

Route

Optimization

Solution

Ashirvad Gupta

Certificate Program in Big Data Analytics and Optimization

April 22nd 2017

Table of Contents

[ABSTRACT 3](#_Toc480583121)

[INTRODUCTION 4](#_Toc480583122)

[Current Process of Route Optimization 4](#_Toc480583123)

[Cost of Route Optimization 5](#_Toc480583124)

[PURPOSE 7](#_Toc480583125)

[MEASURE OF SUCCESS 7](#_Toc480583126)

[ASSUMPTIONS 7](#_Toc480583127)

[STATE OF ART 8](#_Toc480583128)

[DATA STRUCTURE AND SUMMARY 9](#_Toc480583129)

[Attribute Description 9](#_Toc480583130)

[Structure of data 9](#_Toc480583131)

[Summary of Data 10](#_Toc480583132)

[DATA PREPROCESSING 11](#_Toc480583133)

[Methodology 11](#_Toc480583134)

[Implementation 11](#_Toc480583135)

[APPROACH 18](#_Toc480583136)

[MODEL BUILDING 20](#_Toc480583137)

[Depot’s Nearest Neighbors Model 20](#_Toc480583138)

[Genetic Algorithm 21](#_Toc480583139)

[Defining Objective Function 22](#_Toc480583140)

[Initial Population Generation 23](#_Toc480583141)

[Mutation 24](#_Toc480583142)

[Crossover 25](#_Toc480583143)

[Selection 25](#_Toc480583144)

[Stopping Criteria and parameter selection 25](#_Toc480583145)

[Genetic Algorithm Implementation 26](#_Toc480583146)

[RESULTS 27](#_Toc480583147)

[Results for all parameters 28](#_Toc480583148)

[Depot’s Nearest Neighbors Model 28](#_Toc480583149)

[Genetic Algorithm using Random Initial Population 31](#_Toc480583150)

[Genetic Algorithm using DNN Generated Initial Population 34](#_Toc480583151)

[ANALYSIS 38](#_Toc480583152)

[Model Comparison 38](#_Toc480583153)

[Genetic Algorithm with Depot’s Nearest Neighbors Model 44](#_Toc480583154)

[CONCLUSION 50](#_Toc480583155)

[APPENDIX 51](#_Toc480583156)

[Iterations vs Fitness Value Plot 51](#_Toc480583157)

[GA Model using DNN Generated Initial Population 51](#_Toc480583158)

[GA Model using Random Initial Population 52](#_Toc480583159)

[Best Solution for Stops Range: 4 to 6 and Time window range: -60 to +60 minutes 52](#_Toc480583160)

[R Code 53](#_Toc480583161)

# ABSTRACT

Route Optimization is a practical and challenging problem in logistics and supply chain management.

We can extend it to a study of service vehicles used for providing cleaning services in multiple cities.

Route Optimization is a technique to develop an optimal routing solution which enables timely and qualitative service for the customers.

This is a software which enables the company vehicles to operate efficiently on optimal routes by maximizing the utilization of resources with minimum distance travelled and time taken, while satisfying customer demand.

The proposed solution framework in this report will provide a near-optimal solution which can help Spring Cleaning Services achieve seamless transportation, reduce cost expenses related to travel distances and labor, optimize vehicles and resource utilization, and improve end-customer services.

# INTRODUCTION

Spring-Cleaning offers a wide variety of cleaning services. They provide carpet cleaning, tile and grout cleaning, upholstery cleaning, hardwood floor cleaning and air duct cleaning, etc.

They have presence in 48 cities nationwide with various depots servicing specific regions.

Spring-Cleaning currently has an in-house software that collects and maintains the services and schedules and decides vehicle routes. We need to work with Spring-Cleaning to develop an optimal routing solution for their vehicles. This will be accomplished by designing a solution using advanced optimization techniques. Spring-Cleaning strives to service requests by dynamically scheduling their resources under various constraints.

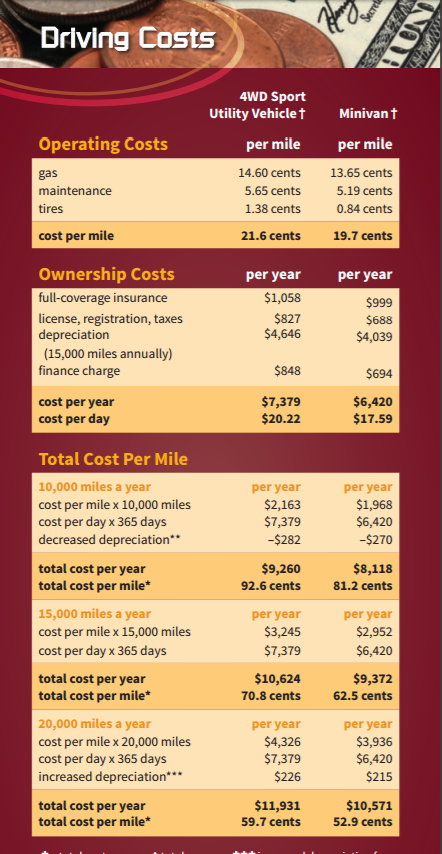
## Current Process of Route Optimization

Spring-Cleaning currently has an in-house software that collects and maintains the services and schedules and decides vehicle routes. Spring-Cleaning strives to service requests by dynamically scheduling their resources under various constraints.

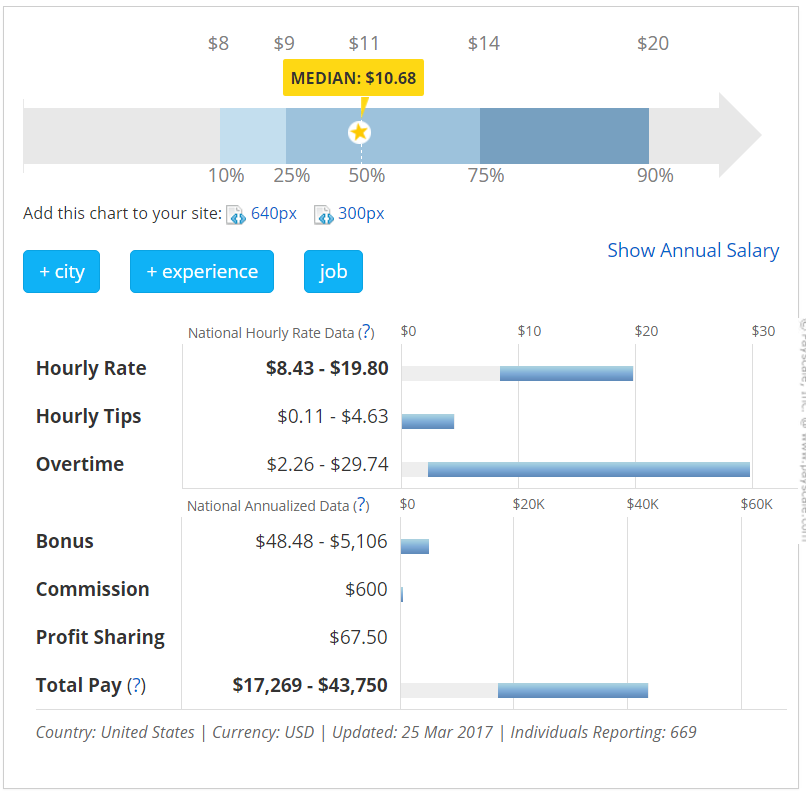
1. Each vehicle must leave from and return to the same depot.
2. Each customer is served exactly once by one vehicle.
3. The customer demand along the route does not exceed the vehicle capacity.
4. All the customers with known demands are assigned to vehicles.
5. The total time of a route does not exceed the maximum vehicle route time. The maximum vehicle route times are equivalent to the time window of the depot.
6. The time window constraints of each customer delivery must be satisfied.
7. The difference between the arrival times of delivery and installation vehicles at the same customer must not exceed the service level (maximum allowable time interval).
8. Stops for every vehicle in a depot are assigned customers(stops) for the day.
9. Constraints like length of the time to complete the job, distance between stops and if the customers have specified a time when they would like to get the service done are considered.

## Cost of Route Optimization

* Cost should be a factor of travel time, penalty time, service time, and number of vehicles involved.
* **Vehicles Cost**: Every vehicle has fixed and variable cost.
  + Fixed Costs included Vehicle purchase or lease, Insurance, Registration and vehicle taxes
  + Variable Costs includes Maintenance and repair, Fuel, fuel taxes and oil, Paid parking and tolls
  + More the number of vehicles, more the fixed cost, variable cost and driver cost.
* As per American Automobile Association, the vehicles cost is:
  + **Operating Costs: 19.7 cents per mile**
  + **Ownership Costs: 17.59 dollars per day**
  + **Driver Salary: 37.5 cents per mile**



* Assuming in an hour a van can travel around average **25 miles per hour**.
* Total vehicles cost excluding ownership cost per hour will come around **25$ per hour.**
* Ownership cost will be fixed cost per day which will come around **17.59$ per day.**
* **Service Cost:** Assuming average salary of a cleaner is **14$ per hour.**
* **Penalty Cost:** Penalty Time high means either customer is getting services late or employees are working late. High Penalty will have its own cost. Penalty cost can be extra cost incurred for employees who are working late and late service cost. Assuming 50% weightage of service cost can be given for Penalty cost. As you can see overtime average is **14$ per hour**, so we can assume 14$ per hour overtime allowance plus **3$ per hour** as late service cost.



# PURPOSE

To determine optimal set of fleet routes for satisfying delivery demands in several depots at different locations.

It is considered a practical problem in the fields of transportation, distribution, and logistics.

# MEASURE OF SUCCESS

* Comprehensive visibility to transportation in motion that tracks all shipments in transit.
* Proactive rather than reactive problem solving.
* Routing and SLA compliance.

# ASSUMPTIONS

* Vehicles Cost, Driver Cost, Service Cost and Penalty Cost is assumed.
* Assuming only one cleaner and driver per vehicle.
* If the travel time is given from a depot/stop to other depot/stop and return travel time is not given, then same travel time is assumed for return.

# STATE OF ART

ORTEC and DPDgroup won the Technology Award at the prestigious annual World Mail Awards. The project launched by DPDgroup and ORTEC relies on the power of predictability, interconnecting mobility, cloud-enabled scalability and route optimization. This results in better service levels, lower operational costs and a reduced carbon footprint, for example in the 1-Hour Predict delivery service.

Another solution by Fleetmatics’ Routist delivers routes and updates directly to the drivers in the field via mobile device. With Fleetmatics REVEAL, vehicle tracking solution behind Routist, drivers’ progress can be monitored in real-time via Live Map. Drivers get routes and updates in real-time via the Fleetmatics Field App.

# DATA STRUCTURE AND SUMMARY

## 

## Attribute Description

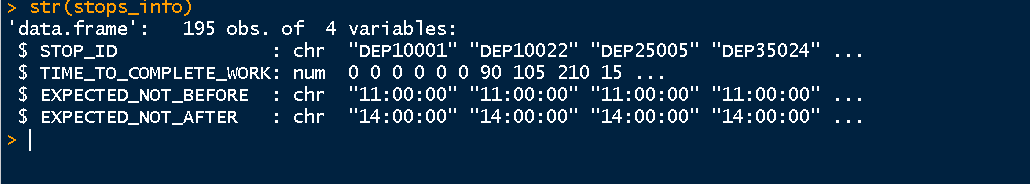
The dataset available is 3 different tables containing 12 different attributes. The tables are stored in MySQL database with names stops\_info\_db, travel\_time\_matrix\_db and parameters\_info\_db.

The description of data:

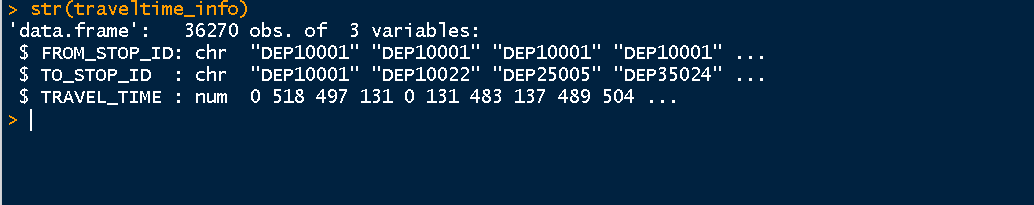
1. Stops Information:
2. STOP\_ID: 8-character unique stop identifier.
3. TIME\_TO\_COMPLETE\_WORK: Total time to complete the task/work at each stop/customer location.
4. EXPECTED\_NOT\_BEFORE: Customer not expecting before this time.
5. EXPECTED\_NOT\_AFTER: customer not expecting after this time.
6. Travel Time Information:
7. FROM\_STOP\_ID: Source Stop/Depot.
8. TO\_STOP\_ID: Target Stop/Depot.
9. TRAVEL\_TIME: Travel time between a stop/depot and stop/depot.
10. Parameters/Constraints Information:
11. MAX\_STOPS\_PER\_ROUTE: Maximum stops threshold per route. For example, 5 stops per route or 7 stops per route.
12. MIN\_STOPS\_PER\_ROUTE: Minimum stops threshold per route. For example, 3 stops per route or 5 stops per route.
13. MAX\_ROUTE\_TIME: Maximum time (travel and job execution time) threshold per route. (660 minutes or 11 hours).
14. MAX\_EARLY\_TIMEWINDOW: Allowable time window and large violations window, if reaches early.
15. MAX\_LATE\_TIMEWINDOW: Allowable time window and large violations window, if reaches late.

## 

## Structure of data



There are 195 records in stops\_info data frame. There are 6 depots and 189 stops.

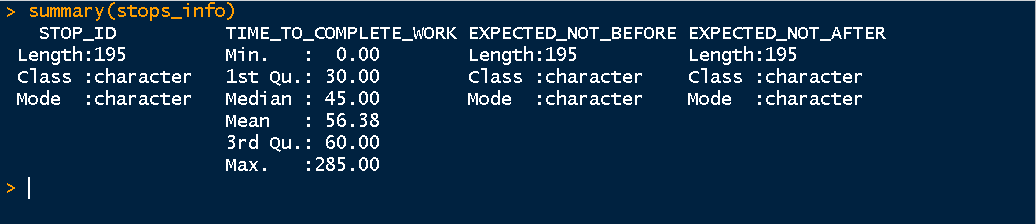


There are 3 variables in the traveltime\_info data frame. It contains information about travel time from one depot/stop to other depot/stop.

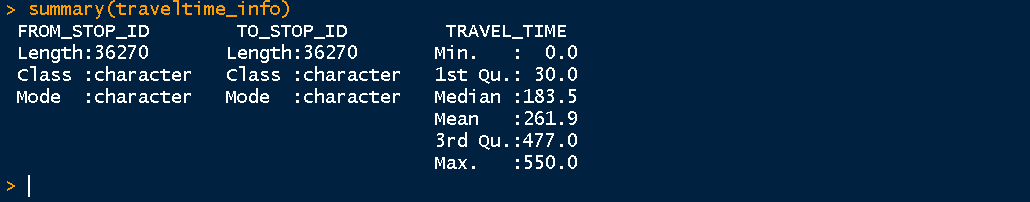
There are 6 set of parameters in the parameters\_info table. Each set of parameter has minimum and maximum number of stops to cover in each route, minimum and maximum allowable time to serve customers, maximum route time including travel and service execution time to cover in each route.

## 

## Summary of Data



As can be seen from the output of summary command, minimum time to complete the work is 0(for depots), maximum time to complete the work is 285 and mean time to complete the work is 56.38. There are no missing values in the above table.



There are 36270 records in travel\_time\_info data frame. There should be total 195\*195=38025 records in the travel\_time\_info. There are 1755 missing values in the travel\_time\_info which we will fill in data preprocessing step.

# DATA PREPROCESSING

## 

## Methodology

As seen in the summary of data, there are 1755 missing values in the dataset.

Steps:

1. Find out for which source and destination, travel time is missing.
2. Find out if time taken for travelling between one stop/depot to other depot/stop and

returning to same stop/depot is same or not.

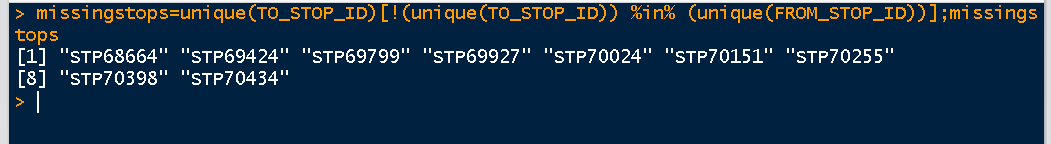
1. If it’s same, then populate the missing travel time for such stops/depot.
2. Travel time will be zero if source and destination depot/source is same.
3. Find out the nearest stop/depot for rest of the missing travel time values and impute the travel time.
4. Convert the data frame into 195 X 195 matrix.

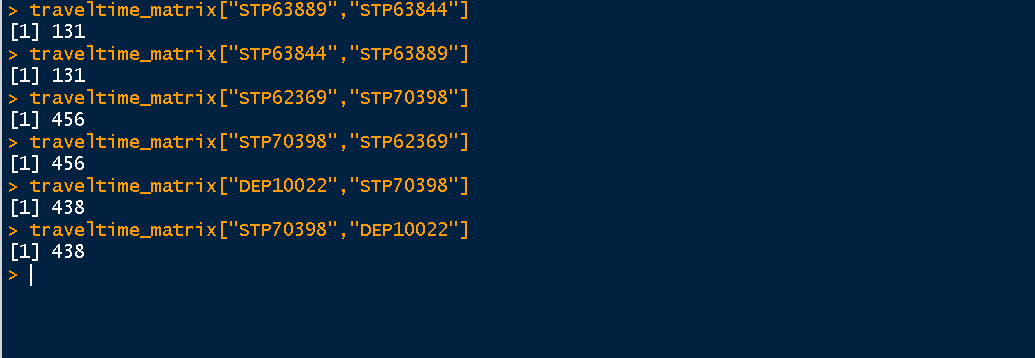
## Implementation

Checking the unique FROM\_STOP\_ID and TO\_STOP\_ID in traveltime\_info data frame.

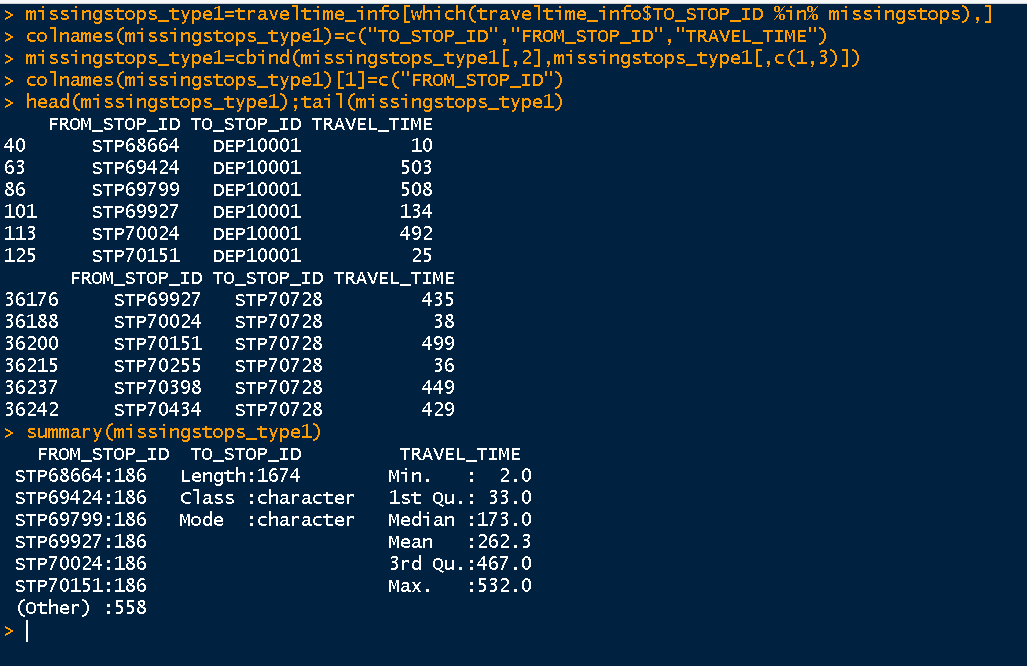


As unique FROM\_STOP\_ID are 186, compute which sources have missing values.



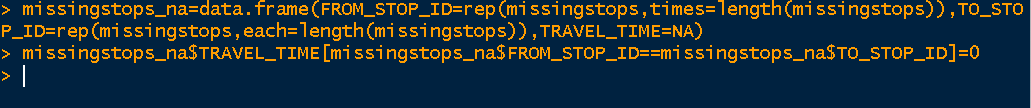
Checking if time taken for travelling between one stop/depot to other depot/stop and returning to same stop/depot is same or not. 

Populating the travel time by reverse lookup in traveltime\_info data frame.

