Route Optimization for Spring Cleaning Services

# CPEE Batch 24 Project Report

## Deva Chandra Raju Malla

# 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. Broadly, they are categorized as residential and commercial cleaning. They have presence in 48 cities nationwide with various depots servicing specific regions.

Spring-Cleaning currently has in house software that collects and maintains the services and schedules and decides vehicle routes. You 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 for a given data with routes scheduled on one day for each of their depots. Every vehicle in a depot would have a set of stops for the day, where a stop means customers. In addition, they have to consider 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, etc.

# Business Objective

The objective of this effort is to optimize schedule for Spring-Clean resources under various constraints for a given data with routes scheduled on one day for each of their depots. Also parameterize the cost per mile per vehicle and service cost per minute to estimate the operational costs and profits.

# State of the Art

Route Optimization Solution for Spring-Cleaning is kind of Multiple Travelling Salesmen Problem with Time Windows (VRPTW).

Vehicle Routing Problem with Time Windows (VRPTW) has multiple objectives. The goal is to determine the minimum number of vehicle routes and the optimal sequence of customers/requests visited by each vehicle, such that all customer/requests are served and all constraints imposed are satisfied.

Following are some of the route-finding approaches.

* Construction heuristics

A construction heuristic is an algorithm that determines a tour according to some construction rules, but does not try to improve upon this tour. A tour is successively built and parts already built remain unchanged throughout the algorithm.

* Locals Search

Local search methods are the preferred methods used in practice. The most successful algorithms rely on the well-known 2-opt operator, which removes two edges from a current tour and connects the resulting two parts by two other edges such that a different tour is obtained. Despite the success of these algorithms for a wide range of TSP instances, it is still hard to understand 2-opt from a theoretical point of view.

* Ant Colony Optimization

Ant Colony Optimization (ACO) is a heuristic algorithm which has been proven a successful technique and applied to a number of combinatorial optimization (CO) problems. The traveling salesman problem (TSP) is one of the most important combinatorial problems. ACO is taken as one of the high performance computing methods for TSP. It still has some drawbacks such as stagnation behavior, long computational time, and premature convergence problem of the basic ACO algorithm on TSP. Those problems will be more obvious when the considered problems size increases.

* Branch and Bound

Branch and Bound algorithm (B&B) is an exact method for finding an optimal solution to an NP-Hard problem. It is an enumerative technique that can be applied to a wide class of combinatorial optimization problems.

Finds an optimal solution, if the problem is of limited size and enumeration can be done in reasonable time. Extremely time-consuming: the number of nodes in a branching tree can be too large.

* Genetic Algorithm

Genetic Algorithms are relatively new stochastic search algorithms which based on evolutionary biology- and computer science principles. Due to the effective optimization capabilities of GAs, it makes GA technique suitable for solving route optimization i.e. TSP and mTSP problems.

* Simulated Annealing

Simulated Annealing (SA) has been successfully applied and adapted to give an approximate solution for the TSP. SA is basically a randomized local search algorithm similar but do not allow path exchange that deteriorates the solution.

In recent past majority of state-of-the-art methods are hybridizations of several concepts rather than one pure implementation. As a simple rule of thumb, a classical and successful recipe is to combine

1) efficient “strengthening” procedures via local searches with

2) “diversifications” methods such as crossovers, restarts, or decomposition phases.

# Spring Clean Data Analysis

Data has provided in a database “spring\_clean” on MySQL. Using library(RMySQL) library(dbConnect).

* Stop information - Contains stops (customer numbers), service required time and service requested time window.
* Travel Time information – contains the travel time between depots to stops and stops to stops.
* Parameter information – Contains stops contains and time window constrains

***stops\_info\_db:***

- consists 195 rows (189 stops and 6 depots).

- STOP\_ID : 8 character unique stop id.

- TIME\_TO\_COMPLETE\_WORK: Total time to complete the task/work at each stop/customer Location.

- EXPECTED\_NOT\_BEFORE: customer not expecting before this time.

- EXPECTED\_NOT\_AFTER: customer not expecting after this time.

***travel\_time\_matrix\_db:***

- consists 36270 rows.

- missing rows corresponds to 9 stops.

- FROM\_STOP\_ID : From STOP\_ID/Depot.

- TO\_STOP\_ID : To STOP\_ID/Depot.

- TRAVEL\_TIME : Travel time between stop/depot - stop/depot.

***parameters\_info\_db:***

- consists of 6 rows.

- columns:

- MAX\_STOPS\_PER\_ROUTE : Maximum stops threshold per route. (5/6/7)

- MIN\_STOPS\_PER\_ROUTE : Minimum stops threshold per route. (3/4/5)

- MAX\_ROUTE\_TIME : Maximum time (travel and job execution time) threshold per route. (660mins - 11hours)

- MAX\_EARLY\_TIMEWINDOW : Allowable time window and large violations window, if reaches early.

- MAX\_LATE\_TIMEWINDOW : Allowable time window and large violations window, if reaches late.

Service times and travel time is provided in “*minutes*”, so the optimal solution would result a minutes.

## Observations

* There are 189 stops and 6 depots.
* Out of 6 depots 2 depots ("*DEP45024*", "*DEP50002*”) are located at same place, as there is no limitation on number of vehicles per depot and to simplify the problem considered only 4 depots.
* There are missing rows corresponds to 9 stops.
* The average distance from each depot to all stops is approx. 260.
* The distance (in minutes) between stops is asymmetric.



Figure 4: Distance between depots and stops.

## Impute missing stops distance

The travel time from 9 stops to any other stops or depots is missing and hence needed to be imputed.

I.e. 9 \* 195 = 1755 stops travel time missing in database.

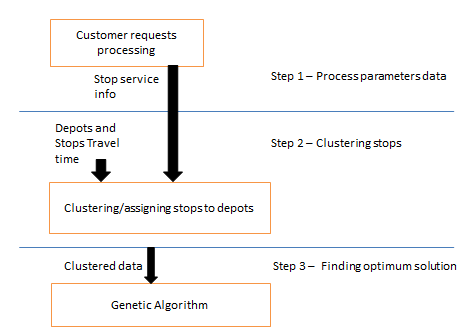
In the database travel time to missing stops are available. Though the travel time between already available stops are asymmetric, to impute the missing stops assumed travel as symmetric and imputed from information available in database.

This leaves 81 missing distances (9 \* 9) needs an imputation, which are not available in database. For these missing stops imputed with an average time of the row.

# Approach

To provide an optimum solution for Spring-Cleaning, designed a solution using problem specific clustering mechanism (main problem) and Genetic Algorithm based heuristic approach (sub problem).

Assigned all customers near to each depot based on the distance and applied genetic algorithm on clustered stops at depot. The solution of the original problem is obtained through iterative interactions between the main problem and the set of sub-problems (refer Figure 5a).



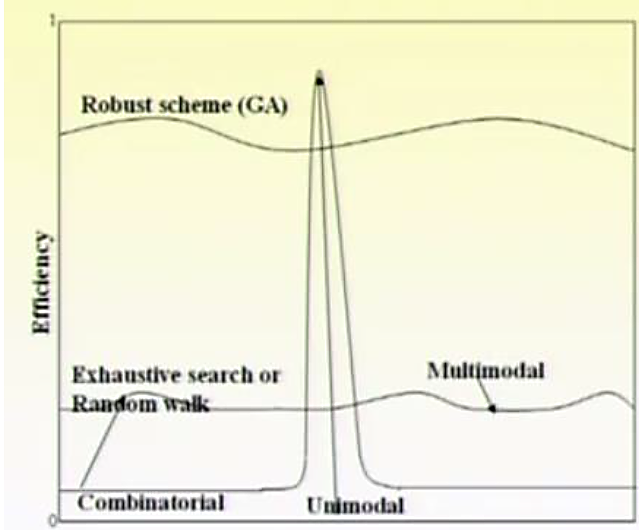
Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution; specially those follow the principles first laid down by Charles Darwin of "survival of the fittest".

Figure 5a: High level approach.

## Why Genetic Algorithm?

The optimization task can be described as follows: given a fleet of vehicles, a common depot and several requests by the customers, find the set of routes with overall minimum route cost which service all the demands. Because of the fact that TSP is already a complex, namely an NP-complete problem, heuristic optimization algorithms, like genetic algorithms (GAs) need to take into account.

Genetic Algorithms are relatively new stochastic search algorithms which based on evolutionary biology- and computer science principles. Due to the effective optimization capabilities of GAs, it makes GA technique suitable for solving route optimization i.e. TSP and mTSP problems.



Advantages of GA:

* Concepts are easy to understand
* Genetic Algorithms are intrinsically parallel
* Always gives answer and answer gets better with iterations
* Chances of getting optimal solutions are more.

### 

### Figure 5b: GA vs other searches.

## Why clustering instead of random selection of depots and stops?

To understand the reasons for clustering please refer below some of the observation from Figure 1 i.e. Distance between depots to stops

* The actual distance between from each depot to stops covers a wide range of distances from 0 to 600.
* The average distance for each depot is approx. 260.
* The 25th quartile is 35 miles for most of the depots

So, intuitively this explains random selection will most likely start with very higher travelling distance because 50 % of the stops distances for each depot are above 150 miles, because of this genetic algorithm required huge number iterations to converge to optimum solution.

