**ROUTE OPTIMIZATION SOLUTION**

**By**

**Eswar L**

**Reg No: 0704**

**Batch: 17**

**Email:** [**eswar3192L@gmail.com**](mailto:eswar3192L@gmail.com)

Contents

[1. Purpose 3](#_Toc464410575)

[1.1 Problem Description 3](#_Toc464410576)

[1.2 Project scope 3](#_Toc464410577)

[1.3 Constraints 3](#_Toc464410578)

[2. State-of-the-art 4](#_Toc464410579)

[2.1 Route optimization solution 4](#_Toc464410580)

[2.2 Trending Solutions 5](#_Toc464410581)

[3. Method 8](#_Toc464410582)

[3.1 Possibilities 8](#_Toc464410583)

[3.2 Action points 9](#_Toc464410584)

[3.3 Assumptions 9](#_Toc464410585)

[4. Data 10](#_Toc464410586)

[4.1 MYSQL Database Data 10](#_Toc464410587)

[4.2 Data Processing 11](#_Toc464410588)

[4.3 Key Algorithm notes 12](#_Toc464410589)

[5. Results 12](#_Toc464410590)

[5.1 User input 1 12](#_Toc464410591)

[5.2 User input 2 15](#_Toc464410592)

[5.3 User input 3 19](#_Toc464410593)

[6. Analysis 23](#_Toc464410594)

[6.1 Decision point 23](#_Toc464410595)

[6.2 Scope of enhancements 23](#_Toc464410596)

[7. Appendices 24](#_Toc464410597)

# Purpose

## Problem Description

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 an 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. You will be 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 stops 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.

## Project scope

This project is to study closely the problem statement of the company “Spring-Cleaning” and to come up with the optimized vehicle routing algorithm based on the service bookings they receive online or offline from the customers. The goal is to create solutions based on all the constraints given as in the sub-topic 1.3. Plan of action for this problem description is to use advanced optimization techniques to transform the existing data and the constraints to a solution that can save time and money to the company. This problem is referred as “Vehicle Routing Problem” by the industry and solutions are designed based on the objective of the problem. In our case the objective is to “Reduce the total time – Job execution time and Travel time” to serve the customers and plan the depots appropriately with man power and vehicles.

## Constraints

1. Total travel +Service time 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

..

4. Minimize the overall program execution time.

# State-of-the-art

## Route optimization solution

This problem of route optimization is also called as the Vehicle Routing Problem or VRP for short. The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem which asks "What is the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a given set of customers?". It generalises the well-known travelling salesman problem (TSP). It first appeared in a paper by George Dantzig and John Ramser in 1959,in which first algorithmic approach was written and was applied to petrol deliveries. Often, the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. The objective of the VRP is to minimize the total route cost. In 1964, Clarke and Wright improved on Dantzig and Ramser's approach using an effective greedy approach called the savings algorithm.

Determining the optimal solution is an NP-hard problem in combinatorial optimization, so the size of problems that can be solved optimally is limited. The commercial solvers therefore tend to use heuristics due to the size of real world VRPs and the frequency that they may have to be solved.

The VRP concerns the service of a delivery company. How things are delivered from one or more depots which has a given set of home vehicles and operated by a set of drivers who can move on a given road network to a set of customers. It asks for a determination of a set of routes, S, (one route for each vehicle that must start and finish at its own depot) such that all customers' requirements and operational constraints are satisfied and the global transportation cost is minimized. This cost may be monetary, distance or otherwise.

The road network can be described using a graph where the arcs are roads and vertices are junctions between them. The arcs may be directed or undirected due to the possible presence of one way streets or different costs in each direction. Each arc has an associated cost which is generally its length or travel time which may be dependent on vehicle type.

To know the global cost of each route, the travel cost and the travel time between each customer and the depot must be known. To do this our original graph is transformed into one where the vertices are the customers and depot and the arcs are the roads between them. The cost on each arc is the lowest cost between the two points on the original road network. This is easy to do as shortest path problems are relatively easy to solve. This transforms the sparse original graph into a complete graph. For each pair of vertices i and j, there exists an arc (i,j) of the complete graph whose cost is written as and is defined to be the cost of shortest path from i to j. The travel time is the sum of the travel times of the arcs on the shortest path from i to j on the original road graph.

Sometimes it is impossible to satisfy all of a customer's demands and in such cases solvers may reduce some customers' demands or leave some customers unserved. To deal with these situations a priority variable for each customer can be introduced or associated penalties for the partial or lack of service for each customer given.