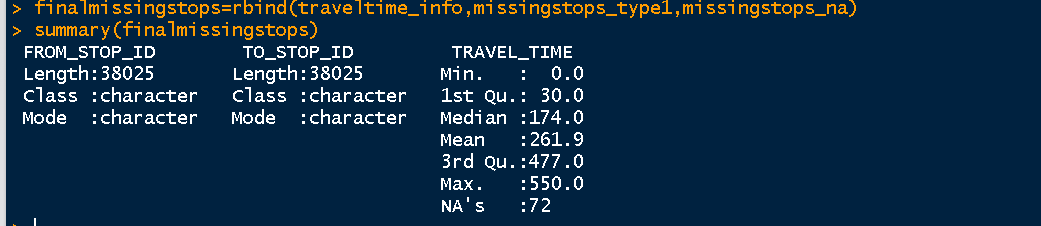


As seen in the output of summary command above, 1674 records are populated by reverse lookup.

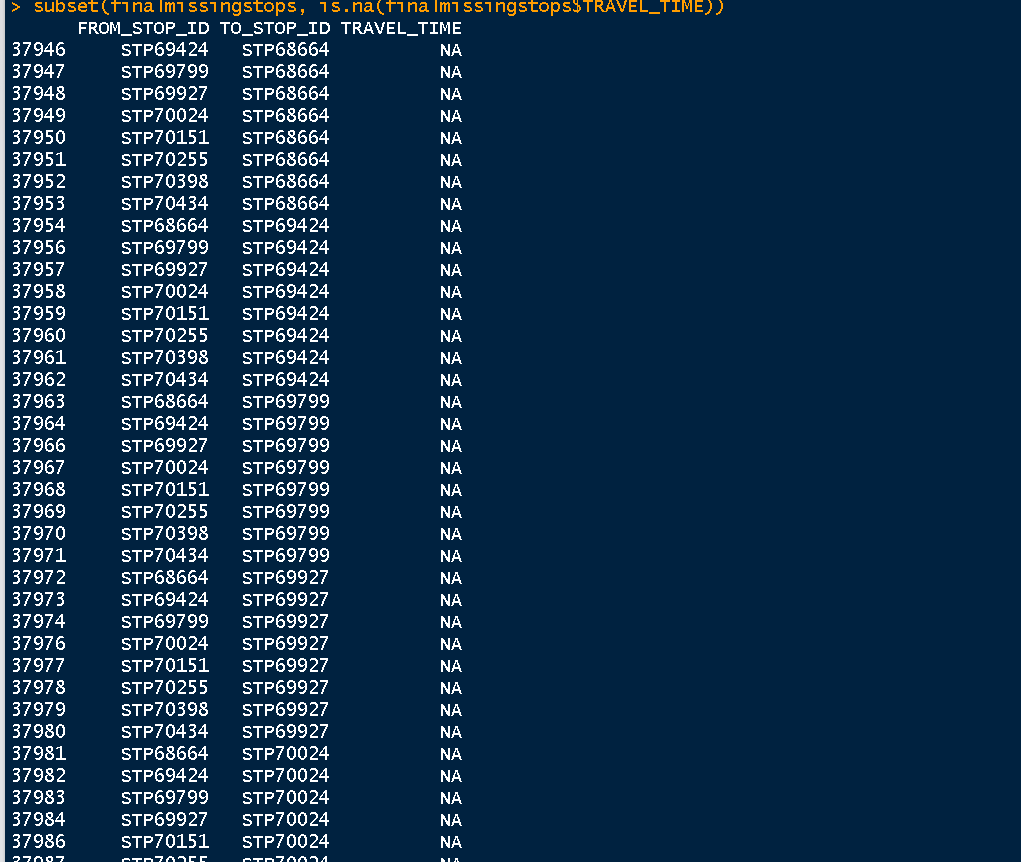
Adding NA for rest of the missing values and populating the traveltime\_info with zero travel time for depots/stops with themselves.



Adding the above computed values to the traveltime\_info data frame.

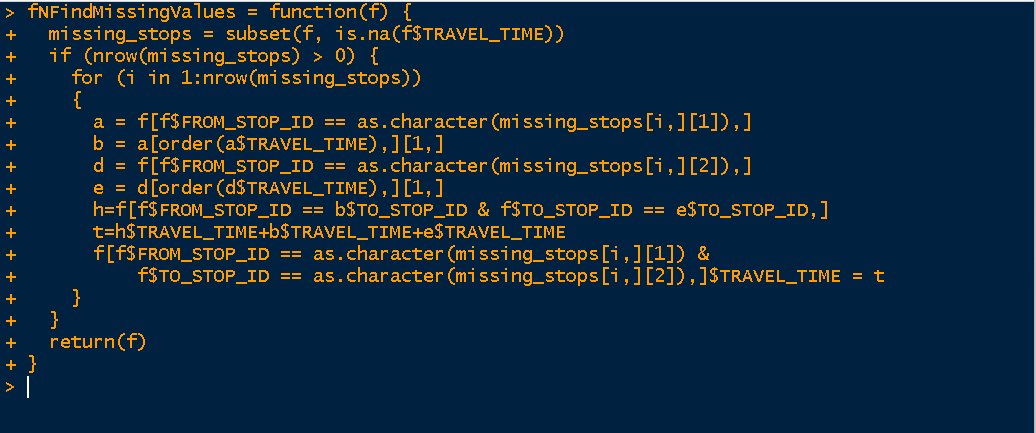


As seen in the output of summary command above, only 72 missing values left. Checking which records has missing travel time.

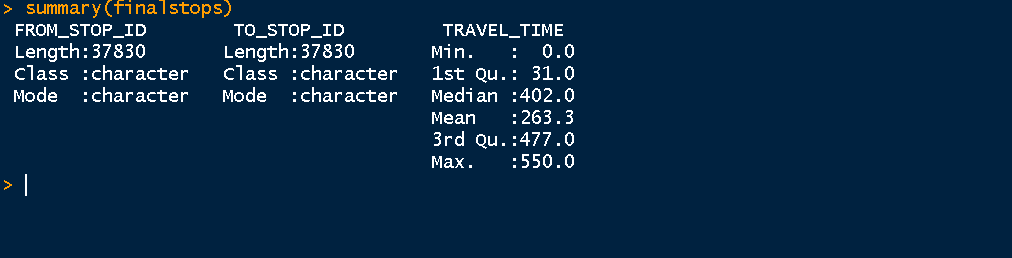


As nearest distance from the missing source and destination needs to be calculated, removing all the rows in which source and destination are same.

Writing the function for computing the rest of missing travel time values.

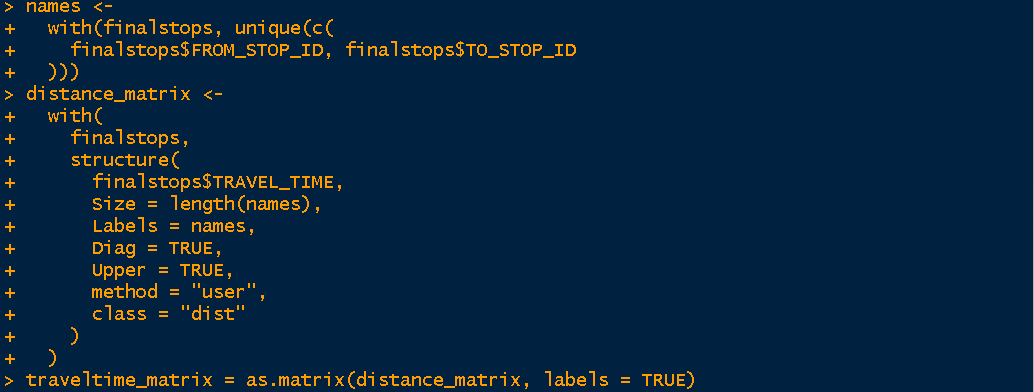


Above function will calculate travel time from missing source and destination stop/depot to their nearest stops/depots. Also, Travel time between nearest depots/stops is calculated. Then, sum of all the travel time will give travel time between missing stops.

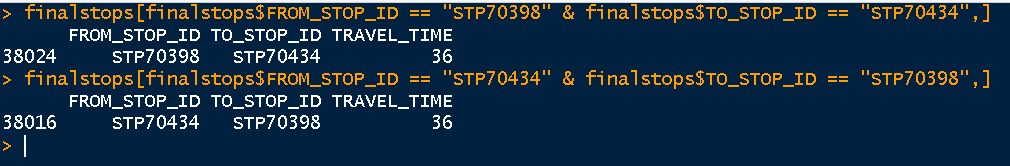


As seen in the above summary command output, there are no records with missing values.

Total number of records are 38025-195=37830.

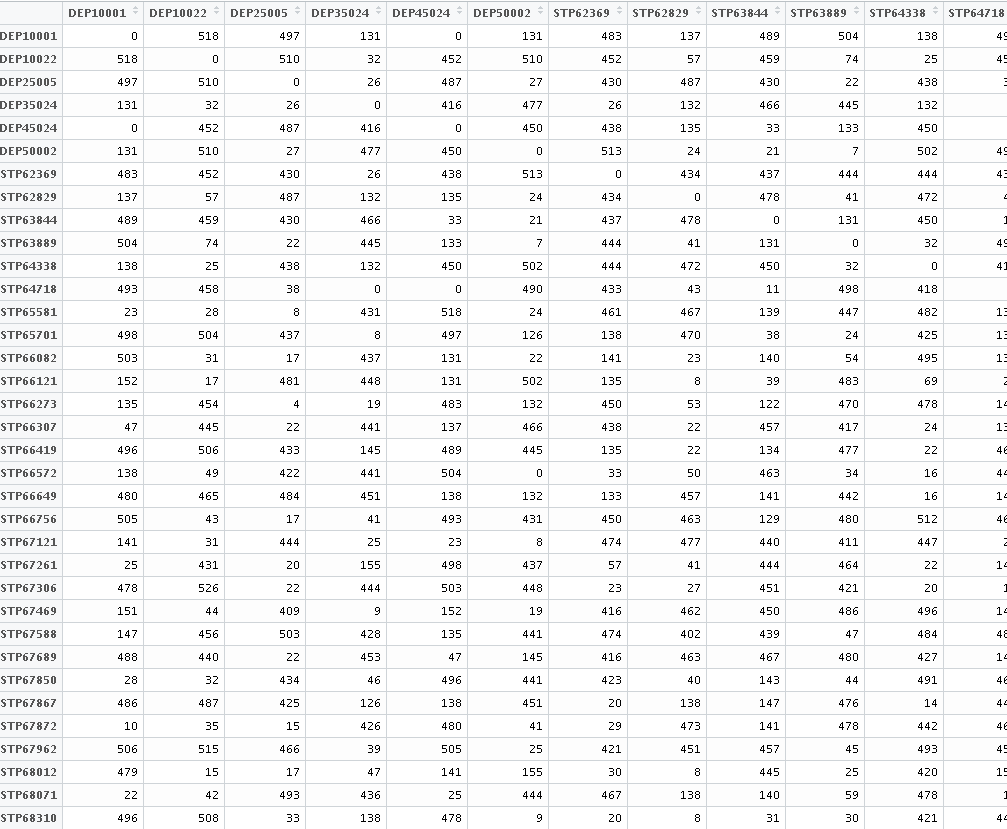


Checking for a record in which travel time is computed by the above function fNFindMissingValues.



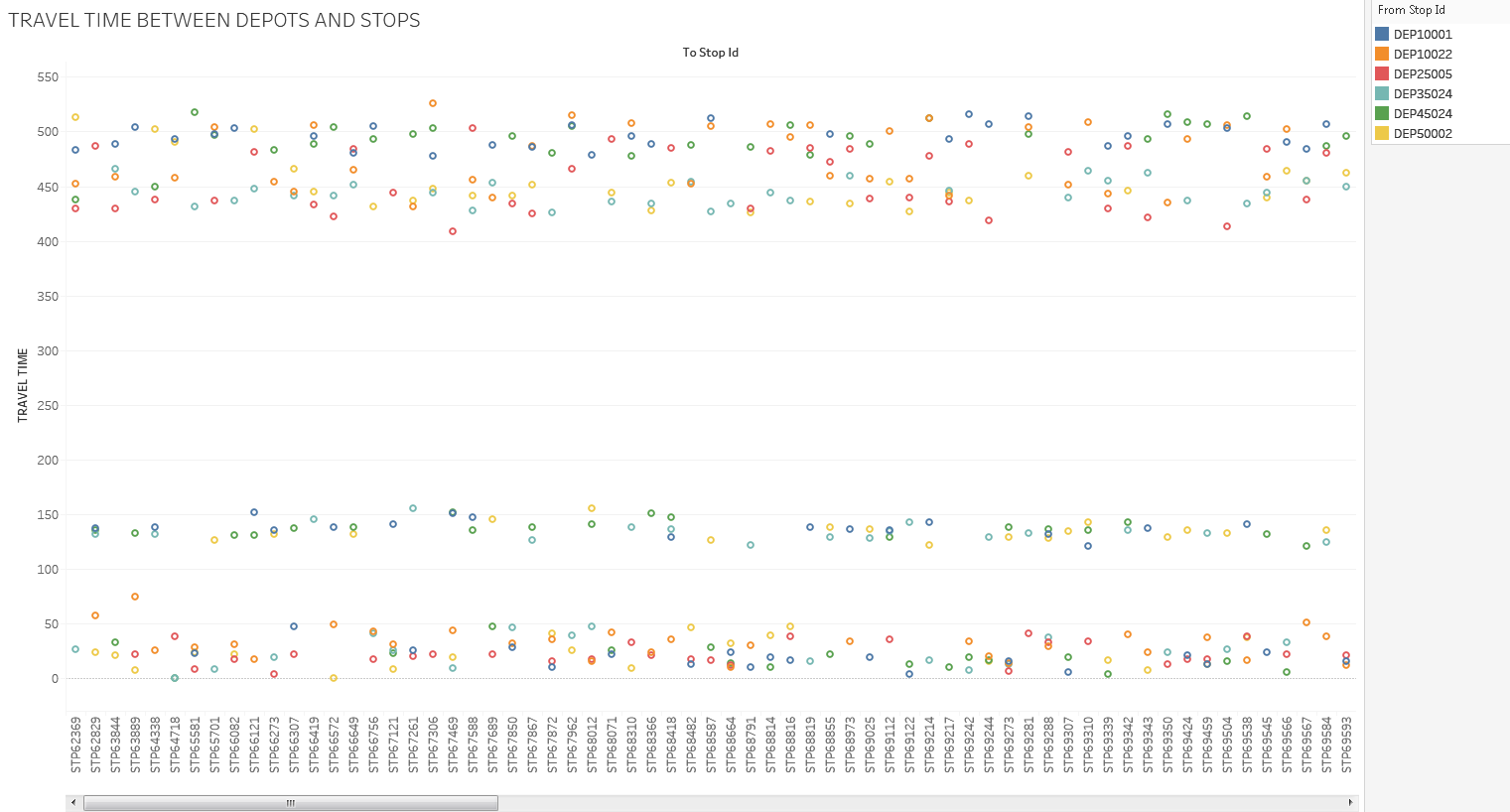
Converting the data frame to matrix for fast computation.

Below is the snapshot of traveltime\_matrix:

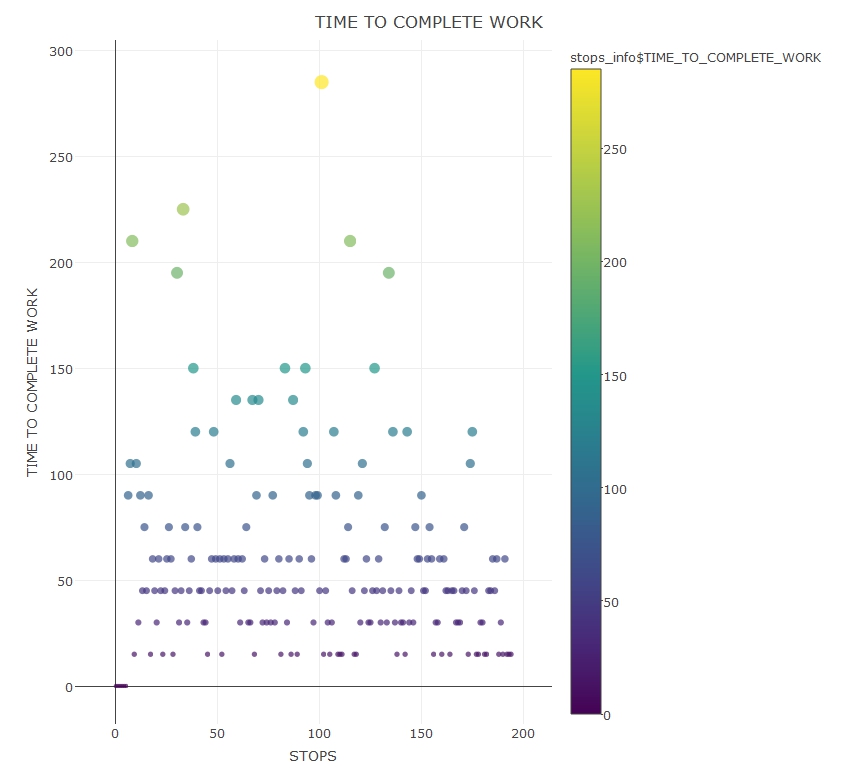




Visualizing the Travel Time between depots and stops.



Visualizing the Job Execution Time (Time to complete work)



# APPROACH

Following are the steps to be followed for using Genetic Algorithm to get the optimal set of routes for a set of parameters:

* Reading the data from MySQL tables into R.
* Understanding the data.
* Variables considered are time taken for service, travel time between depots/stops, time frame specified by the customers for the cleaning service.
* Identifying if there are missing values or not
* If missing values are there, then imputing the missing values using reverse lookup and nearest neighbor approach.
* In Reverse Lookup approach, if travel time is given from a depot/stop to other depot/stop is missing and return travel time is not given, then same travel time is filled for the return travel time.
* In Nearest Neighbors approach, if travel time is not given from a depot/stop to other depot/stop, then nearest neighbors of source and destination depot/stop are identified. Travel time between missing source/depots will be sum of travel time from source to its nearest neighbor depot/stop, travel time between source nearest neighbor and destination nearest neighbor and travel time between destination nearest neighbor to destination depot/stop.
* Convert the data frame to matrix for fast lookup of data.
* Identify which STOP\_ID are depots and stops.
* Find probability of assigning depots based upon depots distance from all stops.
* Identify the Objective function which is minimize the overall time (travel time and job execution time) considering the constraints. If time is minimized, cost will also be minimized, customer and employees will be happy.

Minimize ∑xij + ∑yj + ∑p

₳ xij = xji

where xij is the travel time from stop i to j

and yj is the job execution time at stop j

and pis the penalty for a route.

* Define the penalties:

1. Total travel and job execution must not exceed 660mins for any route.

Apply penalty if exceeds,

1 - 60 mins - exceeded minutes + 10% of total route time

61 - 120 mins - 2 times the exceeded minutes + 20% of total route time

121 - 180 mins - 3 times the exceeded minutes + 30% of total route time

181 - 240 mins - 4 times the exceeded minutes + 40% of total route time ...

2. If any route has more/less than the stops threshold

1 stop - 10% of total route time

2 stops - 20% of total route time

3 stops - 30% of total route time...

3. If a vehicle reaches a customer before/after small time window (30 mins) - no penalty

within small and large time window - 10% of total route time

beyond large time window

1st 60mins - exceeded minutes + 10% of total route time

2nd 60mins - 2 times the exceeded minutes + 20% of total route time.

* Identify the algorithms which can be used to solve the route optimization problem
* Depot’s Nearest Neighbors Model and Genetic Algorithm can be tried.
* There are different set of parameters for which solution is to be computed.
* Run the GA model for all the parameter sets 10 times.
* Each time number of routes can be different.
* Each time the value of objective function value will be different.
* To pick up best solutions among those, cost needs to be calculated.
* Identify the constraints for which most optimum results are obtained.

# MODEL BUILDING

## Depot’s Nearest Neighbors Model

Depot’s Nearest Neighbors algorithm for the Route Optimization considering cleaning services vehicles can be developed. Each service stops are assigned to its nearest depot and constructs routes for the customer set of each depot.

