Bournemouth University

Faculty of Science and Technology

Department of Computing and Informatics

Search and Optimisation Assignment - 2021/22

Group 08

Names: Abhilash Singh Bharatpur, Sidhart Negi, Zoya Ahmad, Sanjay Negi

Problem Definition

We have been employed by **SaO Gas Ltd** as **data scientists**. We are to produce delivery schedules for their fleet of **25 tanker lorries** operating from **5 depots** in the country of Optilandia (which surprisingly use £ as their currency), minimising the overall cost of delivery for the distributor under certain constraints.

The speed of tank lorry is 50 miles/hour on average by considering safety, regardless of the lorry's capacity. There are multiple objectives of the delivery schedules to be considered, including minimizing the overall dispatch time used, minimizing the overall cost of delivery for the distributor, whilst certain constraints may be applied to the objectives, which will be detailed in the optimization problems below.

Tanker type	Capacity [tonnes]	Cost per mile [£]	Cost per mile per tonne [£]
Small	5	1.00	1.50
Medium	12	1.60	1.00
Large	22	2.20	0. 50

As an example of the cost calculating, a large tanker can be loaded with up to 22 tonnes of LPG at one of the depots (you can assume that depots never run out of gas). It costs the distributor £2.2 per mile to use the large tanker (even if it is empty) plus £0. 50 per mile for every tonne of LPG the tanker carries. So, a large tanker with 10 tonnes of LPG travelling 20 miles costs: 20

[miles] x (2. 2 [£] + 10 [tonnes] x 0. 50[£]) = 144 [£]. However, after dropping 2 tonnes of LPG at a customer, the next 20 miles travelled will cost less as the lorry is now lighter: 20 [miles] x (2.2 [£] + 8 [tonnes] x 0.50 [£]) = 124 [£].

As an example of the dispatch time calculating, the loading time at depot can be omitted (not included as dispatch time), however, the travel time and offloading time will be included into the dispatch time. The offloading time consists of constant fixing time of 5 minutes at a customer, and 10 minutes per tonne for dropping the gas from tank to customer. With above mentioned large tank with 10 tonnes of LPG travelling 20 miles to drop 2 tonnes to a customer, the dispatch time imposed is (20 [miles] / 50 [miles/ hours]) x 60 (x 60 is to change to minutes) + 2 tonnes * 10 [minutes/tonne] + 5 minutes = 49 minutes.

Problem 1

Winter is coming, so fully fill all customers' tanks is urgent. Schedule LPG delivery to all customers in order to fully fill their tanks while minimising the overall dispatch time. There are no additional constrains apart from the tanker lorry capacity, and the overall delivery cost is not a concerned objective.

Problem 2

2. Schedule LPG delivery in order to maximise the cost efficiency, where the cost efficiency is defined as Cost efficiency = total gas delivered to all customers / overall cost of delivery, while observing the following constraints in addition to tanker lorry capacity:

   a. Each lorry must stop for 20 minutes for a rest after 2 hours continuous driving

   b. The lorry with small tank can only stop up to 4 times, the lorry with medium tank can only stop up to 8 times, the lorry with medium tank can only stop up to 16 times, this only includes customer deliveries.

   c. Each lorry can refill and end its journey at any depot.

   d. At the end, all customers should have more than 50% of gas in their tanks, or the Gas Ltd will receive complaint from the customer, and will incur a penalty of £1,000 for each such customer.

Methodology

To solve the first optimization problem to minimize dispatch time while fully filling customer's tanks with LPG gas. We are going to assign customers to depots closer to them, to do this we are using dijkstra algorithm which will help in finding the shortest path between evey two nodes in the network. Now, we will have customers assigned to the depot nearest to them. We are going to use Greedy Search method to solve first optimization problem of minimizing dispatch time. We have written a class named greedy which would return the fitness of a sequence. In our case sequence is the set of customer and routes. The lorry jounery would end if all the customers have been served or the demand of the next customer is more than the gas left in the lorry. All the lorries flee at once and will return to depot to refill only if they run out of gas or all deliveries are made.

The second optimization problem is solved using genetic algorithm. Genetic algorithm requires population function, fitness function, parent selection function, crossover function, mutation function. Starting with population function, as we have a list of customers assigned to each depot these customers are randomly swapped in the list and population is generated. Next we have assigned lorries to customers from the list using genotypetophenotype function. In the next step fitness is calculated and we have used rank sequence function to select parents for crossover based on their rank. Ranking is done based on the fitness score. Later, the crossover is done to produce a offspring. Mutation function is used to mutate in the next step and a new generation is produced. This process continues until the number of generations are given and returns the best route.

Importing necessary libraries

In [1]:

```
import numpy as np
import pandas as pd
import random
import math
import networkx as nx
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform
```

Load data

Customer data

In [2]:

```
## read the files

edge_df=pd.read_csv("SaO_Optilandia_links.csv")
locs = pd.read_csv('SaO_Optilandia_locations.csv')
#calculating demand adding it to locs dataframe
locs['demand'] = locs['capacity'] - locs['level']
locs.head()
```

Out[2]:

	id	x	y	is_depot	is_customer	capacity	level	demand
0	0	49464.6277	5928.80907	False	False	NaN	NaN	NaN
1	1	49694.4968	6336.56398	False	False	NaN	NaN	NaN
2	2	49568.0405	6390.31359	False	False	NaN	NaN	NaN
3	3	49746.6430	6162.56676	False	False	NaN	NaN	NaN
4	4	49779.8686	5761.12212	False	False	NaN	NaN	NaN

Lorry data

We load the details of lorries in a dataframe

In [ ]:

```python
import json

# Opening JSON file
f = open('SaO_Optilandia_depot_lorries.json')

# returns JSON object as a dictionary
data = json.load(f)

f.close()


df_523 = pd.DataFrame(data['523'])
df_124 = pd.DataFrame(data['124'])
df_373 = pd.DataFrame(data['373'])
df_167 = pd.DataFrame(data['167'])
df_127 = pd.DataFrame(data['127'])

lorry_df = pd.concat([df_124, df_127, df_167, df_373, df_523])
lorry_df = lorry_df.reset_index(drop = True)

# "depot" having the depot_id of the each lorry.
lorry_df['depot']=lorry_df['lorry_id'].apply(lambda x: int(x.split('-')[0]))


lorry_df.head(10)
```

Visualize network

Create a graph

In [4]:

