VehShare

Often we see there are multiple sources and destinations for a ride and due to inefficient path following, transportation costs increase abundantly.
So the goal of this project is to make an efficient system which can give an efficient path by taking locations of various sources and destinations keeping in mind the constraints to be optimised like time, profit, fuel, etc.
Our system will be using some heuristic based and genetic algorithms.

Formally the problems are described in three categories

1. ORIENTEERING PROBLEM
   Objective : Find an s − t path of total length at most B so as to maximize the profit collected by the path.

2. ORIENTEERING PROBLEM WITH TIME WINDOWS
   Objective: Find an s − t path within the budget and satisfying time windows, so as to maximize the profit collected by the path.

3. CONSTRAINED ROUTING FOR RIDESHARING
   Objective: Find an s − t driver route within the budget, satisfying all constraints and pick up-drop off customers.

System:

- 4 gb RAM
- OS - Ubuntu 20.04.2 LTS, 64bit

Install libraries:

- Gcc compiler for ubuntu
- curlpp
  commands
  sudo apt update
  sudo apt install libcurlpp-dev

Project folder structure:

-NiravRoute16

-Code	//contains all the code
-Orienteering	//Algorithms implementation
-Orienteering_with_time_windows	
-Ridesharing_system	//Our main code lies in this
-Other important docs	//useful documents

-papers //research papers
-presentation //project ppt
-Thesis report //report - contains detailed report

Of the above folders, “Ridesharing_system” folder consists of all the code.

There are two modules in our Ridesharing system viz. Ridesharing algorithm and Allocation module.

Ridesharing Algorithm
This module basically does route allocation. Here we will have some “n” drivers and “m” customers. Then we are going to assign these “m” customers to “n” drivers keeping in mind the path and time window constraints.

Following is the important and detailed information regarding the Ridesharing Algorithm module
Input to the Ridesharing Algorithm
Inside Ridesharing Algorithm folder go to “*project/day1_20_21/unfiltered_data*”. This is where we will store our input files(requests file).
There will be one request file per driver with the naming convention as “requests52561.txt” where 52561 as the driverID.
Inside requests{driverID} file:

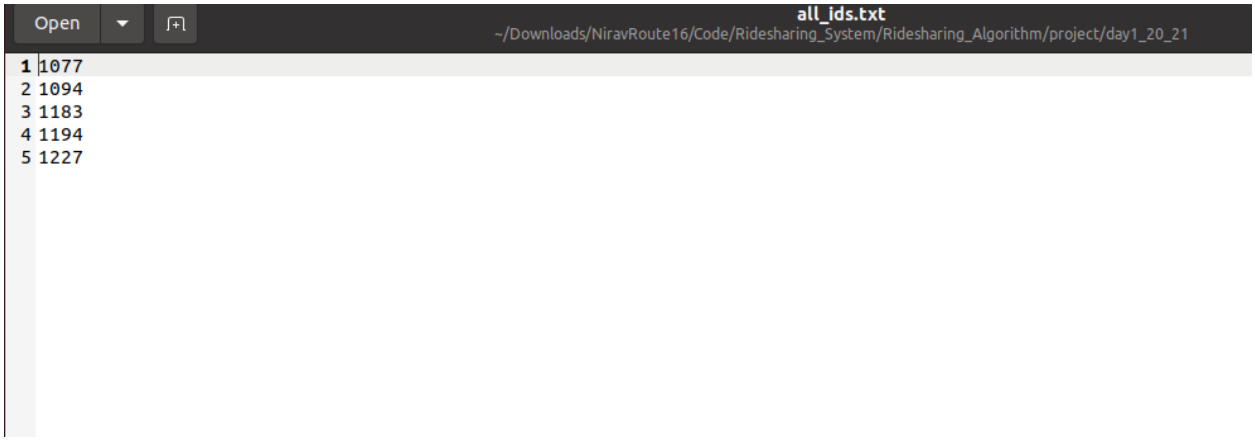
1. Firstline will contain
   tmax driver_tolerance
   Where
   tmax = maximum duration for which the driver can provide service
   driver_tolerance = waiting time of driver after reaching the pickup point

2. Secondline will contain:
   driverID source_latitude source_longitude dest_latitude dest_longitude opening_time profit
   Where
   driverID - ID assigned to driver(unique and same as ID part of file name)
   Opening_time - time at which driver is willing to start the trip
   Profit -