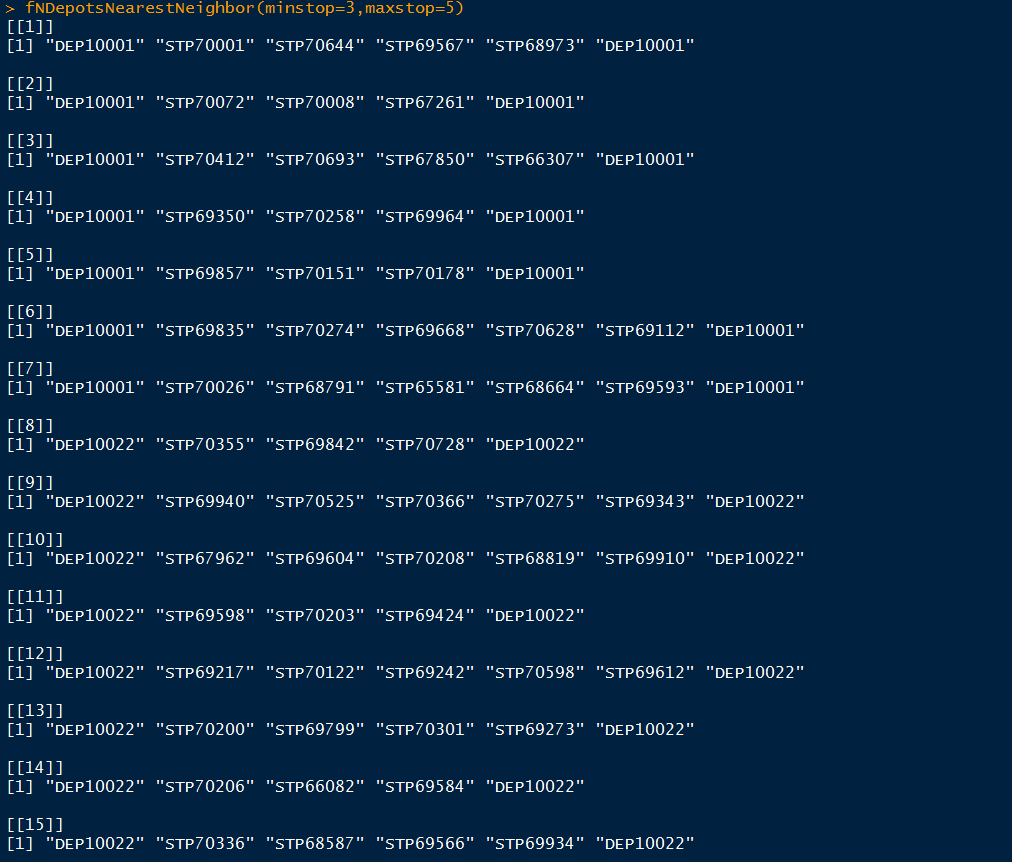
Specifically, the following heuristic procedure is repeated until all the service stops within each group are assigned to a route:

1. Randomly choose route size for each depot based on their capacity.
2. Calculate the travel time for every pair of stops including stops and a depot.
3. Select a pair of stops with the minimum travel time.
4. Connect the stops from a start stop to an end stop by a visiting sequence.
5. Connect the next stops as close as possible to the previously connected end stop to make a route until all the stops are equal to route size.
6. Arrange the route so that a vehicle leaves from a depot and returns to the same depot.

One of the main characteristics of this algorithm is that infeasible solutions are not allowed throughout the search. This algorithm also not guarantee for Feasible solutions. Nearby Feasible solutions are achieved through the simple assignment of stops in a route. Feasible routes are made by a sequential assignment method of a stop to a route that meets the maximum route capacity constraints at each stop. In other words, we assign vehicle k iteratively to the constructed route if its capacity is not done.

Also, there are other constraints in the problem earliest job start time to customer and latest job finish time, maximum route time, and there is penalty if constraints are not met, Genetic Algorithm is the best approach in which initial population can be output of the above algorithm. By following this approach feasible solutions can be easily achieved in less time.

In R, **fNDepotsNearestNeighbor** function the above algorithm is implemented. The output of the function is list of routes.



## Genetic Algorithm

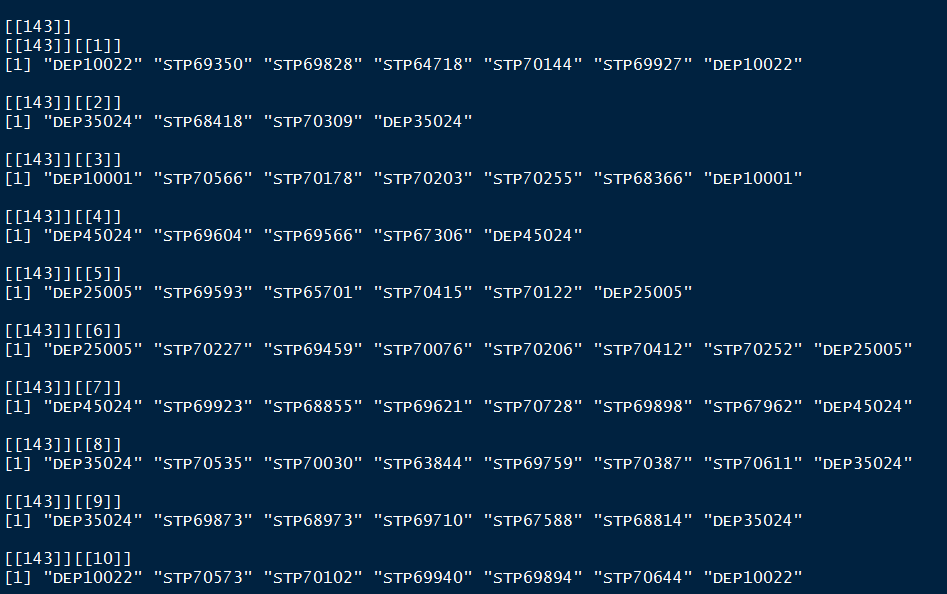
Genetic algorithms are stochastic search algorithms based on the mechanisms of natural selection. Most basic unit is a Gene. In our case, gene is a depot/stop.



Each route in the population is called a *chromosome* and represents a solution to the problem.



Genetic algorithm starts with an initial set of randomly generated solutions called a *population.* In this case, population is set of routes.



A chromosome evolves through successive iterations called *generations (or iterations).*

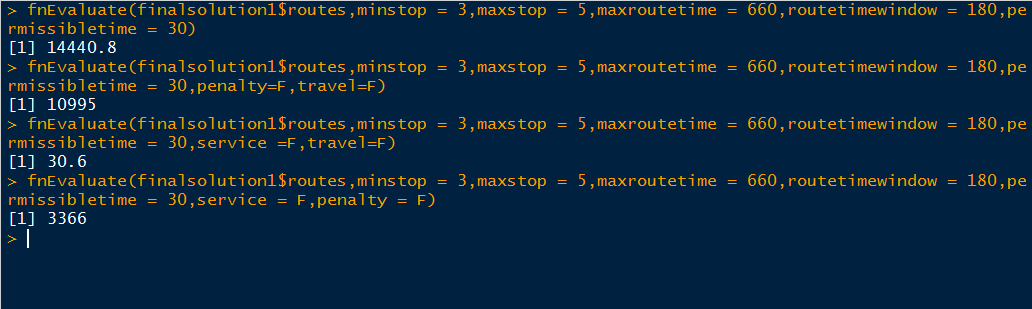
During each generation, only strong elite population is eligible for genetic operations.

During each generation, genetic operators, such as crossover, mutation, and selection applied on strong population which will create better chromosomes or routes.

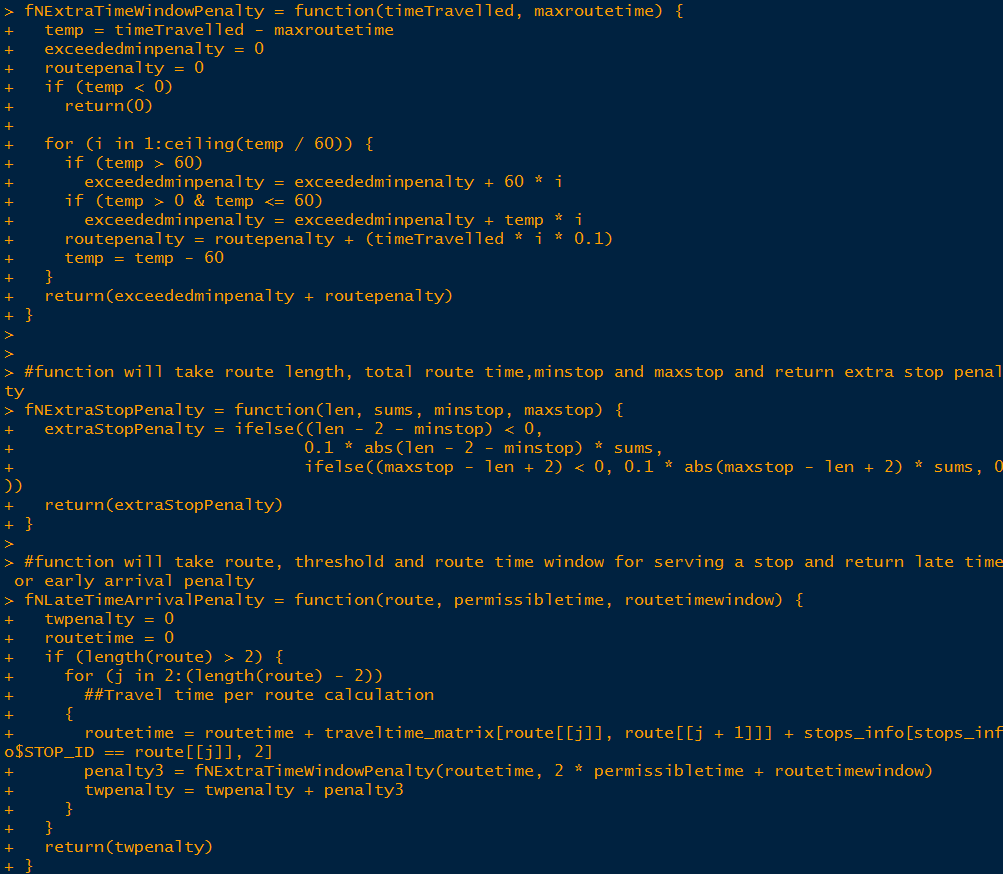
Each chromosome is then evaluated by measure of fitness as a solution. If it is better than the population, it will be added in population otherwise discarded.

### Defining Objective Function

The value of objective function is total time travelled by all the vehicles on the routes, total time spent in executing the job and penalty time. **fnEvaluate** function will calculate the objective function value. This function also return travel time, service time and penalty time separately.



Penalty calculation is implemented in three methods: **fNExtraTimeWindowPenalty, fNExtraStopPenalty, fNLateTimeArrivalPenalty.**

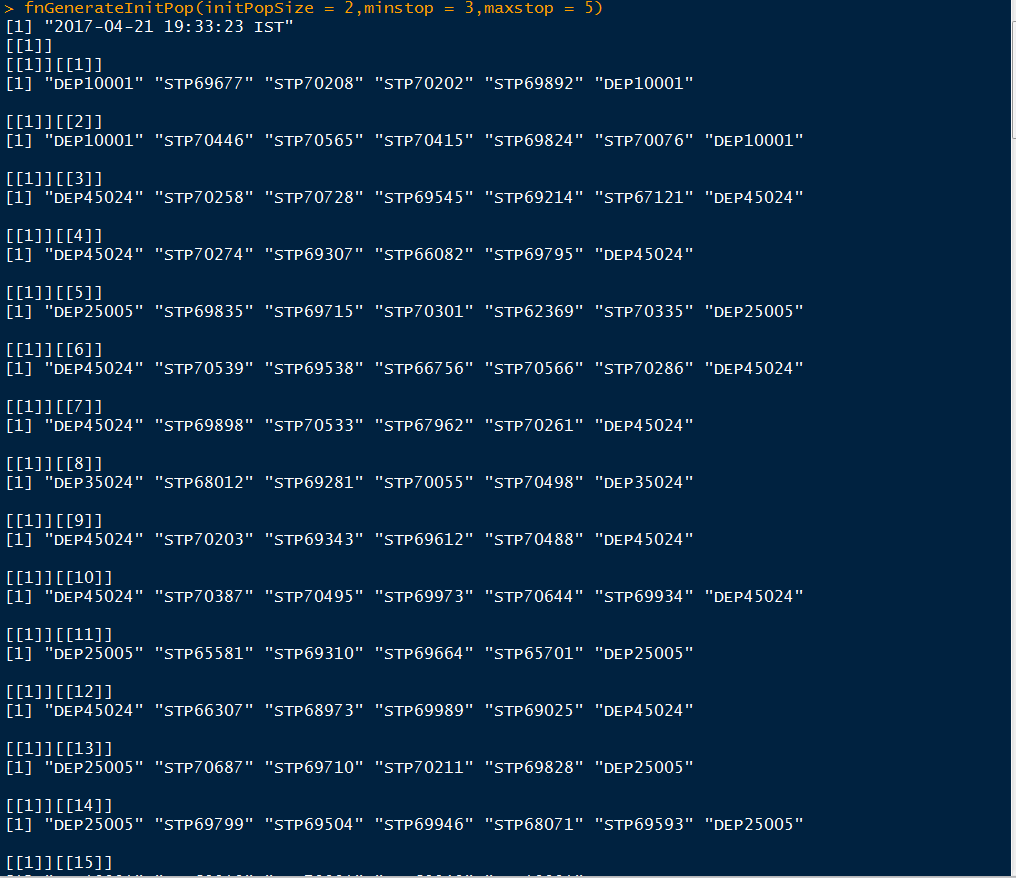


### Initial Population Generation

Initial Population is generated randomly. Every time seed is different while generating initial population. While generating initial population, all the routes sizes are randomly decided between minimum stops and maximum stops. Stops are randomly assigned to the routes.

It’s not necessary that number of routes in each set of chromosome will be equal.

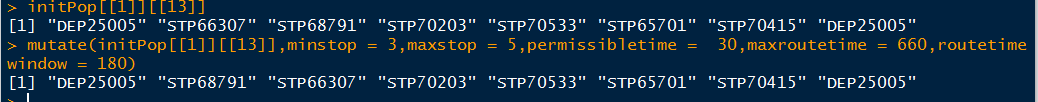
**fnGenerateInitPop** function will take initial Population size as input and output that many number of set of solutions in a list.



### Mutation

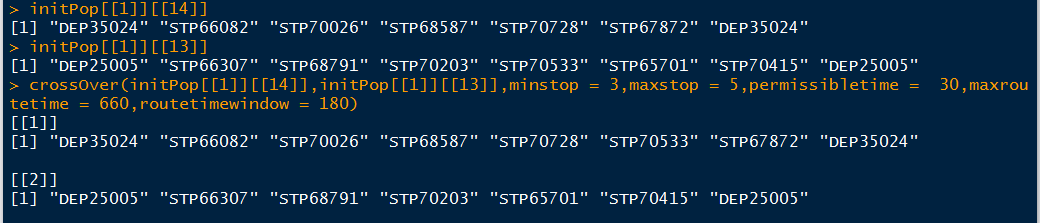
**Mutation** is a [genetic operator](https://en.wikipedia.org/wiki/Genetic_operator) used to maintain [genetic diversity](https://en.wikipedia.org/wiki/Genetic_diversity) from one generation of a population of [genetic algorithm](https://en.wikipedia.org/wiki/Genetic_algorithm) [chromosomes](https://en.wikipedia.org/wiki/Chromosome_(genetic_algorithm)) to the next. It is analogous to biological [mutation](https://en.wikipedia.org/wiki/Mutation). Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to mutation probability which is user defined. This probability should be set low. If it is set too high, the search will turn into a primitive random search. In our problem, mutation probability is initially set to 0.8 and it decays with iterations.

In mutation, a random chromosome is chosen and two random stops will be swapped.



### Crossover

In [genetic algorithms](https://en.wikipedia.org/wiki/Genetic_algorithm), **crossover** is a [genetic operator](https://en.wikipedia.org/wiki/Genetic_operator) used to vary the programming of a chromosome or [chromosomes](https://en.wikipedia.org/wiki/Chromosome_(genetic_algorithm)) from one generation to the next. It is analogous to [reproduction](https://en.wikipedia.org/wiki/Reproduction) and [biological crossover](https://en.wikipedia.org/wiki/Chromosomal_crossover), upon which genetic algorithms are based. Crossover is a process of taking more than one parent solutions and producing a child solution from them. In our problem, Single point crossover and Two-point crossover is used. When the size of randomly selected two routes (or chromosomes) are equal, then one random point is selected and swapped between two chromosomes and children are formed for next generation. When the size of randomly selected two chromosomes are unequal, then one random stop and last stop are selected in chromosome and swapped between two parents and children are formed. If the number of stops are less than minimum stops in one chromosome, then random stop is selected from other chromosome and added in random position in between depots. If the number of stops are 2 less than minimum stops in one chromosome, then that parent is discarded and all the stops are added to other chromosome.



### Selection

Selection is a process in which individuals create a new population for the next generation based on their fitness function values. In this solution, parents are chosen with a rank-based mechanism. Only top 20 solutions are used to create new population.

### Stopping Criteria and parameter selection

Stopping criteria is if in 50 consecutive iterations/generations fitness value is not improved or maximum number of iterations reached. With lot of experimentation, following

* Population size : 100 or 150
* Maximum Iterations: 1000
* Mutation start rate ( Pm ) : 0.8
* Crossover rate ( Pc ) : 1-Pm

To converge faster, multiple crossovers and mutations can be done in one iteration.

* Mutation: (0.2\* vehiclefleetsize) where vehiclefleetsize is number of routes/chromosomes in a solutions.
* Crossover: (0.3\* vehiclefleetsize)

**Note:** Above values are decided by experimentation.

### Genetic Algorithm Implementation

Following Steps needs to be followed to get best solution:

1. Initial Population is generated first.
2. Evaluate the initial population.
3. Repeat until stopping criteria meets.
   1. Select one random solution from top 20 solutions
   2. Do mutation or crossover based upon mutation start probability in that solution.

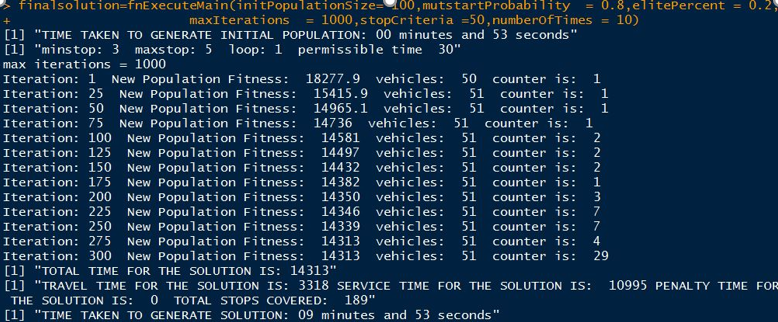
Output from the **fnRunGeneticAlgo** function is best solution for the given parameters/constraints.

In this solution, Genetic Algorithm executed for two different initial populations

1. Randomly Generated Initail Population
2. Population generated using Depot’s Nearest Neighbors model.

In **fnExecuteMain** function, if random=TRUE, Initial Population is generated randomly, otherwise

population is generated using Depot’s Nearest Neighbors model. This function will output a list with solutions for all the parameters.

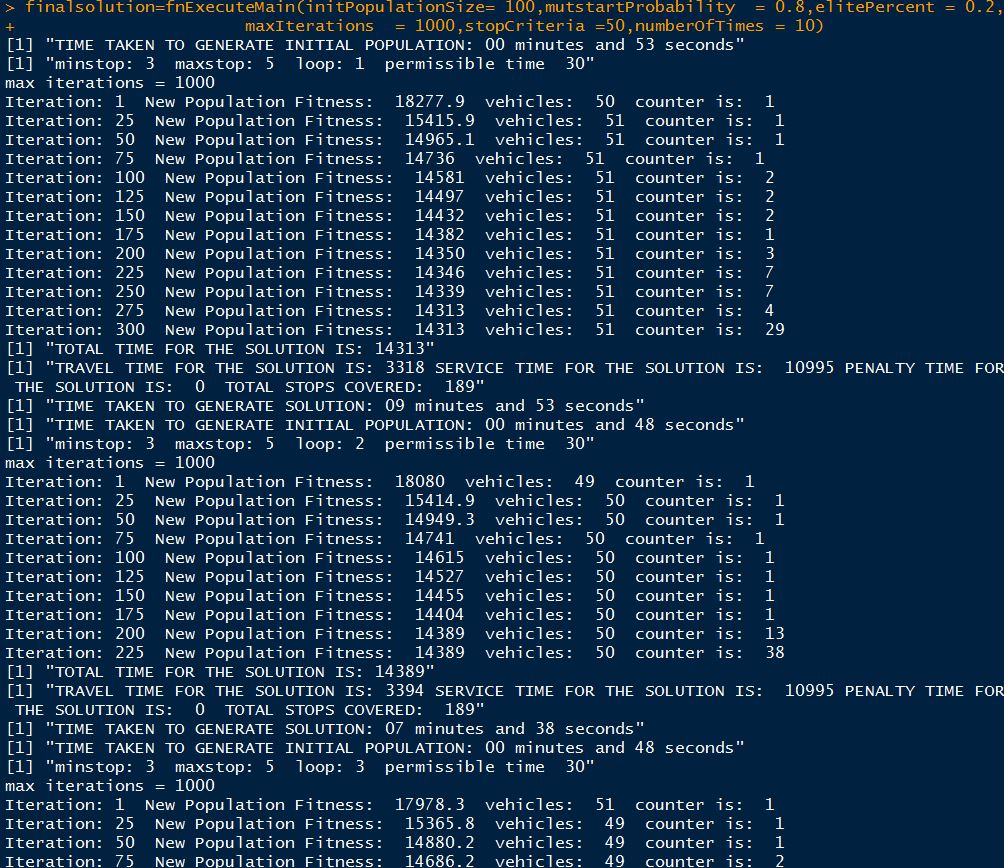


# RESULTS

The below function will run 10 times with maximum number of iterations as 1000, elite percentage is 20%,

Mutation Start Probability is 0.8 and Initial Population Size is 100.

By default, it will run Genetic Algorithm with Population generated with Depot’s Nearest Neighbors model. If optional parameter **random=TRUE** argument is passed in the function, then it will run Genetic Algorithm with Randomly generated Initial Population. If optional argument **dnn=TRUE** argument is passed in the function, then it give solution based on Depot’s Nearest Neigbors model.