The objective function of a VRP can be very different depending on the particular application of the result but a few of the more common objectives are:

• Minimize the global transportation cost based on the global distance travelled as well as the fixed costs associated with the used vehicles and drivers

• Minimize the number of vehicles needed to serve all customers

• Least variation in travel time and vehicle load

• Minimize penalties for low quality service

The VRP has many obvious applications in industry. In fact, the use of computer optimization programs can give savings of 5% to a company as transportation is usually a significant component of the cost of a product (10%) - indeed the transportation sector makes up 10% of the EU's GDP. Consequently, any savings created by the VRP, even less than 5%, are significant.

Here, the most commonly used techniques for solving Vehicle Routing Problems are listed. Near all of them are heuristics and meta heuristics because no exact algorithm can be guaranteed to find optimal tours within reasonable computing time when the number of cities is large. This is due to the NP-Hardness of the problem. Next we can find a classification of the solution techniques we have considered.

## Trending Solutions

*Genetic Algorithm:*

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 randomised, 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.". Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

It is better than conventional AI in that it is more robust. Unlike older AI systems, they do not break easily even if the inputs changed slightly, or in the presence of reasonable noise. Also, in searching a large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithm may offer significant benefits over more typical search of optimization techniques. (linear programming, heuristic, depth-first, breath-first, and praxis.)

GAs simulate the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome that we see in our DNA. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of evolution.

GAs are based on an analogy with the genetic structure and behaviour of chromosomes within a population of individuals using the following foundations:

* Individuals in a population compete for resources and mates.
* Those individuals most successful in each 'competition' will produce more offspring than those individuals that perform poorly.
* Genes from `good' individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
* Thus each successive generation will become more suited to their environment.

A population of individuals are is maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet {0,1}. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A fitness score is assigned to each solution representing the abilities of an individual to `compete'. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective `breeding' of the solutions to produce `offspring' better than the parents by combining information from the chromosomes.

The GA maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted. In this way it is hoped that over successive generations better solutions will thrive while the least fit solutions die out.

New generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Each successive generation will contain more good `partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

**Based on Natural Selection**

After an initial population is randomly generated, the algorithm evolves the through three operators:

* selection which equates to survival of the fittest;
* crossover which represents mating between individuals;
* mutation which introduces random modifications.

1. Selection Operator

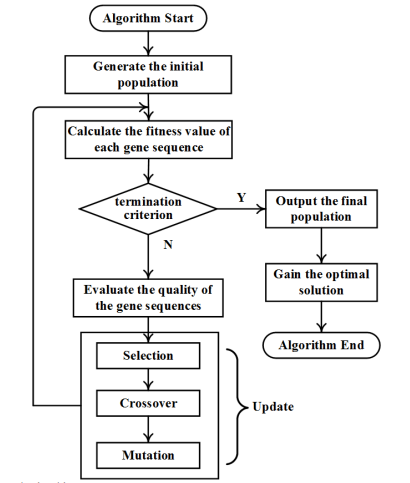
* Key idea: give preference to better individuals, allowing them to pass on their genes to the next generation.
* The goodness of each individual depends on its fitness.
* Fitness may be determined by an objective function or by a subjective judgement.

2. Crossover Operator

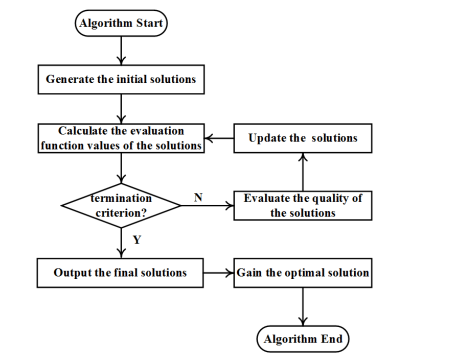
* Prime distinguished factor of GA from other optimization techniques
* Two individuals are chosen from the population using the selection operator
* A crossover site along the bit strings is randomly chosen
* The values of the two strings are exchanged up to this point
* If S1=000000 and s2=111111 and the crossover point is 2 then S1'=110000 and s2'=001111
* The two new offspring created from this mating are put into the next generation of the population
* By recombining portions of good individuals, this process is likely to create even better individuals

