

# 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

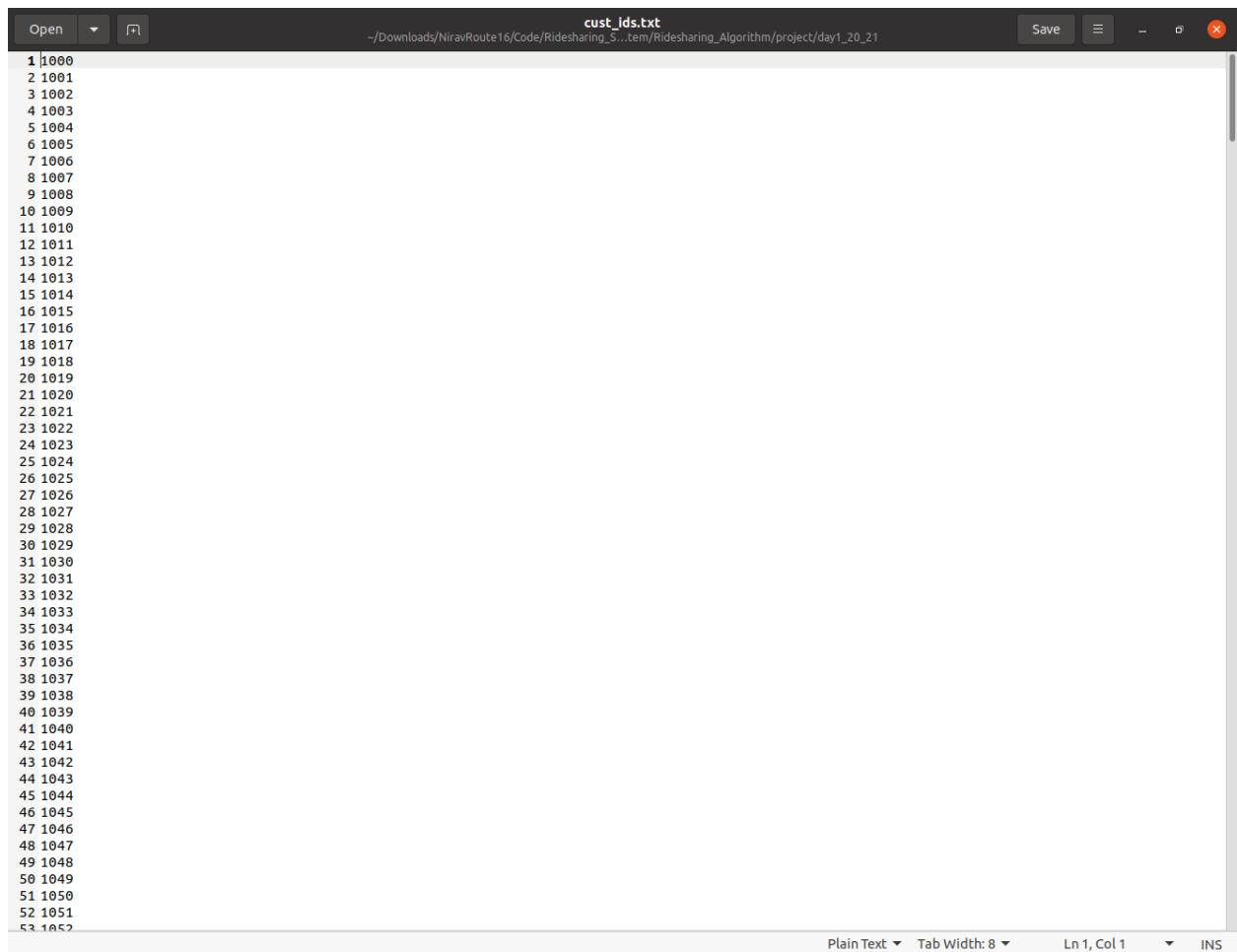
### **Screenshot:**



The screenshot shows a text editor window with the title bar 'all\_ids.txt'. The file path is '~/.Downloads/NiravRoute16/Code/Ridesharing\_System/Ridesharing\_Algorithm/project/day1\_20\_21'. The editor contains a list of five driver IDs, each preceded by a line number from 1 to 5.

```
1 1077
2 1094
3 1183
4 1194
5 1227
```