## Results for all parameters

### Depot’s Nearest Neighbors Model

1. Stops Range: 3 to 5 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 27595.3 | 4018 | 10995 | 12582 | 50 | 8148 |
| 27889.2 | 4036 | 10995 | 12858 | 49 | 8214 |
| 24214.6 | 3975 | 10995 | 9245 | 47 | 7138 |
| 22203.1 | 4193 | 10995 | 7015 | 51 | 6638 |
| 20190.4 | 3993 | 10995 | 5202 | 49 | 6033 |
| 29059.4 | 4007 | 10995 | 14057 | 50 | 8563 |
| 21817.2 | 4129 | 10995 | 6693 | 49 | 6494 |
| 25807.8 | 4002 | 10995 | 10811 | 47 | 7589 |
| 20733.8 | 4156 | 10995 | 5583 | 51 | 6222 |
| 25571 | 4101 | 10995 | 10475 | 50 | 7575 |

1. Stops Range: 4 to 6 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 35118.3 | 3793 | 10995 | 20330 | 40 | 10104 |
| 40603.9 | 3796 | 10995 | 25813 | 38 | 11623 |
| 37919.1 | 3767 | 10995 | 23157 | 41 | 10915 |
| 39407.7 | 3874 | 10995 | 24539 | 38 | 11284 |
| 33944.8 | 3712 | 10995 | 19238 | 39 | 9754 |
| 31250.2 | 3723 | 10995 | 16532 | 39 | 8990 |
| 45397.7 | 3743 | 10995 | 30660 | 37 | 12964 |
| 32107.9 | 3772 | 10995 | 17341 | 41 | 9269 |
| 27920.1 | 3751 | 10995 | 13174 | 41 | 8082 |
| 32556 | 3851 | 10995 | 17710 | 41 | 9396 |

1. Stops Range: 5 to 7 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 55695.2 | 3619 | 10995 | 41081 | 33 | 15811 |
| 53389.7 | 3615 | 10995 | 38780 | 33 | 15158 |
| 61528.4 | 3840 | 10995 | 46693 | 35 | 17499 |
| 51354.8 | 3572 | 10995 | 36788 | 33 | 14581 |
| 55170.5 | 3642 | 10995 | 40533 | 35 | 15697 |
| 40576 | 3699 | 10995 | 25882 | 35 | 11562 |
| 43179.4 | 3708 | 10995 | 28476 | 36 | 12318 |
| 46686.9 | 3621 | 10995 | 32071 | 33 | 13259 |
| 59725.2 | 3558 | 10995 | 45172 | 32 | 16935 |
| 52529.1 | 3578 | 10995 | 37956 | 34 | 14932 |

1. Stops Range: 3 to 5 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 18901.1 | 4118 | 10995 | 3788 | 52 | 5720 |
| 19898.1 | 4008 | 10995 | 4895 | 50 | 5968 |
| 20938.1 | 4208 | 10995 | 5735 | 49 | 6245 |
| 17268.4 | 4146 | 10995 | 2127 | 53 | 5275 |
| 16774.7 | 4128 | 10995 | 1652 | 48 | 5047 |
| 24322.2 | 4053 | 10995 | 9274 | 49 | 7203 |
| 17297 | 4123 | 10995 | 2179 | 50 | 5231 |
| 20427.4 | 4151 | 10995 | 5281 | 51 | 6135 |
| 17615.1 | 4090 | 10995 | 2530 | 50 | 5321 |
| 19515.9 | 4105 | 10995 | 4416 | 48 | 5824 |

1. Stops Range: 4 to 6 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 28452.5 | 3668 | 10995 | 13789 | 42 | 8250 |
| 27194.9 | 3877 | 10995 | 12323 | 41 | 7877 |
| 23905 | 3888 | 10995 | 9022 | 41 | 6945 |
| 29928.9 | 3875 | 10995 | 15059 | 41 | 8651 |
| 25742.4 | 3731 | 10995 | 11016 | 39 | 7430 |
| 26459.2 | 3823 | 10995 | 11641 | 39 | 7633 |
| 39016.2 | 3916 | 10995 | 24105 | 42 | 11244 |
| 25181.8 | 3846 | 10995 | 10341 | 41 | 7306 |
| 20647.8 | 3812 | 10995 | 5841 | 40 | 6004 |
| 22677 | 3848 | 10995 | 7834 | 41 | 6597 |

1. Stops Range: 5 to 7 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 40219.5 | 3740 | 10995 | 25485 | 34 | 11444 |
| 44830 | 3529 | 10995 | 30306 | 33 | 12733 |
| 32694.6 | 3650 | 10995 | 18050 | 35 | 9329 |
| 39808.7 | 3731 | 10995 | 25083 | 35 | 11345 |
| 43667.8 | 3581 | 10995 | 29092 | 34 | 12421 |
| 24598 | 3758 | 10995 | 9845 | 36 | 7053 |
| 44766.9 | 3656 | 10995 | 30116 | 33 | 12715 |
| 61281.6 | 3523 | 10995 | 46764 | 33 | 17394 |
| 41684.1 | 3731 | 10995 | 26958 | 34 | 11859 |
| 40272.7 | 3637 | 10995 | 25641 | 34 | 11459 |

### Genetic Algorithm using Random Initial Population

1. Stops Range: 3 to 5 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 18412.6 | 5914 | 10995 | 1504 | 49 | 5529 |
| 18379.9 | 6000 | 10995 | 1385 | 50 | 5537 |
| 17450.3 | 5697 | 10995 | 758 | 50 | 5274 |
| 18706.2 | 6421 | 10995 | 1290 | 51 | 5647 |
| 17380.9 | 5445 | 10995 | 941 | 49 | 5237 |
| 19544 | 6703 | 10995 | 1846 | 49 | 5850 |
| 16872.7 | 5526 | 10995 | 352 | 52 | 5146 |
| 17367.5 | 5590 | 10995 | 782 | 50 | 5250 |
| 18275.3 | 6219 | 10995 | 1061 | 51 | 5525 |
| 19171.7 | 6666 | 10995 | 1511 | 51 | 5779 |

1. Stops Range: 4 to 6 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 25494.4 | 5695 | 10995 | 8804 | 38 | 7342 |
| 22788.1 | 6677 | 10995 | 5116 | 40 | 6610 |
| 21148.5 | 5975 | 10995 | 4178 | 40 | 6146 |
| 20943.6 | 5051 | 10995 | 4898 | 41 | 6106 |
| 23378.2 | 5796 | 10995 | 6587 | 39 | 6760 |
| 23295.3 | 6970 | 10995 | 5330 | 41 | 6772 |
| 24448.2 | 5938 | 10995 | 7515 | 39 | 7063 |
| 19879.6 | 5406 | 10995 | 3479 | 39 | 5769 |
| 19012.9 | 5634 | 10995 | 2384 | 41 | 5558 |
| 24600.7 | 6256 | 10995 | 7350 | 39 | 7107 |

1. Stops Range: 5 to 7 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 37509.9 | 6740 | 10995 | 19775 | 34 | 10676 |
| 34876.9 | 6405 | 10995 | 17477 | 32 | 9895 |
| 41204.2 | 5896 | 10995 | 24313 | 32 | 11688 |
| 42733 | 6088 | 10995 | 25650 | 32 | 12121 |
| 37113.7 | 6001 | 10995 | 20118 | 32 | 10529 |
| 41584.4 | 6447 | 10995 | 24142 | 32 | 11795 |
| 35441.5 | 5735 | 10995 | 18711 | 33 | 10072 |
| 38181.9 | 6045 | 10995 | 21142 | 32 | 10831 |
| 29134.4 | 5473 | 10995 | 12666 | 34 | 8303 |
| 29048 | 5228 | 10995 | 12825 | 33 | 8261 |

1. Stops Range: 3 to 5 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 17692.3 | 6379 | 10995 | 318 | 51 | 5360 |
| 16669.1 | 5421 | 10995 | 253 | 51 | 5070 |
| 18216.3 | 6710 | 10995 | 511 | 49 | 5473 |
| 16902.3 | 5600 | 10995 | 307 | 50 | 5119 |
| 17107.9 | 5977 | 10995 | 136 | 51 | 5195 |
| 16774.9 | 5408 | 10995 | 372 | 51 | 5100 |
| 17192.7 | 6042 | 10995 | 156 | 53 | 5254 |
| 16638 | 5273 | 10995 | 370 | 51 | 5061 |
| 18371.8 | 6884 | 10995 | 493 | 51 | 5553 |
| 18120.5 | 6724 | 10995 | 402 | 53 | 5517 |

1. Stops Range: 4 to 6 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 18745.4 | 6731 | 10995 | 1019 | 41 | 5483 |
| 20284.9 | 6185 | 10995 | 3105 | 40 | 5901 |
| 20405.6 | 6296 | 10995 | 3115 | 40 | 5936 |
| 19562 | 6373 | 10995 | 2194 | 40 | 5696 |
| 18127 | 5670 | 10995 | 1462 | 39 | 5272 |
| 18733.7 | 5477 | 10995 | 2262 | 39 | 5444 |
| 19422.6 | 6004 | 10995 | 2424 | 39 | 5639 |
| 17619.2 | 5497 | 10995 | 1127 | 39 | 5128 |
| 18373.9 | 5950 | 10995 | 1429 | 41 | 5377 |
| 19481.1 | 5981 | 10995 | 2505 | 38 | 5638 |

1. Stops Range: 5 to 7 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 27733.4 | 5695 | 10995 | 11043 | 32 | 7871 |
| 38580.3 | 7027 | 10995 | 20558 | 33 | 10962 |
| 20096.3 | 5708 | 10995 | 3393 | 34 | 5742 |
| 23577.4 | 5673 | 10995 | 6909 | 32 | 6693 |
| 26828.6 | 6090 | 10995 | 9744 | 32 | 7615 |
| 25950.2 | 5875 | 10995 | 9080 | 34 | 7401 |
| 23642.1 | 5058 | 10995 | 7589 | 32 | 6712 |
| 27661.9 | 6319 | 10995 | 10348 | 33 | 7868 |
| 21839.6 | 5750 | 10995 | 5095 | 34 | 6236 |
| 25606.8 | 5732 | 10995 | 8880 | 32 | 7268 |

### Genetic Algorithm using DNN Generated Initial Population

1. Stops Range: 3 to 5 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 14440.8 | 3366 | 10995 | 80 | 52 | 4457 |
| 14676.6 | 3483 | 10995 | 199 | 52 | 4523 |
| 14554.8 | 3400 | 10995 | 160 | 50 | 4454 |
| 14415 | 3420 | 10995 | 0 | 51 | 4432 |
| 14686.4 | 3503 | 10995 | 188 | 48 | 4456 |
| 14523.8 | 3369 | 10995 | 160 | 49 | 4427 |
| 14377 | 3382 | 10995 | 0 | 51 | 4421 |
| 14305.8 | 3151 | 10995 | 160 | 49 | 4366 |
| 15083.5 | 3385 | 10995 | 704 | 50 | 4604 |
| 14811.6 | 3513 | 10995 | 304 | 52 | 4562 |

1. Stops Range: 4 to 6 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 15190.7 | 3356 | 10995 | 840 | 41 | 4476 |
| 15537.2 | 3398 | 10995 | 1144 | 43 | 4609 |
| 14850.5 | 3359 | 10995 | 496 | 43 | 4414 |
| 14741 | 3468 | 10995 | 278 | 42 | 4366 |
| 14452 | 3217 | 10995 | 240 | 41 | 4266 |
| 15134.5 | 3251 | 10995 | 888 | 44 | 4512 |
| 14778.1 | 3264 | 10995 | 519 | 41 | 4359 |
| 14577.1 | 3340 | 10995 | 242 | 43 | 4337 |
| 14753.6 | 3423 | 10995 | 336 | 41 | 4352 |
| 14628.4 | 3448 | 10995 | 185 | 44 | 4369 |

1. Stops Range: 5 to 7 and Time window range: -30 to +30 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 18133.6 | 3236 | 10995 | 3903 | 36 | 5221 |
| 19204.9 | 3296 | 10995 | 4914 | 35 | 5507 |
| 19904.6 | 3180 | 10995 | 5730 | 35 | 5706 |
| 19074.8 | 3462 | 10995 | 4618 | 36 | 5488 |
| 19136.7 | 3434 | 10995 | 4708 | 36 | 5506 |
| 18068.9 | 3239 | 10995 | 3835 | 35 | 5185 |
| 18881.7 | 3405 | 10995 | 4482 | 35 | 5416 |
| 17580.1 | 3069 | 10995 | 3516 | 34 | 5029 |
| 17585.4 | 3300 | 10995 | 3290 | 35 | 5048 |
| 17817.1 | 3210 | 10995 | 3612 | 36 | 5132 |

1. Stops Range: 3 to 5 and Time window range: -60 to +60 minutes

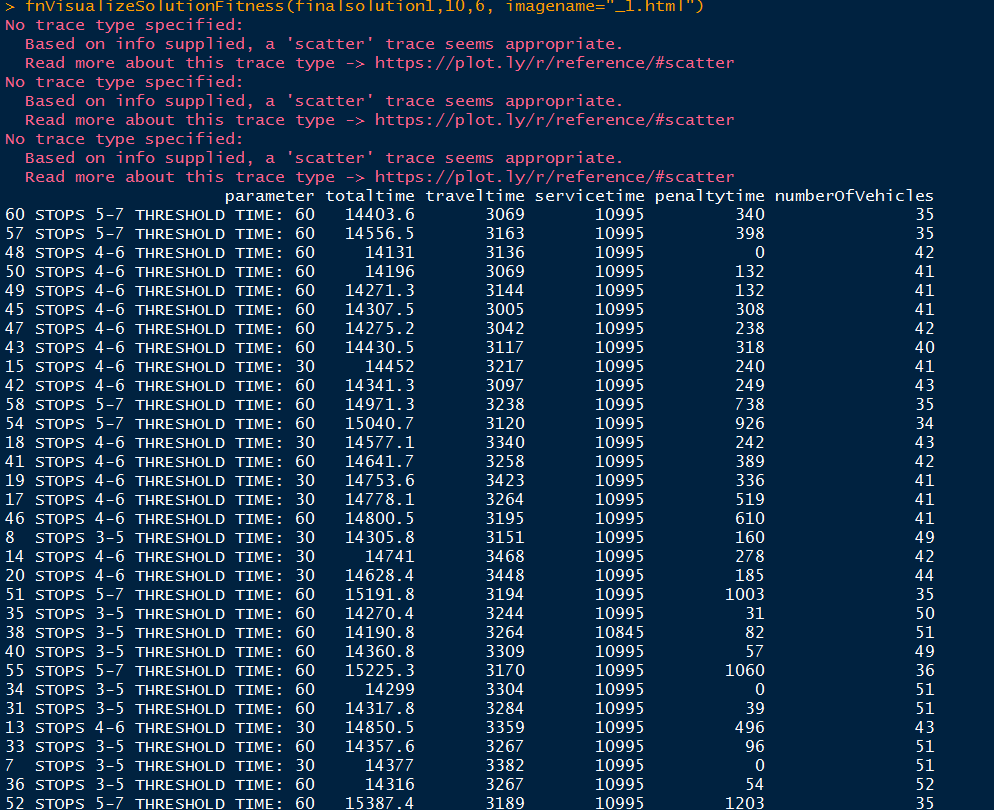
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 14317.8 | 3284 | 10995 | 39 | 51 | 4404 |
| 14408.6 | 3318 | 10995 | 96 | 51 | 4430 |
| 14357.6 | 3267 | 10995 | 96 | 51 | 4415 |
| 14299 | 3304 | 10995 | 0 | 51 | 4399 |
| 14270.4 | 3244 | 10995 | 31 | 50 | 4373 |
| 14316 | 3267 | 10995 | 54 | 52 | 4421 |
| 14659.5 | 3358 | 10995 | 306 | 50 | 4483 |
| 14190.8 | 3264 | 10995 | 82 | 51 | 4376 |
| 14410.4 | 3278 | 10995 | 137 | 52 | 4448 |
| 14360.8 | 3309 | 10995 | 57 | 49 | 4381 |

1. Stops Range: 4 to 6 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 14641.7 | 3258 | 10995 | 389 | 42 | 4338 |
| 14341.3 | 3097 | 10995 | 249 | 43 | 4270 |
| 14430.5 | 3117 | 10995 | 318 | 40 | 4242 |
| 14923.4 | 3223 | 10995 | 705 | 43 | 4435 |
| 14307.5 | 3005 | 10995 | 308 | 41 | 4225 |
| 14800.5 | 3195 | 10995 | 610 | 41 | 4365 |
| 14275.2 | 3042 | 10995 | 238 | 42 | 4234 |
| 14131 | 3136 | 10995 | 0 | 42 | 4193 |
| 14271.3 | 3144 | 10995 | 132 | 41 | 4215 |
| 14196 | 3069 | 10995 | 132 | 41 | 4194 |

1. Stops Range: 5 to 7 and Time window range: -60 to +60 minutes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FITNESS VALUE** | **TRAVEL TIME** | **SERVICE TIME** | **PENALTY TIME** | **NUMBER OF VEHICLES** | **TOTAL COST** |
| 15191.8 | 3194 | 10995 | 1003 | 35 | 4370 |
| 15387.4 | 3189 | 10995 | 1203 | 35 | 4426 |
| 17095.4 | 3212 | 10995 | 2888 | 35 | 4909 |
| 15040.7 | 3120 | 10995 | 926 | 34 | 4310 |
| 15225.3 | 3170 | 10995 | 1060 | 36 | 4397 |
| 16613.7 | 3294 | 10995 | 2325 | 35 | 4773 |
| 14556.5 | 3163 | 10995 | 398 | 35 | 4190 |
| 14971.3 | 3238 | 10995 | 738 | 35 | 4308 |
| 15602.4 | 3199 | 10995 | 1408 | 35 | 4486 |
| 14403.6 | 3069 | 10995 | 340 | 35 | 4147 |



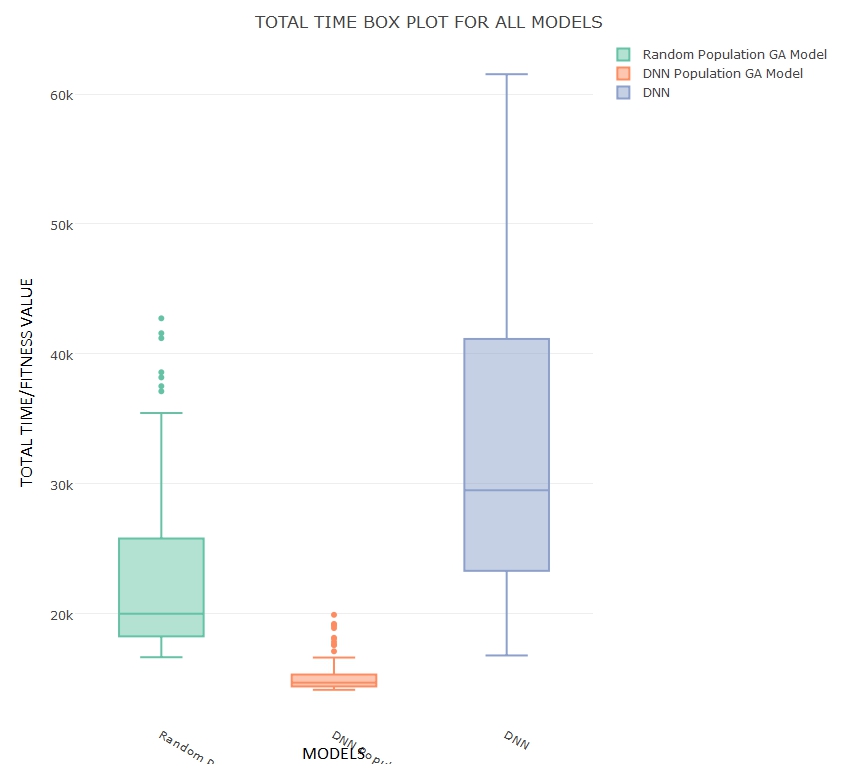
# ANALYSIS

For all three the models, performance is reported.

Evaluation metric is better fitness values and cost in less time for all the 6 sets of parameters.

This is important for this case as we have to provide optimal set of routes to the customer on daily basis which would help customer in optimizing the resources and increasing customer satisfaction.