3. Mutation Operator

* With some low probability, a portion of the new individuals will have some of their bits flipped.
* Its purpose is to maintain diversity within the population and inhibit premature convergence.
* Mutation alone induces a random walk through the search space
* Mutation and selection (without crossover) create a parallel, noise-tolerant, hill-climbing algorithms



*Simulated Annealing:*



The process of annealing can be simulated with the Metropolis algorithm, which is based on Monte Carlo techniques. We can apply this algorithm to generate a solution to combinatorial optimization problems assuming an analogy between them and physical many-particle systems with the following equivalences:

* Solutions in the problem are equivalent to states in a physical system.
* The cost of a solution is equivalent to the “energy” of a state.
* To apply simulated annealing with optimization purposes we require the following:
* A successor function that returns a “close” neighboring solution given the actual one. This will work as the “disturbance” for the particles of the system.
* A target function to optimize that depends on the current state of the system. This function will work as the energy of the system.
* The distribution used to decide if we accept a bad movement is known as Boltzmann distribution.
* Decrease the temperature slowly; accepting less bad moves at each temperature level until at very low temperatures the algorithm becomes a greedy hill-climbing algorithm.
* This distribution is very well known is in solid physics and plays a central role in simulated annealing. Where γ is the current configuration of the system, E γ is the energy related with it, and Z is normalization constant.

# Method

## Possibilities

* Of the 3 algorithms, picked Genetic Algorithm as it has highest convergence rate with better performance than the other approaches.
* Linear programming for this data is quite a tough to impute constraints on the problem.
* This problem can be addressed in two methods
* Method 1:
  + Applying Genetic Algorithm between all Depots and Service points in the data space
* Method 2:
  + Applying Genetic Algorithm after clustering Service points to its nearest Depots
* Irrespective of the methods algorithm produces better results when the best coding to handle mutations and crossovers is done.

Moving on, Method 2 is implemented using Genetic Algorithm.

Genetic Algorithm terminologies in this problem:

* Gene – Stops: Depots and Service points
* Chromosome – Group of stops for one route/vehicle
* Population – group of chromosomes with all possible solutions
* Crossover – Take a route and bifurcate in to two parts and swap
* Mutation – Pick up a service point from the worst route and add in the chromosome of elite population
* Fitness function – Total time of all the routes from all the Depots

## Action points

* Process the data
* Cluster the service points to its nearest Depot
* Get user input parameters
* Generate initial population
* Create crossover function
* Create mutation function
* Pick up elite population from initial population
* Function to generate new population
* Pickup elite population and apply crossover and mutate
* Iterate the loop until the convergence point
* Convergence points can be Fixed Iterations, No change in Fitness value, etc…
* Visualize the output and its advantages

## Assumptions

* Service data is given for the next working day.
* All the routes start from Depot and end at the same Depot.
* User input parameter on number of stops is considered as the number of service points excluding Depots in the route.
* In each route, first service point is served on time.
* To calculate the total cost, assume fixed costs on each vehicle as 100$ and each minute of time costs 1$.
* If the time travel from one point to other point is 0, then two points are at the same location. Applicable to only Depots.
* All the service points have to be served in a day.
* No customer points to be visited before 11:00 – MaxEarly time and after 14:00 + MaxDelay time. Which is 180 minutes. Used in the algorithm.

# Data

## MYSQL Database Data

stops\_info\_db:

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

- columns:

- 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.

- columns:

- 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.

## Data Processing

* From the data processing, it is identified that 2 of 6 Depots are in the same location as the other two.
* So we can remove the data related to two depots to reduce the data and get faster results.

Depo\_dist <- sqldf ('select \* from data\_travel\_time where FROM\_STOP\_ID like "DEP%" and TO\_STOP\_ID like "DEP%"')

# FROM\_STOP\_ID TO\_STOP\_ID TRAVEL\_TIME

# DEP10001 DEP45024 0

# DEP35024 DEP50002 0

* Removing DEP45024, DEP50002 related data from all the tables given.
* Imputing missing values

#convert travel\_time into a 195x195 matrix like dataframe

travel\_time\_df <- data.frame(matrix(NA, nrow = 195, ncol = 195))

#Assigning Row and Column Names

stopnames <- service\_time$STOP\_ID

colnames(travel\_time\_df) <- stopnames

row.names(travel\_time\_df) <- stopnames

for (i in 1:nrow(travel\_time))