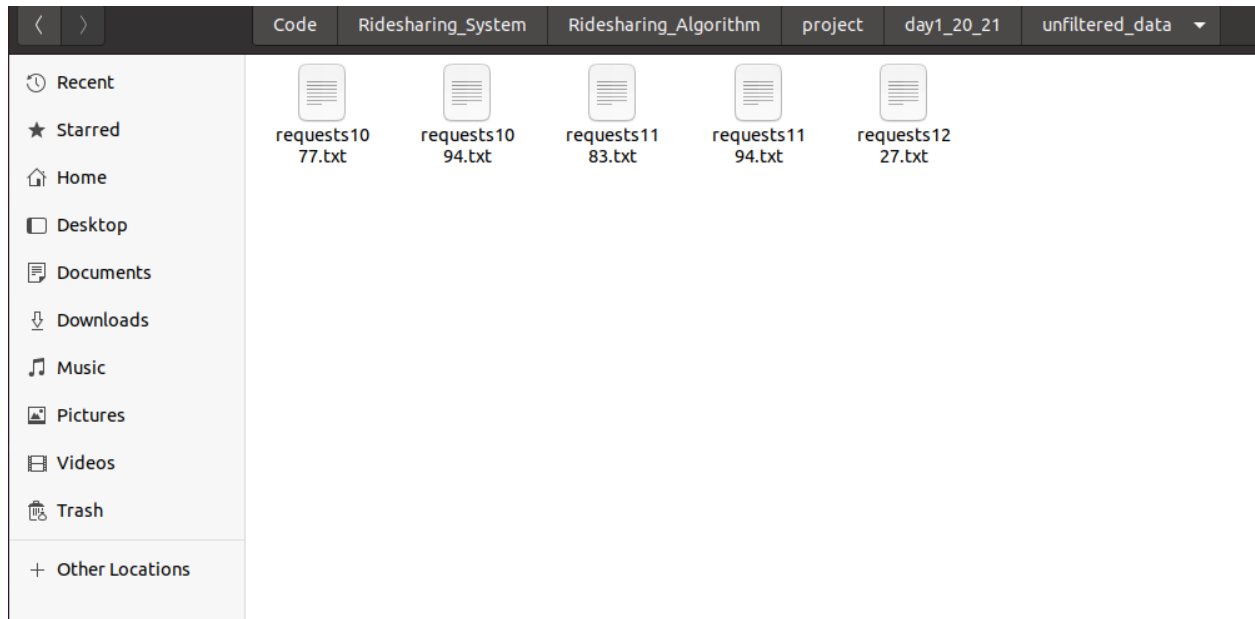
*all\_ids.txt(driverID's of all the drivers)*



The screenshot shows a text editor window with the title bar 'cust\_ids.txt'. The file path is '~/.Downloads/NiravRoute16/Code/Ridesharing\_System/Ridesharing\_Algorithm/project/day1\_20\_21'. The editor contains a list of 53 customer IDs, each preceded by a line number from 1 to 53. The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 1, Col 1', and 'INS'.

```
1 1000
2 1001
3 1002
4 1003
5 1004
6 1005
7 1006
8 1007
9 1008
10 1009
11 1010
12 1011
13 1012
14 1013
15 1014
16 1015
17 1016
18 1017
19 1018
20 1019
21 1020
22 1021
23 1022
24 1023
25 1024
26 1025
27 1026
28 1027
29 1028
30 1029
31 1030
32 1031
33 1032
34 1033
35 1034
36 1035
37 1036
38 1037
39 1038
40 1039
41 1040
42 1041
43 1042
44 1043
45 1044
46 1045
47 1046
48 1047
49 1048
50 1049
51 1050
52 1051
53 1052
```