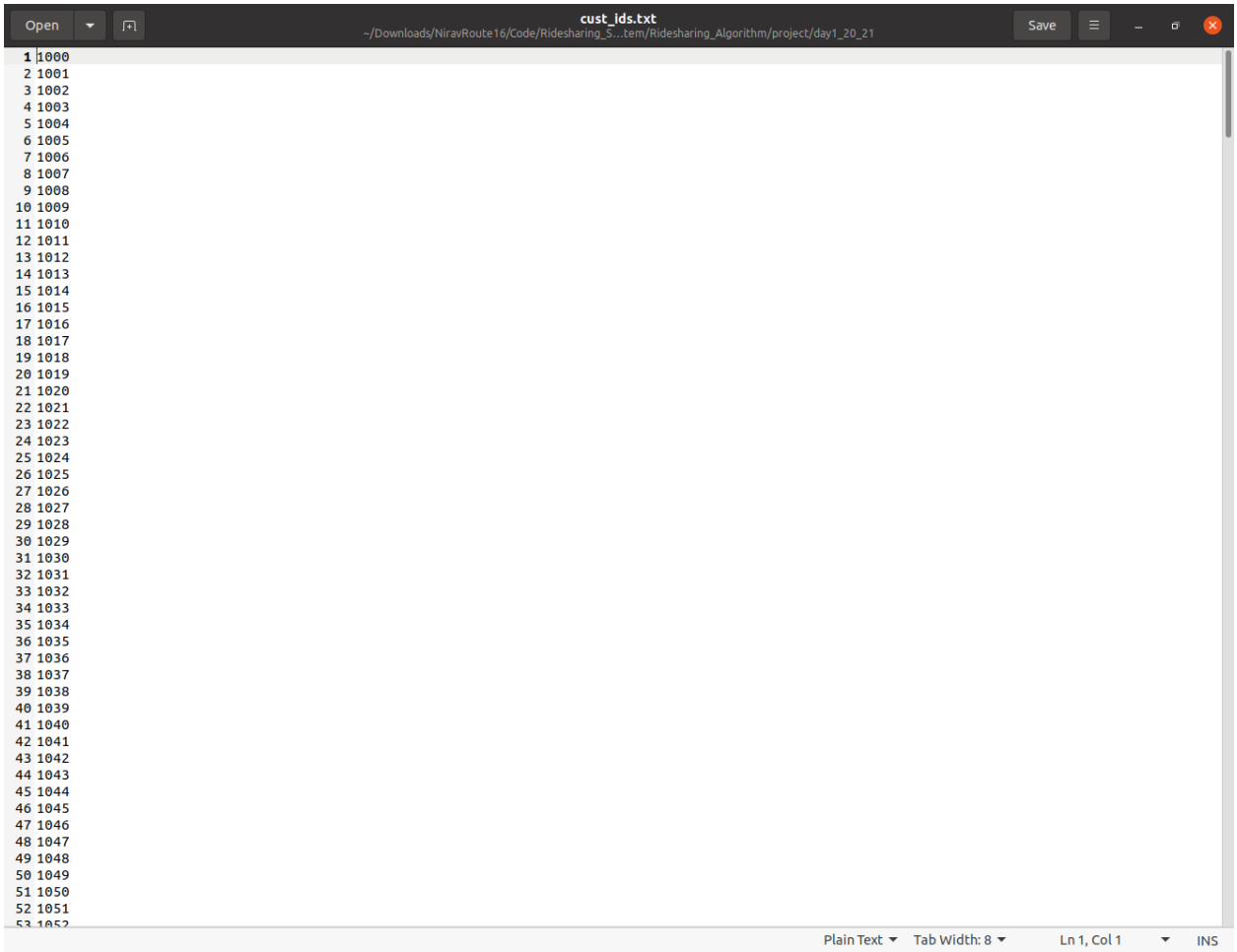
3. Third line onwards, details of customers in the form
   custID source_latitude source_longitude dest_latitude dest_longitude profit opening_time closing_time

   Profit = can be some fare for the trip
   Opening_time = time at which customer will be available for getting service
   Closing_time = time before which the trip should get completed