{

travel\_time\_df[travel\_time$FROM\_STOP\_ID[i], travel\_time$TO\_STOP\_ID[i]] <-travel\_time$TRAVEL\_TIME[i]

}

#Imputing missing values(based on colmax)

colMax <- function(data)

{

apply(data,2,FUN=max,na.rm=TRUE )

}

max <- colMax(travel\_time\_df)

#Imputing for missing stops

for (i in c("STP68664","STP69424","STP69799","STP69927","STP70024","STP70151","STP70255","STP70398","STP70434"))

{

for (j in 1:length(stopnames))

{

travel\_time\_df[i, j] <- max[j]

}

}

## Key Algorithm notes

* Mutation probability is given 0.30 and reduced over iteration using formula mut <- mut/i where i the iteration.
* Number of iterations for each cluster is 200 for the current run.
* Maximum population size in the program is 200.
* Elite population is picked after applying penalties or constraints given in the problem description.
* Convergence criteria in the program is based on the number of iterations and not any other parameter.
* Visualizations – Plots on Total costs, Total time captured from elite population on every iteration.

# Results

## User input 1

Minroutes – 3

Maxroutes – 5

Maxearlytime – 30

Maxdelaytime – 30

Iterations – 200

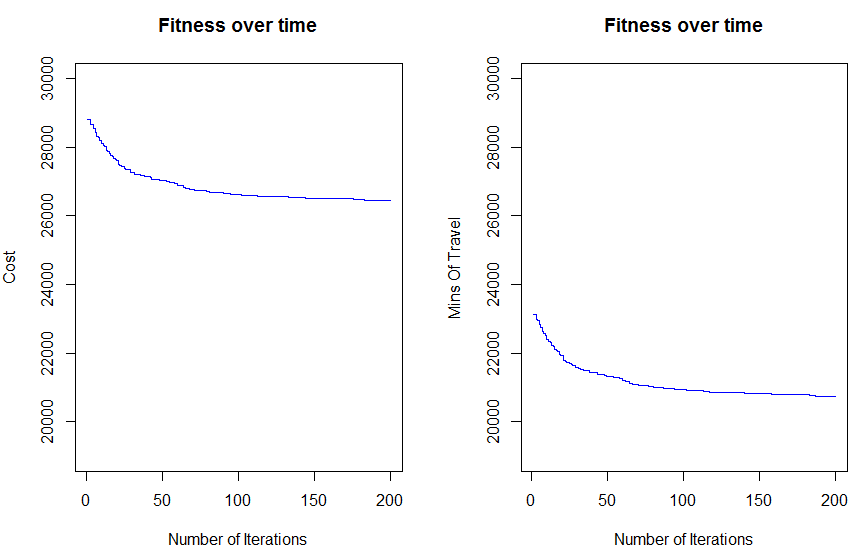
Original cost: 28817.28

Reduced cost: 26440.72

Original time: 23117.28

Reduced Time: 20740.72

Savings: 2300



## User input 2

Minroutes – 4

Maxroutes – 6

Maxearlytime – 30

Maxdelaytime – 30

Iterations – 200

Original cost: 28812.44

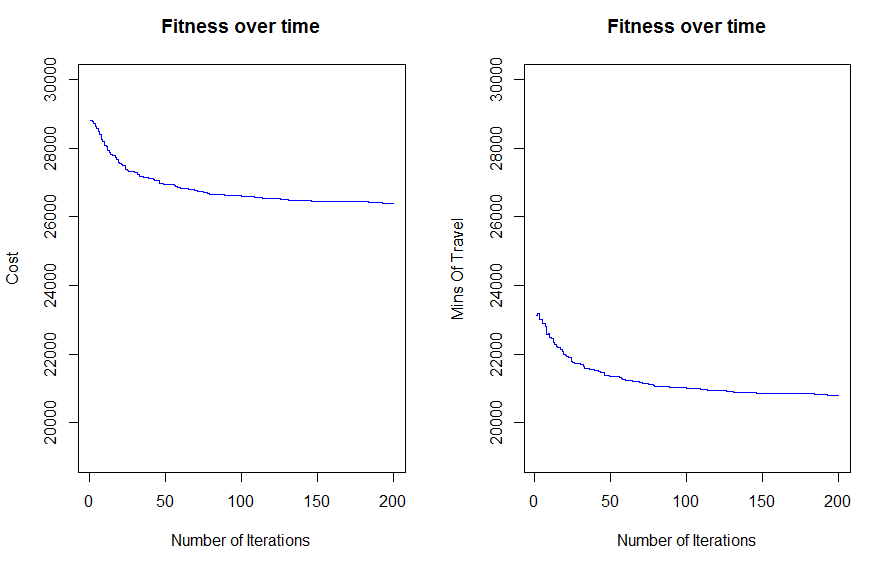
Reduced cost: 26385.82

Savings: 2426

Original time: 23112.44

Reduced time 20785.82

Savings: 2326



## User input 3

Minroutes – 5

Maxroutes – 7

Maxearlytime – 30

Maxdelaytime – 30

Iterations – 200

Original cost: 32740.686

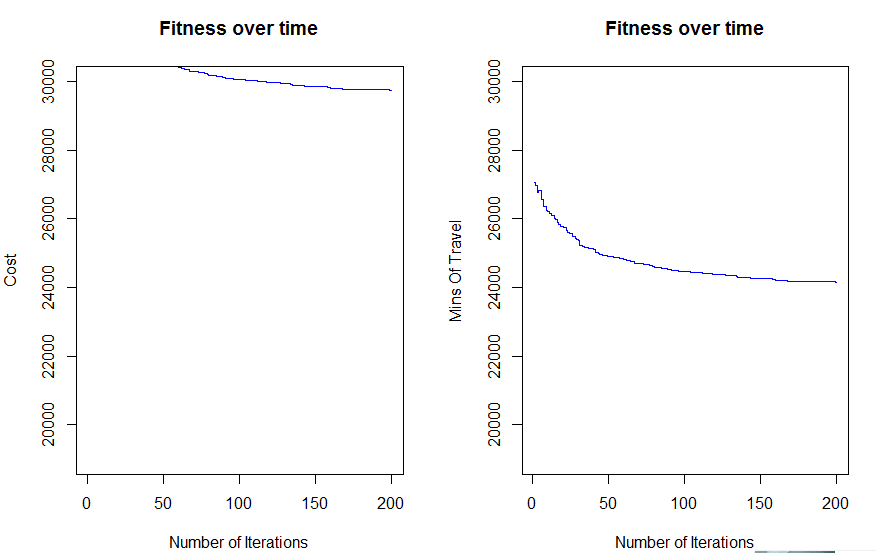
Reduced cost: 29967.34

Savings: 2773