*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             |
| -Orienteeing                   | //Algorithms implementation         |
| -Orienteeing_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:

```
rohit@ubuntu: ~/Downloads/NiravRoute16/Code/Ridesharing_System/AllocationModule/src
5
Drivers data size: 5
Getting components module started-----
Check 1
t:0
t:1
t:2
t:3
t:4
Check 2
0 - 0
1 - 1
2 - 2
3 - 3
4 - 4
File Wise Driver separation
0:0
1:1
2:2
3:3
4:4
Total Input Files to be generated = 5
-----INPUT CREATION FINISHED-----

---Printing tour---
ID: 1123   Index: 6   vt: 1   w:64.00 ot: 14:15:41 ct: 15:15:41 sst: 03:02:22 at:03:02:22 maxshift:3067999 wait:0   shift:135301

---Printing tour---
ID: 1027   Index: 6   vt: 1   w:77.00 ot: 13:59:01 ct: 14:59:01 sst: 07:22:22 at:07:22:22 maxshift:977799 wait:0   shift:50901

---Printing tour---
ID: 1354   Index: 4   vt: 1   w:97.00 ot: 19:39:01 ct: 18:39:01 sst: 17:55:42 at:17:55:42 maxshift:14262199 wait:0   shift:2042901

---Printing tour---
ID: 1294   Index: 4   vt: 1   w:96.00 ot: 07:39:01 ct: 08:39:01 sst: 17:19:02 at:17:19:02 maxshift:3770399 wait:0   shift:449901

---Printing tour---
ID: 1438   Index: 6   vt: 1   w:98.00 ot: 18:52:21 ct: 19:52:21 sst: 00:44:02 at:00:44:02 maxshift:241699 wait:0   shift:6501
5
-----FULL TOUR-----
Total Non unique:5
Total unique customers:5
Number of drivers who will share:5
Number of repeated customers: 0
Total Cost: 1419
Total Cost after Modules: 329
rohit@ubuntu:~/Downloads/NiravRoute16/Code/Ridesharing_System/AllocationModule/src$
```

**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.

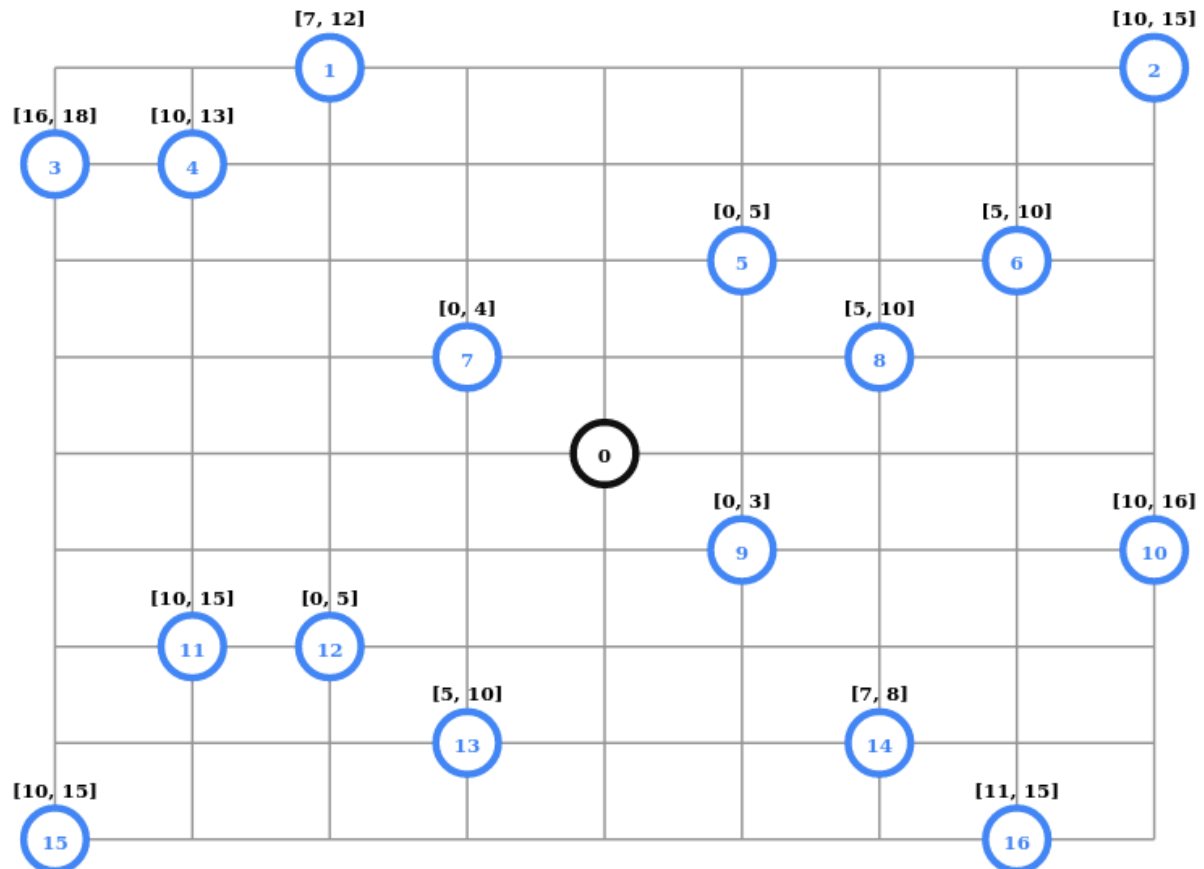


Image source: <https://developers.google.com/optimization/routing/vrptw>

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.