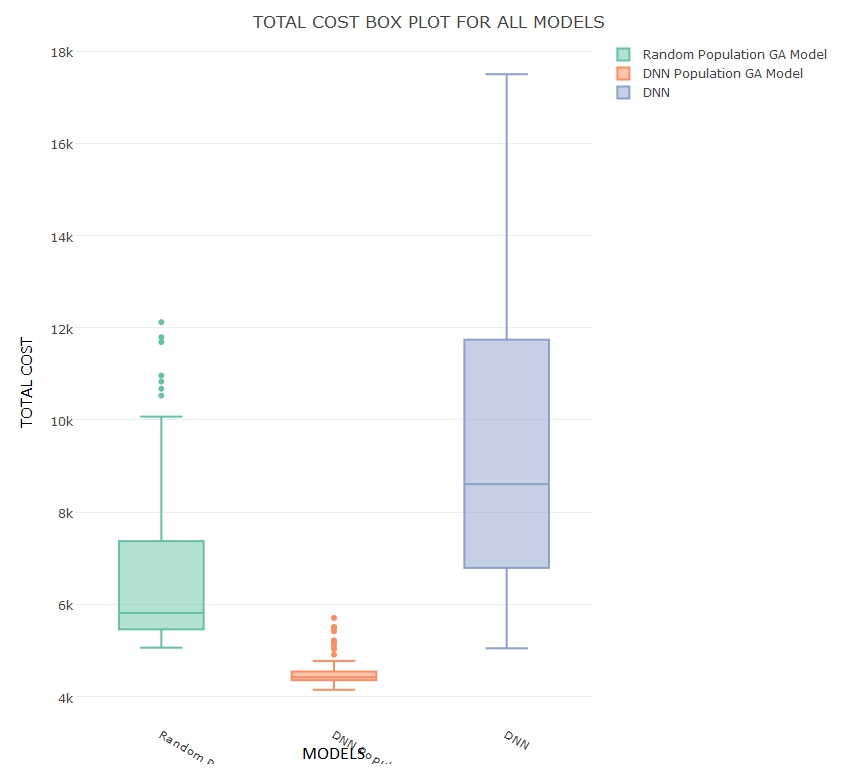
## Model Comparison



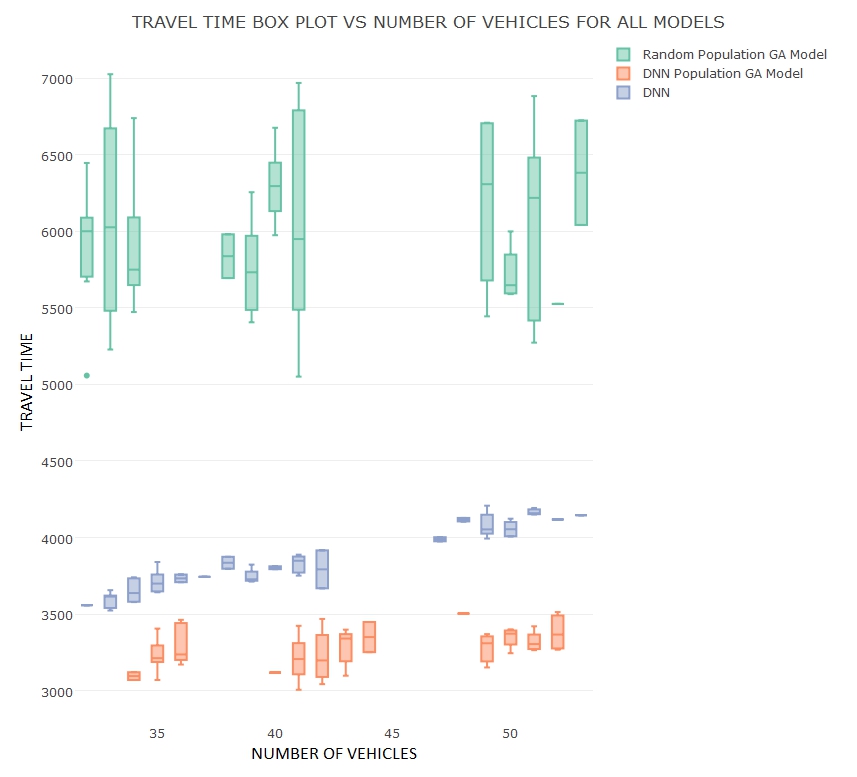
The three different models are compared by running the model against data. Box Plot and Scatter Plot is designed for all models for Total Time, Total Cost.

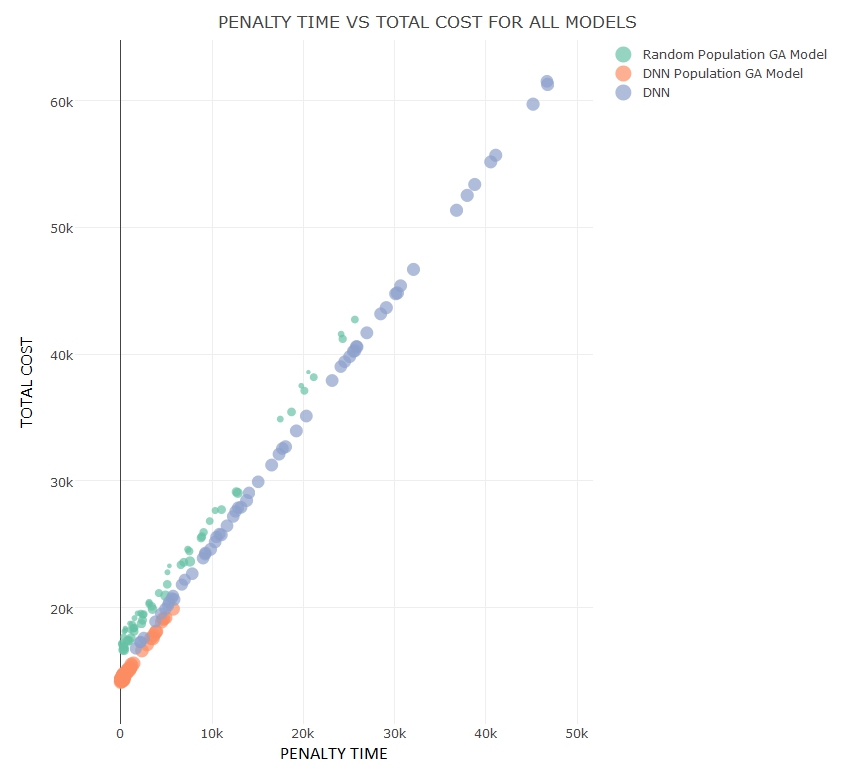
Total Time Box Plot for All Models: As seen in the below chart, total time/fitness value is lowest in DNN Population GA Model and highest in DNN model. So, DNN Population GA Model is giving best results. In DNN model, its high because we considered low distance initially but in the last stops with more distances will be left. That may be the reason, total time/fitness value is high.

Total Cost Box Plot for All Models: As seen in the below chart, total cost value is lowest in DNN Population GA Model. So, DNN Population GA Model is giving best results.



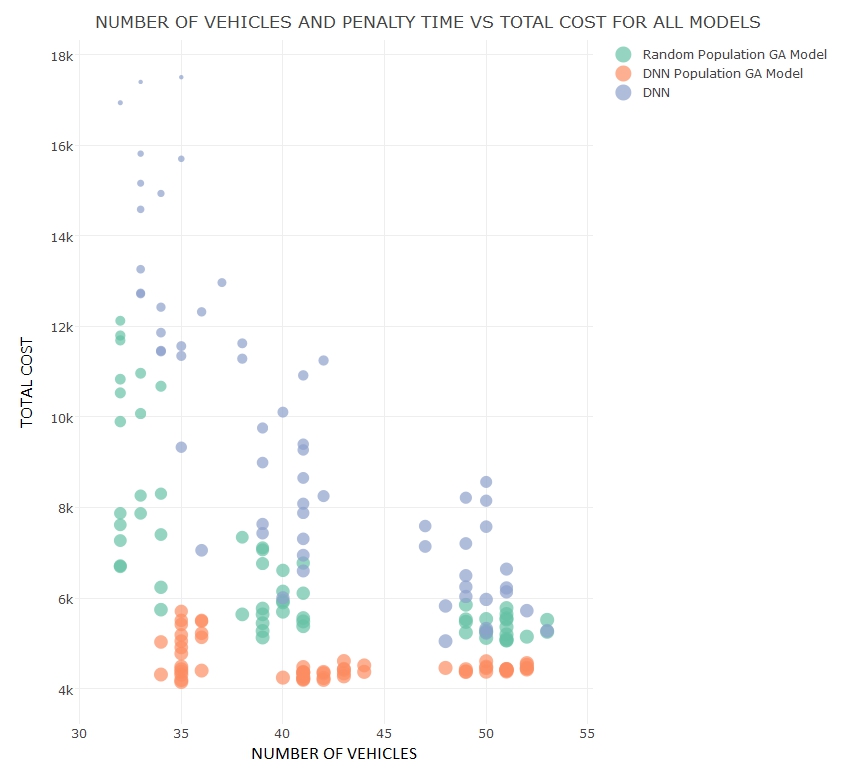
Travel Time Box Plot for All Models: As seen in the below chart, travel time value is lowest in DNN Population GA Model. So, DNN Population GA Model is giving best results.



Penalty Time effect on Total Cost: As seen in the below plot, As Penalty Time increases, Total Cost is also increasing. Penalty Time is low for DNN Population GA Model as compared to other two models.

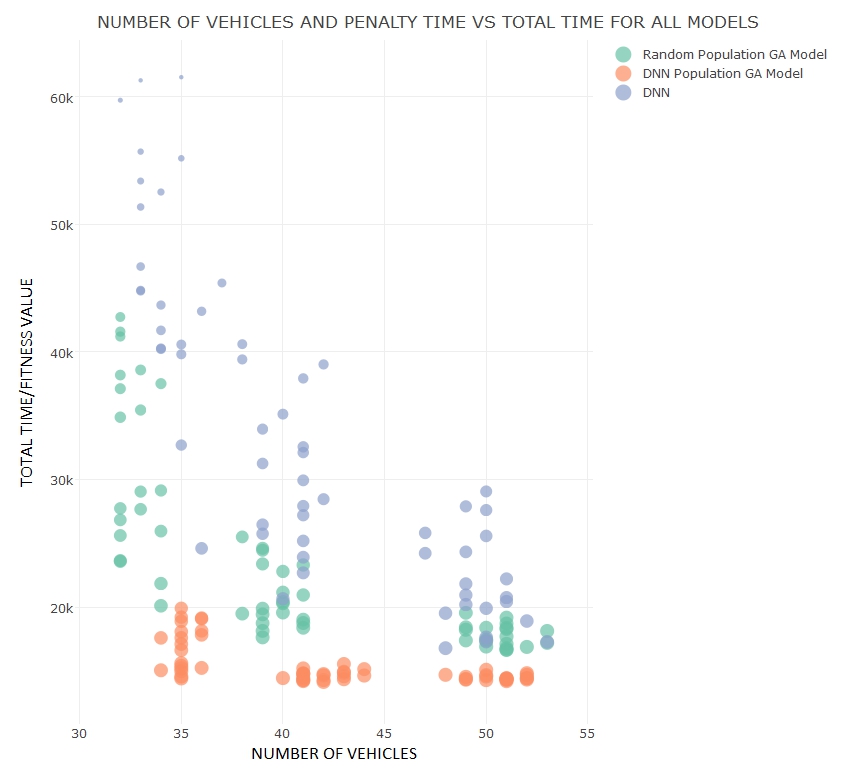
Number of Vehicles and Penalty Time Vs Total Cost for All Models: As seen in the model if number of vehicles are low and penalty time is low, then total cost will be low.

Large Size of Bubble depicts low penalty time and small size depicts high penalty time



Number of Vehicles and Penalty Time Vs Total Cost for All Models: As seen in the model if number of vehicles are low and penalty time is low, then total cost will be low.

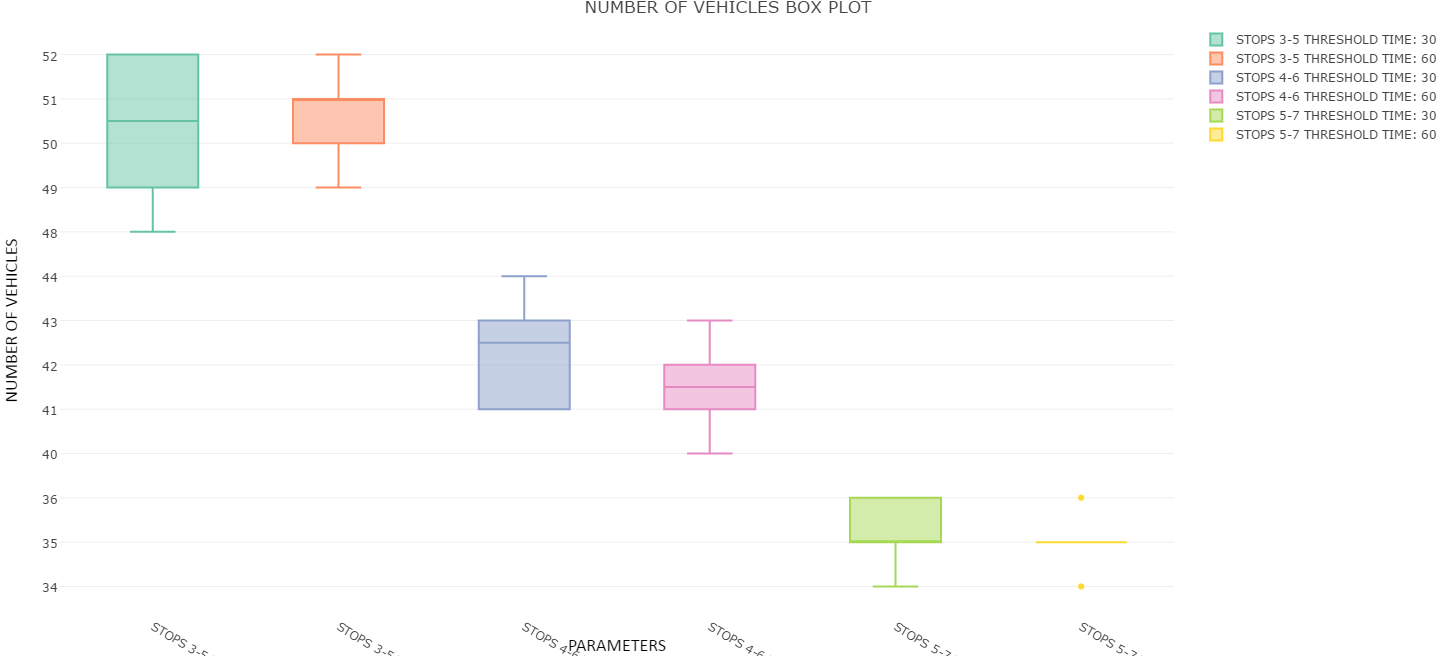
Large Size of Bubble depicts low penalty time and small size depicts high penalty time



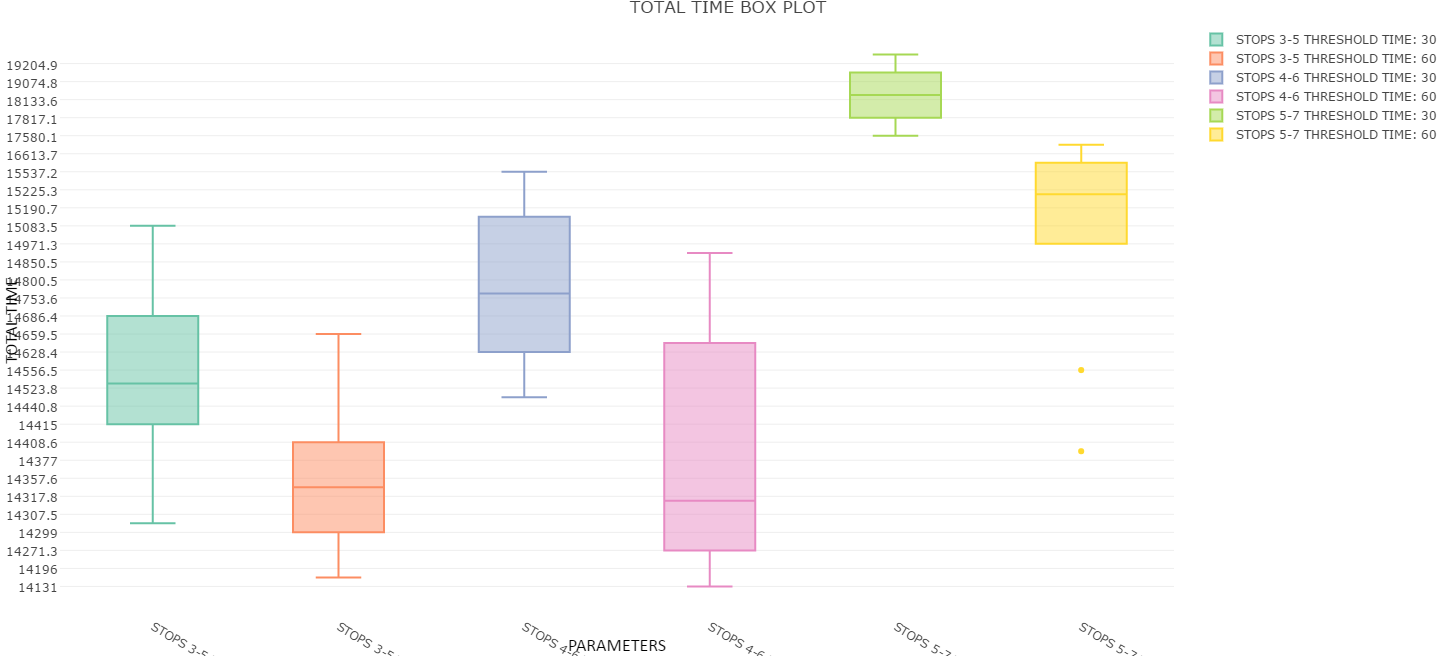
Number of Vehicles and Penalty Time Vs Total Time for All Models: As seen in the model if number of vehicles are low and penalty time is low, then total time will be low.

Large Size of Bubble depicts low penalty time and small size depicts high penalty time

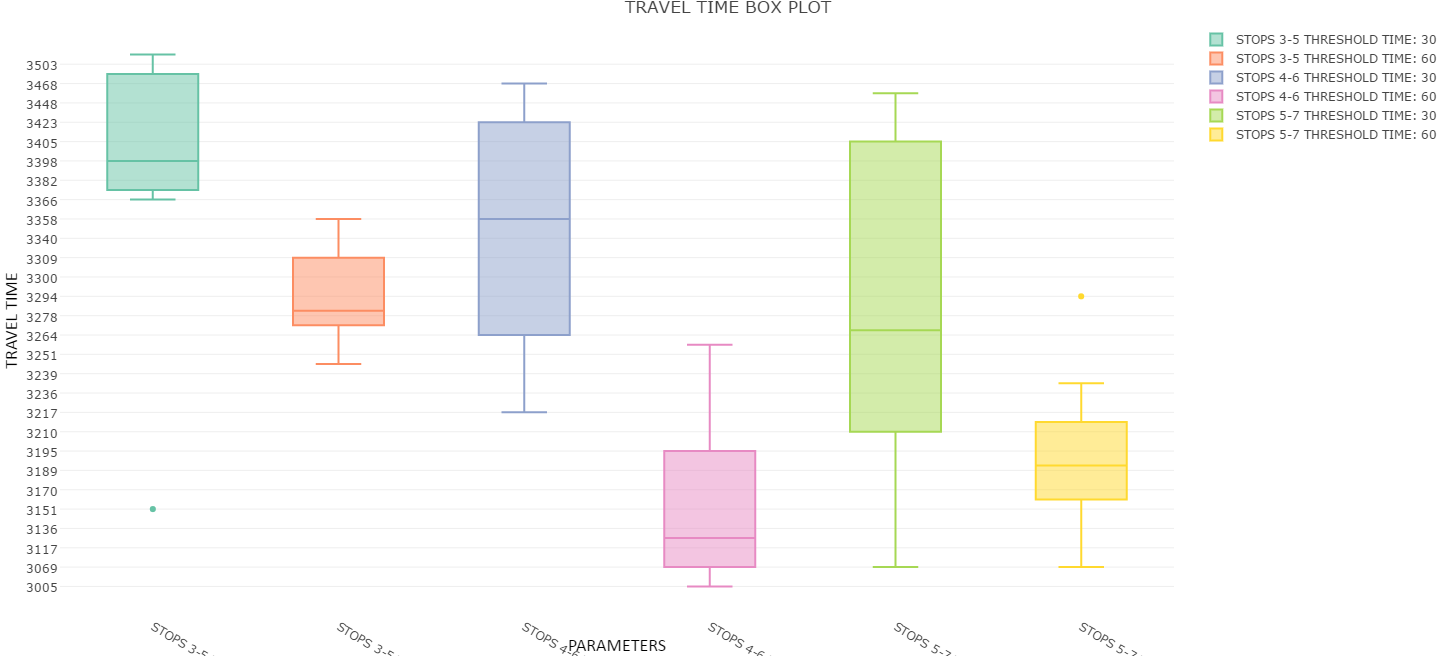
## Genetic Algorithm with Depot’s Nearest Neighbors Model



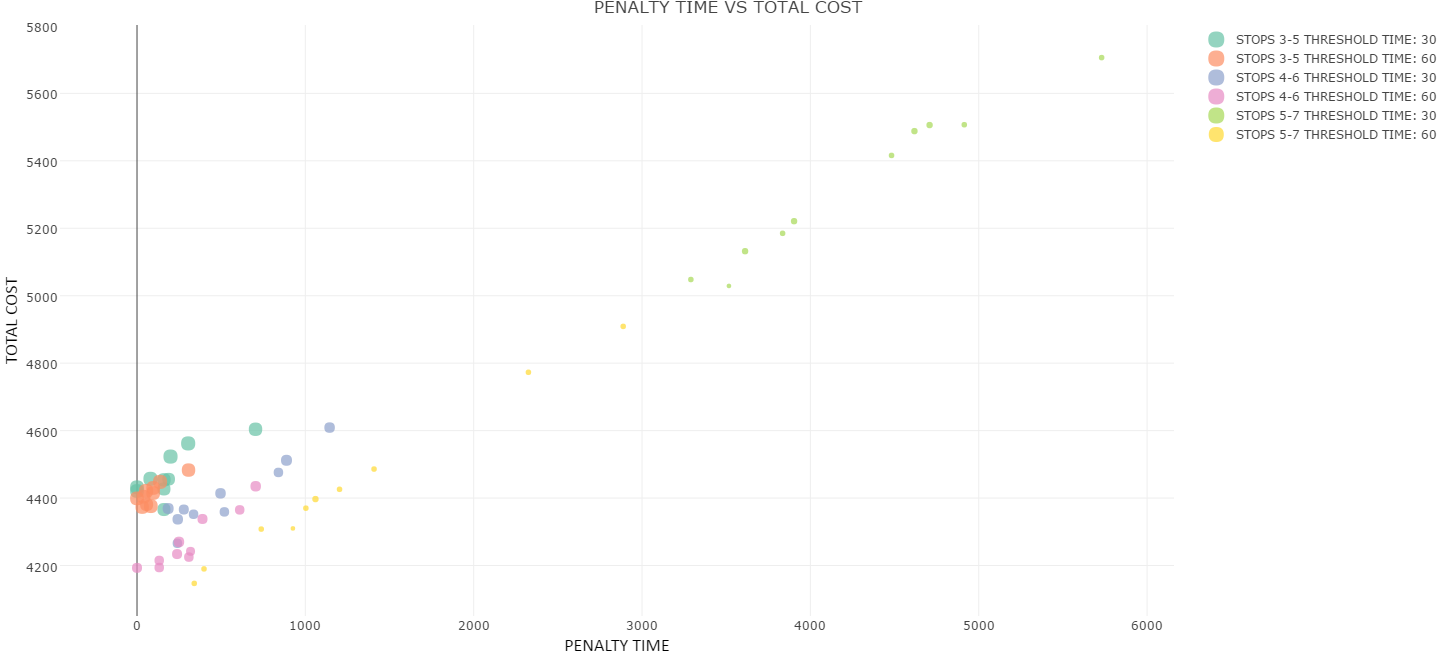
As seen in the above plot, for parameter values Stops ranging 3 to 5 and max early and late time arrival 30 minutes, number of vehicles are between 48 and 52, stops ranging 3 to 5 and max early and late time arrival 60 minutes, number of vehicles are between 49 and 52, stops ranging 4 to 6 and max early and late time arrival 30 minutes, number of vehicles are between 41 and 44, stops ranging 4 to 6 and max early and late time arrival 60 minutes, number of vehicles are between 40 and 43, stops ranging 5 to 7 and max early and late time arrival 30 minutes, number of vehicles are between 34 and 36, stops ranging 5 to 7 and max early and late time arrival 60 minutes, number of vehicles are between 34 and 36.