Refer below *Figure 5c,* which explains how clustering will help in reducing the initial travel time.

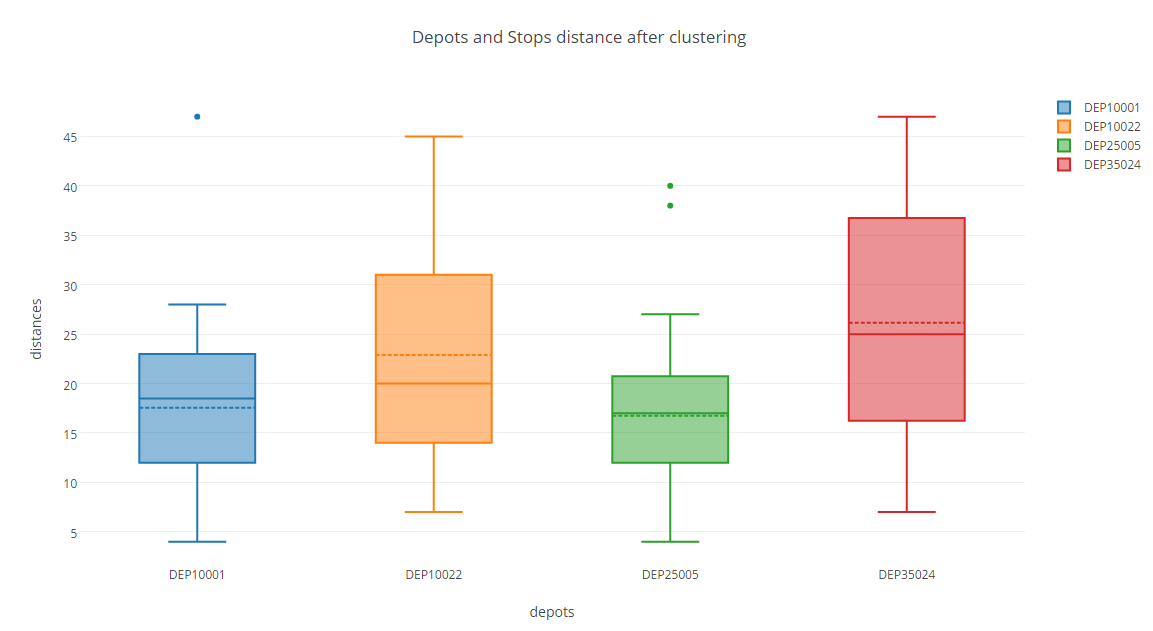
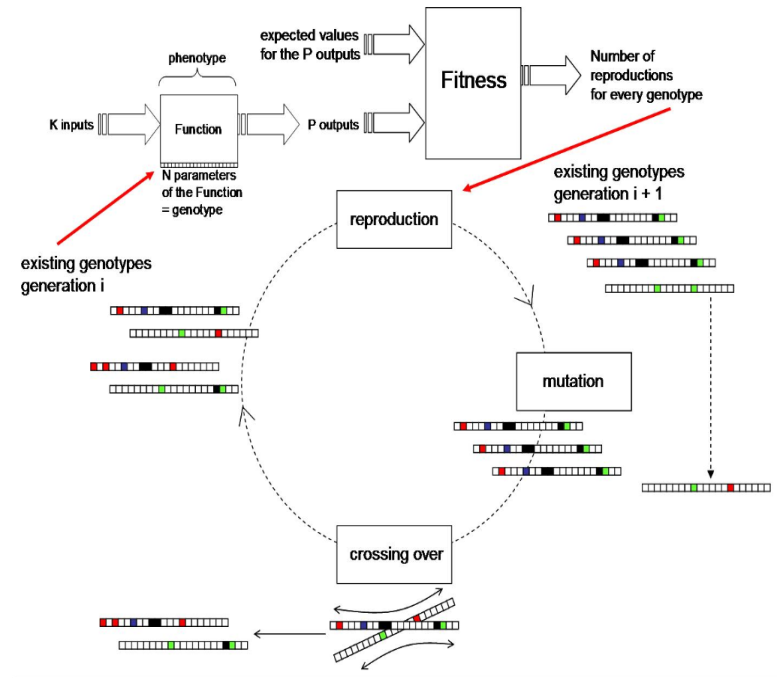


Figure 5c: Distance between depots and stops (in minutes) after clustering

After clustering 47 miles is highest distance assigned to depot DEP10001. Stops distances to all other distance range narrowed from 0 – ~600 to 0 - ~50 and three of the depots 75th quartiles are less than 30 miles.

This allows genetic algorithm to converge to optimum solution with less number of iterations.

# Genetic Algorithm for Route Optimization

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions.

Spring clean data contains 189 customer requests/stops which need to be serviced for a day with an optimum travel cost. Regular GA-based approach has used single chromosome representation for solving the route optimization has used single chromosome for representation so far. For this problem followed a multi-chromosome approach, which separates the vehicles routes from each other thus may present a more effective approach.

All vehicles routes from each route collectively form a main chromosome.

In this problem each customer request/stop represents a gene, eachvehicle routerepresents a sub portion of main chromosome and collection of all vehicles routes for each depot represents chromosome.

* Gene – each customer requests/stop.
* Chromosome - all depots vehicle routes includes every city once and only once from either of the depot. Refer Figure 6 a sample vehicles routes from different depots.









Figure 6 a: subset of a chromosome*.*

* Population - collection of all chromosomes with different route combinations.
* Top Elite - accounted multiple top elite ranges for different test runs i.e. from 20% to 50%.
* Mutation probability – Mutation probability started with 0.9 and eventually reduces over the iterations.

## Special genetic operators

### Crossover and mutation are two basic and special operators of GA. Performance of GA very depend on them. Type and implementation of operators depends on encoding and also on a problem. See below for the mutation and crossover techniques used in this approach.

### Mutation

### There are two sets of mutation operations followed for mutation, the so-called In-route mutations and the Cross-route mutations.

* + ***In-route mutation***operation work inside vehicle route. The operator chooses a random subsection of a vehicle route and inverts the order of the genes inside it. An example is illustrated on Fig. 6b.

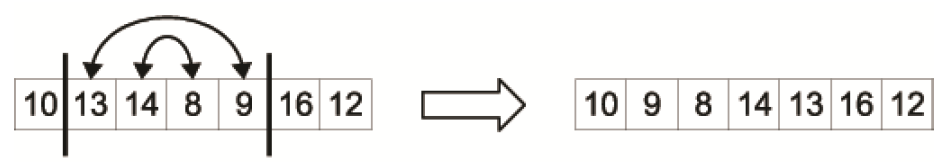


Figure 6.1a: In-route mutation – gene sequence inversion.

* + ***Cross-route mutation*** operates on multiple chromosomes. If we think about the distinct chromosomes as individuals, this method could be similar to the regular crossover operator. Fig. 3 illustrates the method when randomly chosen subparts of two chromosomes are transposed. If the length of one of the chosen subsections equal to zero, the operator could transform into an interpolation.

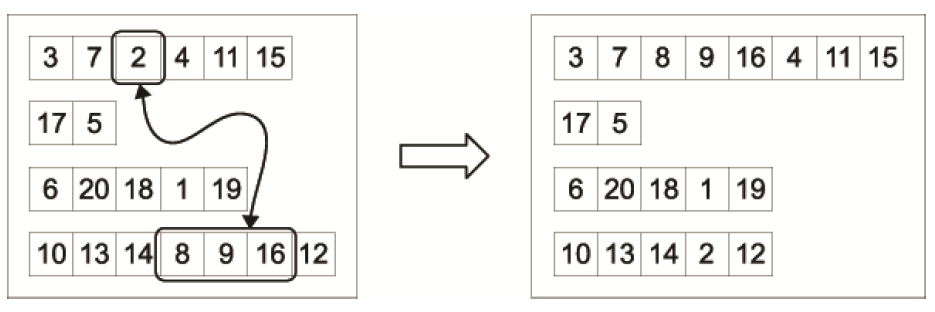


Figure 6b: In-route mutation – gene sequence inversion.

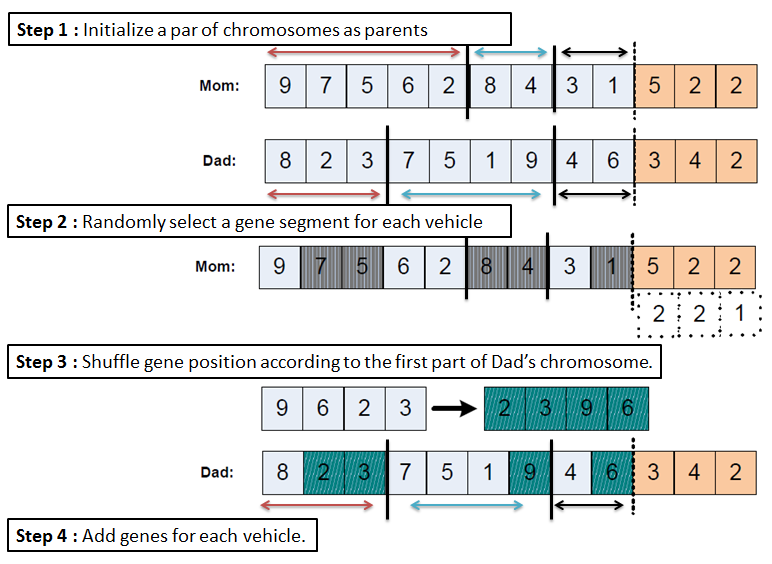
### Crossover

The key to finding a good solution using GAs lies in developing a good chromosome representation of candidate solutions to the problem. Ideally, a good chromosome representation should reduce or eliminate redundant chromosomes from the GA population. Redundancy in the chromosome representation refers to a solution that is capable of being represented in more than one way and appearing in the population multiple times. These multiple representations increase the search space unnecessarily and inhibit the search process.