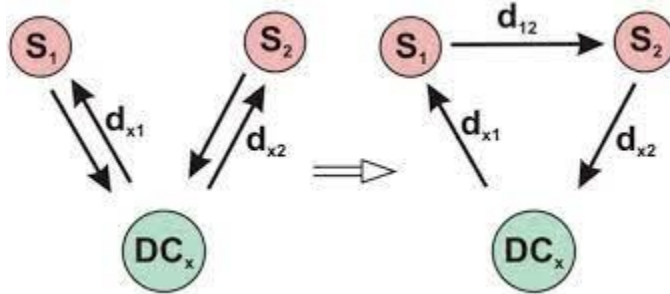
For example node 1 will be available for service for time window [7,12], node 2 for [10,15] and so on.

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

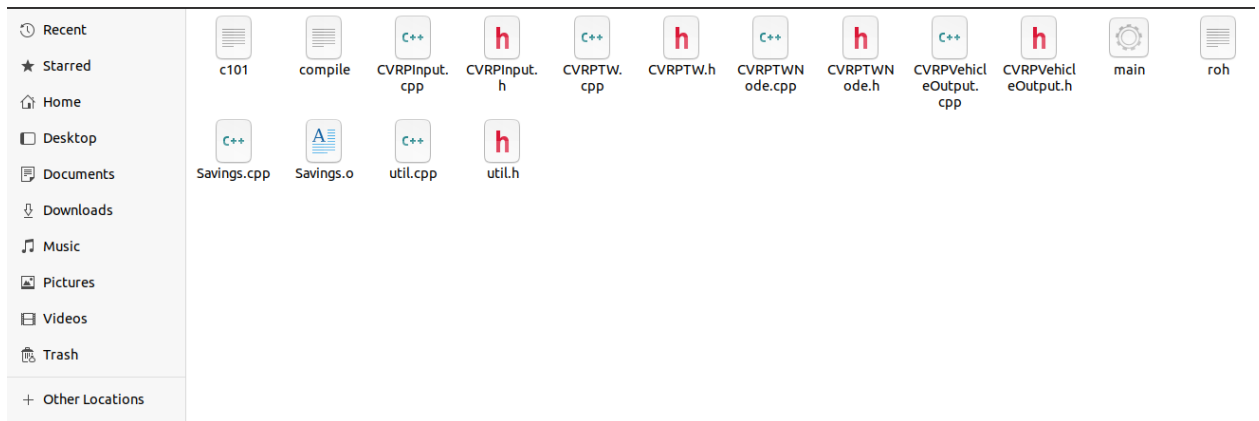
Savings algorithm is a kind of greedy algorithm for solving vehicle routing problem.