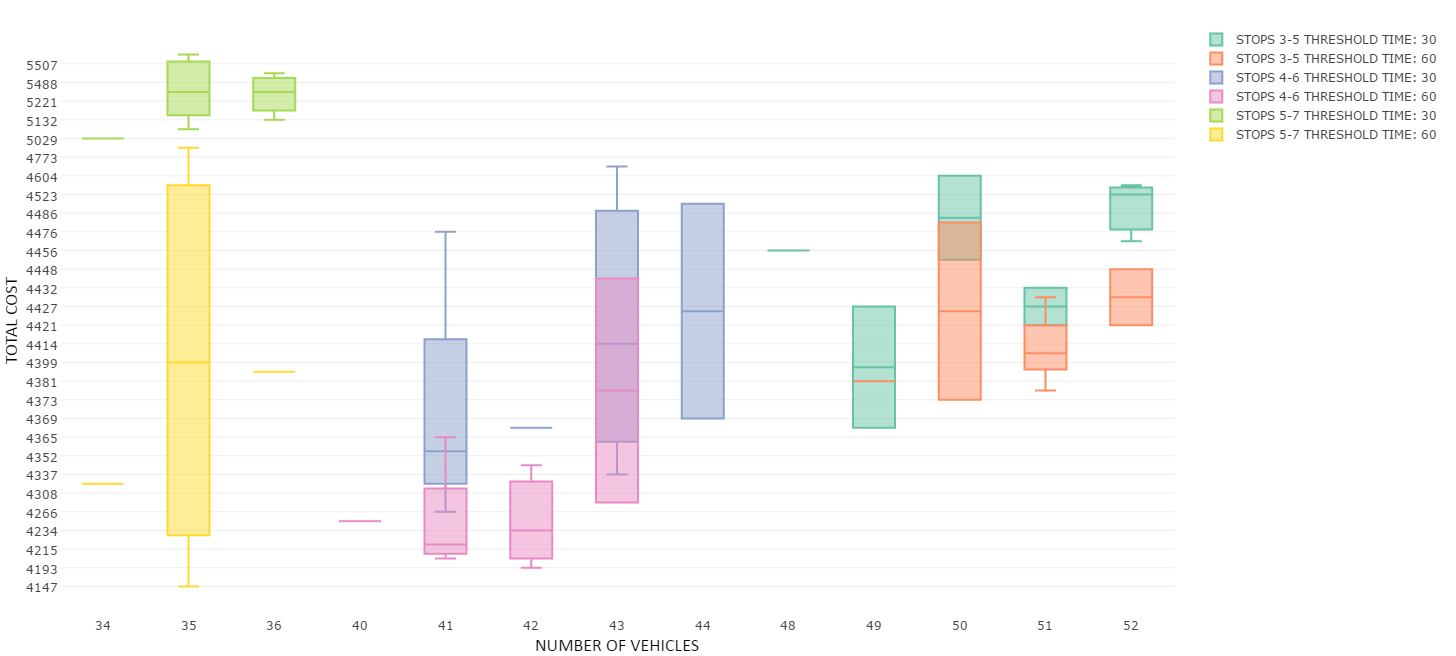
As seen in the above plot, for parameter values Stops ranging 3 to 5 and max early and late time arrival 30 minutes, total time is between 14306 and 15803, stops ranging 3 to 5 and max early and late time arrival 60 minutes, total time is between 14190 and 14660, stops ranging 4 to 6 and max early and late time arrival 30 minutes, total time is between 14452 and 15537, stops ranging 4 to 6 and max early and late time arrival 60 minutes, total time is between 14131 and 14923, stops ranging 5 to 7 and max early and late time arrival 30 minutes, total time is between 17580 and 19904, stops ranging 5 to 7 and max early and late time arrival 60 minutes, total time is between 14403 and 17095.



As seen in the above plot, for parameter values Stops ranging 3 to 5 and max early and late time arrival 30 minutes, travel time is between 3366 and 3513, stops ranging 3 to 5 and max early and late time arrival 60 minutes, travel time is between 3244 and 3358, stops ranging 4 to 6 and max early and late time arrival 30 minutes, travel time is between 3217 and 3468, stops ranging 4 to 6 and max early and late time arrival 60 minutes, travel time is between 3005 and 3258, stops ranging 5 to 7 and max early and late time arrival 30 minutes, travel time is between 3069 and 3462, stops ranging 5 to 7 and max early and late time arrival 60 minutes, travel time is between 3069 and 3294.

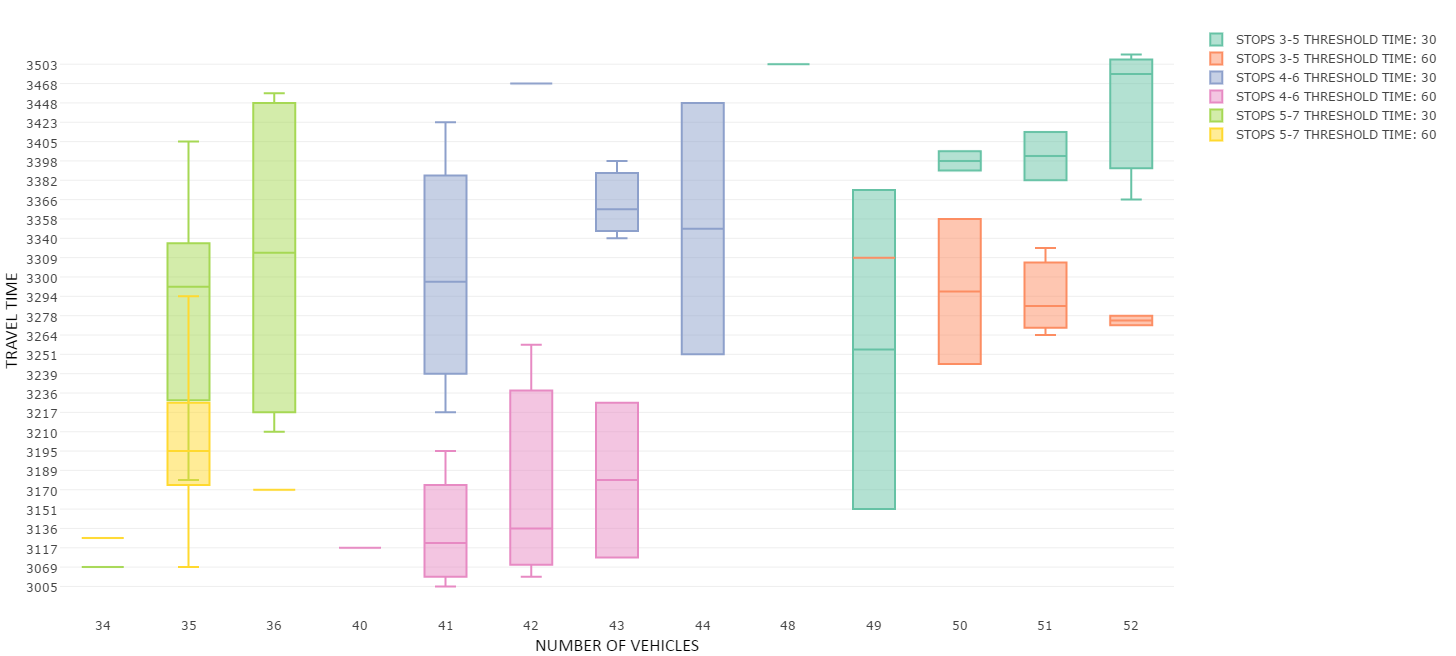


As seen in the above plot, penalty time is more for stops range 5 to 7 because there are more stops to cover for each route, late time penalty and total route time penalty is more whereas penalty is less for stops range 3 to 5, because there are less number of stops in each route in comparison to other stops ranges. As penalty increases, total cost also increases.



As seen in the above plot, as number of vehicles increases, total cost also increases.

As seen in the above plot, penalty time has more important role in adding costs. For 35 vehicles, total cost shoot up to 5507. Penalty time is huge for 35 vehicles.



As seen in the above plot, travel time is in between 3000 and 3500 for all set of parameters. If number of vehicles are low, then total travel time by the vehicles is also low. For stops range 4 to 6, travel time is also low.

# CONCLUSION

In GA Model if we use DNN generated Population, routes are further optimized for all set of parameters.

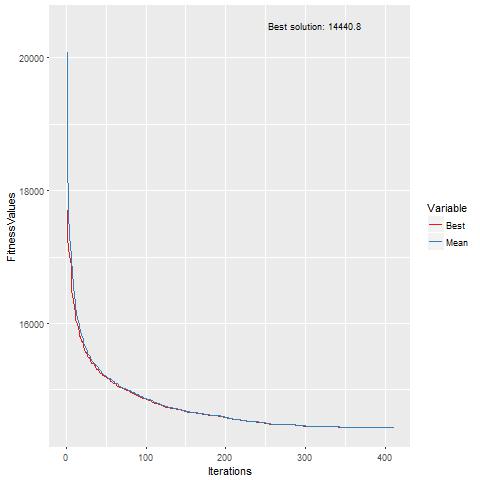
This model gives most optimal solution for stops range 4 to 6. Here the penalty on time is also minimum most of the time. Though we see an overhead of 10 - 15% on number of vehicles as compared to the solution for stops range 5 to 7. But solution for stops range 5 to 7 has high penalty on time which renders it not suitable if customer satisfaction is to be taken care of.

There is one more solution for stop range 3 to 5 which looks even better if we value the minimum time penalty but number of vehicles are increased and hence it has increased resources cost.

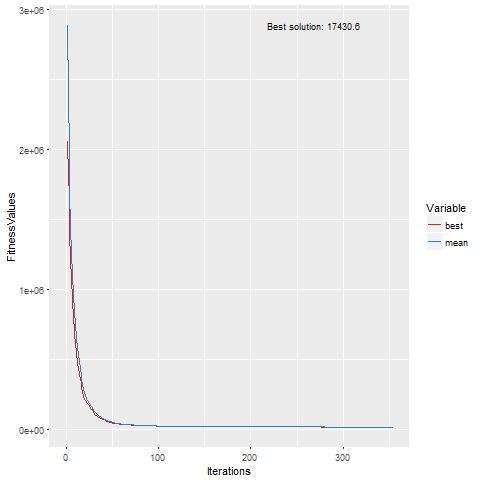
# APPENDIX

## Iterations vs Fitness Value Plot

### GA Model using DNN Generated Initial Population



### GA Model using Random Initial Population



## Best Solution for Stops Range: 4 to 6 and Time window range: -60 to +60 minutes

Solution for Stops Range: 4 to 6 -60 min ‑



## R Code

Attaching the R Code, MySQL Dump.