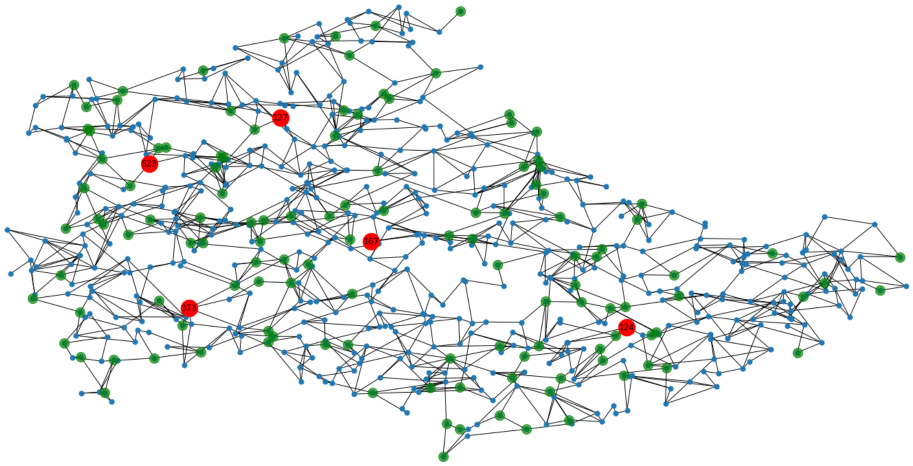
```python
depot_loc, customer_loc= np.where(locs.is_depot)[0], np.where (locs.is_customer

depot_loc_labels={}
for i in depot_loc:
    depot_loc_labels.update({i:i})

# Calculate the Euclidean distance between every two nodes in the network
dist_ecu = squareform(pdist(locs[['x','y']]))

edges = [(id1,id2, dist_ecu[id1,id2]) for _,(id1,id2) in edge_df.iterrows()]
pos = {k:v.values for k,v in locs[['x','y']].iterrows()}


G = nx.Graph()
G.add_nodes_from(locs['id'].to_numpy())
G.add_weighted_edges_from(edges)
```

In [5]:

```
def network( locs_df, graph, regions = False, **details):
    '''

    function to draw network.

    If regions = False:
        This function will draw the full network with all the customers and all

    else:
        This function will draw the network of a region allocated to a depot.

        ***NOTE :  in this case you need to supply the

        depot_id : id of the depot ; type = int
        customer_ids : ids of customers allocated to the depot; type = list

        as the args.**

    '''

    plt.figure(figsize=(20,10))

    depot_ids = np.where(locs_df.is_depot)[0]

    depot_loc_labels={}
    for i in depot_ids:
        depot_loc_labels.update({i:i})


    if not regions:

        customer_ids = np.where (locs.is_customer)[0]

        nx.draw(graph, with_labels = False, pos = pos, node_size=50)
        nx.draw_networkx_nodes(graph, pos = pos, nodelist= depot_ids, node_colo
        nx.draw_networkx_labels(graph, pos, depot_loc_labels)
        nx.draw_networkx_nodes(graph, pos, nodelist = customer_ids, node_color

        plt.show()

    else:

        depot_id = details['depot_id']
        depot_labels={depot_id : depot_id}

        customer_ids = details['customer_ids']

        plt.title(f"Customers allocated for depot {depot_id} ")
        nx.draw(graph, with_labels = False, pos = pos, node_size = 50)
        nx.draw_networkx_nodes(graph, pos = pos, nodelist = depot_ids, node_col
        nx.draw_networkx_nodes(graph, pos = pos, nodelist = [depot_id], node_co
        nx.draw_networkx_labels(graph, pos, depot_labels)
        nx.draw_networkx_nodes(graph, pos, nodelist = customer_ids, node_color

        plt.show()
```

In [6]:

```
network(locs, G, regions = False)
```



In the above graph, the red nodes represents the location of the depots and the green nodes represent the location of the customers.

assign customers to depots

In [7]:

```
## Shortest path distance between every two nodes in the network

dist_matrix = np.zeros([len(locs),len(locs)])

temp = nx.all_pairs_dijkstra_path_length(G)

for node, paths in temp:
    for i in sorted(paths):
        dist_matrix[node,i] = dist_matrix[i,node] = paths[i]
```

In [8]:

```
depot = {124:[], 127:[], 167:[], 373:[], 523:[]}


def depot_allocation(customer_location):

    dist={}
    for i in depot_loc:


        dist.update({i:dist_matrix[i,customer_location]})

    nearest_depot=list(dist.keys())[list(dist.values()).index(min(dist.values()

    depot[nearest_depot].append(customer_location)


for i in customer_loc:
    depot_allocation(i)
```

In [9]:

```
for i in depot:
    print(f'No. of customers alloted to depot {i} (based on distance) are: ', l
```

```
No. of customers alloted to depot 124 (based on distance) are:
37
No. of customers alloted to depot 127 (based on distance) are:
23
No. of customers alloted to depot 167 (based on distance) are:
19
No. of customers alloted to depot 373 (based on distance) are:
30
No. of customers alloted to depot 523 (based on distance) are:
16
```

In [10]:

```
# example
depot[523]
```

Out[10]:

```
[8, 27, 70, 94, 136, 183, 225, 235, 257, 276, 378, 391, 491, 51
9, 603, 632]
```

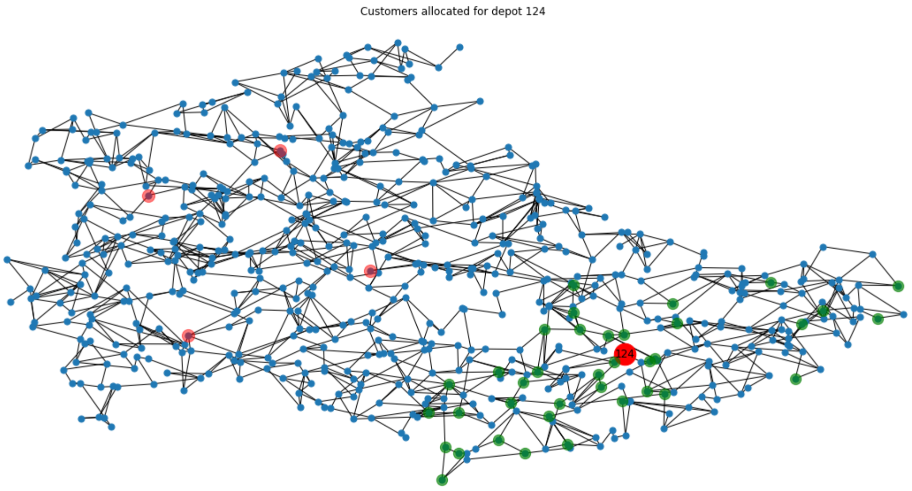
In [11]:

```
# make a copy of the depot allocations dictionary
import copy

depot_customers = copy.deepcopy(depot)
```

Visualising regions

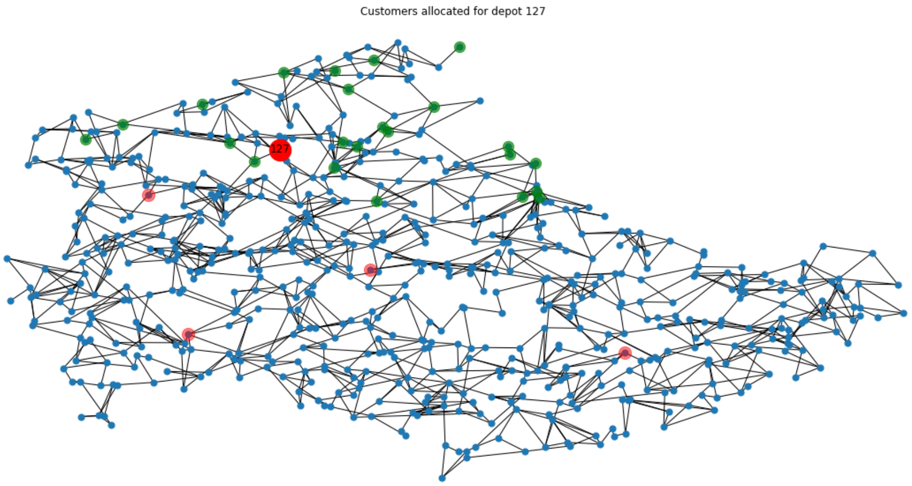
In [12]:

```
network(locs, G, regions = True, depot_id = 124, customer_ids = depot[124] )
```