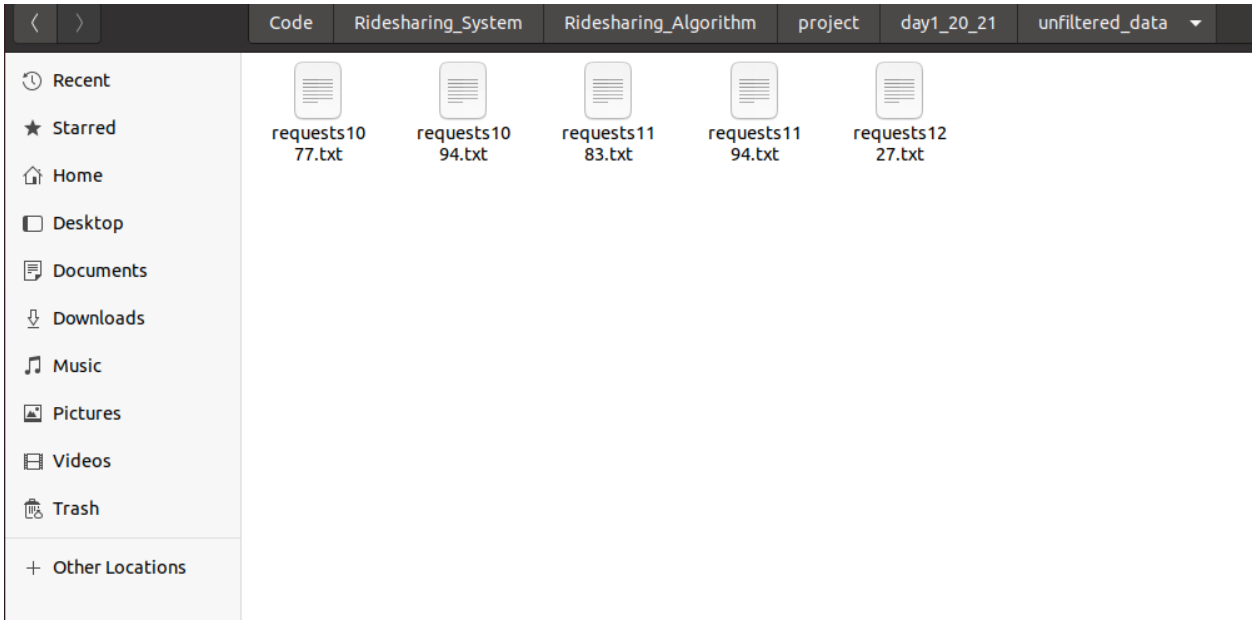
Screenshot:



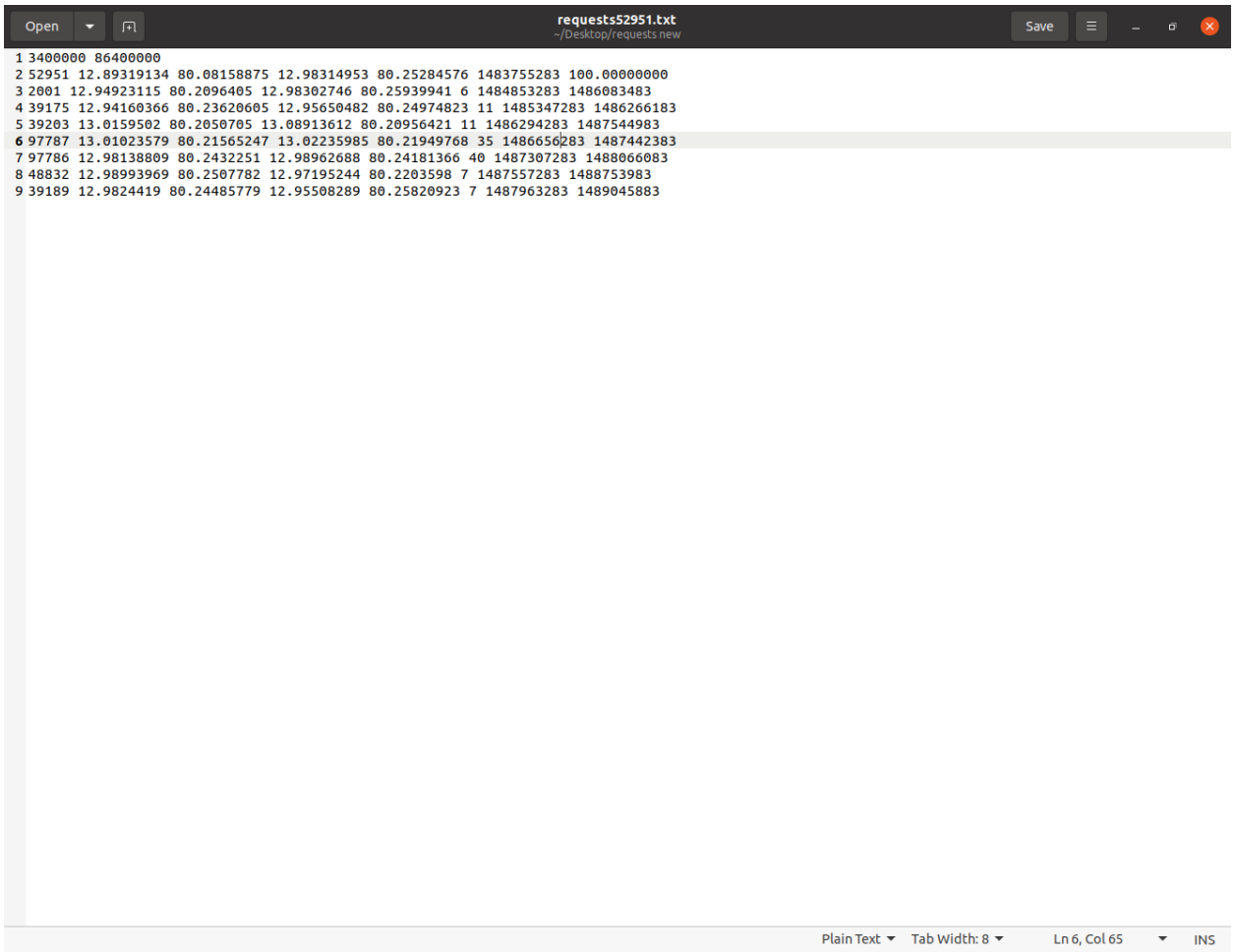
all_ids.txt(driverID's of all the drivers)



cust_ids.txt(contains custID's of all the customers)