|  |
| --- |
| rm(list = ls())  #setting working directory  setwd("C:/Insofe/Project/test")    #loading necessary libraries for the project  library(RMySQL)  library(animation)  library(igraph)  library(networkD3)  library(magrittr)  library(ggplot2)  library(reshape2)  library(plotly)  library(htmlwidgets)    #turning warnings off  options(warn=-1)    #connecting to mysql and get the data  db\_details = c("gupas05-e7440", "root", "ashirvad", "spring\_clean")  connection = dbConnect(  MySQL(),  user = db\_details[2],  password = db\_details[3],  dbname = db\_details[4],  host = db\_details[1]  )    #Listing the tables in the database  dbListTables(connection)    #readiing tables  traveltime\_info = dbReadTable(connection, "travel\_time\_matrix\_db")  stops\_info = dbReadTable(connection, "stops\_info\_db")    #fnGetParameters will return constraints values after fetching from mysql tables  fnGetParameters = function()  {  connection = dbConnect(  MySQL(),  user = db\_details[2],  password = db\_details[3],  dbname = db\_details[4],  host = db\_details[1]  )  parameters = dbReadTable(connection, "parameters\_info\_db")  parameters2 = dbGetQuery(  connection,  "SELECT truncate(time\_to\_sec(TIMEDIFF(EXPECTED\_NOT\_AFTER,EXPECTED\_NOT\_BEFORE))/60,0) as routetimewindow FROM spring\_clean.stops\_info\_db;"  )  stops\_rs = dbGetQuery(  connection,  "select sum(TIME\_TO\_COMPLETE\_WORK) as sum\_stops\_execution\_time,count(STOP\_ID) as number\_stops from stops\_info\_db where STOP\_ID like 'STP%'"  )  sum\_stops\_execution\_time = as.numeric(stops\_rs[1])  number\_stops = as.numeric(stops\_rs[2])  uniqueroutetimewindow = unique(parameters2)  parameters = data.frame(parameters,  (uniqueroutetimewindow),  sum\_stops\_execution\_time,  number\_stops)  return(parameters)  }    attach(traveltime\_info)    #checking structure of data: traveltime\_info and stops\_info  head(traveltime\_info)  str(traveltime\_info)  str(stops\_info)  head(stops\_info)  summary(stops\_info)    #total execution time is 10995  sum(stops\_info$TIME\_TO\_COMPLETE\_WORK)  #checking summary of traveltime\_info  summary(traveltime\_info)    #no of unique stops  length(unique(FROM\_STOP\_ID))  length(unique(TO\_STOP\_ID))      #check missing values in the data  ##finding the missing stops  missingstops = unique(TO\_STOP\_ID)[!(unique(TO\_STOP\_ID)) %in% (unique(FROM\_STOP\_ID))]    #imputing the missing stop values with reverse lookup  missingstops\_type1 = traveltime\_info[which(traveltime\_info$TO\_STOP\_ID %in% missingstops), ]  colnames(missingstops\_type1) = c("TO\_STOP\_ID", "FROM\_STOP\_ID", "TRAVEL\_TIME")  missingstops\_type1 = cbind(missingstops\_type1[, 2], missingstops\_type1[, c(1, 3)])  colnames(missingstops\_type1)[1] = c("FROM\_STOP\_ID")  nrow(missingstops\_type1)  missingstops\_na = data.frame(  FROM\_STOP\_ID = rep(missingstops, times = length(missingstops)),  TO\_STOP\_ID = rep(missingstops, each = length(missingstops)),  TRAVEL\_TIME = NA)  missingstops\_na$TRAVEL\_TIME[missingstops\_na$FROM\_STOP\_ID == missingstops\_na$TO\_STOP\_ID] =0  finalstops = rbind(traveltime\_info, missingstops\_type1, missingstops\_na)  finalstops = finalstops[!finalstops$FROM\_STOP\_ID == finalstops$TO\_STOP\_ID, ]  summary(missingstops\_na)  head(finalstops)  summary(finalstops)    #no of missing values=72  sum(is.na(finalstops))  #Rows with missing values  subset(finalstops, is.na(finalstops$TRAVEL\_TIME))    #funtion to fill missing values with nearest neighbor  fNFindMissingValues = function(f) {  missing\_stops = subset(f, is.na(f$TRAVEL\_TIME))  if (nrow(missing\_stops) > 0) {  for (i in 1:nrow(missing\_stops))  {  a = f[f$FROM\_STOP\_ID == as.character(missing\_stops[i, ][1]), ]  b = a[order(a$TRAVEL\_TIME), ][1, ]  c = f[f$FROM\_STOP\_ID == as.character(missing\_stops[i, ][2]) &  f$TO\_STOP\_ID == b$TO\_STOP\_ID, ]  d = f[f$FROM\_STOP\_ID == as.character(missing\_stops[i, ][2]), ]  e = d[order(d$TRAVEL\_TIME), ][1, ]  g = f[f$FROM\_STOP\_ID == as.character(missing\_stops[i, ][1]) &  f$TO\_STOP\_ID == e$TO\_STOP\_ID, ]  t = round((  b$TRAVEL\_TIME + g$TRAVEL\_TIME + e$TRAVEL\_TIME + c$TRAVEL\_TIME  ) / 2)  f[f$FROM\_STOP\_ID == as.character(missing\_stops[i, ][1]) &  f$TO\_STOP\_ID == as.character(missing\_stops[i, ][2]), ]$TRAVEL\_TIME = t  }  }  return(f)  }    #filling missing values  finalstops = fNFindMissingValues(finalstops)  sum(is.na(finalstops))    #converting it to distance matrix for more efficient lookup  traveltime\_matrix = acast(data = finalstops, FROM\_STOP\_ID ~ TO\_STOP\_ID, fill = 0)  View(traveltime\_matrix)    sum(is.na((traveltime\_matrix)))  dim(traveltime\_matrix)  rm(missingstops, missingstops\_na, missingstops\_type1)  #Now distance matrix has no missing values      #function to output stops and depots based on type  #if type=stops, then it will return stops otherwise depots  calcdepostops <- function(stopid, type) {  depots = vector()  stops = vector()  for (i in unique(stopid)) {  if (startsWith(i, "DEP"))  depots = c(depots, i)  else  stops = c(stops, i)  }  if (type == "stops")  return(stops)  else  return(depots)  }    #getting stops  servicestops = calcdepostops(stops\_info$STOP\_ID, type = "stops")    #getting depots  depots = calcdepostops(stops\_info$STOP\_ID, type = "depots")    summary(t(traveltime\_matrix[depots, servicestops]))    #function to find minimum values in a vector and randomly return one of the index if more than one values or minimum  fNMinimum = function(x) {  minimum = which(x == min(x))  mins = sample(length(minimum), 1)  return(minimum[mins])  }    #funtction to get nearest stops per depots  #if the distance is from one stop to two or more depots, then it will randomly assign the depots among those depots  #if names ==TRUE then it will return list of list of stops with names as depots, otherwise it will simply return list of slist of stops nearer to depots  fNnearestdepots = function(traveltime\_matrix,  depots,  servicestops,  name = TRUE) {  depotstopsdf = as.data.frame(t(traveltime\_matrix[depots, servicestops]))  nearestdepots = list(names(depotstopsdf)[apply(depotstopsdf, 1, fNMinimum)], rownames(depotstopsdf))  listneareststops = list()  for (i in 1:length(depots))  {  an = list(servicestops[which(nearestdepots[[1]] == depots[i])])  listneareststops = c(listneareststops, an)  }  if (name == TRUE)  names(listneareststops) = depots  return(listneareststops)  }    #function to find probability of selection of depots based upon number of stops near to a depot  fnProbabilityDepots = function(listneareststops, depots, middlestops) {  probdepots = vector()  for (j in 1:length(depots))  {  probdepots = c(probdepots,  length(listneareststops[[j]]) / length(middlestops))  }  return(probdepots)  }    #function to get job service time in a particular stop  fNGetServiceTime = function(stp) {  return(stops\_info[stops\_info$STOP\_ID == stp, c("TIME\_TO\_COMPLETE\_WORK")])  }    #function to get a dataframe which is a subset of traveltime\_matrix containing information about those depots and list of stops which are passed to it in arguments  fNGetAllTime <- function(depotname, stopsvector) {  listvec = c(depotname, stopsvector)  #ls1=setdiff(names,ls)  t = traveltime\_matrix[rownames(traveltime\_matrix) %in% listvec, colnames(traveltime\_matrix) %in% listvec]  df = setNames(melt(t), c('FROM\_STOP\_ID', 'TO\_STOP\_ID', 'TRAVEL\_TIME'))  df = df[df$FROM\_STOP\_ID != df$TO\_STOP\_ID,]  return(df)  }    #function will take min and max and total as arguments and return a vector which contains randomly generated elements between min and max arguments which sums upto total value  #last element can take any value which is left in last  fNrandomVector = function(min , max , total)  {  ran\_vec = vector()  while (total > 0) {  temp = sample(min:max, 1, replace = TRUE)  if (total - temp > 0)  ran\_vec = c(ran\_vec, temp)  else  ran\_vec = c(ran\_vec, total)  total = total - temp  }  return(ran\_vec)  }    #function which implemented nearest depots neighbour algorithm  #fNDepotsNearestNeighbor will take minimum and maximum stops as input  #this function will return routes based upon stops vicinity with respect to depots  fNDepotsNearestNeighbor = function(minstop, maxstop) {  listneareststops = fNnearestdepots(traveltime\_matrix, depots, servicestops, name = FALSE)  listneareststopswithnames = fNnearestdepots(traveltime\_matrix, depots, servicestops)  routes = list()  depnames = names(listneareststopswithnames)  vehicles = 0  for (i in 1:length(listneareststops))  {  au = vector()  routesize = fNrandomVector(minstop, maxstop, length(listneareststops[[i]]))  vehiclefleetsize = length(routesize)  vehicles = vehicles + vehiclefleetsize  lookupvec = unlist(listneareststops[i])  depname = depnames[i]  atime = fNGetAllTime(depname, lookupvec)  EXECUTIONTIME = unlist(lapply(atime$TO\_STOP\_ID, fNGetServiceTime))  atime = cbind(atime, EXECUTIONTIME)  TOTALTIME = unlist(lapply(atime$TRAVEL\_TIME + atime$EXECUTIONTIME, sum))  atime = cbind(atime, TOTALTIME)  atime = atime[order(atime$TOTALTIME), ]  for (j in 1:vehiclefleetsize)  {  rt = vector()  rt[1] = depname  size = routesize[j] + 1  atime = subset(atime,!atime$TO\_STOP\_ID %in% depots)  counter = 1  while ((size) > length(rt))  {  #assigning stops to the route based upon minimum distance from that stop  atime = subset(atime,!atime$TO\_STOP\_ID %in% au)  nrow(atime)  rte = as.character(atime[atime$FROM\_STOP\_ID == rt[counter],][1,]$TO\_STOP\_ID)  au = c(au, rte)  rt = append(rt, rte)  counter = counter + 1  }  rt = append(rt, depnames[i])  #adding route to list of routes  routes = append(routes, list(rt))  }  }  return(routes)  }    #function will generate initial population of set of chromosome based upon nearest depot neighbors algorithm  #it will take minstop,maxstop and initialPopulationSize as arguments  fNGenerateInitalNN = function(minstop, maxstop, initialPopulationSize) {  poplist = list()  for (i in 1:initialPopulationSize)  {  #calling fNDepotsNearestNeighbor to get one set of chromosomes/routes  routes = fNDepotsNearestNeighbor(minstop, maxstop)  poplist = c(poplist, list(routes))  }  return(poplist)  }      #function will take two timestamps as arguments and return diiference between them in format minute and seconds  fNMinutesSeconds = function(endTime, startTime) {  dif = as.numeric(difftime(endTime, startTime, units = 'min'))  return(paste(  sprintf('%02d', as.integer(dif)),  "minutes and"  ,  sprintf('%02.0f', (dif - as.integer(dif)) \* 60),  "seconds"  ))  }    #function will take minstop,maxstop and initial Population size as arguments  #generate initial population randomly  fnGenerateInitPop <- function(initPopSize, minstop, maxstop) {  initPopulation = list()  seed = Sys.time()  set.seed(seed)  print(seed)  seeds = sample(6000:7000, initPopSize, replace = FALSE)  listneareststops = fNnearestdepots(traveltime\_matrix, depots, servicestops)  probdepots = fnProbabilityDepots(listneareststops, depots, servicestops)  for (i in 1:initPopSize) {  set.seed(seeds[i])  range\_min = minstop  range\_max = maxstop  chromosome\_stopssize = fNrandomVector(range\_min, range\_max, length(servicestops))  vehiclefleetsize = length(chromosome\_stopssize)  chromosome\_depo = sample(1:length(depots),  vehiclefleetsize,  replace = TRUE,  prob = probdepots)  chromosome\_stops = sample(1:length(servicestops), replace = FALSE)  solution = list()  chromosomes\_count = 0  for (j in 1:vehiclefleetsize)  {  chromosome = vector()  #creating one route/chromosome  chromosome = ifelse(  chromosome\_stopssize[j] > 0,  c(depots[chromosome\_depo[j]], servicestops[chromosome\_stops[chromosomes\_count +  1:chromosome\_stopssize[j]]], depots[chromosome\_depo[j]]),  c(depots[chromosome\_depo[j]], depots[chromosome\_depo[j]])  )  if (chromosome\_stopssize[j] > 0)  chromosome = c(depots[chromosome\_depo[j]], servicestops[chromosome\_stops[chromosomes\_count +  1:chromosome\_stopssize[j]]], depots[chromosome\_depo[j]])  else  chromosome = c(depots[chromosome\_depo[j]], depots[chromosome\_depo[j]])  chromosomes\_count = chromosomes\_count + chromosome\_stopssize[j]  #adding one route to set of routes  solution = c(solution, list(chromosome))  }  #adding one set of chromosomes/routes to population list  initPopulation = c(initPopulation, list(solution))  }  return(initPopulation)  }    #function will take time travelled in a route and maximum route time as input and return extra time window penalty  fNExtraTimeWindowPenalty = function(timeTravelled, maxroutetime) {  temp = timeTravelled - maxroutetime  exceededminpenalty = 0  routepenalty = 0  #if timetravelled is less than maximum route time then penalty is 0  if (temp < 0)  return(0)    for (i in 1:ceiling(temp / 60)) {  if (temp > 60)  exceededminpenalty = exceededminpenalty + 60 \* i  if (temp > 0 & temp <= 60)  exceededminpenalty = exceededminpenalty + temp \* i  routepenalty = routepenalty + (timeTravelled \* i \* 0.1)  temp = temp - 60  }  return(exceededminpenalty + routepenalty)  }      #function will take route length, total route time,minstop and maxstop and return extra/less stops penalty  fNExtraStopPenalty = function(len, sums, minstop, maxstop) {  #penalty for less/extra stops  extraStopPenalty = ifelse((len - 2 - minstop) < 0,  0.1 \* abs(len - 2 - minstop) \* sums,  ifelse((maxstop - len + 2) < 0, 0.1 \* abs(maxstop - len + 2) \* sums, 0))  return(extraStopPenalty)  }    #function will take route, threshold and route time window for serving a stop and return late time or early arrival penalty  fNLateTimeArrivalPenalty = function(route, permissibletime, routetimewindow) {  twpenalty = 0  routetime = 0  if (length(route) > 2) {  for (j in 2:(length(route) - 2))  ##Travel time per route calculation  {  routetime = routetime + traveltime\_matrix[route[[j]], route[[j + 1]]] + stops\_info[stops\_info$STOP\_ID == route[[j]], 2]  penalty3 = fNExtraTimeWindowPenalty(routetime, 2 \* permissibletime + routetimewindow)  twpenalty = twpenalty + penalty3  }  }  return(twpenalty)  }    #We define the Objective function as follows.  #function will take chromosome/set of chromosomes, minstop,maxstop,permissible time,max route time, route time window as input  #optional arguments are penalty,service and travel  #function will return objective function fitness value if optional arguments are not given  #function will return only penalty time if penalty =TRUE and other are set as FALSE  #function will return only service time if service =TRUE and other are set as FALSE  #function will return only travel time if travel =TRUE and other are set as FALSE  fnEvaluate <-  function(chromosome,  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow,  penalty = TRUE,  service = TRUE,  travel = TRUE) {  sumtotal = 0  routetime = 0  for (i in chromosome)  {  latearrivalpenalty = 0  stp = strsplit(i, split = " ")  sums = 0  #calculating route time  for (j in 1:(length(stp) - 1))  {  if (service)  sums = sums + stops\_info[stops\_info$STOP\_ID == stp[[j]], 2]  if (travel)  sums = sums + traveltime\_matrix[stp[[j]], stp[[j + 1]]]  }  #calculating penalties  extratimepenalty = fNExtraTimeWindowPenalty(sums, maxroutetime)  extrastoppenalty = fNExtraStopPenalty(length(stp), sums, minstop, maxstop)  latearrivalpenalty = fNLateTimeArrivalPenalty(stp, permissibletime, routetimewindow)  #print(paste("sum is",sums," penalty1 ",extratimepenalty," penalty2 ",extrastoppenalty," penalty3 ",latearrivalpenalty," k is ",length(stp)-2))  if (penalty)  sums = sums + extratimepenalty + extrastoppenalty + latearrivalpenalty  sumtotal = sumtotal + sums  }  return(sumtotal)  }    #mutate function will take a route,minstop,maxstop,permissible time,max route time, route time window as input  #function will return a route after random swapping if its fitness value is better than input  #otherwise it will return input route  mutate <- function(individual,  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow) {  #Interchange the values of two of the attributes  tempindividual = individual  individualfitness = fnEvaluate(  list(individual),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow  )  if (length(individual) > 3) {  indlen = length(individual) - 1  rnd = sample(seq(2, indlen, 1), size = 2, replace = FALSE)  temp = individual[rnd[1]]  individual[rnd[1]] = individual[rnd[2]]  individual[rnd[2]] = temp  }    individualfitness\_mut = fnEvaluate(  list(individual),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow  )  #checking if mutated chromosome is better than input chromosome  if (individualfitness > individualfitness\_mut)  return(individual)  else  return(tempindividual)  }  #crossover function will take two routes,minstop,maxstop,permissible time,max route time, route time window as input  #function will return two routes after swapping/cutting/adding the stops in between two routes if sum of fitness values of transformed routes is better than sum of fitness values of input routes  crossOver = function(p1,  p2,  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow) {  p1fitness = fnEvaluate(list(p1),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow)  p2fitness = fnEvaluate(list(p2),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow)    tempp1 = p1  tempp2 = p2  p1len = length(p1) - 1  p2len = length(p2) - 1  bool = length(p1) > length(p2)  if (length(p1) == length(p2)) {  swap = floor(length(p1) \* 0.5)  if (p1len != 2)  pos = sample(2:p1len, swap, replace = F)  else  pos = 2  for (l in pos) {  v = p2[l]  k = p1[l]  p1[[l]] = v  p2[[l]] = k  }  }  else{  if (bool) {  if (p2len != 2)  sp = sample(2:p2len, size = 1, replace = F)  else  sp = 2  }  else{  if (p1len != 2)  sp = sample(2:p1len, size = 1, replace = F)  else  sp = 2  }  if (length(p1) == 3)  {  p1temp = vector()  p2temp = c(p2[1:(sp)], p1[sp], p2[(sp + 1):length(p2)])  }  else if (length(p2) == 3)  {  p1temp = c(p1[1:(sp)], p2[sp], p1[(sp + 1):length(p1)])  p2temp = vector()  }  if (length(p1) != 3 & length(p1) < (minstop + 2))  {  p1temp = c(p1[1:sp], p2[sp], p1[(sp + 1):length(p1)])  p2temp = c(p2[1:(sp - 1)], p2[(sp + 1):length(p2)])  }  else if (length(p2) != 3 & length(p2) < (minstop + 2))  {  p1temp = c(p1[1:(sp - 1)], p1[(sp + 1):length(p1)])  p2temp = c(p2[1:sp], p1[sp], p2[(sp + 1):length(p2)])  } else if (length(p2) > (maxstop + 2))  {  p1temp = c(p1[1:sp], p2[sp], p1[(sp + 1):length(p1)])  p2temp = c(p2[1:(sp - 1)], p2[(sp + 1):length(p2)])  }  else if (length(p1) > (maxstop + 2))  {  p1temp = c(p1[1:(sp - 1)], p1[(sp + 1):length(p1)])  p2temp = c(p2[1:sp], p1[sp], p2[(sp + 1):length(p2)])  }  else{  p1temp = c(p1[1:(sp - 1)], p2[sp:(length(p2) - 1)], p1[1])  p2temp = c(p2[1:(sp - 1)], p1[sp:(length(p1) - 1)], p2[1])  }  p1 = p1temp  p2 = p2temp  #print(p1temp)  #print(p2temp)  }  p1tempfitness = fnEvaluate(list(p1),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow)  p2tempfitness = fnEvaluate(list(p2),  minstop,  maxstop,  permissibletime,  maxroutetime,  routetimewindow)  #checking if childrens are better than parents  if ((p1fitness + p2fitness) > (p1tempfitness + p2tempfitness))  return(list(p1, p2))  else  return(list(tempp1, tempp2))  }    #plotGraph function will take fitnessVector and meanFitnessVector as input  #function will plot a graph of fitness value vs iterations/generations  plotGraph <- function(fitnessVector, meanfitnessVector) {  i = length(fitnessVector)  temp <-  data.frame(  Iterations = c(seq(1, i), seq(1, i)),  Variable = c(rep("Mean", i), rep("Best", i)),  FitnessValues = c(meanfitnessVector, fitnessVector)  )  pl <-  ggplot(temp,  aes(  x = Iterations,  y = FitnessValues,  group = Variable,  colour = Variable  )) + geom\_line() + scale\_x\_continuous(limits = c(0, length(fitnessVector)))  p2 = pl + annotate(  "text",  x = i \* 0.9,  y = max(temp$FitnessValues) + i,  hjust = 1,  size = 3,  color = "black",  label = paste("Best solution:", min(temp$FitnessValues))  ) + scale\_colour\_brewer(palette = "Set1")  print(p2)  }    #animatePlot function will take fitnessVector and meanFitnessVector as input  #function will plot a animated graph of fitness value vs iterations/generations  animatePlot <- function(fitnessVector, meanfitnessVector) {  for (i in seq(1, length(fitnessVector))) {  temp <-  data.frame(  Iterations = c(seq(1, i), seq(1, i)),  Variable = c(rep("Mean", i), rep("Best", i)),  FitnessValues = c(meanfitnessVector[1:i], fitnessVector[1:i])  )  pl <-  ggplot(temp,  aes(  x = Iterations,  y = FitnessValues,  colour = Variable  )) + geom\_line() + scale\_x\_continuous(limits = c(0, length(fitnessVector)))  p2 = pl + annotate(  "text",  x = i \* 0.9,  y = max(temp$FitnessValues) + i,  hjust = 1,  size = 3,  color = "black",  label = paste("Best solution:", min(temp$FitnessValues))  ) + scale\_colour\_brewer(palette = "Set1")  print(p2)  }  }    #fnRunGeneticAlgo will take initPopulation, mutstartProbability,elitePercent, maxIterations,minstop,maxstop,permissibleTime,maxroutetime,routetimewindow,stopcriteria,imageName as input  #function will run genetic algorithm and return best route after stopping criteria meets or max iterations.  fnRunGeneticAlgo <-  function(initPopulation,  mutstartProbability,  elitePercent,  maxIterations,  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow,  stopcriteria,  imageName) {  swf = FALSE  counter = 0 # is used for stopping criteria  Plist = list()  fitnessVector = vector()  meanfitnessVector = vector()  cat("max iterations =", maxIterations, "\n")  # How many winners from each generation?  