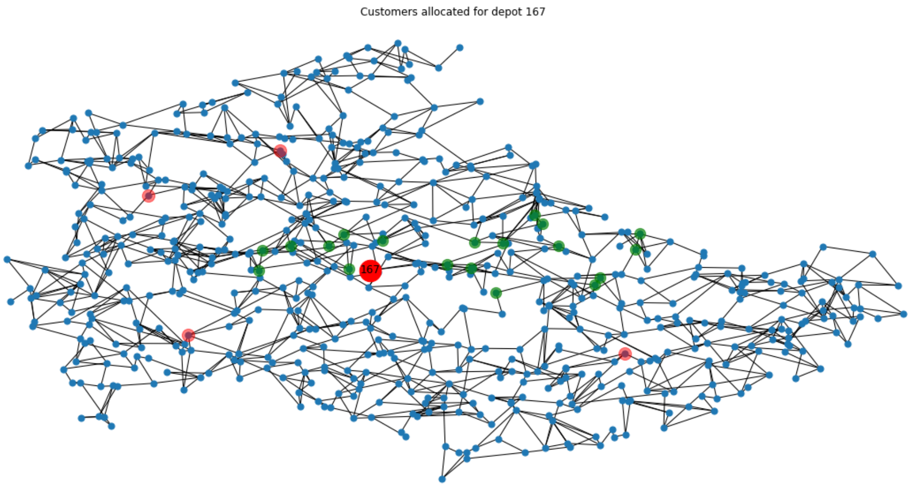
In [13]:

```
network(locs, G, regions = True, depot_id = 127, customer_ids = depot[127])
```



In [14]:

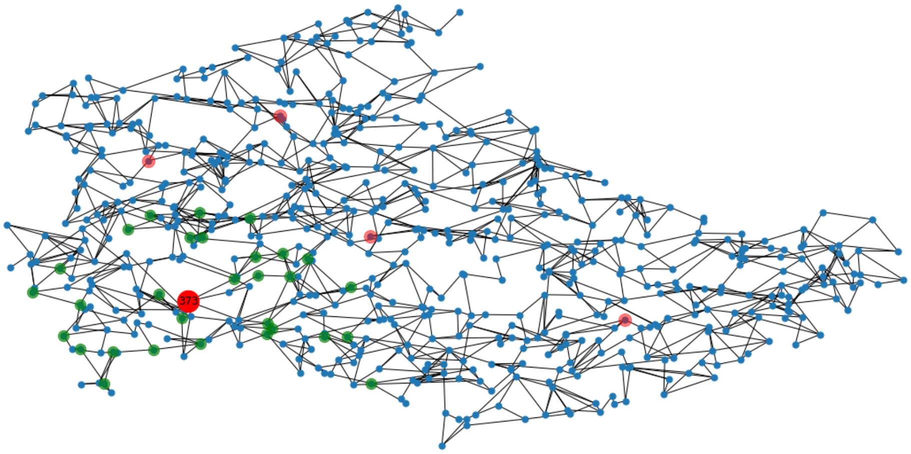
```
network(locs, G, regions = True, depot_id = 167, customer_ids = depot[167] )
```



In [15]:

```
network(locs, G, regions = True, depot_id = 373, customer_ids = depot[373] )
```

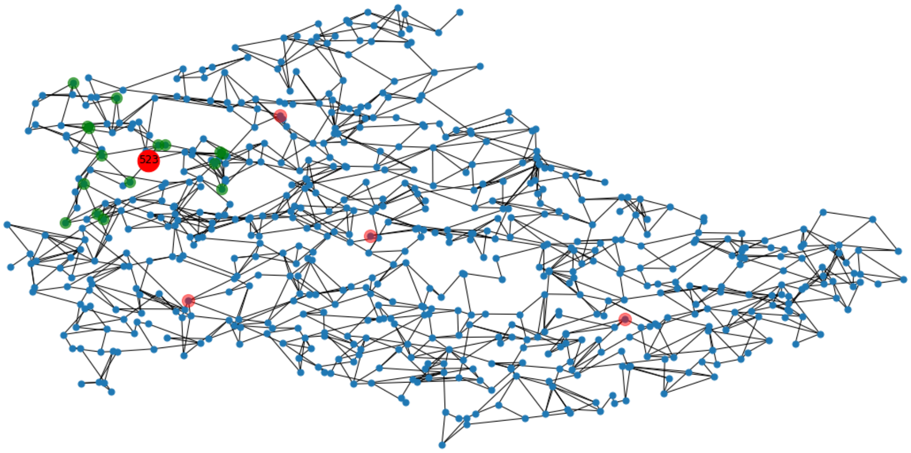
Customers allocated for depot 373



In [16]:

```
network(locs, G, regions = True, depot_id = 523, customer_ids = depot[523] )
```

Customers allocated for depot 523



PROBLEM 1

Objective : To minimize dispatch time of all the deliveries.