As part of design to find an optimum solution and to build a population, followed a two part chromosome technique, as the name implies, divides the chromosome into two parts. The first part of length n represents a permutation of n cities and the second part of length m gives the number of cities assigned to each vehcile. Therefore, the total length of the chromosome is n + m in this representation. The m values present in the second part of the chromosome must sum to n in order to represent a valid solution.

Two part crossover (TCX) treats each vehicles separately when performing crossover in the first part of the chromosome. This ensures that highly fit building blocks that may be present in the routes of parental chromosomes are maintained during the reproduction process and inherited by the offspring chromosomes.

An example is used below to illustrate the process of using TCX to generate a child chromosome of (9 cities and 3 vehicles). Five basic steps involved. Firstly, a pair of parent two-part chromosomes (Mom and Dad) are initialized, and the Mom chromosome is used as the base for generating a Child in this example. Secondly, we randomly select a gene segment (i.e. sub-tour) from the first part of Mom‟s chromosome. In this case, the selected gene segments are <7, 5> for vehicle 1, <8, 4> for vehicle 2 and <1> for vehicle 3. Hence, the numbers of randomly selected genes are (2, 2, and 1) as shown in the dashed area of Step 2. Next, the order of the remaining genes is sorted according to the positions in the first part of Dad‟s chromosome. In this example, the remaining genes in the first part of Mom‟s chromosome are <9, 6, 2,3>, and the order of these genes is shuffled to <2, 3, 9, 6> according to the first part of Dad‟s chromosome. Next, based on the number of unselected genes, we sequentially pick the genes (stops) based on the number of stops for every vehicle from Mom‟s chromosome, to determine how many new genes will be added for each vehicle in the child chromosome. Here, the unselected genes are <2, 3, 9, 6> and if, for example, we sequentially pick the genes (3, 0, 1), then in the Child chromosome, vehicle 1 gets genes <2, 3, 8>, vehicle 2 gets <none>, and vehicle 3 gets <6>.



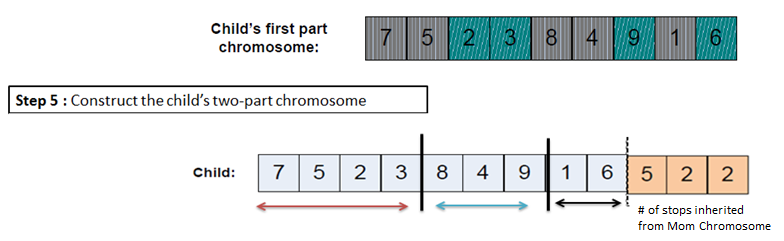


Figure 6.1.2 a: *Two-part crossover operation.*

### Convergence criteria

A genetic algorithm is usually said to converge when there is no significant improvement in the values of fitness of the population from one generation to the next.

The common GA terminating conditions are:

* + When fixed number of generations are reached
  + A optimal solution is obtained that satisfies the optimization criteria
  + When successive GA iterations no longer produce better results
  + Allocated budget (computational time / cost) reached

As this is a NP-Hard optimization problem and main objective is to identify the optimum solution, so chosen two conditions *‘When fixed number of generations are reached ‘* and *‘When 100 successive GA iterations no longer produce better results’* to terminate or as a convergence criteria for GA.

# Penalty Constraints

There are penalty clauses based on certain conditions on a vehicles number of stops, stops reach time and vehicles total travel time.

Constraints:

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.

1. 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...

1. Service time window for customer is 11:00 AM to 14:00 PM. If a vehicle reaches a customer before/after small time window (30 mins) i.e. 10:30 AM to 11:00 AM and 2:00 PM to 2:30 PM - no penalty
   * Within small and large time window( 10:00 AM to 10:300 AM and 2:30 PM to 3:00 PM) - 10% of total route time
   * Beyond large time window (Before 10:00 AM and after 3:00 PM)

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

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

As there is no vehicle start time constraint from depot, assumed each vehicle will reach first customer in a no penalty zone.

Designed a program to apply slab basis penalty incase if vehicle not meets constraint 1 and 3.

# Test Conditions and assumptions

Solution has been designed to perform multiple test conditions through a **‘comma delimited’** file. There are assumptions made to compare multiple run of Genetic algorithm and to suggest best vehicle range and service cost to Spring Clean Services (Refer Figure 8a).

Program expects the following fields to iterate and find an optimum solution:

* **Stops limit** – This field accepts three values ‘0’ for stops range 3 to 5,’1’ for stops range 4 to 6 and ’2’ for stops range ‘5’ to ‘7’.
* **Iterations** – Number of times reproduction need to be performed
* **Mutate Probability** – Probability to apply a mutation technique. Value provided will be the initial value for mutation, program will reduce probability eventually.
* **Top elite percent** – percentage of top elite population for next reproduction after fitness calculation.
* **Population size** – Initial number of chromosomes or population need to be produced from mutation and crossover
* **Cost per mile** –This is to analyze total travel cost from the total miles traveled.
* **Vehicle daily rentals** – Vehicle daily rental charges.
* **Service cost per min** – Average service cost for Spring Clean services per min.

Program calculates the following attributes and writes to a **‘comma delimited’** file.

* **Total travel minutes** – Total travel time of entire route which including all vehicles of all four depots.
* **Number of vehicles** – Number of vehicles required for optimal travel route.
* **Totals service time** – Total service time of all customer requests.
* **Total cost of route** – Estimated cost of a route considering cost per mile

i.e. (**Total travel minutes** \* **Cost per mile) + (Number of vehicles \* Vehicle daily rentals**

* **Total service income –** Expected services income i.e. **Total service time \* Service cost per minute**
* **Process execution time –** Time taken to execute the GA.

**Assumptions:**

* To convert travel minutes in to miles, assumed vehicle average travel as mile per minute.
* As per web sources considered **Cost per mile** as 1.1$
* As per web sources considered average **Vehicle daily rentals** as 150$.
* Though service cost varies based on type of requested services (ex: Carpet cleaning, furniture, chimney and office space etc...), considered average service cost as 3 to 6$.

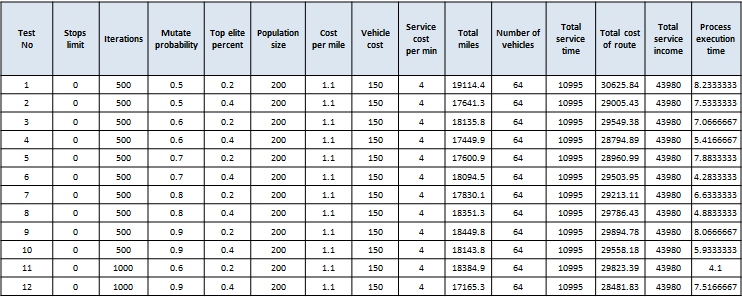


Figure 8 a: *Sample test conditions CSV file.*

# Result Analysis

There are multiple tests has been performed with different test conditions. Performed multiple tests to compare the effect of crossover & without cross, different mutation techniques, multiple stop range conditions and multiple population ranges.

## Crossover effectiveness

Two-part chromosome crossover technique is effectively generation. The offspring have a better chance of inheriting highly-fit routes from the parental chromosomes, because the mapping between each route and the multiple vehicles sub tour are considered separately in two-part chromosome. This allows children to have an opportunity to reach any feasible combinations within the entire search space.

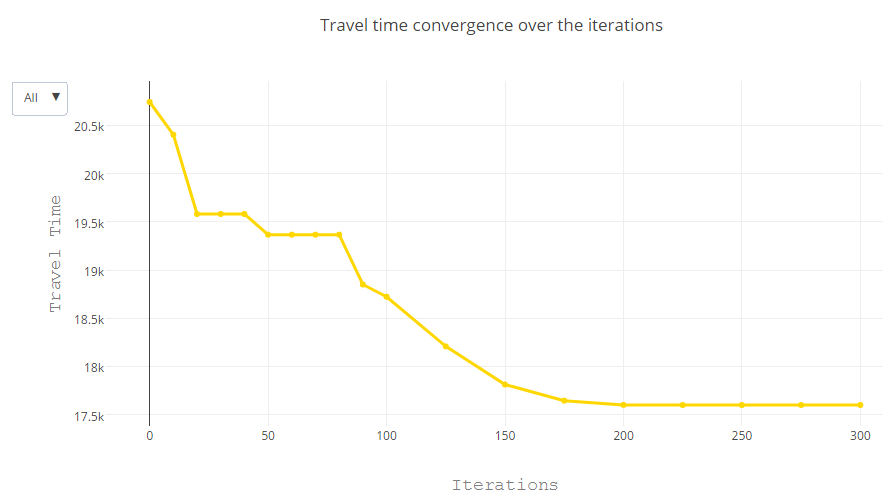


Figure 9.1 a: Travel time convergence*.*

## Mutation Techniques

Multiple tests performed to compare the effectiveness of three mutation techniques, such as ‘Cross route’, ‘In-route’ and combined.