Requests file for each driver inside /project/day1_20_21/unfiltered_data



This is the requests{driverID}.txt file which is basically our input file(every line explained above)

Note: tmax, driver_tolerance, opening_time, closing_time all will be in milliseconds

Inside Ridesharing Algorithm folder go to "*project/day1_20_21/all_ids.txt*"
In *all_ids.txt* we will have all the ID's of the **drivers**

Also there is "*cust_ids.txt*" which contains ID's of all the **customers**

This is all about input files, **output** of our **Ridesharing Algorithm** is a "*serial.txt*" file at location "*/project/day1_20_21/serial.txt*"

How to compile and run?
The main file for "Ridesharing Algorithm" is in "*newOP.cpp*" file.
So for compilation, command is:
g++ newOP.cpp -o newOP -lcurl -lgomp

-lcurl to link the curlcpp as we are sending http requests and using responses from osrm server.
-lgomp to link gcc module for getting time

Command to run
There are three steps in the Ridesharing Algorithm namely step1, step2 and step3
So for running we need to specify what step we want to run as command line argument
./newOP step1
./newOP step2
./newOP step3

step1 - Inside newOP.cpp there is a function step1() which basically deals with filtering the requests and assigning it to drivers considering coordinates(stored at /project/day1_20_21/filtered_requests_folder)

step2 - This function basically deals with the distance matrix creation(stored at /project/day1_20_21/final_data_20_tmax_1.5/distance_matrix)
Also it remove drivers from data and generate final files

Step3 - This function is the main implementation of the genetic algorithm(GA.cpp) of this module.

On successful completion of all the three steps, a "**serial.txt**" file will be created under "/projet/dat1_20_21". This is our output file which is in binary format and acts as an input to the Allocation module.

Allocation Module

This module basically deals with the allocation of customers to drivers by identifying any conflicts(like the same customer assigned to more than one driver), resolving them and final allocation of customers to drivers.

Project folder structure:

-NiravRoute16
- -Code //contains all the code
  - -Orienteering //Algorithms implementation
  - -Orienteering_with_time_windows
  - -Ridesharing_system //Our main code lies in this
    - -Allocation module //main folder for allocation module
    - -Ridesharing Algorithm
- -Other important docs //useful documents
- -papers //research papers
- -presentation //project ppt
- -Thesis report //report - contains detailed report

Inside Allocation module folder there are

-src //contains all code for this module

-Readme

-Ridesharing input

Input to Allocation Module:

Copy paste the “serial.txt” file from Ridesharing Algorithm module to “Allocation module/src”. This file will be our input to the Allocation module.

How to compile and run?