Greedy Search

In [17]:

```
# class which will help to calculate the fitness of a sequence.
class Greedy:

    def __init__(self, lorry_df, depot_allocations):

        self.depot_df = lorry_df
        self.depot_loc = [124, 127, 167, 373, 523]
        self.depot_allocations = depot_allocations

        self.capacity = self.depot_df.capacity.to_list()
        self.routes = {}
        self.vehicles_used = {}
        self.total_time = []



    def dist(self, xa, ya, xb ,yb):
        '''
        function to calculate euclidean distance

        '''
        d = np.sqrt( np.square(xa-xb) + np.square(ya-yb) )

        return d




    def cal_time(self, id1, id2, df):
        '''
        function to calculate time
        '''

        if id1 == id2:
            return 0.0

        else:

            from_ = df[ df['id'] == id1]
            to_ = df[ df['id'] == id2]


            xa = int(from_['x'])
            ya = int(from_['y'])


            xb = int(to_['x'])
            yb = int(to_['y'])
            distance = self.dist(xa,ya, xb,yb)

            time = (( distance / 50) * 60)

            return float(time)


    def nearest_customer(self, curr_loc, customer_list):
        '''
        function to get the nearest customer location along with the time of tr
        '''

        dist={}

        for i in customer_list:

            if i != curr_loc:
                dist.update({i : self.cal_time(i, curr_loc, locs)})

        res = {key: val for key, val in sorted(dist.items(), key = lambda ele:

        return [list(res.keys())[0],res[list(res.keys())[0]]]



    def nearest_depot(self, current_loc):
        '''
        function to get the nearest depot from current location
        '''

        dist={}

        for i in self.depot_loc:
            dist.update({i : self.cal_time(i, current_loc, locs)})

        res = {key: val for key, val in sorted(dist.items(), key = lambda ele:

        return [list(res.keys())[0], res[list(res.keys())[0]]]



    def journey(self, lorry_id, customer_list):
        '''
        function to run a journey till the lorry capacity is more than demand o
        has been made.
        '''

        # intialisation
        lorry_capacity = int(self.depot_df[self.depot_df['lorry_id'] == lorry_i

        # set the current location and depot
        curr_loc = int(self.depot_df[self.depot_df['lorry_id'] == lorry_id]['de
        curr_depot = int(self.depot_df[self.depot_df['lorry_id'] == lorry_id]['

        # update journey
        self.routes[lorry_id] =[]
        self.routes[lorry_id].append((curr_loc, lorry_capacity))

        # initialise next customer capacity
        next_customer_capacity = 0


        # run loop till the lorry capacity at current location is more than the
        # or till all the deliveries are made
        while ( lorry_capacity >= next_customer_capacity) and (len (customer_li

            # get the next closest location to the current location
            next_loc, time_of_travel = self.nearest_customer(curr_loc, customer

            # get the demand of the next location
            demand = float(round(locs.loc[next_loc]['demand'],2))


            # Updating the journey
            self.routes[lorry_id].append((next_loc,-demand))


            # add offloading time to the time of travel
            time_of_travel += (((demand * 10) + 5))
            self.total_time.append(time_of_travel)


            # make delivery and update lorry capacity
            lorry_capacity -= demand


            # removing the location to which lpg has been delivered from the cu
            customer_list.remove(next_loc)


            # update the current location
            curr_loc = next_loc

            # if the all the deliveries are not complete then update the next_c
            if len(customer_list)!=0:
                next_location, _ = self.nearest_customer(curr_loc, customer_lis
                next_customer_capacity = float(round(locs.loc[next_location]['d



        # AFTER THE WHILE LOOOP, THE JOURNEY CAN END TWO WAYS

        # 1. if lorry capacity is less than the demand of the next customer the
        # depot with the updated customer list to avoid double delivery and end
        if lorry_capacity < next_customer_capacity:

            # update the customer list of the depot (the customers to whom the
            self.depot_allocations[curr_depot] = customer_list
            print(f"Lorry {lorry_id} ended its journey at (not carrying enough




        # 2. if all the deliveries have been made then end the lorry journey
        elif len(customer_list) == 0:
            print(f"Lorry {lorry_id} ended its journey at (no customers left to



    def run(self):

        # for each depot
        for region, depot in enumerate(self.depot_loc):

            print(f'\n\tREGION: {region + 1}')
            # get the capacity of each lorry
            lorry_ids = self.depot_df[self.depot_df['depot'] == depot]['lorry_i
            lorry_capacities = self.depot_df[self.depot_df['depot'] == depot]['

            # track lorry count
            num_vehicle = 0

            # for each lorry
            for lorry_no in range(len(lorry_ids)):

                # start its journey
                print(f"Lorry {lorry_ids[lorry_no]} has started")
                self.journey(lorry_ids[lorry_no], self.depot_allocations[depot]

                # update lorry count
                num_vehicle += 1


                # if after journey, the deliveries are not complete and all the
                # then refuel and journey again
                if len(self.depot_allocations[depot]) != 0 and (num_vehicle ==


                    # until all the customers have not been served keep refilli
                    while(len(self.depot_allocations[depot]) != 0):

                        # treat the last visited location as the current locati
                        # nearest depot for refuelling
                        lorry_id = lorry_ids[lorry_no]
                        curr_loc = self.routes[lorry_id][-1][0]

                        near_depot, time_of_travel = self.nearest_depot(curr_lo
                        self.total_time.append(time_of_travel)

                        # Refuel and update the journey
                        print(f"Lorry {lorry_id} is refilling their tank at {ne
                        self.routes[lorry_id].append((near_depot, lorry_capacit

                        # start delivering again and update the journey
                        self.journey(lorry_id, self.depot_allocations[depot])



                # if all the deliveries are complete then stop
                elif (len(self.depot_allocations[depot]) == 0):

                    # track the num of vehicles used
                    self.vehicles_used[depot] = num_vehicle

                    break


        print('\n\n')

        time_to_depot = []
        print('all vehicles went to depot to end journey...')

        # we assume that all the lorries start to go back to depot to end journ
        for i, row in self.depot_df.iterrows():

            lorry_id = row['lorry_id']
            if lorry_id in self.routes:
                curr_loc = self.routes[lorry_id][-1][0]
                near_depot, time = self.nearest_depot(curr_loc)

                # update journey and append time to time_to_depot list
                time_to_depot.append(time)
                self.routes[lorry_id].append((near_depot, 0))
        # add the max time taken to reach the depot to the total_time list
        self.total_time.append(max(time_to_depot))


        print('\n\n')
        for i in self.vehicles_used:
            print(f'No. of vehices used from depot {i} :', self.vehicles_used[i


        return  sum(self.total_time), self.routes
```

In [18]:

```
O = Greedy(lorry_df, depot_customers )
time , journeys = O.run()
```

```

        REGION: 1
Lorry 124-0 has started
Lorry 124-0 ended its journey at (not carrying enough fuel fo
r next location) 204
Lorry 124-1 has started
Lorry 124-1 ended its journey at (not carrying enough fuel fo
r next location) 118
Lorry 124-2 has started
Lorry 124-2 ended its journey at (not carrying enough fuel fo
r next location) 177
Lorry 124-3 has started
Lorry 124-3 ended its journey at (no customers left to delive
r) 400

        REGION: 2
Lorry 127-0 has started
Lorry 127-0 ended its journey at (not carrying enough fuel fo
r next location) 65
Lorry 127-1 has started
Lorry 127-1 ended its journey at (not carrying enough fuel fo
r next location) 41
Lorry 127-2 has started
Lorry 127-2 ended its journey at (no customers left to delive
r) 534

        REGION: 3
Lorry 167-0 has started
Lorry 167-0 ended its journey at (not carrying enough fuel fo
r next location) 110
Lorry 167-1 has started
Lorry 167-1 ended its journey at (not carrying enough fuel fo
r next location) 64
Lorry 167-2 has started
Lorry 167-2 ended its journey at (not carrying enough fuel fo
r next location) 73
Lorry 167-3 has started
Lorry 167-3 ended its journey at (no customers left to delive
r) 437

        REGION: 4
Lorry 373-0 has started
Lorry 373-0 ended its journey at (not carrying enough fuel fo
r next location) 22
```

```
Lorry 373-1 has started
Lorry 373-1 ended its journey at (not carrying enough fuel fo
r next location) 86
Lorry 373-2 has started
Lorry 373-2 ended its journey at (no customers left to delive
r) 135

        REGION: 5
Lorry 523-0 has started
Lorry 523-0 ended its journey at (not carrying enough fuel fo
r next location) 519
Lorry 523-1 has started
Lorry 523-1 ended its journey at (not carrying enough fuel fo
r next location) 603
Lorry 523-2 has started
Lorry 523-2 ended its journey at (no customers left to delive
r) 391



all vehicles went to depot to end journey...



No. of vehices used from depot 124 : 4
No. of vehices used from depot 127 : 3
No. of vehices used from depot 167 : 4
No. of vehices used from depot 373 : 3
No. of vehices used from depot 523 : 3
```