Original time: 27040.686

Reduced time: 24367.34

Savings: 2673



# Analysis

## Decision point

Considering solution with 3 minimum and 5 maximum routes, client spends only 26440 $ per day.

* Assume the similar service bookings happen for the complete month with 22 working days on average, then

## Scope of enhancements

* In the current model:
  + Mutation probabilities can be improved over iterations.
  + Stop Level Clustering could be done
  + Customer loyalty value can be used to as priority to provide service
  + More Random Solutions for depots with more workloads
  + Simulated Annealing can be tried, but GA theoretically and ideally is better over SA.

# Appendices

Genetic Algorithm code:

rm(list = ls(all = TRUE))

library(rlist)

library(base)

library(gdata)

library(Rmalschains)

library(rgp)

library(rgpui)

library(mcga)

library(DBI)

library(RMySQL)

library(ggplot2)

library(gridExtra)

#Given data #Importing from MySQL to R

dbcon <- dbConnect(RMySQL::MySQL(),username = "root",host = "127.0.0.1",port = 3306,dbname = "springs\_clean",password = .rs.askForPassword("Enter password:"))

travel\_time <- dbReadTable(dbcon, "travel\_time\_matrix\_db")

service\_time <- dbReadTable(dbcon, "stops\_info\_db")

penalty\_db <- dbReadTable(dbcon, "parameters\_info\_db")

#Service Time Visualization

stime<-service\_time[7:195,]

length<-length(stime$TIME\_TO\_COMPLETE\_WORK)

Service\_Time <-stime$TIME\_TO\_COMPLETE\_WORK

p1 <- ggplot(stime, aes(x =c(1:length) , y = stime$TIME\_TO\_COMPLETE\_WORK))

p2<-p1 + labs(x="Stops",y = "Service Time") + scale\_y\_continuous("Service Time",limits=c(1,300),breaks=seq(0, 300, 50))

p3<-p2 + geom\_point(aes(color = factor(Service\_Time )))

p4<- p3+ scale\_color\_discrete(name ="Service Time")

plot(p4)

summary(Service\_Time)