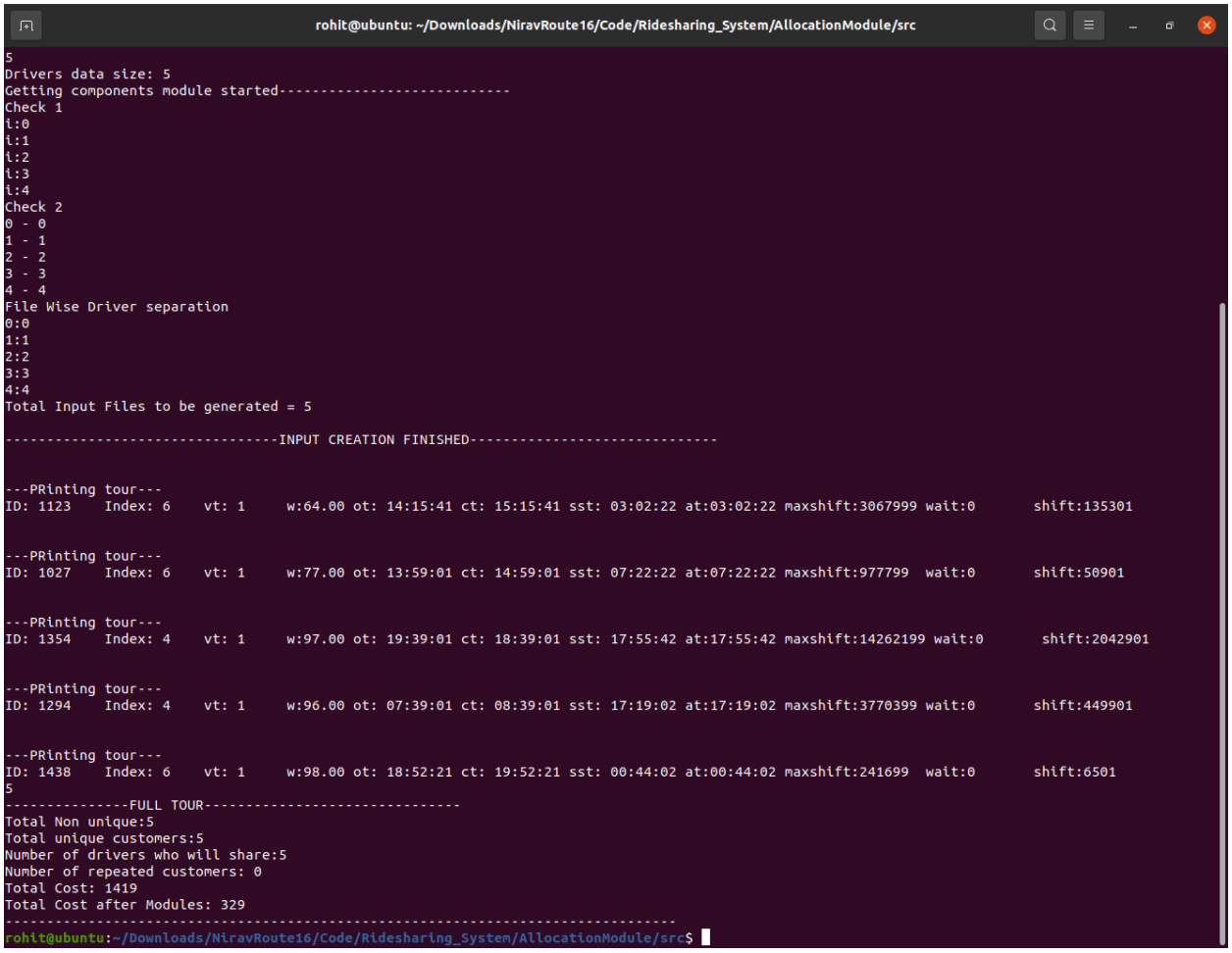
InputProcessor.cpp contains the main file and hence we can compile our code by following command

g++ InputProcessor.cpp -o InputProcessor

For running

./InputProcessor

Screenshot:



Note: Will update explanation of each field afterwards

Output file

Output for Allocation module is generated under “/Allocation module/src/finalfiles”
It shows the trip information.

Sample output:

1-1077 1-1123 2-1123 2-1077
1-1094 1-1027 2-1027 2-1094
1-1183 1-1354 2-1354 2-1183
1-1194 1-1294 2-1294 2-1194
1-1227 1-1438 2-1438 2-1227

Explanation:

Here, 1-1077 means driver with driverID 1077 has started the trip(1 stands for starting the trip)

Then 1-1123 means driver has picked customer with custID 1123(again 1 as trip has started for 1123) then 2-1123 represents customer 1123's trip has ended(2 for ending the trip) and then 2-1077 means that the trip for driver has ended.

OSRM installation(Not required for now but for future reference)
https://paris-fire-brigade.github.io/data-challenge/post/2019/06/13/06-set-up-an-osrm-server-on-ubuntu.html
This is a fully working tutorial for installing and configuring osrm module.
Note: While installing there comes a step which asks to download Berlin map,if you want India map then download it from osrm website(*http://download.geofabrik.de/asia/india-latest.osm.pbf*)

Constrained Vehicle Routing With Time Windows(CVRPTW)

In the capacitated vehicle routing problem **with time-windows** (CVRPTW), a fleet of delivery vehicles with uniform capacity must service customers with known demand and opening hours for a single commodity. The vehicles start and end their routes at a common depot. Each customer can only be served by one vehicle.