In [19]:

```python
print(f'Time taken to make all deliveries: {time/60} hrs')
```

```
Time taken to make all deliveries: 174.33671621124404 hrs
```

In [20]:

```python
customer_unserved = 0

for i in depot_loc:
    customer_unserved += len(depot_customers[i])

customer_served = len(customer_loc) - customer_unserved

print(f"Number of customers served {customer_served}")
```

```
Number of customers served 125
```

In [21]:

```
journeys
```

Out[21]:

```
{'124-0': [(124, 5),
  (542, -0.47),
  (633, -0.18),
  (260, -1.62),
  (507, -1.15),
  (204, -0.69),
  (124, 0)],
 '124-1': [(124, 12),
  (14, -0.33),
  (36, -0.81),
  (254, -1.28),
  (206, -0.88),
  (147, -0.85),
  (397, -0.7),
  (171, -1.32),
  (264, -0.86),
```

In [22]:

```
type(journeys)
```

Out[22]:

```
dict
```

In [23]:

```
result_dict=[]

for i in journeys.keys():

    result_dict.append({"lorry_id":i,"loc":journeys[i]})
```

In [25]:

```
import json

class Encoder(json.JSONEncoder):
    def default(self, obj):
         if isinstance(obj, np.integer):
            return int(obj)
         elif isinstance(obj, np.floating):
            return float(obj)
         elif isinstance(obj, np.ndarray):
            return obj.tolist()
         else:
            return super(NpEncoder, self).default(obj)


with open('solution-1.json', 'w') as filepath:
    json.dump(result_dict, filepath, cls=Encoder)
```

PROBLEM 2

Objective : To maximize cost efficiency i.e. total gas delivery / total cost

Genetic Algorithm

Customers assignment

In [26]:

```
customers_124 = depot[124]
customers_127 = depot[127]
customers_167 = depot[167]
customers_373 = depot[373]
customers_523 = depot[523]

print('total customers for depot 124:', len(customers_124))
print('total customers for depot 127:', len(customers_127))
print('total customers for depot 167', len(customers_167))
print('total customers for depot 373:', len(customers_373))
print('total customers for depot 523:', len(customers_523))
```

```
total customers for depot 124: 37
total customers for depot 127: 23
total customers for depot 167 19
total customers for depot 373: 30
total customers for depot 523: 16
```

Functions to calculate cost

In [27]:

```
def dist(xa, ya, xb ,yb):
    '''
    function to calculate euclidean distance'''
    d = np.sqrt( np.square(xa-xb) + np.square(ya-yb) )

    return d




def cal_dist(id1, id2, df):
    '''function to calculate distance between two locations'''

    if id1 == id2:
        return 0.0

    else:

        from_ = df[ df['id'] == id1]
        to_ = df[ df['id'] == id2]


        xa = int(from_['x'])
        ya = int(from_['y'])


        xb = int(to_['x'])
        yb = int(to_['y'])


        distance = dist(xa,ya, xb,yb)

    return float(distance)




def cal_time(id1, id2, df):
    '''function to calculate time of going from one location to another'''

    if id1 == id2:
        return 0.0

    else:

        from_ = df[ df['id'] == id1]
        to_ = df[ df['id'] == id2]
        xa = int(from_['x'])
        ya = int(from_['y'])


        xb = int(to_['x'])
        yb = int(to_['y'])

        distance = dist(xa,ya, xb,yb)

        time = (( distance / 50) * 60)

        return float(time)
```