#convert travel\_time into a 195x195 matrix like dataframe

travel\_time\_df <- data.frame(matrix(NA, nrow = 195, ncol = 195))

#Assigning Row and Column Names

stopnames <- service\_time$STOP\_ID

colnames(travel\_time\_df) <- stopnames

row.names(travel\_time\_df) <- stopnames

for (i in 1:nrow(travel\_time))

{

travel\_time\_df[travel\_time$FROM\_STOP\_ID[i], travel\_time$TO\_STOP\_ID[i]] <-travel\_time$TRAVEL\_TIME[i]

}

#Imputing missing values(based on colmax)

colMax <- function(data)

{

apply(data,2,FUN=max,na.rm=TRUE )

}

max <- colMax(travel\_time\_df)

#Imputing for missing stops

for (i in c("STP68664","STP69424","STP69799","STP69927","STP70024","STP70151","STP70255","STP70398","STP70434"))

{

for (j in 1:length(stopnames))

{

travel\_time\_df[i, j] <- max[j]

}

}

#cluster stops nearer to depots using average distance

depot\_to\_stop <-as.matrix(travel\_time\_df[1:6,1:195])

stop\_to\_depot <-t(travel\_time\_df[1:195,1:6])

avg\_distance<-as.data.frame((depot\_to\_stop+stop\_to\_depot)/2)

#assigning depot numbers as first element (d1,d2,...d6)

cluster<-list()

for(j in 1:6)

{

cluster[[j]]<-j

}

#clustering stops based on shortest time to travel from depots

for(i in 7:195)

{

temp <-which(avg\_distance[,i] == min(avg\_distance[,i]))

if(length(temp)==1)

{

cluster[[temp]] <-c(cluster[[temp]],i)

}

else

{

min<-length(stopnames)

depot<-0

for(ii in 1:length(temp))

{

if(length(cluster[[temp[ii]]])<min)

{

min<-length(cluster[[temp[ii]]])

depot<-temp[ii]

}

}

cluster[[depot]] <-c(cluster[[depot]],i)

}

}

#Reading input parameters from user

input\_param <- function()

{

cat("Enter Min stops per vehicle")

m <- as.integer(readline(prompt = ""))

cat("Enter Max stops per vehicle")

n <- as.integer(readline(prompt = ""))

cat("Enter Max Route Time \n")

o<-as.integer(readline(prompt = ""))

cat("Enter Max\_Early Time Windows\n")

p<-as.integer(readline(prompt = ""))

cat("Enter Max\_Late Time Windows \n")

q<-as.integer(readline(prompt = ""))

cat("Enter Fixed Cost Per Vehicle ")

r<-as.integer(readline(prompt = ""))

cat("Enter Fixed Cost for Per Min Travel ")

s<-as.integer(readline(prompt = ""))

return(c(m,n,o,p,q,r,s))

}

# Calculate Total Route Time(trt) (trt =Travel Time + Service Time)

trtcalc <-function(x)

{

stime <-0

ttime <-0

for(kk in 1:(length(x)-1))

{

ttime=as.numeric(travel\_time\_df[x[kk],x[kk+1]])+ttime

stime=service\_time[x[kk],2]+stime

}

t <-stime+ttime

return(t)

}

#Generating initial population

fn\_initpop <-function(x,Minstop,Maxstop)

{

set.seed(1234\*Maxstop)

mlist<-list()

depot<-x[1]

x<-setdiff(x, depot)

for (ii in 1:InitPopSize)

{

clist <-x

templist<-list()

for(jj in 1: length(clist))

{

if (length(clist) >=MaxStops)

{

a <-resample(clist,size = 3,replace = FALSE)

clist<-setdiff(clist, a)

b<-trtcalc(c(depot,a,depot))

zz<-trtcalc(a)

if(b>=MaxRTime)

{

templist[[jj]] <-c(depot,a,depot)

}

else

{

c<-NULL

while((length(a)<MaxStops) & (length(clist)> 0) & zz<180)

{

c <-resample(clist,size = 1)

b<-trtcalc(c(depot,a,c,depot))

zz<-trtcalc(c(a,c))

if(b<=MaxRTime & zz<=180)

{

a<-c(a,c)

clist<-setdiff(clist, a)

}

else

{

break

}

}

templist[[jj]] <-c(depot,a,depot)

}

}

else if ((length(clist)<MaxStops) & (length(clist) > 0))

{

a <- resample(clist, size =length(clist),replace = FALSE)

templist[[jj]] <-c(depot,a,depot)

clist<-setdiff(clist, a)

}

if (length(clist) == 0)

break

}

mlist[[ii]]<-templist

}

return(mlist)

}

#We define the Penalty functions as follows.

