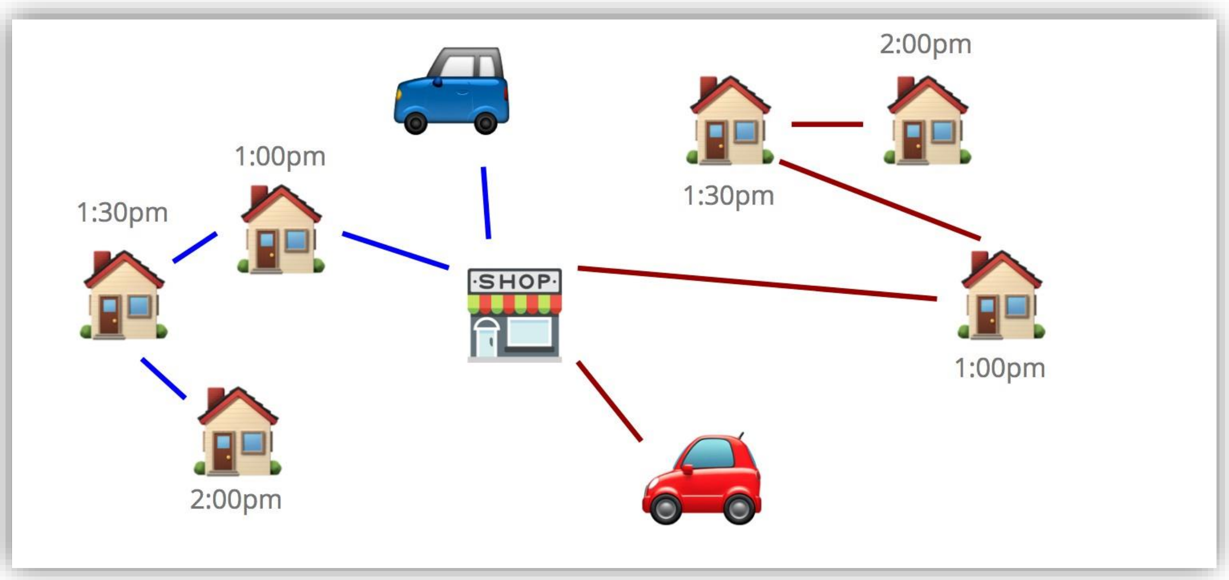
Vehicle Routing Problem with Time Windows



VEHICLE ROUTING PROBLEM WITH TIME WINDOWS (VRPTW) CAN BE DEFINED AS CHOOSING ROUTES FOR LIMITED NUMBER OF VEHICLES TO SERVE A GROUP OF CUSTOMERS IN THE TIME WINDOWS. EACH VEHICLE HAS A LIMITED CAPACITY. IT STARTS FROM THE DEPOT AND TERMINATES AT THE DEPOT. EACH CUSTOMER SHOULD BE SERVED EXACTLY ONCE.

SUBMITTED BY- SHOURYA MEHRA

TABLE OF CONTENTS

Introduction

The core of this problem is similar to variations of vehicle routing with time windows**(VRPTW)**. Specifically, **stochastic** or **dynamic vehicle routing with time windows**.

A maintenance company typically consists of technicians who drive around & attend maintenance jobs and Service Managers who manage & help these technicians to go around in an optimal way, driving as less distance as possible and attending as many maintenance jobs as possible. Service Managers plan technician visits in advance for all regular checks.

The plan includes a schedule (like a timetable) indicating which technician goes where, at what time. These **maintenance jobs** have a **specific duration** (60 minutes in this problem statement, including **travel time**).

Problem Statement

The challenge is that technicians get **callouts and emergency calls** during the day. This could be for various reasons like - AC is not working, people got stuck inside an Elevator etc. When Technicians go to attend these on demand orders, **they deviate from the plan and their route may not be optimal anymore**.

The task is to attempt finding a way that could automate our (Service Manager's) manual work of creating a plan for Technicians.

Also, the solution should consider how to handle the scenarios of callouts and emergencies.

<u>Deliverable</u>

1. Create a private GitHub, BitBucket or GitLab repository to share your code including all the commits you make along the way.
2. Submit a solution that works, it would be great if you can make it easily runnable with helper scripts/code.
3. Include a README detailing your approach, its limitations

Evaluation

1. Understanding the problem or the business use case -your understanding of various scenarios and corner cases.
2. Presentation of both the problem and the solution -data chosen to try out with, steps that were taken while solving and documenting them.
3. Documentation of limitations.

Research

The vehicle routing problem with time windows is an extension of the well-known vehicle routing problem (Crainic and Laporte, 2000, Toth and Vigo, 2002).

One of the best-known routing problem is at the same times the simplest one namely the travelling salesman problem (TSP). The route has to be constructed in order to minimize the distance to be travelled. The vehicle routing problem (VRP) is the m-TSP .**If we add a time window to each customer, we get the vehicle routing problem with time windows, where a vehicle now has to visit a customer within a certain time frame.** The vehicle may arrive before the time window opens but the customer cannot be serviced until the time windows open.

Many decision problems in business and economics, notably including those in manufacturing, location, routing and scheduling may be formulated as **optimization problems**. Typically, these problems are too difficult to be solved exactly within a reasonable amount of time and heuristics become the methods of choice. In cases where simply obtaining a feasible solution is not satisfactory, but where the quality of solution is critical, it becomes important to investigate efficient procedures to obtain the best possible solutions within time limits deemed practical.