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

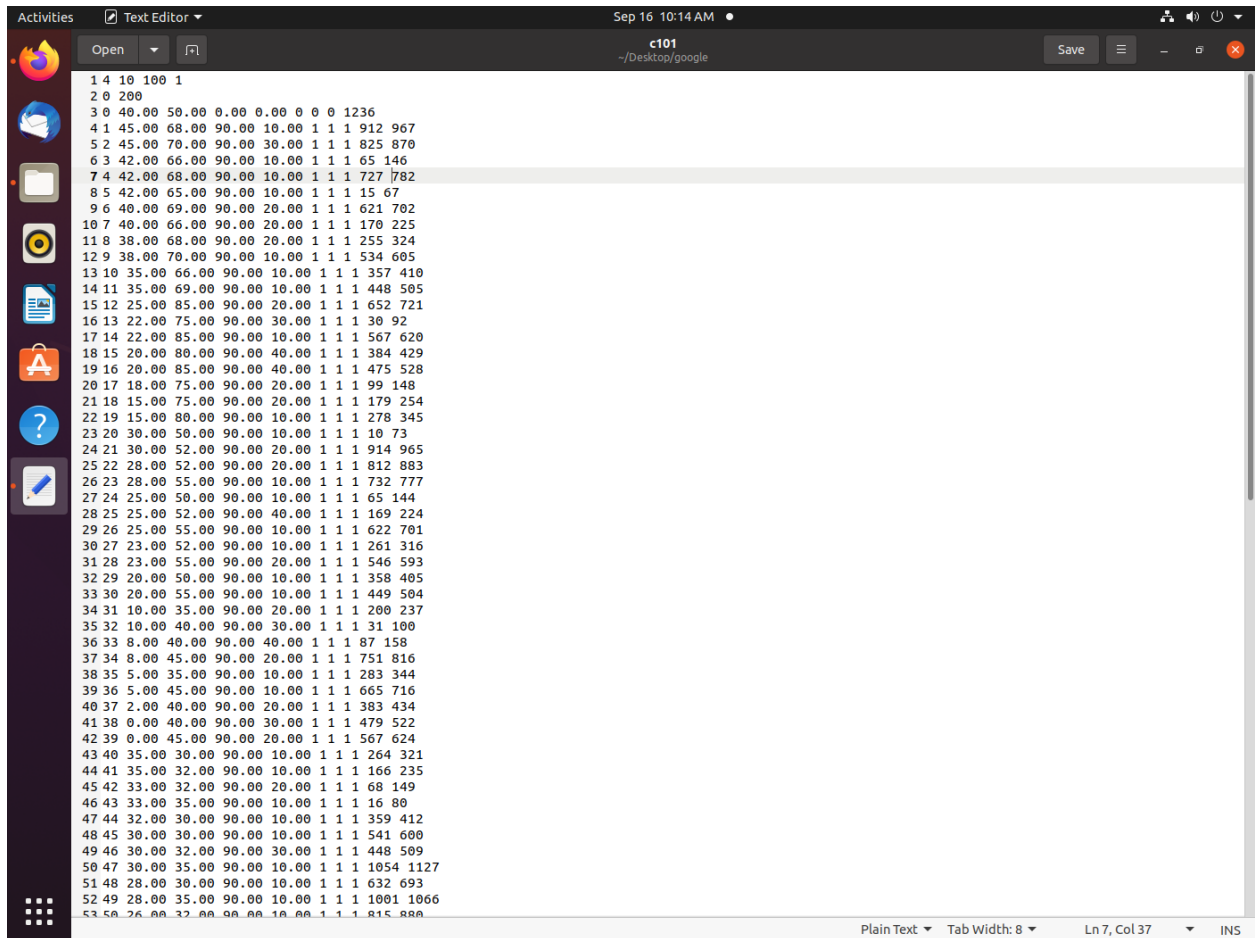
### Our System:

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

A screenshot of a Linux desktop environment. The top bar shows the date and time as 'Sep 16 10:14 AM'. The main window is a text editor titled 'c101' with the file path '~/Desktop/google'. The editor contains a CVRP instance data file with the following structure:

```
1 4 10 100 1
2 0 200
3 0 40.00 50.00 0.00 0.00 0 0 0 1236
4 1 45.00 68.00 90.00 10.00 1 1 1 912 967
5 2 45.00 70.00 90.00 30.00 1 1 1 825 870
6 3 42.00 66.00 90.00 10.00 1 1 1 65 146
7 4 42.00 68.00 90.00 10.00 1 1 1 727 782
8 5 42.00 65.00 90.00 10.00 1 1 1 15 67
9 6 40.00 69.00 90.00 20.00 1 1 1 621 702
10 7 40.00 66.00 90.00 20.00 1 1 1 170 225
11 8 38.00 68.00 90.00 20.00 1 1 1 255 324
12 9 38.00 70.00 90.00 10.00 1 1 1 534 605
13 10 35.00 66.00 90.00 10.00 1 1 1 357 410
14 11 35.00 69.00 90.00 10.00 1 1 1 448 505
15 12 25.00 85.00 90.00 20.00 1 1 1 652 721
16 13 22.00 75.00 90.00 30.00 1 1 1 30 92
17 14 22.00 85.00 90.00 10.00 1 1 1 567 620
18 15 20.00 80.00 90.00 40.00 1 1 1 384 429
19 16 20.00 85.00 90.00 40.00 1 1 1 475 528
20 17 18.00 75.00 90.00 20.00 1 1 1 99 148
21 18 15.00 75.00 90.00 20.00 1 1 1 179 254
22 19 15.00 80.00 90.00 10.00 1 1 1 278 345
23 20 30.00 50.00 90.00 10.00 1 1 1 10 73
24 21 30.00 52.00 90.00 20.00 1 1 1 914 965
25 22 28.00 52.00 90.00 20.00 1 1 1 812 883
26 23 28.00 55.00 90.00 10.00 1 1 1 732 777
27 24 25.00 50.00 90.00 10.00 1 1 1 65 144
28 25 25.00 52.00 90.00 40.00 1 1 1 169 224
29 26 25.00 55.00 90.00 10.00 1 1 1 622 701
30 27 23.00 52.00 90.00 10.00 1 1 1 261 316
31 28 23.00 55.00 90.00 20.00 1 1 1 546 593
32 29 20.00 50.00 90.00 10.00 1 1 1 358 405
33 30 20.00 55.00 90.00 10.00 1 1 1 449 504
34 31 10.00 35.00 90.00 20.00 1 1 1 200 237
35 32 10.00 40.00 90.00 30.00 1 1 1 31 100
36 33 8.00 40.00 90.00 40.00 1 1 1 87 158
37 34 8.00 45.00 90.00 20.00 1 1 1 751 816
38 35 5.00 35.00 90.00 10.00 1 1 1 283 344
39 36 5.00 45.00 90.00 10.00 1 1 1 665 716
40 37 2.00 40.00 90.00 20.00 1 1 1 383 434
41 38 0.00 40.00 90.00 30.00 1 1 1 479 522
42 39 0.00 45.00 90.00 20.00 1 1 1 567 624
43 40 35.00 30.00 90.00 10.00 1 1 1 264 321
44 41 35.00 32.00 90.00 10.00 1 1 1 166 235
45 42 33.00 32.00 90.00 20.00 1 1 1 68 149
46 43 33.00 35.00 90.00 10.00 1 1 1 16 80
47 44 32.00 30.00 90.00 10.00 1 1 1 359 412
48 45 30.00 30.00 90.00 10.00 1 1 1 541 600
49 46 30.00 32.00 90.00 30.00 1 1 1 448 509
50 47 30.00 35.00 90.00 10.00 1 1 1 1054 1127
51 48 28.00 30.00 90.00 10.00 1 1 1 632 693
52 49 28.00 35.00 90.00 10.00 1 1 1 1001 1066
53 50 26.00 32.00 90.00 10.00 1 1 1 815 888
```

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

```
-----
Route 0 : 0 -> 57 -> 55 -> 54 -> 53 -> 56 -> 58 -> 60 -> 59 -> 0 ->
Capacity this route:- 200
Cost of this route:- 101.883
-----
Route 1 : 0 -> 98 -> 96 -> 95 -> 94 -> 92 -> 93 -> 97 -> 100 -> 99 -> 3 -> 0 ->
Capacity this route:- 200
Cost of this route:- 97.6316
-----
Route 2 : 0 -> 11 -> 12 -> 14 -> 16 -> 19 -> 15 -> 17 -> 18 -> 13 -> 0 ->
Capacity this route:- 200
Cost of this route:- 101.777
-----
Route 3 : 0 -> 91 -> 89 -> 88 -> 85 -> 84 -> 82 -> 83 -> 86 -> 87 -> 90 -> 0 ->
Capacity this route:- 170
Cost of this route:- 76.0696
-----
Route 4 : 0 -> 32 -> 33 -> 31 -> 35 -> 37 -> 38 -> 39 -> 36 -> 34 -> 0 ->
Capacity this route:- 200
Cost of this route:- 97.2272
-----
Route 5 : 0 -> 47 -> 49 -> 52 -> 50 -> 51 -> 48 -> 45 -> 44 -> 46 -> 42 -> 40 -> 41 -> 43 -> 0 ->
Capacity this route:- 160
Cost of this route:- 66.2415
-----
Route 6 : 0 -> 66 -> 68 -> 64 -> 61 -> 72 -> 81 -> 78 -> 76 -> 71 -> 70 -> 73 -> 77 -> 79 -> 80 -> 0 ->
Capacity this route:- 200
Cost of this route:- 142.19
-----
Route 7 : 0 -> 75 -> 1 -> 2 -> 4 -> 6 -> 9 -> 8 -> 7 -> 5 -> 10 -> 0 ->
Capacity this route:- 160
Cost of this route:- 59.7877
-----
Route 8 : 0 -> 20 -> 21 -> 22 -> 24 -> 25 -> 27 -> 28 -> 29 -> 26 -> 23 -> 0 ->
```

In the output routes are generated for each driver.