# Max Route Time (660 mins check)

penalty1 <-function(x)

{

if(x>MaxRTime)

{

c <-trunc((x-MaxRTime)/60)+1

c<-(c\*10\*x)/100

return(x+c)

}

else

{

return(x)

}

}

#Minimum Number Of Stops in a route check

penalty2 <-function(x,y)

{

d <-length(x)-2

if(d<MinStops)

{

sc<-(MinStops-d)\*10

sc<-((sc)/100)\*y

sc<-y+sc

return(sc)

}

else

return(y)

}

#vehicles should reach customers in the slot(11Am -2 PM)

penalty3 <-function(x,y)

{

# x is route, y is score of the route

if(y<=180)

{

return(y)

}

else

{

x<-setdiff(x,x[1])

if(length(x)==1)

{

return(y)

}

else

{

stime <-0

ttime <-0

for(kk in 1:(length(x)-1))

{

ttime=as.numeric(travel\_time\_df[x[kk],x[kk+1]])+ttime

stime=service\_time[x[kk],2]+stime

}

ptime <-stime+ttime

zz<-180+unlist(ETWindow)+unlist(LTWindow)

if(ptime<=zz)

{

z<-y+0.1\*y

return(z)

}

else

{

diff<-(ptime-(180+ETWindow+LTWindow))/60

diff<-trunc(diff)+1

diff<-(diff\*10\*y)/100

z<-y+diff

return(z)

}

}

}

}

#We define the Cost/Objective function as follows.

#Final Score Per Solution

fscore <-function(x)

{

sum(unlist(x))

}

#Route Count Per Solution

Rcount <-function(x)

{

length(x)

}

# 3) Cost Per Solution

cost <-function(x,y)

{

# x--Route count, Y score value

return((CPRoute\*x)+(CPMin\*y))

}

#Function to pick elite solution

pickElite <-function(x,y)

{

x[c(y)]

}

#function nto generate new population

fnnewgen <- function(x,y,z)

{

while (length(x) < InitPopSize)

{

# Mutation

if (runif(1,0,1)< z)

{

parent<-resample(x,size =1)

temp<-parent[[1]]

t<-lapply(temp, function(z) swap(z))

x<-list.append(x,t)

}

# Crossover

else

{

y<-setdiff(y,y[1])

# length of elements to be swapped

elements\_to\_be\_swapped<-resample(y,size = round((resample(0.90:0.95, size = 1, replace = F)\*length(y)/10),0),replace = FALSE)

#Pick 2 parents

parents<-resample(x,size =2,replace = FALSE)

while(length(elements\_to\_be\_swapped)>1)

{

d <-resample(elements\_to\_be\_swapped,size = 2, replace = FALSE)

elements\_to\_be\_swapped<-setdiff(elements\_to\_be\_swapped,d)

parents<-rapply(parents, function(z) ifelse(z == d[1],d[2],ifelse(z==d[2],d[1], z)), how = "list")

}

x<-list.append(x,parents[[1]])

x<-list.append(x,parents[[2]])

}

}

return(x)

}

swap <-function(x)

{

if(length(x)==3)

return(x)

else

{

temp<-x[length(x)-1]

x[length(x)-1]<-x[2]

x[2]<-temp

return(x)

}

}

#Print function for best results in every iteration

bestsolution <-function(cost,best)

{

bcost<-mapply(function(x,y) x[[unlist(y)]] ,cost,best)

return(sum(bcost))

}

# Execute the genetic algorithm

geneticAlgo<- function(PopList,maxiterations,mutProb,elite\_percent)

{

cat("max iterations =", maxiterations, "\n")

#Applying Penalty on Inital Population

rt<-rapply(PopList, function(x) trtcalc(x),how = "list")

p1<-rapply(rt, function(x) penalty1(x),how = "list")

p2 <-mapply(function(x,y) mapply(function(x,y) mapply(function(x,y) penalty2(x,y),x,y,SIMPLIFY=FALSE),x,y,SIMPLIFY=FALSE),PopList,p1,SIMPLIFY=FALSE)

p3<-mapply(function(x,y) mapply(function(x,y) mapply(function(x,y) penalty3(x,y),x,y,SIMPLIFY=FALSE),x,y,SIMPLIFY=FALSE),PopList,p2,SIMPLIFY=FALSE)

