraint vehicle routing problem).

Scenario 1:

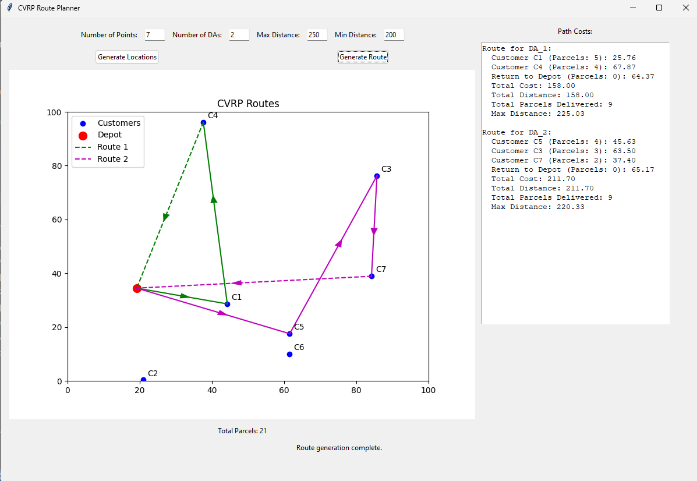
In this scenario, the number of DA’s is set to 2, points are 7, and the distance range is set to 400 - 700. As both constraints are satisfied, the programs execute with no visible problems (DA returning to the depot is part of the CVRP requirement thus it is not the problem)

Scenario 2:

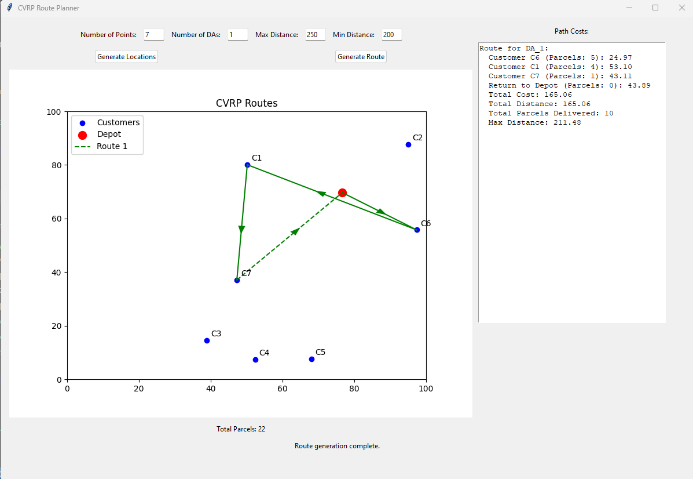
The distance range has not changed, as we have lowered the number of DA’s to 1 and increased the points to 10, now as seen, due to the lower number of DA’s and increased points, one DA must make two trips back to the depot, however does not have enough distance to travel to the C10. This does indeed mean that the distance range is not satisfied either, however the solution does not complete mainly due to the lack of multiple DA’s and increase in points rather than distance constraint, as if we regenerate the solution, there may be a point where a singular DA can satisfy the distance constraint, but would always need to make multiple trips back to the Depot due to the increase in points and decrease in DA’s (as shown below)



Scenario 3:

The distance range has changed to 200 – 250 and the DA number and points have returned to the same as scenario 1, as you can see, if we use scenario 1 as the basis for the standard solution, in theory this solution satisfies the capacity constraint, however the distance range is too small to allow the DA’s to travel to all points. If the number of points was lower, the distance may have been satisfied.

Scenario 4:

One thing to note through the previous scenarios is that at most, only one point was left unvisited, even in other iterations where some points were a great distance away, at most 2 points would be left unvisited. However, in the case of both the Number o DA’s and the capacity constraint being insufficient, the singular DA can barely visit any points and is forced to return to the depot due to distance constraint, thus this is not a feasible solution

In conclusion, if one constraint is not satisfied, it affect the other constrain too, if we set scenario 1 as a standard precedent to be compared to, both constraints were satisfied, hence the solution being all locations are visited, with minimal return to the depot, however with the lack of one constraint satisfaction, other constraints are affected and the solution eventually becomes incomplete/unfeasible.

# 6.0 Some critical analysis of the implementation

The main goal of the program production was to create a CVRP solution which optimizes route creation and ensures each DA completes its basic assigned tasks, to deliver all parcels whilst taking the shortest route possible. Our program successfully met the requirements for this goal, however through out production and now at the end of it, we can review and reflect on production process and implementation of the solution. A major implementation improvement file/directory structure. A major offender of this is the CartesianPlane.py program, as this program only contains one function which allows us to perform random point generation. The idea was to separate certain frequently used function for easier readability, however as the production went on, and eventually towards the end of production we realised that the function saved in a separate file was unnecessary. This perhaps can be considered bad code design, however another aspect we had to consider was to not include functions unrelated to the files purpose in them, thus we kept the CartesianPlane.py file. As the GUI.py program mainly utilised the CartesianPlane.py file, a modification that can be made could be including the scatterplot functions inside the CartesianPlane.py file, thus upholding clean code and readability.   
  
Another improvement of implementation may be the environment we used. A python environment, whilst flexible and readable, does still have scope limitations which could have been easily overcome by the use of Java and JSON intelligent agents, if a remodelling of the program was ever to be done, perhaps the use of Java and JSON could have been more useful in the creation of DA’s.

# 7.0 Summary/Conclusion

The CVRP solution produced by our program fulfils the main criteria of the CVRP. The MRA acts as a central coordination system, which implements the Near Neighbour Search (NNS) technique to optimise the routes taken the DA’s. This NNS technique ensures that DA’s take the shortest path possible to deliver all packages, whilst upholding the constraints required to uphold the problem. Our python environment ensured flexibility in our solution, with eh vast number of libraries and inspiration from Google OR-Tools, we were able to recreate the Routing optimization and the constraints in an efficient and effective manner. The GUI displays the program output and how DAs are updated overtime and what routes they take, it displays the solution in a human readable manner. This way users can test the limits of the solution and study how the problem changes depending on the number of parameters you set. Whilst certain code design and environment improvements can be made in the future, the CVRP program ensure that requirements for problem creation and solution are met to perform and illustrate the problem and solution.