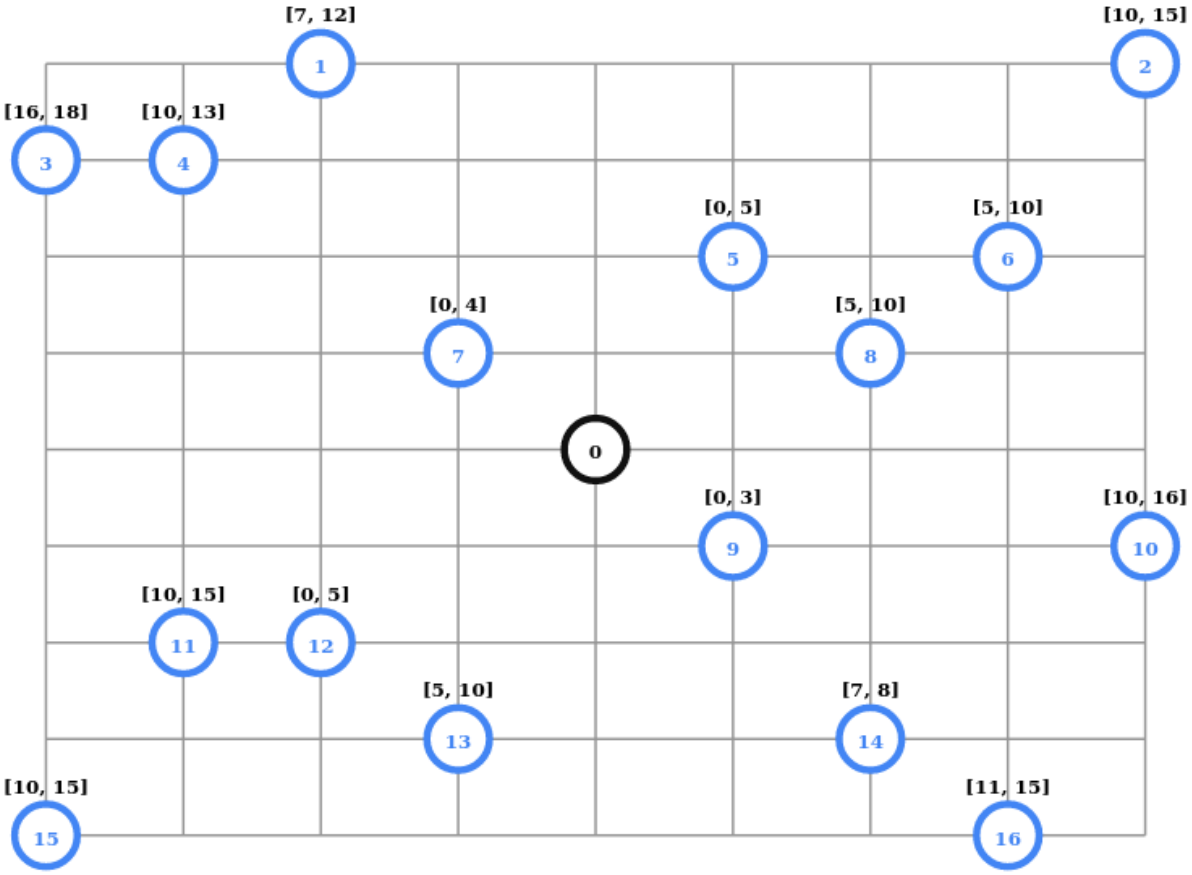


Image source: https://developers.google.com/optimization/routing/vrptw

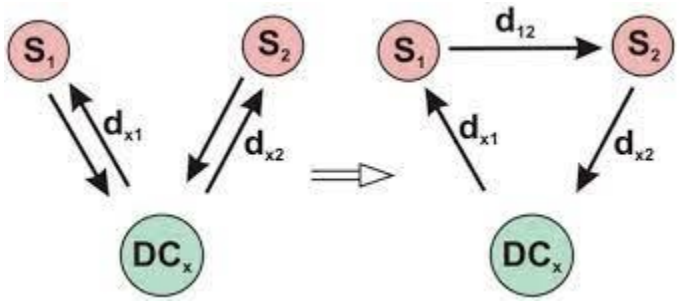
In the above figure node 0 is the depot location and all other nodes are the customer nodes waiting for service with some specific time windows.
For example node 1 will be available for service for time window [7,12], node 2 for [10,15] and so on.

In addition to this in our system every vehicle will have some maximum capacity and every customer node will have some specific demand.

Our task is to find out the optimal route such that time window constraints are satisfied, capacity constraints are satisfied and the total cost which is total distance travelled by a vehicle should be minimised.

Clarke and Wright Savings Algorithm

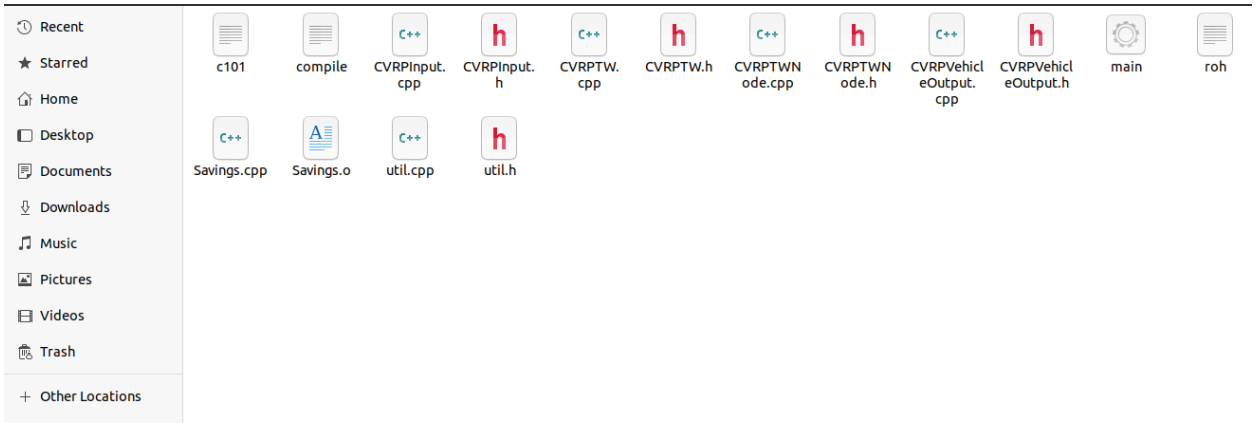
Savings algorithm is a kind of greedy algorithm for solving vehicle routing problem.