GA Functions

In [28]:

```
import random
import numpy as np, operator, pandas as pd, matplotlib.pyplot as plt



# sequence generator
def createSequence(EList):
    route = random.sample(EList, len(EList))
    return route




# population generator

def initialPopulation(popSize, EList):
    population =[]

    for i in range(0, popSize):
        population.append(createSequence(EList))

    return population




# class which will help to calculate the fitness of a sequence.
class Fitness:

    def __init__(self, Eseq, depot_df):
        '''initialisations'''
        self.Eseq = Eseq
        self.depot_df = depot_df
        self.depot = int(self.depot_df.lorry_id[0].split('-')[0])
        self.capacity = self.depot_df.capacity.to_list()
        self.cpm = self.depot_df.cpm.to_list()
        self.cptm = self.depot_df.cptm.to_list()
        self.fitness = 0



    def get_num_customers(self, capacity):
        '''function that returns the num of customer to be assigned to a lorry
        contraint B)'''

        if capacity == 5:
            num_customers = 4
```

```python
        elif capacity == 12:
            num_customers = 8

        else:
            num_customers = 16

        return num_customers

    def genotypeToPhenotype(self, chro):
        '''function to assign customers to each lorry'''

        # calculate the no. of customers to be assigned to each lorry
        num_customers = [self.get_num_customers(i) for i in self.capacity]


        # CONSTRAINT B
        # assign the customers
        index_assigned = 0

        lorry0 = chro[:num_customers[0]]
        index_assigned += num_customers[0]

        lorry1 = chro[index_assigned:index_assigned + num_customers[1]]
        index_assigned += num_customers[1]

        lorry2 = chro[index_assigned:index_assigned + num_customers[2]]
        index_assigned += num_customers[2]

        lorry3 = chro[index_assigned:index_assigned + num_customers[3]]
        index_assigned += num_customers[3]

        lorry4 = chro[index_assigned:]


        return [lorry0, lorry1, lorry2, lorry3, lorry4]



    def taskFitness(self):
        '''function to calculate fitness in terms of cost efficiency keeping th

        lorries = self.genotypeToPhenotype(self.Eseq)

        # initialise variables for grand total of cost and delivery in the regi
        total_cost = 0
        total_delivery = 0
        total_time = []
        lorry_visits = []
        lorry_rest_stops = []
```

```
routes = {}

for lorry_no, lorry_customers in enumerate(lorries):

    #print(f'route for Vehicle: {lorry_no}')

    # Initialize values for a lorry
    lorry_id = self.depot_df.lorry_id[lorry_no]
    lorry_cost = 0
    lorry_delivery = 0
    lorry_volume = self.capacity[lorry_no]
    lorry_cpm = self.cpm[lorry_no]
    lorry_cptm = self.cptm[lorry_no]


    routes[lorry_id] = []
    lorry_time = 0


    # if customers have been assigned to a lorry then make the start an
    # which the lorry belongs
    if len(lorry_customers) !=0:
        lorry_customers = [self.depot] + lorry_customers + [self.depot]
        routes[lorry_id].append((self.depot, lorry_volume))



    # loop for lorry route
    for c in range(len(lorry_customers)):


        if c!= len(lorry_customers)-1 and len(lorry_customers)!= 0:


            # set current and next location ( NOTE: the first and last
            curr_loc = lorry_customers[c]
            next_loc = lorry_customers[c+1]
            #print('Curr :', curr_loc, 'next : ', next_loc)



            # calculate the demand of next location
            if next_loc != self.depot:
                demand = float(locs.loc[next_loc, 'demand'])
            else:
                demand = 0

            #print('demand : ', demand)
```

```
# LORRY CAPACITY CONSTRAINT
# if the volume of lorry is less than the demand of the nex
# and add the cost of travelling to and from depot to the n

#print('l :',lorry_volume,'d :',demand)
if lorry_volume < demand:


    # go to the depot
    #print('refuelling -->')
    # cost -----------------------------------------------
    distance = cal_dist(curr_loc, self.depot, locs)
    cost = distance * (lorry_cpm + lorry_volume * lorry_cpt

    # time -----------------------------------------------
    time = cal_time(curr_loc, self.depot, locs)


    # store vehicle journey
    routes[lorry_id].append((self.depot, lorry_volume))


    # update lorry_cost and lorry_time
    lorry_cost += cost
    lorry_time += time



    # refuel
    lorry_volume = self.capacity[lorry_no]
    # store vehicle state
    routes[lorry_id].append((self.depot, lorry_volume))




    # go to next location
    # cost -----------------------------------------------
    distance = cal_dist(self.depot, next_loc, locs)
    cost = distance * (lorry_cpm + lorry_volume * lorry_cpt

    # time -----------------------------------------------
    time = cal_time(self.depot, next_loc, locs)
    time += ((demand * 10) + 5)



    # store vehicle journey
    routes[lorry_id].append((next_loc, -demand))
```

```
                        # update lorry_cost and lorry_time
                        lorry_cost += cost
                        lorry_time += time


                        # deliver
                        lorry_volume -= demand
                        lorry_delivery += demand



                    else :

                        # go to next location
                        # cost -----------------------------------------------
                        distance = cal_dist(curr_loc, next_loc, locs)
                        cost = distance *  (lorry_cpm + lorry_volume * lorry_cp

                        # time -----------------------------------------------
                        time = cal_time(curr_loc, next_loc, locs)
                        time += ((demand * 10) + 5)