#Applying Cost function on Inital Population

Route\_count<-lapply(PopList,function(x) lapply(x,function(x) Rcount(x) ))

f\_score <-lapply(p3,function(x) lapply(x,function(x) fscore(x)))

c<-mapply(function(x,y) mapply(function(x,y) cost(x,y),x,y,SIMPLIFY = FALSE),Route\_count,f\_score,SIMPLIFY=FALSE)

sorted <-lapply(c,function(x) order(unlist(x),decreasing = FALSE))

#Calculate Elite Population Percentage

elite=round(elite\_percent\*InitPopSize,0)

sorted <-lapply(sorted,function(x) x[1:elite])

# Main Iteration Loop

best\_cost<-c()

best\_min<-c()

best\_Route\_count<-c()

for (i in 1:maxiterations)

{

##Pick Elite Population

Elitepop <-mapply(function(x,y) pickElite(x,y),PopList,sorted,SIMPLIFY = FALSE)

#Mutation Probability

mut =mutProb/i

#Generate New Gen Population

PopList<-mapply(function(x,y,z) fnnewgen(x,y,z),Elitepop,cluster,mutProb,SIMPLIFY = FALSE)

#Penalty functins on new gen

rt<- rapply(PopList, function(x) trtcalc(x),how = "list")

p1<-rapply(rt, function(x) penalty1(x),how = "list")

p2 <-mapply(function(x,y) mapply(function(x,y) mapply(function(x,y) penalty2(x,y),x,y,SIMPLIFY=FALSE),x,y,SIMPLIFY=FALSE),PopList,p1,SIMPLIFY=FALSE)

p3<-mapply(function(x,y) mapply(function(x,y) mapply(function(x,y) penalty3(x,y),x,y,SIMPLIFY=FALSE),x,y,SIMPLIFY=FALSE),PopList,p2,SIMPLIFY=FALSE)

# Cost functions on new gen

Route\_count<-lapply(PopList,function(x) lapply(x,function(x) length(x) ))

f\_score <-lapply(p3,function(x) lapply(x,function(x) fscore(x)))

c<-mapply(function(x,y) mapply(function(x,y) cost(x,y),x,y,SIMPLIFY = FALSE),Route\_count,f\_score,SIMPLIFY=FALSE)

sorted <-lapply(c,function(x) order(unlist(x),decreasing = FALSE))

#Pick Elite

sorted <-lapply(sorted,function(x) x[1:elite])

cat("Iteration", i, "\n")

##Print Best solution of the iterarion

best<-sapply(sorted, `[[`, 1,simplify = F)

best\_cost[i]<- bestsolution(c,best)

best\_min[i]<-bestsolution(f\_score,best)

best\_Route\_count[i]<-bestsolution(Route\_count,best)

best\_sol<-mapply(function(x,y) x[[unlist(y)]] ,PopList,best,SIMPLIFY = FALSE)

print(paste0(best\_sol))

print(paste0("best cost in iteration ", i, "=",best\_cost[i]))

print(paste0("best Mins in iteration ", i, "=",best\_min[i]))

print(paste0("best No of routes iteration ", i, "=",best\_Route\_count[i]))

}

par(mfrow=c(1,2))

plot(c(1:length(best\_cost)),best\_cost,xlab="Number of Iterations",ylab="Cost",xlim=c(1,maxiterations),ylim=c(19000,30000),type='s',col="blue",main="Fitness over time")

plot(c(1:length(best\_min)),best\_min,xlab="Number of Iterations",ylab="Mins Of Travel",xlim=c(1,maxiterations),ylim=c(19000,30000),type='s',col="blue",main="Fitness over time")

return("success")

}

#Reading Parameters

param<-input\_param()

MinStops<-param[1]

MaxStops<-param[2]

MaxRTime<-param[3]

ETWindow<-param[4]

LTWindow<-param[5]

CPRoute<-param[6]

CPMin<-param[7]

InitPopSize=200

#Generate Initial population

InitPopList<-mapply(function(x,y,z) fn\_initpop(x,y,z),cluster,MinStops,MaxStops,SIMPLIFY = FALSE)

#Execute Genetic Algorithm+

system.time(geneticAlgo(InitPopList,200,0.3,0.35))