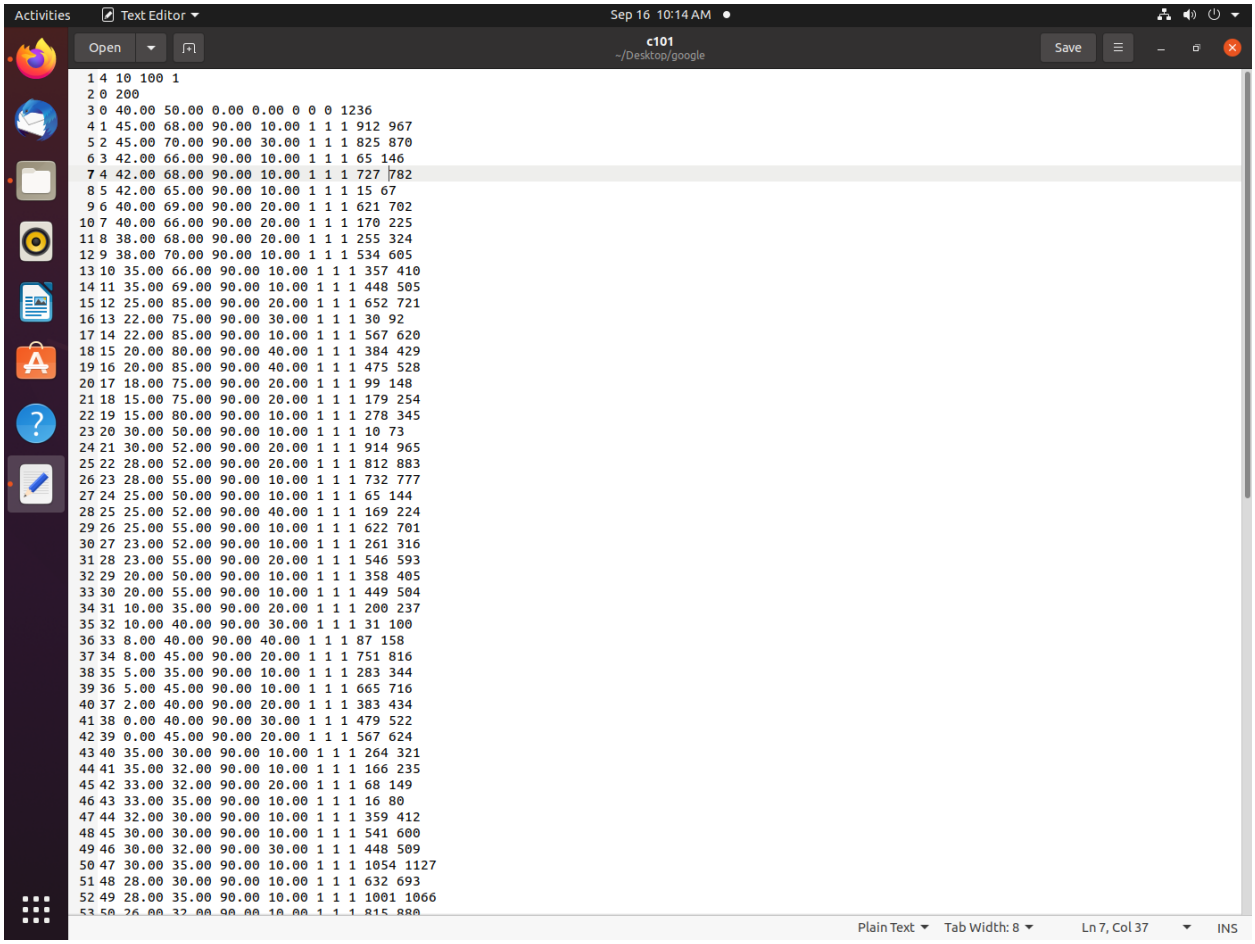
For understanding the Savings algorithm watch this video by nptm from 40:00 - 59:00

Our System:

We have a basic implementation of Savings algorithm for CVRPTW. We've used Euclidean distance for calculating distance between nodes but in real world scenario we should get distance from some map API's. Let's understand that using the following screenshots.