                        # store vehicle journey
                        if next_loc != self.depot:
                            routes[lorry_id].append((next_loc, -demand))


                        else:
                            routes[lorry_id].append((self.depot, 0))



                        # update lorry_cost and lorry_time
                        lorry_cost += cost
                        lorry_time += time


                        # deliver
                        lorry_volume -= demand
                        lorry_delivery += demand


            total_time.append(lorry_time)
            total_cost += lorry_cost
            total_delivery += lorry_delivery
            #print(f'total cost for vehicle {lorry_no} is : {lorry_cost}')
```

```
        lorry_rest_stops = [np.floor(i/120) for i in total_time]
        time_with_rest_vehicle_journey = [ i + j*20 for i, j in zip(total_time,


        self.fitness = total_delivery/total_cost
        #print(f'Efficiency: {self.fitness} ')


        return self.fitness, routes, lorry_rest_stops, time_with_rest_vehicle_j





# sorting the population in descending order according to the fitness value.
def rankSeq(population, depot_data):
    """
    This function sorts the given population in decreasing order of the fitness
    """
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] = Fitness(population[i], depot_data).taskFitness()[0]

    return sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse




def selection(popRanked, eliteSize):
    """
    This function takes in a population sorted in decreasing order of fitness s
    It returns a list of indices of the chosen mating pool in the given populat
    """
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])

    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break

    return selectionResults
```

```python
def matingPool(population, selectionResults):
    """
    This function takes in a population and returns the chosen mating pool whic
    """
    matingpool = []
    for i in range(0, len(selectionResults)):
        index = selectionResults[i]
        matingpool.append(population[index])
    return matingpool




def breed(parent1, parent2):
    """
    This function should breed both parents (sequences) and return a child sequ
    mentioned above. Please fill in the code to do so.
    """

    #print('P1:', len(parent1))
    #print('P2:', len(parent2))

    offspring=[]
    P1_c = []
    P2_c = []

    indexes = list(range(len(parent1)))
    points = random.choices(indexes, k=2)

    Gene_start = min(points)
    Gene_end = max(points)

    #print('gene start:', Gene_start)
    #print('gene end:', Gene_end)

    for i in range(Gene_start,Gene_end):
        P1_c.append(parent1[i])

    P2_c = [item for item in parent2 if item not in P1_c]

    offspring = P1_c + P2_c
    #print('offspring Length: ',len(offspring))

    return offspring
```

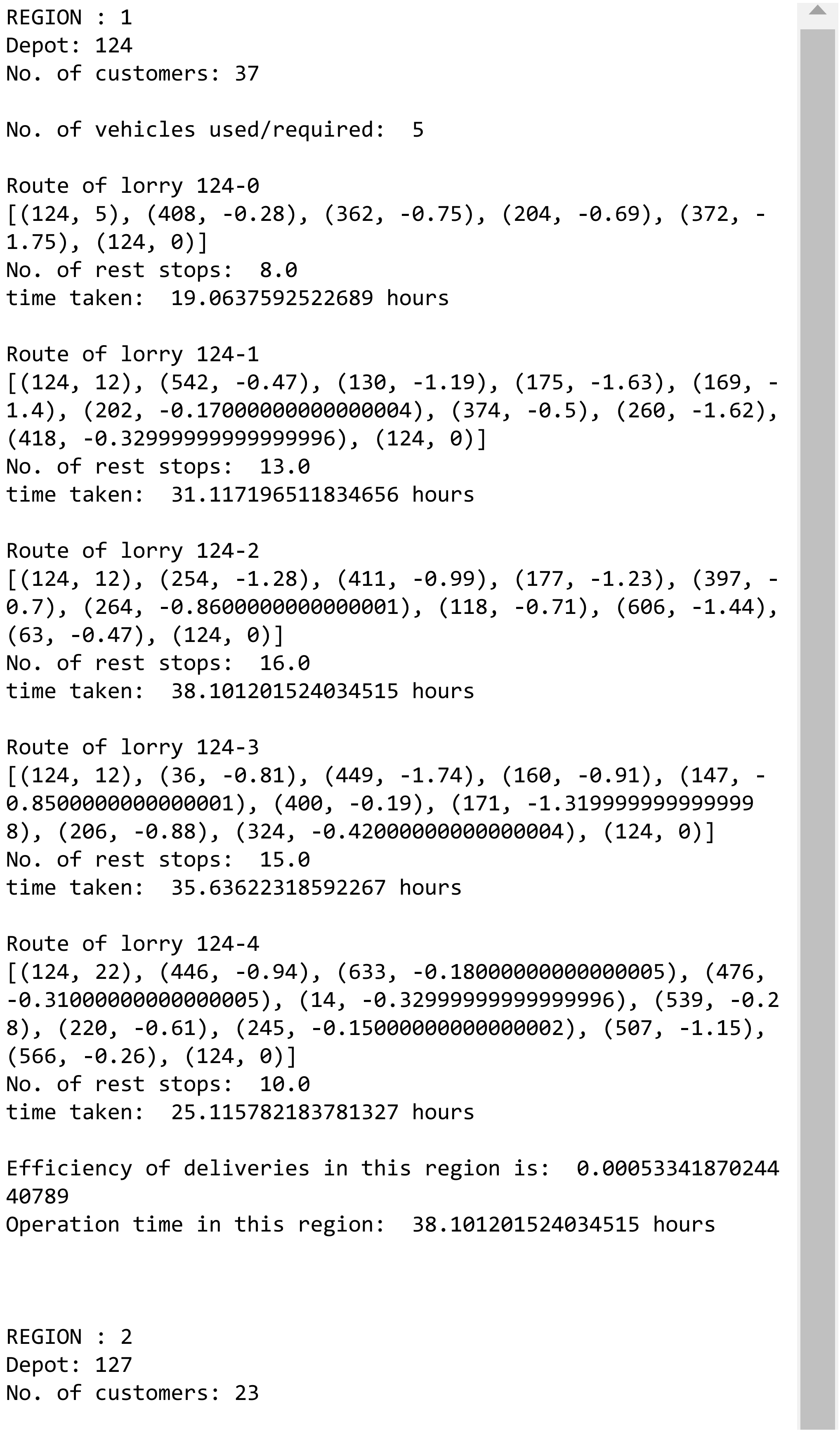
```
def breedPopulation(matingpool, eliteSize):
    """
    This function should return the offspring population from the current popul
    retain the eliteSize best rsequence from the current population. Then it sh
    members of the population, to fill out the rest of the next generation. You
    """

    offsprings = []
    length = len(matingpool) - eliteSize
    pool = random.sample(matingpool, len(matingpool))

    for i in range(0,eliteSize):
        offsprings.append(matingpool[i])

    for i in range(0, length):
        offspring = breed(pool[i], pool[len(matingpool)-i-1])
        offsprings.append(offspring)


    return offsprings




def mutate(individual, mutationRate):
    """
    This function should take in an individual (sequence) and return a mutated
    between 0 and 1. Use the swap mutation to mutate the individual according t
    through each of the employee and swap it with another employee according to
    """

    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            city1 = individual[swapped]
            city2 = individual[swapWith]

            individual[swapped] = city2
            individual[swapWith] = city1

    return individual



def mutatePopulation(population, mutationRate):
    """
    This function should use the above mutate function to mutate each member of
    population and mutate each individual using the mutationRate.
    """
    mutatedPop = []

    for ind in range(0, len(population)):
        mutatedInd = mutate(population[ind], mutationRate)
        mutatedPop.append(mutatedInd)

    return mutatedPop




def nextGeneration(currentGen, eliteSize, mutationRate, depot_data):
    """
    This function takes in the current generation, eliteSize and mutationRate a
    """
    pop_rank = rankSeq(currentGen, depot_data)
    #print('ranked: ',pop_rank)
    selection_results = selection(pop_rank, eliteSize)
    #print('selection= ',len(selection_results))
    matingpool = matingPool(currentGen, selection_results)
    #print('mating pool: ', matingpool)
    offspring = breedPopulation(matingpool, eliteSize)
    #print('offspring: ',offspring)
    nextGeneration = mutatePopulation(offspring, mutationRate)

    #print('Next generation: ',nextGeneration)
    return nextGeneration



def geneticAlgorithm(population, popSize, eliteSize, mutationRate, generations,
    """
    This function creates an initial population, then runs the genetic algorith
    """

    pop = initialPopulation(popSize, population)
    #print("Initial pop: " + str(pop))

    for i in range(0, generations):
        #print('Generation: ',i)
        pop = nextGeneration(pop, eliteSize, mutationRate, depot_data)

    bestRouteIndex = rankSeq(pop,depot_data)[0][0]
    bestRoute = pop[bestRouteIndex]
    #print('Best sequence= ', bestRoute)
    return bestRoute
```