Often the number of customers combined with the complexity of real-life data does not permit solving the problem exactly. In these situations, one can apply approximation algorithms, which are feasible but not necessarily optimal solutions.

Approaches

We came up with two approaches to get the optimal way of attending all our clients:

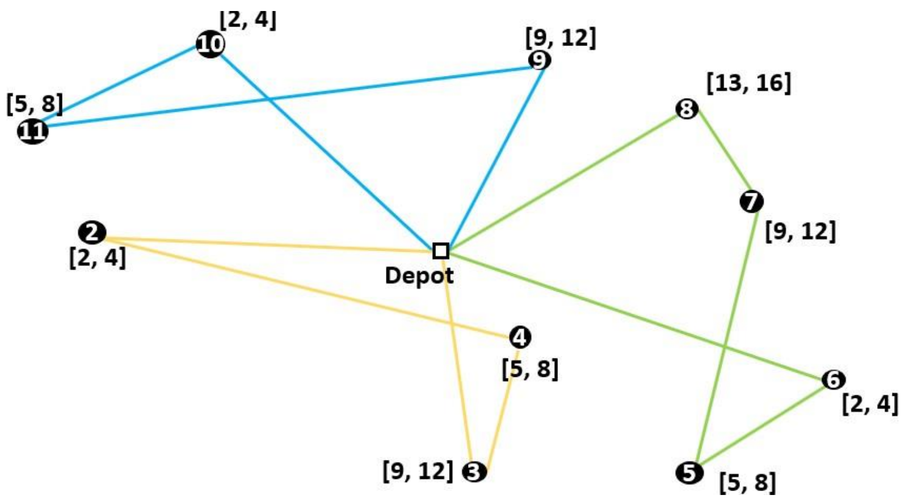
1. All the technicians start from one depot point and **travel radially outwards** from it, attending the client closest to them.
2. Assign **zones** to every technician, all of whose clients the technician must attend in a day.

We went with the second approach as it compartmentalizes the problem efficiently and demands much less improvisation, from both, coding as well as practical point of view.

Details of the chosen approach

All the locations are first clustered using KMeans clustering keeping **number of clusters as total_locations/10 where 10 represents maximum working hours of one technician.**

Each cluster thus formed represents the zone of a particular technician, and every point (representing a client) of the cluster must be attended in a day.



Sequence for the technician is calculated by fixing a starting point in the cluster and **iterating through the shortest point subsequently**. So, if the technician starts from a position ‘A’, the next service location will be the one closest to ‘A’, say ‘B’. **Now the next service location will be the one closest to ‘B’, say ‘C’** and so on.

Emergency case

After every iteration (completing service of one client), the user has an option **to introduce an emergency call.**

If the user does so, **the technician closest to the location of emergency call attends the emergency, while the other technicians carry on with their routine**

sequence.

The technician, after attending the emergency, attends the next regular client's location **closest to the emergency location**. This next regular client's location **may or may not be the same as the one in the sequence before the emergency was introduced. Thus, efficiency is maintained by recalculating the sequence for a particular technician after attending an emergency.**

Challenges faced

1. Since there is no **defined line for the range of duration of travel time and/or time required to get a job done, in 60 minutes,** the solution started to seem vague. To tackle this, **the dataset is first filtered so that the travelling time around 10-20 minutes, hence assigning about 45 minutes to get the job done.**
2. Getting a dataset which addresses
    a. geolocations of buildings,
    b. respects the 60min job time,
    c. and seems realistically feasible, all at the same time.
    d. Workaround to this **was filtering the locations** of dataset such that locations of the same clusters are neither **too close(50meters) to each other nor very far apart(40kms).**
3. Getting appropriate distance and duration of travel from one location to another, as **we cannot take Euclidean distances and travel at constant speed in reality**. On top of that, incorporating the road traffic aspect in travelling. This can be tackled by using **Google Distance Matric API (mentioned in Enhancements later) which returns realistic duration and distance of travel from Google Maps.**
4. Optimal sequence of clients to be attended per cluster.
5. **Attending emergency calls in-between regular clients**, which was made possible by **halting the program after a client is serviced, to take in input for emergency**, if any. Thus, in our case this happens at a maximum of 10 times. (10 hours).

Limitations

1. A technician cannot attend an emergency **before he finishes the ongoing job.**
2. Distance calculated is the Haversine distance, which is the **angular distance between two points on the surface of a sphere.**
3. Every technician does not have equal number of jobs to attend. Maximum being 10, few technicians are only assigned to 6-7 jobs. Thus, **nonuniformity of workforce which can be taken as a loss in efficiency.**
4. To some extent, the solution is centred around particular dataset used, and holds practicality till those datasets, **whose points of a particular cluster are not far apart from each other.**

References

- https://developers.google.com/optimization/routing/vrptw
- https://gurobi.github.io/modeling-examples/technician_routing_scheduling/technician_routing_scheduling.html
- https://developers.google.com/maps/documentation/distance-matrix/overview
- https://www.youtube.com/watch?v=v9tUEsHD6BE
- https://www.codegrepper.com/code-examples/python/python+keyboard+press
- https://pypi.org/project/mpu/