Savings.cpp is the file which contains the algorithm implementation.
c101 is the input file. Let's understand the input format



Now this is the standard cvrp instance available on internet as Solomans cvrptw instances. There are some parameters in this which we are not using so we'll just see the ones which are useful to us.

Line 1: The second parameter is the total number of vehicles in the depot(10), third parameter is the total number of customer nodes(100)

Line 2: The second parameter here is the *maximum capacity* of each vehicle.

Line 3:

- Id - identifier for depot
- X-coordinate of depot
- Y-coordinate of depot
- Service time(0 for depot)
- Demand(0 for depot)
- Next three parameters are not of use
- Last parameter is the closing time that is the time before which the driver needs to return back to the depot.

Line 4 onwards:

- X-coordinate of customer
- Y-coordinate of customer

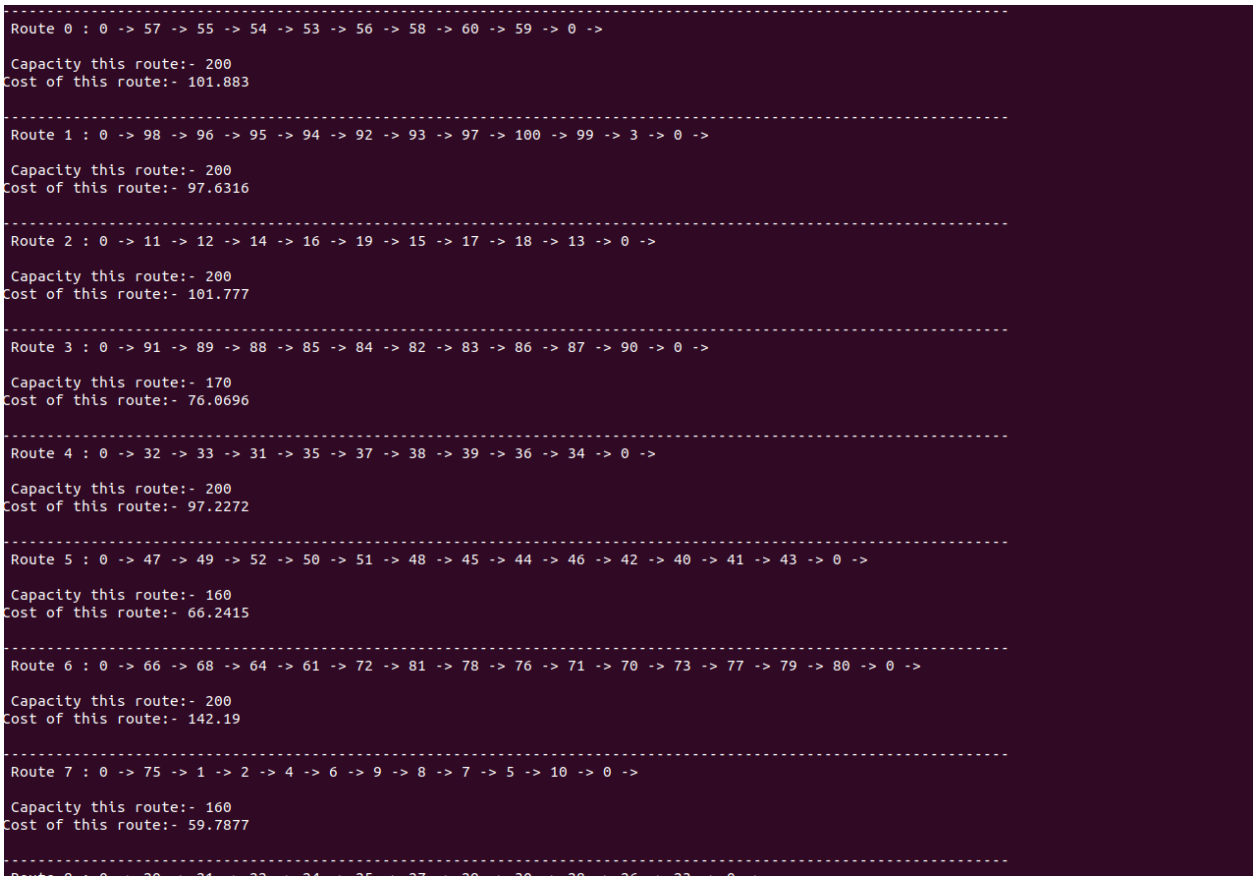
- Service time
- Demand - seats/capacity required by this customer
- Next three parameters are not of use
- Opening time
- Closing time

How to compile and run?

g++ CVRPTWNode.cpp CVRPTW.cpp CVRPInput.cpp CVRPVehicleOutput.cpp util.cpp Savings.cpp -o main

For running the program give input file as an argument
./main c101

Output



In the output routes are generated for each driver.