‘In-route’ mutation approach showed an early convergence, but over the iterations population generated from ‘in-route’ didn’t help crossover technique to produce a better reproduction after 40 iterations whereas ‘Cross route’ and combined techniques closed produced optimal solution. Though ‘Cross route’ and combined closely performed, ‘Cross-route’ mutation approach helped cross-over technique to produce a better reproduction and allowed a more search space from 20th iteration to till 70th iteration than ‘in-route’ and ‘Combined’ techniques.

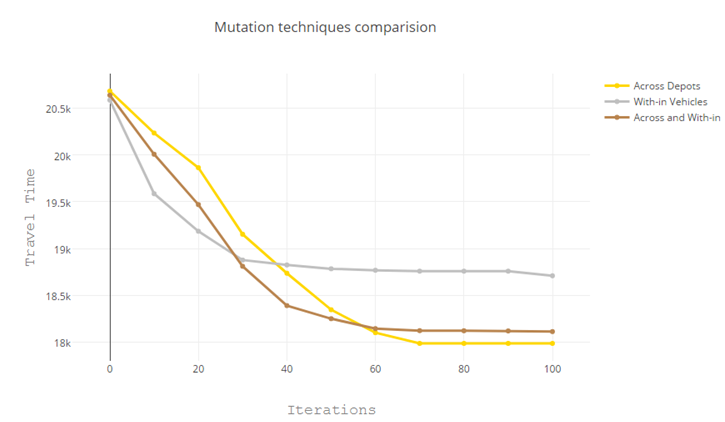


Figure 9.2 a: Mutation techniques compare*.*

## Various Stops range tests.

Objective of the project is to identify optimal solution for given stop ranges for each vehicle. Performed multiple tests and recorded the average travel time for different stop ranges (i.e. for 3 to 5, 4 to 6 and 5 to 7). Test are performed for a similar parameter set multiple times (refer *Table 9.3 a* for test conditions).

**Table 9.3a: Parameters for Various Stops range tests**

**Parameter Value Average Travel Time**

Stops Range random selection **3 to 5**, 17972 miles

**4 to 6** 22245 miles

**5 to 7** 31318 miles

Mutation probability rate in GA 0.8

Top elite 0.4

Iterations 500

Population 200

Number of Tests 12

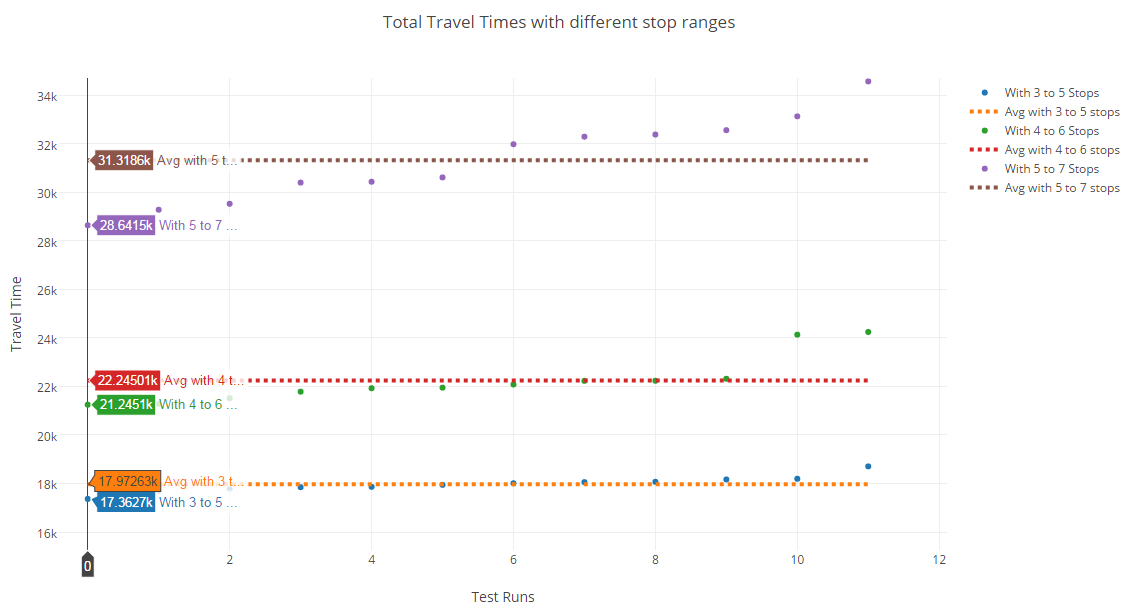


Figure 9.3 a: Travel Times Vs Stops range

## Performance testing

Multiple tests performed to record the execution timing s of an each test conditions with certain parameters (refer *Table 9.4a* for test conditions and *Figure 9.4.a* for test performances without convergence criteria.

**Table 9.4a: Parameters for performance tests**

**Parameter Value**

**----------------------------------------------------------------------------------------------------------------------------------------------------**

Stops Range random selection 3 to 5

Mutation probability rate in GA 0.9

Mutation Technique Cross-Route

Top elite 0.4

Iterations 10 to 1000

Population 50 to 300

Test Type **With** and **Without**

Convergence Criteria

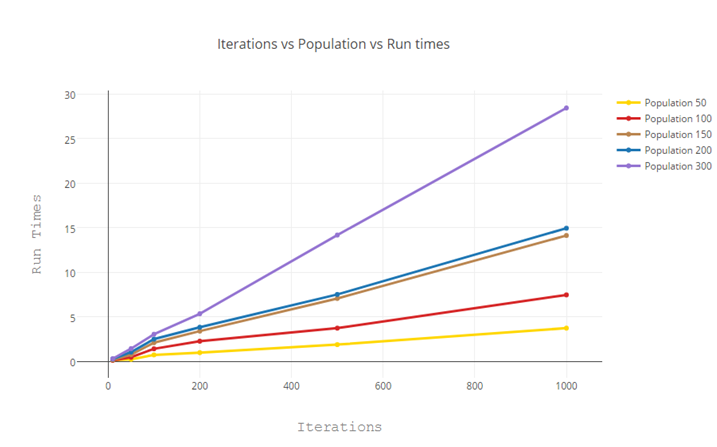
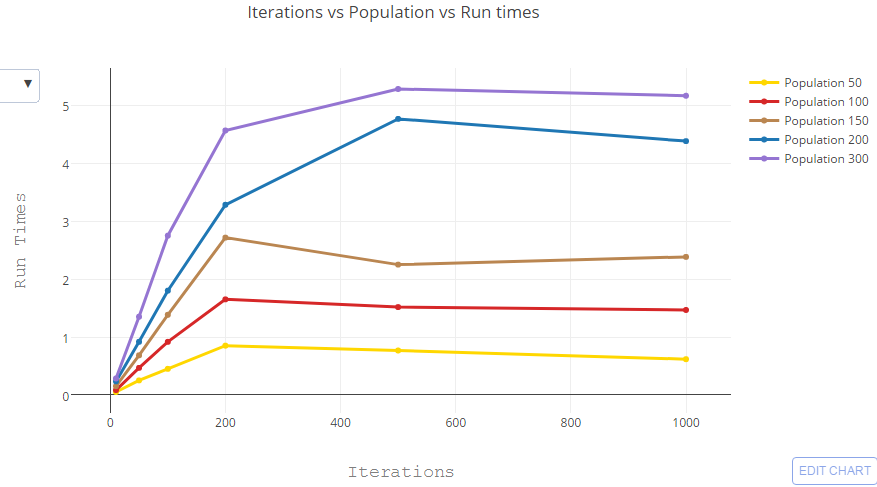
 

Figure 9.4 a: Without Convergence Criteria Figure 9.4 b: With Convergence criteria.

Except four tests out of 30, has completed within 8 mins. Population size with 300 for iterations 500 and 1000 ran for 13 and 27 minutes respectively.

There is a 50% drop in run times for higher iterations with convergence criteria in place where as there is no drastic change in variance of total travel times (refer *Figure 9.4 c* and *9.4 d*).

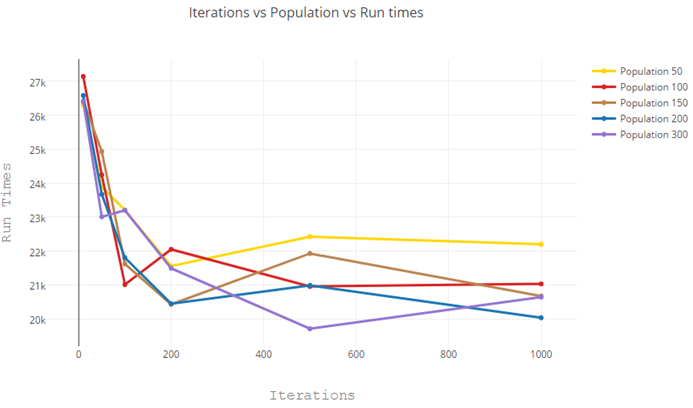
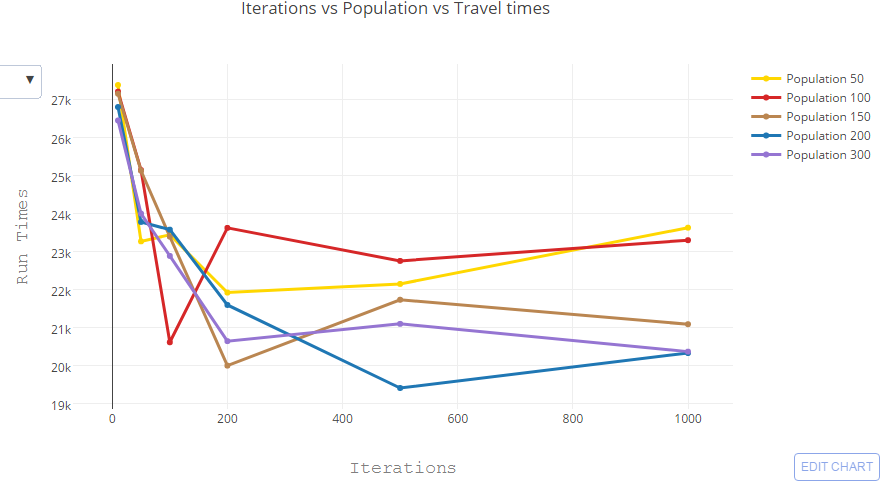
 

Figure 9.4 c: Without Convergence Criteria*.* Figure 9.4 d: With Convergence criteria.

# Conclusion

There are couple assumptions made to come-up with an overall travel cost of the problem.

**Assumptions:**

* As per web sources considered **Cost per mile** as **1.1$,** which includes gas, toll fare and driver wages.
* As per web sources considered average **Vehicle daily rentals** as **150$ per day.**

Average service cost **4$ per minute,** though the service cost varies based on type of requested services (ex: Carpet cleaning, furniture, chimney and office space etc...).

Total service times requested by customers are 10995 minutes, with an assumption of 4$ per minute service expected total service income is 43980$.

Table 10a gives an average travel time and average travel cost comparison for multiple stops ranges. Genetic Algorithm has outperformed for stops range 3 to 5 with an average travel time 17972 miles and with an optimized average travel cost of 29369$.(refer *Figure 10a),* which leads to an margin of 14000$ approx.

**Table 10a: Parameters for comparison tests**

**Stops Range Average Travel Time Average Travel Cost**

3 to 5 17972 miles $29369

4 to 6 22245 miles $31819

5 to 731318 miles $40300

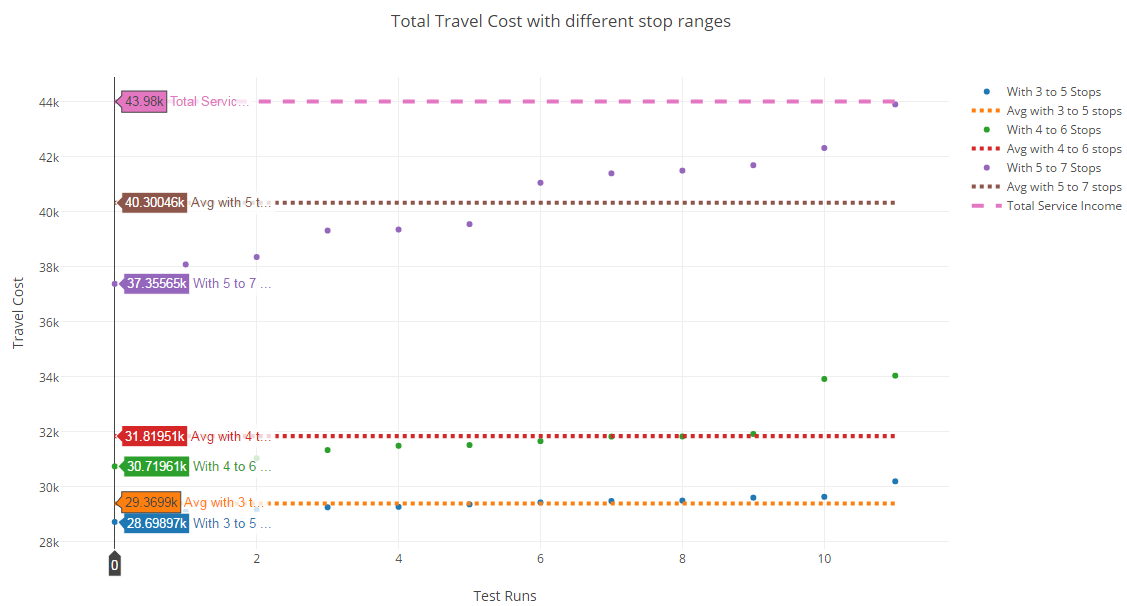


Figure 10a: Travel Cost Incurred vs Expected service Income*.*

# Appendix: Python Code

Program has been designed into 5 programs and functions based the purpose of the code.

1. *SpringClean\_DataPull.py*

This program is to connect to database, pull data from *parameters\_db, stops\_db* and *travel\_time\_db* tables and returns to calling program.

1. *SpringClean\_GA\_Functions.py*

This program contains core functions for Genetic algorithm.

generate\_chromosome – To generate chromosome

generate\_population – To generate population using clustered stops informed and based on pop\_size parameter.

generate\_population\_rand – To generate population using random depot and stops selection.

mutate\_across\_depots - This function is to mutate the vehicle routes across depots.

mutate\_within\_route – This function is to mutate the stops within vehicle route.

crossover – To perform crossover between two chromosomes or on a two solutions of a population.

1. *SpringClean\_Genetic\_Algorithm.py*

This program contains core Genetic algorithm iterative logic and convergence criteria.

1. *SpringClean\_TravelTime.py*

This program is to calculate travel time and penalty for routes which are not meeting constrains. Following functions created to calculate penalty and travel time.

fitness\_calc - Calculates penalty and travel time for vehicles of all depots.

vechile\_route\_time – calculate travel time for one vehicle

penalty\_cal\_slab\_basis – calculates penalty on a slab basis.

penalty\_percent\_calc – calculates penalty on entire travel time.

1. *SpringCleanMain.py*

Main program has a capability to read the data from either database or from CSV file.

* Reads all data into dataframes and converts into numpy array for faster processing.
* Data processing
  + Identifies missing stop information.
  + Imputes missing stops
  + Converts travel time information into 2D matrix i.e. 195 x 195.
* Iterates *SpringClean\_Genetic\_Algorithm.py* for all test conditions provided in spread sheet.
* Writes optimal route to *text file* and also writes a travel time, travel cost and execution time into CSV file.

***Code:***

***SpringClean\_Genetic\_Algorithm.py***

*import numpy as np*

*import copy*

*import random*

*from SpringClean\_GA\_Functions import \**

*from SpringClean\_TravelTime import fitness\_calc*

*def Genetic\_Algorithm(travel\_dist\_matrix,stop\_service\_info,parameters\_info,veh\_range,iterations,mutate\_per,elite\_per,origPopSize):*

*service\_time\_window = parameters\_info[veh\_range,:]*

*penalty\_time\_window = parameters\_info[veh\_range+3,:]*

*### Pull only four depots from array to allocate stops to each depot*

*stops\_depots = travel\_dist\_matrix[:4,]*

*## Generating chrosome and population from clustered stops.*

*InitPopulation = generate\_population(origPopSize,stops\_depots,service\_time\_window)*

*## Generating chromosome and population with randome depot and stop selection*

*#InitPopulation = generate\_population\_rand(origPopSize,stops\_depots,service\_time\_window)*

*chomosome\_time = fitness\_calc(InitPopulation,travel\_dist\_matrix,stop\_service\_info,service\_time\_window,penalty\_time\_window)*

*## This logic is to sort population based on fitness*

*chomosome\_time = np.array(chomosome\_time)*

*srt = np.argsort(chomosome\_time[:,4])*

*InitPopulation = chomosome\_time[srt,:]*

*topElite=int(round(elite\_per\*origPopSize,0))*

*population = InitPopulation[:topElite,]*

*prev\_best\_score, loop\_break\_counter = 0, 0*

*for iter in range(iterations):*

*population = copy.deepcopy(population[:topElite,]) #deepcopy is to copy only content not reference of an object*

*mutate = mutate\_per/(iter + 1)*

*while len(population) < origPopSize:*

*if (np.random.random\_sample(1) < mutate):*

*random\_chromosome = np.random.randint(0,high=topElite,size=1)*

*mutate\_chromosome = copy.deepcopy(population[random\_chromosome[0]])*

*### Random number to select mutation either across depots or within vehicle route*

*mutated = mutate\_across\_depots(mutate\_chromosome)*

*#mutate\_select = np.random.randint(0,high=2,size=1)*

*#if mutate\_select == 0:*

*# mutated = mutate\_across\_depots(mutate\_chromosome)*

*#else:*

*# #mutated = mutate\_within\_route(mutate\_chromosome)*

*# mutated = mutate\_across\_depots(mutate\_chromosome)*

*mutate\_ind = []*

*mutate\_ind.append(copy.deepcopy(mutated[:4].tolist()))*

*chomosome\_time = fitness\_calc(mutate\_ind,travel\_dist\_matrix,stop\_service\_info,service\_time\_window,penalty\_time\_window)*

*else:*

*cross\_over\_parents = random.sample(range(int(topElite)-1),2)*

*parent\_1 = copy.deepcopy(population[cross\_over\_parents[0]])*

*parent\_2 = copy.deepcopy(population[cross\_over\_parents[1]])*

*childs = crossOver(parent\_1,parent\_2)*

*childs = copy.deepcopy(np.array(childs)[:,:4].tolist())*

*chomosome\_time = fitness\_calc(childs,travel\_dist\_matrix,stop\_service\_info,service\_time\_window,penalty\_time\_window)*

*chomosome\_time = np.array(chomosome\_time)*

*population = np.vstack([population,chomosome\_time])*

*srt = np.argsort(population[:,4])*

*population = population[srt,:]*

*Tot\_Vehicles = len(population[0][0])+len(population[0][1])+len(population[0][2])+len(population[0][3])*

*#print("Best fitness score in iteration", iter, "=",population[0][4], Tot\_Vehicles)*

*# Criteria to stop processing if route time is not converging for the last 100 iterations*

*if prev\_best\_score == population[0][4]:*

*loop\_break\_counter = loop\_break\_counter + 1*

*else:*

*loop\_break\_counter = 0*

*prev\_best\_score = copy.deepcopy(population[0][4])*

*if loop\_break\_counter > 100:*

*print("### Information : At",iter,"algorithm not converged for last 100 iterations ### ")*

*break;*

*print("Best fitness Score ", population[0][4])*

*print("Mutate Probability ", mutate\_per)*

*print("Iterations ", iterations)*

*print("origPopSize ", origPopSize)*

*print("Total # of vehicles :",Tot\_Vehicles)*

*return population[0][4],Tot\_Vehicles,population[0]*

***SpringClean\_GA\_Functions.py***

*import random*

*import numpy as np*

*### This is to generate chromosome(which combines routes of vechiles belongs to each depot).*

*### This funtion returns a vehicle route in list of list of lists.*

*def generate\_chromosome(depot\_stops\_array,Min\_Veh,Max\_Veh,depot\_num):*

*random.shuffle(depot\_stops\_array) ## This is random shuffle stops assigned to depot.*

*depot\_vehs\_route = []*

*i = 0*

*while i <= depot\_stops\_array.shape[0]:*

*stops\_number = np.random.randint(Min\_Veh,Max\_Veh+1)*

*veh\_route = [depot\_num]*

*## Below logic is to assign left over stops at the end of route to final vehicle.*

*if i + stops\_number > (depot\_stops\_array.shape[0]):*

*stops\_number = (depot\_stops\_array.shape[0] - i)*

*if stops\_number > 0:*

*for \_ in range(stops\_number):*

*i = i + 1*

*veh\_route.append(depot\_stops\_array[i-1])*

*veh\_route.append(depot\_num)*

*depot\_vehs\_route.append(veh\_route)*

*else:*

*break*

*return depot\_vehs\_route*

*### This function is to generate population from a shortest distance stops for each depot*

*def generate\_population(pop\_size,travel\_time\_depts,veh\_limit\_parms):*

*### Assigning a stops to each depot based on shotest distance. This logic uses argmin funtion to pull the index of shortest distance*

*depot\_stops = [[],[],[],[]]*

*for i in range(travel\_time\_depts.shape[1]):*

*depot\_idx = np.argmin(travel\_time\_depts[:,i],0)*

*depot\_stops[depot\_idx].append(i)*

*InitPopulation = []*

*for \_ in range(pop\_size):*

*chromosome = []*

*for dept in range(len(depot\_stops)):*

*stops\_dept\_np = np.array(depot\_stops[dept][1:])*

*veh\_route = generate\_chromosome(stops\_dept\_np,veh\_limit\_parms[1],veh\_limit\_parms[0],dept)*

*chromosome.append(veh\_route)*

*InitPopulation.append(chromosome)*

*return(InitPopulation)*

*### This function is to generate population from a random stops for each depot*

*def generate\_population\_rand(pop\_size,travel\_time\_depts,veh\_limit\_parms):*

*depot\_stops = [[],[],[],[]]*

*for i in range(travel\_time\_depts.shape[1]):*

*depot\_idx = np.random.randint(0,high=4,size=1)*

*depot\_stops[depot\_idx[0]].append(i)*

*InitPopulation = []*

*for \_ in range(pop\_size):*

*chromosome = []*

*for dept in range(len(depot\_stops)):*

*stops\_dept\_np = np.array(depot\_stops[dept][1:])*

*veh\_route = generate\_chromosome(stops\_dept\_np,veh\_limit\_parms[1],veh\_limit\_parms[0],dept)*

*chromosome.append(veh\_route)*

*InitPopulation.append(chromosome)*

*return(InitPopulation)*

*## This function is to mutate the individuals - Approach is to swap vehicle routes across depots.*

*def mutate\_across\_depots(mutate\_parent):*

*ds = random.sample(range(len(mutate\_parent)-1),2) ## This is to pick the two random depots to mutate*

*# this is to pick the random vehicle under ramdomly selected depot.*

*vh1 = random.sample(range(len(mutate\_parent[ds[0]])-1),1)*

*vh2 = random.sample(range(len(mutate\_parent[ds[1]])-1),1)*

*depot1 = mutate\_parent[ds[0]][vh1[0]][0]*

*depot2 = mutate\_parent[ds[1]][vh2[0]][0]*

*mutate\_parent[ds[0]][vh1[0]] = [x if x != depot1 else depot2 for x in mutate\_parent[ds[0]][vh1[0]]]*

*mutate\_parent[ds[1]][vh2[0]] = [x if x != depot2 else depot1 for x in mutate\_parent[ds[1]][vh2[0]]]*

*### Below is to swap (mutate routes of a vehicle*

*mutate\_parent[ds[0]][vh1[0]], mutate\_parent[ds[1]][vh2[0]] = mutate\_parent[ds[1]][vh2[0]], mutate\_parent[ds[0]][vh1[0]]*

*mutate\_parent[4] = 0*

*return(mutate\_parent)*

*## This function is to mutate the individuals - Approach is to swap stops within vehicle routes.*

*def mutate\_within\_route(mutate\_parent):*

*ds = random.sample(range(len(mutate\_parent)-1),2) ## This is to pick the random two depots to mutate*

*# this is to pick the random vehicle under ramdomly selected depot.*

*vh1 = random.sample(range(len(mutate\_parent[ds[0]])-1),1)*

*vh2 = random.sample(range(len(mutate\_parent[ds[1]])-1),1)*

*# Select the stop randomly for each vehicle route -*

*vh1\_stop = np.random.randint(1,high=(len(mutate\_parent[ds[0]][vh1[0]])-1),size=1)*

*vh2\_stop = np.random.randint(1,high=(len(mutate\_parent[ds[1]][vh2[0]])-1),size=1)*

*### Below is to swap (mutate routes of a vehicle*

*mutate\_parent[ds[0]][vh1[0]][1], mutate\_parent[ds[0]][vh1[0]][vh1\_stop[0]] = mutate\_parent[ds[0]][vh1[0]][vh1\_stop[0]], mutate\_parent[ds[0]][vh1[0]][1]*

*mutate\_parent[ds[1]][vh2[0]][1], mutate\_parent[ds[1]][vh2[0]][vh2\_stop[0]] = mutate\_parent[ds[1]][vh2[0]][vh2\_stop[0]], mutate\_parent[ds[1]][vh2[0]][1]*

*mutate\_parent[4] = 0*

*return(mutate\_parent)*

*def crossOver(parent1,parent2):*

*## produce two childerns using from parent 1 and parent 2*

*child = [parent1]*

*child.append(parent2)*

*number\_of\_stops = 189*

*depots\_nums = [0,1,2,3]*

*for i in range(len(child)):*

*if i == 0:*

*l = child[i+1][0:4]*

*elif i == 1:*

*l = child[i-1][0:4]*

*else:*

*print("Out of Boundary - This CrossOver funtion capable to generate two childs from parents")*

*unsaved\_gene = []*

*stops\_random = random.sample(range(4,number\_of\_stops+5),number\_of\_stops - round(number\_of\_stops \* 0.5))*

*flatten = lambda l: [item for depots in l for vehicles in depots for item in vehicles]*

*parent\_f = flatten(l)*

*for k in range(len(parent\_f)):*

*if ((parent\_f[k] in stops\_random) or*

*(parent\_f[k] in depots\_nums)):*

*pass*

*elif parent\_f[k] not in depots\_nums:*

*unsaved\_gene.append(parent\_f[k])*

*j = 0*

*for depots in range(len(child[i])-1):*

*for vehs in range(len(child[i][depots])):*

*for stop in range(len(child[i][depots][vehs])):*

*if ((child[i][depots][vehs][stop] in stops\_random) or*

*(child[i][depots][vehs][stop] in depots\_nums)):*

*pass*

*elif child[i][depots][vehs][stop] not in depots\_nums:*

*child[i][depots][vehs][stop] = unsaved\_gene[j]*

*j = j + 1*

*child[i][4] = 0 # reset the time calculated to re-calculate for child*

*return(child)*

***SpringClean\_DataPull.py***

*import pymysql.cursors*

*def pull\_from\_mySQL():*

*#### For mysql connectivity*

*#### Connect to mysql using the appropriate credentials to the desired database.*

*connection = pymysql.connect(host='ip-172-31-13-154',*

*user='insofeadmin',*

*password='MDQzZTgyYj',*

*db='insofe\_1047\_spring\_clean',*

*charset='utf8mb4',*

*cursorclass=pymysql.cursors.DictCursor)*

*try:*

*with connection.cursor() as cursor:*

*### Extract data from active\_emp\_details table*

*query = "SELECT \* FROM `parameters\_info\_db`"*

*cursor.execute(query)*

*parameters\_db = cursor.fetchall()*

*### Extract data from dept\_aggr\_by\_gender table*

*query = "SELECT \* FROM `stops\_info\_db`"*

*cursor.execute(query)*

*stops\_db = cursor.fetchall()*

*### Extract data from dept\_aggr table*

*query = "SELECT \* FROM `travel\_time\_matrix\_db`"*

*cursor.execute(query)*

*travel\_time\_db = cursor.fetchall()*

*#print(dept\_aggr\_all)*

*finally:*

*connection.close()*

*return(parameters\_db,stops\_db,travel\_time\_db)*

***SpringClean\_TravelTime.py***

*### This fucntion is to calculate the time taken(fitness) by each vehicle*

*def fitness\_calc(population,travel\_dist\_matrix,stop\_service\_info,service\_time\_window,penalty\_time\_window):*

*population\_temp = population*

*for i in range(len(population)):*

*chromosome\_total = 0*

*for j in range(len(population[i])):*

*Depot\_travel\_time = 0*

*for k in range(len(population[i][j])):*

*vehicle\_route = population[i][j][k]*

*##### Vechile route time funtion to calculate the route times between each stop #######*

*Vehicle\_travel\_time = vechile\_route\_time(vehicle\_route,travel\_dist\_matrix,stop\_service\_info,penalty\_time\_window)*

*Depot\_travel\_time = Depot\_travel\_time + Vehicle\_travel\_time*

*chromosome\_total = chromosome\_total + Depot\_travel\_time*

*population\_temp[i].append(round(chromosome\_total,1))*

*return(population\_temp)*

*### this function is to calculate vehicle time and checks penatly constraints for each vehicle route*

*def vechile\_route\_time(vehicle\_route,travel\_dist\_matrix,stop\_service\_info,penalty\_time\_window):*

*total\_penalty\_time, total\_penalty\_percent, penalty\_time, penalty\_percent = 0, 0, 0, 0*

*stops\_count = len(vehicle\_route) - 2 # substracting first and last depot count from stops count*

*## Calculate the stops threshold penalty*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(2,penalty\_time\_window,stops\_count,' ',0,0)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*Vehicle\_travel\_time, reach\_time = 0, 0*

*penalty\_time, penalty\_percent = 0, 0*

*for l in range(len(vehicle\_route)):*

*if l > 0:*

*from\_stop = vehicle\_route[l-1]*

*to\_stop = vehicle\_route[l]*

*Vehicle\_travel\_time = Vehicle\_travel\_time + travel\_dist\_matrix[from\_stop,to\_stop] + stop\_service\_info[to\_stop,1]*

*reach\_time = reach\_time + stop\_service\_info[to\_stop,1]*

*if l > 1:*

*reach\_time = reach\_time + travel\_dist\_matrix[from\_stop,to\_stop]*

*# Calculates the penatly at every stop based on the reach time and allowed time window*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(3,penalty\_time\_window,0,stop\_service\_info[to\_stop,:],reach\_time,0)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(1,penalty\_time\_window,0,'',0,Vehicle\_travel\_time)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*total\_route\_time\_with\_penalty = Vehicle\_travel\_time + total\_penalty\_time + (total\_penalty\_percent \* Vehicle\_travel\_time)*

*return(total\_route\_time\_with\_penalty)*

*### this function is to calculate vehicle time and checks penatly constraints at every stop.*

*def vechile\_route\_time\_2(vehicle\_route,travel\_dist\_matrix,stop\_service\_info,penalty\_time\_window):*

*total\_penalty\_time, total\_penalty\_percent, penalty\_time, penalty\_percent = 0, 0, 0, 0*

*stops\_count = len(vehicle\_route) - 2 # substracting first and last depot count from stops count*

*### Calculate the stops threshold penalty*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(2,penalty\_time\_window,stops\_count,' ',0,0)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*Vehicle\_travel\_time, reach\_time = 0, 0*

*penalty\_time, penalty\_percent = 0, 0*

*for l in range(len(vehicle\_route)):*

*if l > 0:*

*from\_stop = vehicle\_route[l-1]*

*to\_stop = vehicle\_route[l]*

*Vehicle\_travel\_time = Vehicle\_travel\_time + travel\_dist\_matrix[from\_stop,to\_stop] + stop\_service\_info[to\_stop,1]*

*reach\_time = reach\_time + stop\_service\_info[to\_stop,1]*

*if l > 1:*

*reach\_time = reach\_time + travel\_dist\_matrix[from\_stop,to\_stop]*

*# Calculates the penatly at every stop based on the reach time and allowed time window*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(3,penalty\_time\_window,0,stop\_service\_info[to\_stop,:],reach\_time,0)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*penalty\_time, penalty\_percent = penalty\_cal\_slab\_basis(1,penalty\_time\_window,0,'',0,Vehicle\_travel\_time)*

*total\_penalty\_time = total\_penalty\_time + penalty\_time*

*total\_penalty\_percent = total\_penalty\_percent + penalty\_percent*

*total\_route\_time\_with\_penalty = Vehicle\_travel\_time + total\_penalty\_time + (total\_penalty\_percent \* Vehicle\_travel\_time)*

*return(total\_route\_time\_with\_penalty)*

*## This funtion takes the penalty type and returns the penalty time and percentage of penalty on total route time for a vehicle*

*## penalty\_type catogerized into three to calculate the penalty:*

*## 1 - total travel time penalty, 2 - stops threshold, 3 - before/after defined service window*

*## This function works for customer with different time windows as well.(Not only for time window i.e. 11:00 AM to 14:00 PM)*

*################ Calculate penalty whenever below constraints are not met ###############################################*

*##*

*## ###### NOTE : Not considered below penalty as slab based.. Applied direct.*

*## 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*

*## with in 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*

*##*

*###########################################################################################################################*

*### Function to apply penatly at slab basis if contrainsts are not met.*

*def penalty\_cal\_slab\_basis(penalty\_type,penalty\_parameters,stops\_count,stop\_service\_info,reach\_time,total\_travel\_time):*

*penalty\_time, penalty\_mul\_factor, penalty\_percent = 0,0,0*

*if penalty\_type == 2: ## this is to check for stops threshold*

*if (stops\_count > penalty\_parameters[0]):*

*penalty\_percent = stops\_count - penalty\_parameters[0]*

*penalty\_percent = penalty\_percent \* 0.1*

*elif (stops\_count < penalty\_parameters[1]):*

*penalty\_percent = penalty\_parameters[1] - stops\_count*

*penalty\_percent = penalty\_percent \* 0.1*

*elif penalty\_type == 3: ## penalty before/after defined service window*

*## This is convert service time into minutes for comparion purpose*

*(h, m, s) = stop\_service\_info[2].split(':')*

*time1 = int(h) \* 3600 + int(m) \* 60 + int(s)*

*(h, m, s) = stop\_service\_info[3].split(':')*

*time2 = int(h) \* 3600 + int(m) \* 60 + int(s)*

*allowed\_time = (time2 - time1)/60*

*## Add no penalty window to timediff*

*allowed\_time = allowed\_time + penalty\_parameters[3]*

*if reach\_time > allowed\_time:*

*penalty\_mul\_factor = int((reach\_time - allowed\_time)/60) + 1*

*time\_diff = reach\_time - allowed\_time*

*for i in range(1,penalty\_mul\_factor+1):*

*slab\_check = time\_diff - (60 \* i)*

*if slab\_check < 0:*

*penalty\_time = penalty\_time + (i \* (time\_diff - (60 \* (i-1))))*

*penalty\_percent = penalty\_percent + round(0.1 \* i,2)*

*else:*

*penalty\_time = penalty\_time + ( i \* 60 )*

*penalty\_percent = penalty\_percent + round(0.1 \* i,2)*

*elif penalty\_type == 1: ## total travel time penalty*

*if total\_travel\_time > penalty\_parameters[2]:*

*penalty\_mul\_factor = int((total\_travel\_time - penalty\_parameters[2]) /60) + 1*

*time\_diff = total\_travel\_time - penalty\_parameters[2]*

*for i in range(1,penalty\_mul\_factor+1):*

*slab\_check = time\_diff - (60 \* i)*

*if slab\_check < 0:*

*penalty\_time = penalty\_time + (i \* (time\_diff - (60 \* (i-1))))*

*penalty\_percent = penalty\_percent + round(0.1 \* i,2)*

*else:*

*penalty\_time = penalty\_time + ( i \* 60 )*

*penalty\_percent = penalty\_percent + round(0.1 \* i,2)*

*return(penalty\_time,penalty\_percent)*

*### Function to apply direct penatly on overrall time travelled.*

*def penalty\_percent\_calc(penalty\_type,penalty\_parameters,stops\_count,stop\_service\_info,reach\_time,total\_travel\_time):*

*penalty\_time, penalty\_mul\_factor, penalty\_percent = 0,0,0*

*if penalty\_type == 2: ## this is to check for stops threshold*

*if (stops\_count > penalty\_parameters[0]):*

*penalty\_percent = stops\_count - penalty\_parameters[0]*

*penalty\_percent = penalty\_percent \* 0.1*

*elif (stops\_count < penalty\_parameters[1]):*

*penalty\_percent = penalty\_parameters[1] - stops\_count*

*penalty\_percent = penalty\_percent \* 0.1*

*elif penalty\_type == 3: ## penalty before/after defined service window*

*## This is convert service time into minutes for comparion purpose*

*(h, m, s) = stop\_service\_info[2].split(':')*

*time1 = int(h) \* 3600 + int(m) \* 60 + int(s)*

*(h, m, s) = stop\_service\_info[3].split(':')*

*time2 = int(h) \* 3600 + int(m) \* 60 + int(s)*

*timediff = (time2 - time1)/60*

*## Add no penalty window to timediff*

*timediff = timediff + penalty\_parameters[3]*

*if reach\_time > timediff:*

*penalty\_mul\_factor = int((reach\_time - timediff)/60) + 1*

*penalty\_time = penalty\_mul\_factor \* (reach\_time - timediff)*

*penalty\_percent = round(0.1 \* penalty\_mul\_factor,2)*

*elif penalty\_type == 1: ## total travel time penalty*

*if total\_travel\_time > penalty\_parameters[2]:*

*penalty\_mul\_factor = int((total\_travel\_time - penalty\_parameters[2]) /60) + 1*

*penalty\_time = penalty\_mul\_factor \* (total\_travel\_time - penalty\_parameters[2])*

*penalty\_percent = 0.1 \* penalty\_mul\_factor*

*return(penalty\_time,penalty\_percent)*

***SpringCleanMain.py***

*import numpy as np*

*import pandas as pd*

*import os*

*import copy*

*from time import gmtime, strftime, time*

*from SpringClean\_DataPull import pull\_from\_mySQL*

*from SpringClean\_Genetic\_Algorithm import Genetic\_Algorithm*

*os.chdir('C:/Users/dt81540/Desktop/DataScience/Travelling Salesmen Problem - Project/ROS/')*

*### This is capture the start time*

*start = strftime("%Y-%m-%d %H:%M:%S", gmtime())*

*### This function is to pull spring\_clean data from mySQL database*

*#parameters\_df, stop\_df, travel\_df=pull\_from\_mySQL()*

*#print("parameters\_df :",parameters\_df)*

*parameters\_df = pd.read\_csv('parameters\_info\_db.csv',sep=',')*

*stop\_df = pd.read\_csv('stop\_info\_db.csv',sep=',')*

*travel\_df = pd.read\_csv('travel\_time\_matrix\_db.csv',sep=',')*

*test\_grid = pd.read\_csv('t\_test\_random\_stop\_range.csv',sep=',')*

*### converting dataframes into numpy array for row and column wise operations flexibility*

*parameters\_info\_db = np.array(parameters\_df)*

*stop\_info\_db = np.array(stop\_df)*

*travel\_time\_matrix\_db = np.array(travel\_df)*

*test\_grid\_np = np.array(test\_grid)*

*stop\_unique = np.sort(stop\_info\_db[:,0])*

*### Convert travel time matrix into multi dimentional array and also list all missing stops and corresponding indexes.*

*k = 0*

*j = 0*

*missing\_stops\_idx = []*

*missing\_stops = []*

*## Converting travel time\_matrix into 2D array*

*travel\_time\_MD = np.zeros(shape=[stop\_info\_db.shape[0],stop\_info\_db.shape[0]])*

*for i in range(stop\_info\_db.shape[0]):*

*if stop\_unique[i] == travel\_time\_matrix\_db[k,0]:*

*for l in range(stop\_info\_db.shape[0]):*

*travel\_time\_MD[i,l] = travel\_time\_matrix\_db[k,2]*

*k = k + 1*

*else:*

*missing\_stops\_idx.append(i)*

*missing\_stops.append(stop\_unique[i])*

*###### IMPORTANT NOTE -- This need to be modified ########*

*### this logic is to impute missing stops travel time with assumption that missing stops symmetric.*

*for i in missing\_stops\_idx:*

*travel\_time\_MD[i,:] = travel\_time\_MD[:,i]*

*### dropping similar deopts related columns and rows from array*

*travel\_time\_MD\_unq = np.delete(travel\_time\_MD, [4,5], 1) # Axis = 1 for Columnwise*

*travel\_time\_MD\_unq = np.delete(travel\_time\_MD\_unq, [4,5], 0) # axis = 0 for Row wise*

*stop\_info\_db\_final = np.delete(stop\_info\_db, [4,5], 0) ## to delete duplicate depots*

*###### Genetic\_Alogorithm Parameters ######*

*##Genetic\_Algorithm(travel\_dist\_matrix, ==> two dimentional travel time matrix*

*## stop\_service\_info, ==> stops info database*

*## parameters\_info, ==> parameters info database*

*## stops\_allowed,==> solution stops range: provide 0 for 3 to 5, 1 for 4 to 6 and 2 for 5 to 7*

*## iterations, ==> number of iteration to find a optimum solution*

*## mutate\_per, ==> mutation percentage*

*## elite\_per, ==> elite population percentage*

*## origPopSize): ==> size of population*

*### This logic process the test conditions from csv file and iterates genetic algorithm for different test conditions.*

*optimized\_route = []*

*f = open('bestRoutes\_run\_t\_test\_same.txt', 'w')*

*for test\_iter in range(test\_grid\_np.shape[0]):*

*print("###################### Test condition" , test\_iter + 1 ,"processing started #######################")*

*(h, m, s) = strftime("%H:%M:%S", gmtime()).split(':')*

*test\_start\_time = int(h) \* 3600 + int(m) \* 60 + int(s)*

*### Genetic Alogorithm ####*

*best\_score, tot\_vehicles, best\_route = Genetic\_Algorithm(travel\_time\_MD\_unq,*

*stop\_info\_db\_final,*

*parameters\_info\_db,*

*int(test\_grid\_np[test\_iter,1]),*

*int(test\_grid\_np[test\_iter,2]),*

*test\_grid\_np[test\_iter,3],*

*test\_grid\_np[test\_iter,4],*

*int(test\_grid\_np[test\_iter,5]))*

*test\_grid\_np[test\_iter,9] = best\_score*

*test\_grid\_np[test\_iter,10] = tot\_vehicles*

*test\_grid\_np[test\_iter,11] = np.sum(stop\_info\_db\_final[:,1])*

*test\_grid\_np[test\_iter,12] = float((best\_score \* test\_grid\_np[test\_iter,6]) + (test\_grid\_np[test\_iter,10] \* test\_grid\_np[test\_iter,7]))*

*test\_grid\_np[test\_iter,13] = float(test\_grid\_np[test\_iter,11] \* test\_grid\_np[test\_iter,8])*

*best\_fit = copy.deepcopy(best\_route[0:4])*

*for i in range(len(best\_fit)):*

*for j in range(len(best\_fit[i])):*

*for k in range(len(best\_fit[i][j])):*

*best\_fit[i][j][k] = stop\_info\_db\_final[best\_fit[i][j][k],0]*

*print("Best Route -- ", best\_fit)*

*(h, m, s) = strftime("%H:%M:%S", gmtime()).split(':')*

*test\_end\_time = int(h) \* 3600 + int(m) \* 60 + int(s)*

*test\_grid\_np[test\_iter,14] = (test\_end\_time - test\_start\_time)/60*

*mystr = "Best Route from test condition : " + str(test\_iter + 1) + "\n" + ', '.join([str(veh) for veh in best\_fit]) + "\n" + "\n"*

*f.write(mystr)*

*f.close()*

*test\_grid = pd.DataFrame(test\_grid\_np,columns=test\_grid.columns)*

*test\_grid.to\_csv("test\_grid\_rot\_results\_t\_test\_random\_stop\_same.csv")*

*print("Process Start Time :", start)*

*print("Process End Time :", strftime("%Y-%m-%d %H:%M:%S", gmtime()))*