origPopSize = length(initPopulation)  topElite = round(elitePercent \* origPopSize, 0)  fitN = mapply(  fnEvaluate,  initPopulation,  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow  )  initPopulation = list(initPopulation, fitN)  sort.fitN = sort.list(fitN)  sorted.fitN = fitN[sort.fitN]  initPopulation = initPopulation[[1]][sort.fitN]  initPopulation = list(initPopulation, sorted.fitN)  # Main loop  NewPopulation = initPopulation  for (i in 1:maxIterations) {  ElitePop = list(NewPopulation[[1]][1:topElite], NewPopulation[[2]][1:topElite])  NewPopulation = list(list())  if (i > 25)  mut = (mutstartProbability \* 2 \* (i %/% 5)) / i  else  mut = (mutstartProbability) / i  while (length(NewPopulation[[1]]) <= origPopSize)  {  # Mutation  if (runif(1, 0, 1) < mut) {  c = sample(1:topElite, 1)  premutsol = ElitePop[[1]][[c]]  vehiclefleetsize = length(premutsol)  muts = sample(1:vehiclefleetsize, 0.1\*vehiclefleetsize)  j = 1  while (j < length(muts))  {  premutsol[[muts[j]]] = mutate(  premutsol[[muts[j]]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow  )  j = j + 1  }  NewPopulation[[1]][[length(NewPopulation[[1]]) + 1]] = premutsol  }  else  {  #crossover  randomsample = sample(1:topElite, 1)  randomsol = ElitePop[[1]][[randomsample]]  randomsolobjective = initPopulation[[2]][[randomsample]]  vehiclefleetsize = length(randomsol)  crossoverroutes = sample(1:vehiclefleetsize, 0.2\*vehiclefleetsize)  if (length(crossoverroutes) %% 2 != 0)  crossoverroutes = crossoverroutes[-c(length(crossoverroutes))]  l = 1  while (l < length(crossoverroutes))  {  crosssol = crossOver(  randomsol[[crossoverroutes[l]]],  randomsol[[crossoverroutes[l + 1]]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow  )  if (length(crosssol[[1]]) == 2)  {  randomsol[[crossoverroutes[l]]] = NULL  randomsol[[crossoverroutes[l + 1]]] = crosssol[[2]]  }  else if (length(crosssol[[2]]) == 2)  {  randomsol[[crossoverroutes[l]]] = crosssol[[1]]  randomsol[[crossoverroutes[l + 1]]] = NULL  }  else  {  randomsol[[crossoverroutes[l]]] = crosssol[[1]]  randomsol[[crossoverroutes[l + 1]]] = crosssol[[2]]  }  l = l + 2  }  NewPopulation[[1]][[length(NewPopulation[[1]]) + 1]] = randomsol  }  }  fitN = mapply(  fnEvaluate,  NewPopulation[[1]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow  )  NewPopulation = list(NewPopulation[[1]], fitN)  sort.fitN = sort.list(fitN)  sorted.fitN = fitN[sort.fitN]  NewPopulation = NewPopulation[[1]][sort.fitN]  NewPopulation = list(NewPopulation, sorted.fitN)  # stopping criteria  if (i == 1 | i %% 25 == 0)  cat(  "Iteration:",  i,  " New Population Fitness: ",  as.numeric(NewPopulation[[2]][[1]]),  " vehicles: ",  vehiclefleetsize,  " counter is: ",  counter + 1,  "\n"  )  fitnessVector = c(fitnessVector, NewPopulation[[2]][[1]])  meanfitnessVector = c(meanfitnessVector, mean(NewPopulation[[2]]))  # stopping criteria  if (i > 1) {  if (NewPopulation[[2]][[1]] == Plist[[length(Plist)]][1]) {  counter = counter + 1  if (counter == stopcriteria) {  break()  }  } else{  counter = 0  }  }  #adding population to Plist for comparision  Plist = c(Plist, NewPopulation)  }  jpeg(file = imageName)  plotGraph(fitnessVector, meanfitnessVector)  dev.off()  #animated plot  if (swf == TRUE)  {  plot = animate\_plot(fitnessVector, meanfitnessVector)  saveSWF(  expr = animate\_plot(fitnessVector, meanfitnessVector),  interval = 0.5,  swf.name = paste0(imageName,".swf"),  loop = 1,  out.dir = getwd()  )  }  # Print current best score  #calculating travel time, penalty time and service time  traveltime = fnEvaluate(  NewPopulation[[1]][[1]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow,  penalty = FALSE,  service = FALSE  )  servicetime = fnEvaluate(  NewPopulation[[1]][[1]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow,  penalty = FALSE,  travel = FALSE  )  penaltytime = fnEvaluate(  NewPopulation[[1]][[1]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow,  penalty = TRUE  ) - traveltime - servicetime  num\_vehicles = length(NewPopulation[[1]][[1]])  print(paste(  "TOTAL TIME FOR THE SOLUTION IS:",  fnEvaluate(  NewPopulation[[1]][[1]],  minstop,  maxstop,  permissibleTime,  maxroutetime,  routetimewindow  )  ))  print(  paste(  "TRAVEL TIME FOR THE SOLUTION IS:",  traveltime,  "SERVICE TIME FOR THE SOLUTION IS: ",  servicetime,  "PENALTY TIME FOR THE SOLUTION IS: ",  penaltytime,  " TOTAL STOPS COVERED: ",  length(unique(unlist(NewPopulation[[1]][[1]]))) - 6  )  )  #final solution after GA  final\_list = list(  NewPopulation[[1]][[1]],  NewPopulation[[2]][[1]],  traveltime,  servicetime,  penaltytime,  num\_vehicles  )  names(final\_list) = c(  "routes",  "totaltime",  "traveltime",  "servicetime",  "penaltytime",  "numberOfVehicles"  )  return(final\_list)  }      #closeMySQLconnections will close existing open mysql connections  closeMySQLconnections = function() {  dbListConnections(dbDriver(drv = "MySQL"))  lapply(dbListConnections(dbDriver(drv = "MySQL")), dbDisconnect)  }      #fnExecuteMain will take initial Population Size, Mutation Start Probability,  #Elite Percentage, Max no of iterations,stopping criteria,number of times as input  #random is an optional input  #if random is false then, initial population consists of solutionn generated by nearest depot neighbour algorithm  #otherwise, initial Population consists of randomly generated set of solutions  #function will return best routes for every set of parameter  #function will run for every parametes based upon number of times  #function will output set of routes for one solution in text file  #function also store the total fitness value for a soulution and text file path in mysql table results  fnExecuteMain <-  function(initPopulationSize,  mutstartProbability,  elitePercent,  maxIterations,  stopCriteria,  numberOfTimes,  random=FALSE,dnn=FALSE) {  closeMySQLconnections()  parameters = fnGetParameters()  solutionlist = list()  result\_df = as.data.frame(setNames(  replicate(7, numeric(0), simplify = T),  c(  "MAX\_STOPS\_PER\_ROUTE",  "MIN\_STOPS\_PER\_ROUTE",  "MAX\_ROUTE\_TIME",  "MAX\_EARLY\_TIMEWINDOW",  "MAX\_LATE\_TIMEWINDOW",  "OBJECTIVE\_VALUE",  "FILEPATH"  )  ))  for (p in 1:nrow(parameters))  {  minstop = parameters[p, 2]  maxstop = parameters[p, 1]  permissibleTime = parameters[p, 4]  maxroutetime = parameters[p, 3]  routetimewindow = parameters[p, 6]  for (j in 1:numberOfTimes)  {  old = Sys.time()  #dnn==FALSE, then GA models will execute  if(dnn==FALSE){  #random==TRUE, then initial population is randomly generated  if(random==FALSE)  initPopulation = fNGenerateInitalNN(  minstop = minstop,  maxstop = maxstop,  initialPopulation = initPopulationSize  )  else  initPopulation = fnGenerateInitPop(initPopulationSize, maxstop = maxstop, minstop =  minstop)  new = Sys.time()  print(paste(  "TIME TAKEN TO GENERATE INITIAL POPULATION:",  fNMinutesSeconds(new, old)  ))  print(  paste(  "minstop:",  minstop,  " maxstop:",  maxstop,  " loop:",  j,  " permissible time ",  permissibleTime  )  )  old = Sys.time()  imagename = paste0(minstop,  "\_",  maxstop,  "\_",  permissibleTime,  "\_",  j,  ".jpeg")  solution = fnRunGeneticAlgo(  initPopulation = initPopulation,  mutstartProbability = mutstartProbability,  elitePercent = elitePercent,  maxIterations = maxIterations,  minstop = minstop,  maxstop = maxstop,  permissibleTime = permissibleTime,  maxroutetime = maxroutetime,  routetimewindow = routetimewindow,  stopcriteria = stopCriteria,  imageName = imagename  )  }  else{  routes=fNNearestDepotsNeighbor(minstop,maxstop)  totaltime=fnEvaluate(routes,minstop,maxstop,permissibleTime,maxroutetime,routetimewindow)  traveltime=fnEvaluate(routes,minstop,maxstop,permissibleTime,maxroutetime,routetimewindow,penalty =F,service=F)  penaltytime=fnEvaluate(routes,minstop,maxstop,permissibleTime,maxroutetime,routetimewindow,travel=F,service=F)  servicetime=totaltime-traveltime-penaltytime  numberOfVehicles=length(routes)  solution = list(  routes,  totaltime,  traveltime,  servicetime,  penaltytime,  numberOfVehicles  )  names(solution) = c(  "routes",  "totaltime",  "traveltime",  "servicetime",  "penaltytime",  "numberOfVehicles"  )  }  new = Sys.time()  print(paste(  "TIME TAKEN TO GENERATE SOLUTION:",  fNMinutesSeconds(new, old)  ))  filename = paste0(  minstop,  "\_",  maxstop,  "\_",  permissibleTime,  "\_",  solution$totaltime,  "\_",  j,  ".txt"  )  lapply(solution$routes,  write,  filename,  append = TRUE,  ncolumns = 1000)  filepath = paste0(getwd(), "/", filename)  result\_df = rbind(  result\_df,  data.frame(  maxstop,  minstop,  maxroutetime,  permissibleTime,  permissibleTime,  solution$totaltime,  filepath  )  )  solutionlist = c(solutionlist, solution)  }  }  colnames(result\_df) <-  c(  "MAX\_STOPS\_PER\_ROUTE",  "MIN\_STOPS\_PER\_ROUTE",  "MAX\_ROUTE\_TIME",  "MAX\_EARLY\_TIMEWINDOW",  "MAX\_LATE\_TIMEWINDOW",  "OBJECTIVE\_VALUE",  "FILEPATH"  )  conn = dbConnect(  MySQL(),  user = db\_details[2],  password = db\_details[3],  dbname = db\_details[4],  host = db\_details[1]  )  #results are stored in mysql table results  dbWriteTable(  conn,  value = result\_df,  name = "results",  append = TRUE,  row.names = FALSE,  overwrite = FALSE  )  closeMySQLconnections()  return(solutionlist)  }      #executing fnExecuteMain with dnn=TRUE and storing the results in finalsolutionNN list  finalsolutionNN = fnExecuteMain(initPopulationSize = 100,mutstartProbability = 0.8,  elitePercent = 0.2,maxIterations = 1000,stopCriteria = 70,numberOfTimes = 10,dnn=TRUE)    #executing fnExecuteMain with random =FALSE and storing the results in finalsolution1 list  finalsolution1 = fnExecuteMain(initPopulationSize = 100,mutstartProbability = 0.8,  elitePercent = 0.2,maxIterations = 1000,stopCriteria = 70,numberOfTimes = 10)  #executing fnExecuteMain with random population and storing the results in finalsol list  finalsol = fnExecuteMain(initPopulationSize = 100,mutstartProbability = 0.8,  elitePercent = 0.2,maxIterations = 1000,stopCriteria = 70,numberOfTimes = 10,random = TRUE)      #fNDataFrame will be used by fNVisualization  fNDataFrame = function(x) {  dff <-  as.data.frame(setNames(  replicate(5, numeric(0), simplify = F),  c("src", "target", "value", "route", "val")  ))  counter = 1  cnt = 1  for (i in 1:(length(x) - 1)) {  if ((startsWith(x[i], "DEP") & startsWith(x[i + 1], "DEP"))) {  counter = counter + 1  cnt = 1  }  else  {  dff[i - counter + 1, 1] = x[i]  dff[i - counter + 1, 2] = x[i + 1]  dff[i - counter + 1, 3] = traveltime\_matrix[x[i], x[i + 1]]  dff[i - counter + 1, 4] = counter  dff[i - counter + 1, 5] = cnt  cnt = cnt + 1  }  }  return(dff)  }  #fNTravelMatrix will be used by fNVisualization  fNTravelMatrix = function(finalstops, x, traveltime\_matrix) {  k = length(finalstops)  tm = matrix(rep(0, k \* k), k, k)  rownames(tm) = colnames(tm) = finalstops  for (i in 1:(length(x) - 1))  {  if (!(startsWith(x[i], "DEP") & startsWith(x[i + 1], "DEP"))) {  if (traveltime\_matrix[x[i], x[i + 1]] == 0)  tm[x[i], x[i + 1]] = 1  else  tm[x[i], x[i + 1]] = traveltime\_matrix[x[i], x[i + 1]]  }  }  return(tm)  }  #fNRoutes will be used by fNVisualization  fNRoutes = function(nodes, df) {  grp = rep(1, nrow(nodes))  nodes = data.frame(nodes, grp)  for (i in 1:nrow(df)) {  if (startsWith(df[i, "target"], "ST"))  nodes[nodes$unique\_solvec == df[i, "target"],]$grp = df[i, "route"]  else{  nodes[nodes$unique\_solvec == df[i, "target"],]$grp = max(df[, "route"]) +  1  }  }  return(nodes)  }      #function will take finalsolution and names of nodes(depots/stops) as input  #function will create forced network graph for all the nodes in a solution in html files  fNVisualizations = function(finalsol, names) {  for (i in seq(1, length(finalsol), 6))  {  solvec = unlist(finalsol[i]$routes)  unique\_solvec = names  nodes = data.frame(unique\_solvec)  tm = fNTravelMatrix(names, solvec, traveltime\_matrix)  nd = fNDataFrame(solvec)  nodes = fNRoutes(nodes, nd)  g <- graph.adjacency(tm, weighted = T)  sN = simpleNetwork(  nd,  linkColour = "black",  nodeColour = "green",  opacity = 2,  zoom = TRUE  )  filename\_sn = paste0(  "simpleNw\_",  round(finalsol[i + 1]$totaltime),  "\_",  round(finalsol[i + 5]$numberOfVehicles),  "\_",  i,  ".html"  )  filename\_fn = paste0(  "forcedNw\_",  round(finalsol[i + 1]$totaltime),  "\_",  round(finalsol[i + 5]$numberOfVehicles),  "\_",  i,  ".html"  )  #sN%>% saveNetwork(file = filename\_sn)  graphD3 <- igraph\_to\_networkD3(g, group = nodes)  val = rep(1, nrow(graphD3$links))  graphD3$links = data.frame(graphD3$links, val)  # Create force directed network plot  forcenet = forceNetwork(  Links = graphD3$links,  Nodes = graphD3$nodes,  Source = 'target',  Target = 'source',  Value = "val",  NodeID = 'name',  Group = 'grp',  opacity = 4,  linkColour = "#000",  zoom = TRUE,  opacityNoHover = TRUE,  #colourScale = JS("d3.scaleOrdinal(d3.schemeCategory20);"),  arrows = TRUE  )  forcenet %>% saveNetwork(file = filename\_fn)  }  }    #executing the fNVisualizations for finalsolutionNN  fNVisualizations(finalsolutionNN,names)  #executing the fNVisualizations for finalsolutionNN  fNVisualizations(finalsolution1, names)  #executing the fNVisualizations for finalsolutionNN  fNVisualizations(finalsol, names)    #fnVisualizeSolutionFitness will take finalsoltution, number of times you ran for each parameter set,  #number of parameters, and imagename as input  #function will save the different plots for a solution in working directory  fnVisualizeSolutionFitness = function(finalsol,numberOfTimes,numberOfParameters,imagename) {  closeMySQLconnections()  totalcost = vector()  totaltime = vector()  penaltytime = vector()  traveltime = vector()  servicetime = vector()  numberOfVehicles = vector()  parameters = fnGetParameters()  parameter = vector()  vehiclescostpermin = 17 / 60  servicecostpermin = 14 / 60  penaltycostpermin = 17 / 60  ownershipcostperday=17.59  for (j in 1:numberOfParameters)  {  for (i in seq((j - 1) \* numberOfTimes \* 6 + 1, j \* 6 \* numberOfTimes, 6))  {  parameter = c(  parameter,  paste0(  "STOPS ",  parameters$MIN\_STOPS\_PER\_ROUTE[j],  "-",  parameters$MAX\_STOPS\_PER\_ROUTE[j],  " THRESHOLD TIME: ",  parameters$MAX\_EARLY\_TIMEWINDOW[j]  )  )  totalcost = c(  totalcost,  round(finalsol[i + 2]$traveltime \* vehiclescostpermin + finalsol[i + 3]$servicetime \*  servicecostpermin + round(finalsol[i + 4]$penaltytime) \* penaltycostpermin+finalsol[i+5]$numberOfVehicles\*ownershipcostperday)  )  totaltime = c(totaltime, finalsol[i + 1]$totaltime)  traveltime = c(traveltime, finalsol[i + 2]$traveltime)  servicetime = c(servicetime, finalsol[i + 3]$servicetime)  penaltytime = c(penaltytime, round(finalsol[i + 4]$penaltytime))  numberOfVehicles = c(numberOfVehicles, finalsol[i + 5]$numberOfVehicles)  }    }  dataf = cbind(  parameter,  totaltime,  traveltime,  servicetime,  penaltytime,  numberOfVehicles,  totalcost  )  dataf = as.data.frame(dataf)  #write.csv(dataf,"sol3.csv")  f <- list(family = "Calibri",size = 18,color = "#000")  #number of vehicles vs total cost, size= reverse penalty  p = plot\_ly(dataf,x = numberOfVehicles, y = totalcost,text = paste("Total Cost is: ",totalcost,"Number of Vehicles: ",numberOfVehicles,"Travel Time is:",traveltime,"Total Time is:",round(totaltime),"Penalty Time is:",round(penaltytime),  "Parameters:",parameter),mode = "markers",color = parameter,size = -penaltytime,alpha = 1) %>% layout(title = "NUMBER OF VEHICLES VS TOTAL COST") %>% layout(xaxis = list(title = "NUMBER OF VEHICLES",titlefont = f), yaxis = list(title = "TOTAL COST",titlefont = f))  #penaltytime vs total cost, size=numberofvehicles  p7 = plot\_ly(dataf,x = penaltytime, y = totalcost,text = paste("Total Cost is: ",totalcost,"Number of Vehicles: ",numberOfVehicles,"Travel Time is:",traveltime,"Total Time is:",round(totaltime),"Penalty Time is:",round(penaltytime),  "Parameters:",parameter),mode = "markers",color = parameter,size = numberOfVehicles,alpha = 1) %>% layout(title = "PENALTY TIME VS TOTAL COST") %>% layout(xaxis = list(title = "PENALTY TIME",titlefont = f), yaxis = list(title = "TOTAL COST",titlefont = f))      #number of vehicles box plot, color=parameters  p1 <- plot\_ly(dataf,x=~numberOfVehicles, y = ~totalcost, color = ~parameter, type = "box") %>% layout(xaxis=list(title = "NUMBER OF VEHICLES",titlefont = f),yaxis=list(title = "TOTAL COST",titlefont = f))  p2 <- plot\_ly(dataf,x=~numberOfVehicles, y = ~traveltime, color = ~parameter, type = "box") %>% layout(xaxis=list(title = "NUMBER OF VEHICLES",titlefont = f),yaxis=list(title = "TRAVEL TIME",titlefont = f))    #parameters number of vehicles box plot  p4 <- plot\_ly(dataf, y = ~numberOfVehicles, color = ~parameter, type = "box")%>% layout(yaxis=list(title = "NUMBER OF VEHICLES",titlefont = f)  ,xaxis=list(title = "PARAMETERS",titlefont = f),title=" NUMBER OF VEHICLES BOX PLOT")  #parameters total time box plot  p5 <- plot\_ly(dataf, y = ~totaltime, color = ~parameter, type = "box") %>% layout(yaxis=list(title = "TOTAL TIME",titlefont = f)  ,xaxis=list(title = "PARAMETERS",titlefont = f),title=" TOTAL TIME BOX PLOT")    #parameters travel time box plot  p6 <- plot\_ly(dataf, y = ~traveltime, color = ~parameter, type = "box") %>% layout(yaxis=list(title = "TRAVEL TIME",titlefont = f)  ,xaxis=list(title = "PARAMETERS",titlefont = f),title=" TRAVEL TIME BOX PLOT")    #number of vehicles vs travel time  p8 = plot\_ly(dataf,x = numberOfVehicles, y = traveltime,text = paste("Total Cost is: ",traveltime,"Number of Vehicles: ",numberOfVehicles,"Travel Time is:",traveltime,"Total Time is:",round(totaltime),"Penalty Time is:",round(penaltytime),  "Parameters:",parameter),mode = "markers",color = parameter,size = -penaltytime,alpha = 1) %>% layout(title = "NUMBER OF VEHICLES VS TRAVEL TIME") %>% layout(xaxis = list(title = "NUMBER OF VEHICLES",titlefont = f), yaxis = list(title = "TRAVEL TIME",titlefont = f))  saveWidget(p, paste0("cost-vehicles",imagename))  saveWidget(p1, paste0("vehicles-cost-box",imagename))  saveWidget(p2, paste0("vehicles-traveltime-box",imagename))  saveWidget(p3, paste0("penaltytime-box",imagename))  saveWidget(p4, paste0("vehicles-box",imagename))  saveWidget(p5, paste0("totaltime-box",imagename))  saveWidget(p7, paste0("penalty-cost",imagename))  saveWidget(p6, paste0("traveltime-box",imagename))    totalcost\_order = order(dataf$totalcost)  besttotal = dataf[totalcost\_order, ]  return(besttotal)  }    #executing the fnVisualizeSolutionFitness for all the solutions  viz=fnVisualizeSolutionFitness(finalsolutionNN,10,6, imagename="\_2.html")  viz1=fnVisualizeSolutionFitness(finalsolution1,10,6, imagename="\_1.html")  viz2=fnVisualizeSolutionFitness(finalsol,10,6, imagename="\_3.html")      #plots for model comparision  sol1=read.csv("sol1.csv",header=T)  Type=rep("Random Population GA Model",60)  sol1=cbind(sol1,Type)  sol2=read.csv("sol21.csv",header=T)  Type=rep("DNN Population GA Model",60)  sol2=cbind(sol2,Type)  sol3=read.csv("sol3.csv",header=T)  Type=rep("DNN",60)  sol3=cbind(sol3,Type)  compsol=rbind(sol1,sol2,sol3)  #compsol=read.csv("models.csv",header = T)  attach(compsol)  #plotting the graphs using plotly  t <- list(  family = "calibri",  size = 14,  color = toRGB("grey50"))  q = plot\_ly(compsol,x = compsol$numberOfVehicles, y = compsol$totalcost,text = paste("Model:",compsol$Type,"Total Cost is: ",compsol$totalcost,"Number of Vehicles: ",compsol$numberOfVehicles,"Travel Time is:",compsol$traveltime,"Total Time is:",round(compsol$totaltime),"Penalty Time is:",round(compsol$penaltytime),  "Parameters:",compsol$parameter),mode = "markers",color = compsol$Type,size = -compsol$penaltytime,alpha = 1) %>% layout(title = "NUMBER OF VEHICLES AND PENALTY TIME VS TOTAL COST FOR ALL MODELS") %>% layout(xaxis = list(title = "NUMBER OF VEHICLES",titlefont = f), yaxis = list(title = "TOTAL COST",titlefont = f))  q    q1 = plot\_ly(compsol,x = compsol$numberOfVehicles, y = compsol$totaltime,text = paste("Model:",compsol$Type,"Total Cost is: ",compsol$totalcost,"Number of Vehicles: ",compsol$numberOfVehicles,"Travel Time is:",compsol$traveltime,"Total Time is:",round(compsol$totaltime),"Penalty Time is:",round(compsol$penaltytime),  "Parameters:",compsol$parameter),mode = "markers",color = compsol$Type,size = -compsol$penaltytime,alpha = 1) %>% layout(title = "NUMBER OF VEHICLES AND PENALTY TIME VS TOTAL TIME FOR ALL MODELS") %>% layout(xaxis = list(title = "NUMBER OF VEHICLES",titlefont = f), yaxis = list(title = "TOTAL TIME/FITNESS VALUE",titlefont = f))  q1    q2 = plot\_ly(compsol,x = compsol$penaltytime, y = compsol$totaltime,text = paste("Model:",compsol$Type,"Total Cost is: ",compsol$totalcost,"Number of Vehicles: ",compsol$numberOfVehicles,"Travel Time is:",compsol$traveltime,"Total Time is:",round(compsol$totaltime),"Penalty Time is:",round(compsol$penaltytime),  "Parameters:",compsol$parameter),mode = "markers",color = compsol$Type,size = -compsol$traveltime,alpha = 1) %>% layout(title = "PENALTY TIME VS TOTAL COST FOR ALL MODELS") %>% layout(xaxis = list(title = "PENALTY TIME",titlefont = f), yaxis = list(title = "TOTAL COST",titlefont = f))  q2    q5 <- plot\_ly(dataf, x= ~compsol$Type,y = ~compsol$totaltime, color = ~compsol$Type, type = "box") %>% layout(yaxis=list(title = "TOTAL TIME/FITNESS VALUE",titlefont = f)  ,xaxis=list(title = "MODELS",titlefont = f),title=" TOTAL TIME BOX PLOT FOR ALL MODELS")  q5  q6 <- plot\_ly(dataf,x=~compsol$Type, y = ~compsol$totalcost, color = ~compsol$Type, type = "box") %>% layout(text=totalcost,yaxis=list(title = "TOTAL COST",titlefont = f)  ,xaxis=list(title = "MODELS",titlefont = f),title=" TOTAL COST BOX PLOT FOR ALL MODELS")  q6      q6 <- plot\_ly(dataf,x=~compsol$numberOfVehicles, y = ~compsol$traveltime, color = ~compsol$Type, type = "box") %>% layout(yaxis=list(title = "TRAVEL TIME",titlefont = f)  ,xaxis=list(title = "NUMBER OF VEHICLES",titlefont = f),title=" TRAVEL TIME BOX PLOT VS NUMBER OF VEHICLES FOR ALL MODELS")  q6 |