Assumptions:

- All the lorries will leave their respective depots at the same time thus the maximum time taken by a lorry will be the total time in which the delivery operations complete.
- All lorries would rest for 20 minutes after a 2 hour drive. Thus, the time to complete the operation would also include these rest times.
- A small lorry can make delivery to 4 customers only, a medium lorry can make delivery to 8 customers only and a large lorry can make delivery to 16 customers only. We don't include the stops made for rest and refuelling in this count.
- The capacity of the lorry is checked before going to the next location. If the lorry_capacity is less than the demand of the next location, then the lorry goes to the depot from its current location for refuelling and from their it goes to the next location to make the delivery.

In [29]:

```
depot_dfs = [df_124, df_127, df_167, df_373, df_523]
customers = [customers_124, customers_127, customers_167, customers_373, custom

journeys = [] # List of dictionaries to store routes
total_efficiency = 0
total_time = []
ctr = 1

for lorry_info, depot_customers in zip(depot_dfs, customers):

    best_route = geneticAlgorithm(population= depot_customers, popSize=100, eli
                                  generations=100, depot_data= lorry_info)

    print('REGION :', ctr)
    print('Depot:', lorry_info.lorry_id[0].split('-')[0])
    print('No. of customers:', len(best_route))


    efficiency, routes, num_rest_stops, vehicle_journey_times = Fitness(best_ro
    journeys.append(routes)
    total_efficiency += efficiency
    total_time.extend(vehicle_journey_times)


    vehicles_used = 0
    for i in routes:
        if len(routes[i])!=0:
            vehicles_used +=1
    print('\nNo. of vehicles used/required: ', vehicles_used)


    for idx, i in enumerate(routes):
        if len(routes[i]) != 0:
            print('\nRoute of lorry', i)
            print(routes[i])
            print('No. of rest stops: ', num_rest_stops[idx])
            print('time taken: ', vehicle_journey_times[idx]/60, 'hours')

    print('\nEfficiency of deliveries in this region is: ', efficiency)
    print('Operation time in this region: ', max(vehicle_journey_times)/60, 'ho
    print('\n\n')
    ctr += 1


print('\nTotal efficieny of the operation: ', total_efficiency)
print('Total time taken to deliver in all regions: ', max(total_time)/60, 'hour
```



```
REGION : 1
Depot: 124
No. of customers: 37

No. of vehicles used/required:  5

Route of lorry 124-0
[(124, 5), (408, -0.28), (362, -0.75), (204, -0.69), (372, -
1.75), (124, 0)]
No. of rest stops:  8.0
time taken:  19.0637592522689 hours

Route of lorry 124-1
[(124, 12), (542, -0.47), (130, -1.19), (175, -1.63), (169, -
1.4), (202, -0.17000000000000004), (374, -0.5), (260, -1.62),
(418, -0.32999999999999996), (124, 0)]
No. of rest stops:  13.0
time taken:  31.117196511834656 hours

Route of lorry 124-2
[(124, 12), (254, -1.28), (411, -0.99), (177, -1.23), (397, -
0.7), (264, -0.8600000000000001), (118, -0.71), (606, -1.44),
(63, -0.47), (124, 0)]
No. of rest stops:  16.0
time taken:  38.101201524034515 hours

Route of lorry 124-3
[(124, 12), (36, -0.81), (449, -1.74), (160, -0.91), (147, -
0.8500000000000001), (400, -0.19), (171, -1.319999999999999
8), (206, -0.88), (324, -0.42000000000000004), (124, 0)]
No. of rest stops:  15.0
time taken:  35.63622318592267 hours

Route of lorry 124-4
[(124, 22), (446, -0.94), (633, -0.18000000000000005), (476,
-0.31000000000000005), (14, -0.32999999999999996), (539, -0.2
8), (220, -0.61), (245, -0.15000000000000002), (507, -1.15),
(566, -0.26), (124, 0)]
No. of rest stops:  10.0
time taken:  25.115782183781327 hours

Efficiency of deliveries in this region is:  0.00053341870244
40789
Operation time in this region:  38.101201524034515 hours



REGION : 2
Depot: 127
No. of customers: 23
No. of vehicles used/required:  4

Route of lorry 127-0
[(127, 5), (41, -1.03), (65, -0.72), (624, -0.51), (534, -1.1
9), (127, 0)]
No. of rest stops:  7.0
time taken:  17.06469331910235 hours

Route of lorry 127-1
[(127, 5), (398, -0.89), (255, -0.16999999999999998), (550, -
0.45999999999999996), (5, -0.09999999999999998), (127, 0)]
No. of rest stops:  4.0
time taken:  11.277591006008752 hours

Route of lorry 127-2
[(127, 12), (146, -1.45), (612, -0.79), (77, -1.47), (82, -0.
6), (387, -1.24), (113, -0.44999999999999996), (265, -0.48),
(628, -1.27), (127, 0)]
No. of rest stops:  18.0
time taken:  43.93660012947123 hours

Route of lorry 127-3
[(127, 12), (585, -1.41), (511, -0.24), (584, -0.219999999999
99997), (380, -0.15999999999999998), (243, -0.669999999999999
9), (31, -0.56), (308, -0.6799999999999999), (127, 0)]
No. of rest stops:  7.0
time taken:  18.199782655565652 hours

Efficiency of deliveries in this region is:  0.00051903304122
11565
Operation time in this region:  43.93660012947123 hours



REGION : 3
Depot: 167
No. of customers: 19

No. of vehicles used/required:  4

Route of lorry 167-0
[(167, 5), (64, -0.96), (214, -1.6099999999999999), (569, -0.
13), (274, -0.16999999999999993), (167, 0)]
No. of rest stops:  5.0
time taken:  13.102982492491751 hours

Route of lorry 167-1
[(167, 5), (44, -0.71), (110, -1.99), (621, -0.24), (598, -0.
97), (167, 0)]
```

```
No. of rest stops:  5.0
time taken:  12.339136741388971 hours

Route of lorry 167-2
[(167, 5), (271, -0.9), (393, -1.12), (453, -0.61), (105, -1.
27), (167, 0)]
No. of rest stops:  6.0
time taken:  14.824238709395352 hours

Route of lorry 167-3
[(167, 12), (210, -0.47), (32, -0.92), (73, -0.22999999999999
998), (437, -0.94), (459, -1.98), (80, -0.22999999999999998),
(528, -0.12), (167, 0)]
No. of rest stops:  6.0
time taken:  15.25117230844246 hours

Efficiency of deliveries in this region is:  0.00099353933966
66152
Operation time in this region:  15.25117230844246 hours



REGION : 4
Depot: 373
No. of customers: 30

No. of vehicles used/required:  5

Route of lorry 373-0
[(373, 5), (205, -0.9099999999999999), (180, -0.76), (543, -
0.54), (474, -0.3999999999999999), (373, 0)]
No. of rest stops:  6.0
time taken:  15.11410025303667 hours

Route of lorry 373-1
[(373, 5), (337, -0.44999999999999996), (29, -0.54), (135, -
1.33), (172, -0.91), (373, 0)]
No. of rest stops:  10.0
time taken:  24.74249934819893 hours

Route of lorry 373-2
[(373, 12), (497, -0.98), (216, -0.27), (431, -0.39), (103, -
0.43999999999999995), (13, -0.19999999999999996), (455, -0.
3), (531, -0.10999999999999999), (86, -1.03), (373, 0)]
No. of rest stops:  12.0
time taken:  29.934211273287303 hours

Route of lorry 373-3
[(373, 12), (364, -0.89), (144, -0.45999999999999996), (22, -
1.68), (294, -0.81), (389, -0.07), (332, -0.79), (547, -0.9
5), (200, -0.27), (373, 0)]
No. of rest stops:  10.0
time taken:  24.939978713512893 hours

Route of lorry 373-4
[(373, 22), (377, -0.95), (270, -0.21999999999999997), (78, -
0.63), (520, -0.44), (100, -1.02), (190, -0.360000000000000
1), (373, 0)]
No. of rest stops:  7.0
time taken:  16.479782264801436 hours

Efficiency of deliveries in this region is:  0.00044481734936
56769
Operation time in this region:  29.934211273287303 hours



REGION : 5
Depot: 523
No. of customers: 16

No. of vehicles used/required:  3

Route of lorry 523-0
[(523, 5), (276, -0.9299999999999999), (70, -1.46), (632, -1.
24), (519, -0.21000000000000002), (523, 0)]
No. of rest stops:  4.0
time taken:  10.53205755893233 hours

Route of lorry 523-1
[(523, 5), (183, -0.3999999999999999), (136, -0.3399999999999
9997), (603, -0.86), (391, -1.0), (523, 0)]
No. of rest stops:  4.0
time taken:  9.983265560545627 hours

Route of lorry 523-2
[(523, 12), (225, -0.64), (8, -0.19999999999999996), (257, -
0.19999999999999996), (27, -1.02), (491, -0.3399999999999999
7), (235, -0.36), (94, -1.52), (378, -0.54), (523, 0)]
No. of rest stops:  7.0
time taken:  18.097712394167367 hours

Efficiency of deliveries in this region is:  0.00091601386693
84274
Operation time in this region:  18.097712394167367 hours




Total efficieny of the operation:  0.0034068222996359553
Total time taken to deliver in all regions:  43.9366001294712
3 hours
```

In [30]:

```
journeys
```

Out[30]:

```
[{'124-0': [(124, 5),
   (408, -0.28),
   (362, -0.75),
   (204, -0.69),
   (372, -1.75),
   (124, 0)],
  '124-1': [(124, 12),
   (542, -0.47),
   (130, -1.19),
   (175, -1.63),
   (169, -1.4),
   (202, -0.17000000000000004),
   (374, -0.5),
   (260, -1.62),
   (418, -0.32999999999999996),
   (124, 0)],
  '124-2': [(124, 12),
```

In [32]:

```
# creating the right format for solution
j={}

for i in journeys:
  j.update(i)
```

In [33]:

```
j
```

Out[33]:

```
{'124-0': [(124, 5),
  (408, -0.28),
  (362, -0.75),
  (204, -0.69),
  (372, -1.75),
  (124, 0)],
 '124-1': [(124, 12),
  (542, -0.47),
  (130, -1.19),
  (175, -1.63),
  (169, -1.4),
  (202, -0.17000000000000004),
  (374, -0.5),
  (260, -1.62),
  (418, -0.32999999999999996),
  (124, 0)],
 '124-2': [(124, 12),
```

In [34]:

```
result_dict1=[]

for i in j.keys():

    result_dict1.append({"lorry_id":i,"loc":j[i]})
```

In [35]:

```
# saving the result as json file
with open('solution-2.json', 'w') as filepath:
    json.dump(result_dict1, filepath, cls=Encoder)
```

Conclusion

As data scientists at SaO Gas Ltd, we have achieved solutions for both the optimization problems with the given constraints in check.For the first optimization problem we have acheived the time taken to make all deliveries and minimizing dispatch time as 174.33671621124404 hrs. In the second optimization problem we have achieved total efficiency of 0.0034068222996359553 which is calculated as total gas delivered to customers by overall cost of the delivery.

In [ ]: