1 Facility Location Problem

1.1 Model

Facility Location Problem can be formulated as follow:

$$\begin{aligned} min \ & \sum_{j=1}^{n} c_j y_j + \sum_{i,j=1}^{m,n} d_{i,j} x_{i,j} \\ s.t. \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad for\ all\ i \\ & x_{i,j} \leq y_i \quad for\ all\ i,j \\ & x_{i,j} \in \{0,1\},\ y_j \in \{0,1\} \end{aligned} \tag{1}$$

It is equivalent to the formulation below:

$$\begin{aligned} min \ & \sum_{j=1}^{n} c_j y_j + \sum_{i,j=1}^{m,n} d_{i,j} x_{i,j} \\ s.t. \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad for\ all\ i \\ & x_{i,j} \leq y_i \quad for\ all\ i,j \\ & 0 \leq x_{i,j} \leq 1,\ y_j \in \{0,1\} \end{aligned} \tag{2}$$

Proof: First, we claim that if we want to guarantee that the optimal soltuion to (2) is integral, we should make sure that $d_{i,j}$ is different from each other for every fixed i.
We just need to show that there exists one integral optimal soltuion in (2), and we prove this by contradiction.
Assume that all the optimal soltuions of (2) cannot be integral. Then, we can select x_{i,j_t} such that $0 < x_{i,j_t} < 1$, $t = 1, 2, ..., k$ and it is an optimal solution to (2). We also have $\sum_{t=1}^{k} x_{i,j_t} = 1$. Choose $d_{i,j_s} = min\{d_{i,j_t}\}$, and we can easily check that $x_{i,j_s} = 1, x_{i,j} = 0$ for all $j \neq j_s$ is still an optimal solution, which contradicts to our assumption. This just proves the equivalence.

We also have an alternative and equivalent MILP formulation:

$$\begin{aligned} min \ & \sum_{j=1}^{n} c_j y_j + \sum_{i,j=1}^{m,n} d_{i,j} x_{i,j} \\ s.t. \quad & \sum_{j=1}^{n} x_{i,j} = 1 \quad for\ all\ i \\ & \sum_{i=1}^{m} x_{i,j} \leq m y_j \quad for\ all\ j \\ & 0 \leq x_{i,j} \leq 1,\ y_j \in \{0,1\} \end{aligned} \tag{3}$$

1.2 Solutions to Facility Location Problem

Q(a): Count the number of linear inequalities in (FLP) and (AFL). Which formulation has fewer number of inequalites?
Solution of (a) :

In FLP, there are **mn** linear inequalities.
In AFL, there are **n** linear inequalities.
Therefore, AFL has fewer number of inequalities.

Q(b): Show that the set of feasible solutions to FLP and AFL are equal. Conclude that these two formulations are equivalent.
Solution of (b) :

Proof: We need to show two things, one is the set of feasible solutions to two formulations are equal, and the other is these two formulations are equivalent.

Firstly, we show that the set of feasible solutions are equal.
On one hand, the feasible solution of FLP is obviously a feasible solution of AFL. That is, $x_{i,j} \leq y_i$ for all i, j implies that $\sum_{i=1}^{m} x_{i,j} \leq m y_j$ for all j.
On the other hand, the feasible soltuion of AFL satisfies that $\sum_{i=1}^{m} x_{i,j} \leq m y_j$ for all j. When $y_j = 0$, it implies that $x_{i,j} = 0$ for $i = 1, 2, .., m$, i.e., we are able to attain $x_{i,j} \leq y_i$ for all i, j. When $y_j = 1$, it is trivial that $x_{i,j} \leq 1 = y_i$ for all i, j.
Hence, we prove that these two sets of feasible solutions are equal.

Then, we show that these two formulations are equivalent.
Since we prove that the sets of feasible soltuions to two formulations are equal, and they share the same objective, it is obvious that these two formulations are equivalent. To be more precise, they have the same feasible region and the same objective, which implies that they are equivalent.

Q(c): Derive the linear relaxation of FLP.
Solution of (c) :

$$\begin{aligned} min \ & \sum_{j=1}^{n} c_j y_j + \sum_{i,j=1}^{m,n} d_{i,j} x_{i,j} \\ s.t. \ & \sum_{j=1}^{n} x_{i,j} = 1 \quad for\ all\ i \\ & x_{i,j} \leq y_i \quad for\ all\ i, j \\ & 0 \leq x_{i,j} \leq 1,\ 0 \leq y_j \leq 1 \end{aligned} \tag{4}$$

Q(d): Derive the linear relaxation of AFL.
Solution of (d) :

$$\begin{aligned} min \ & \sum_{j=1}^{n} c_j y_j + \sum_{i,j=1}^{m,n} d_{i,j} x_{i,j} \\ s.t. \ & \sum_{j=1}^{n} x_{i,j} = 1 \quad for\ all\ i \\ & \sum_{i=1}^{m} x_{i,j} \leq m y_j \quad for\ all\ j \\ & 0 \leq x_{i,j} \leq 1,\ 0 \leq y_j \leq 1 \end{aligned} \tag{5}$$

Q(e): What is the relationship between FLP-Val, AFL-Val, FLP-LR-Val and AFL-LR-Val?
Solution of (e) :

From Question(b), we show that FLP and AFL are equivalent, which implies that **FLP-Val** = **AFL-Val**.
Since Linear Relaxation will enlarge the feasible region of Primal Problem, then we will have **FLP-LR-Val** $\leq$ **FLP-Val** and **AFL-LR-Val** $\leq$ **AFL-Val**,

Q(f): Generate random instances of facility location problem.
Solution of (f) :

In this question, we need to use uniform random variables to generate the location of facilities and customes. We also need to generate the cost from uniform random variable. The result can be visualized as follow:

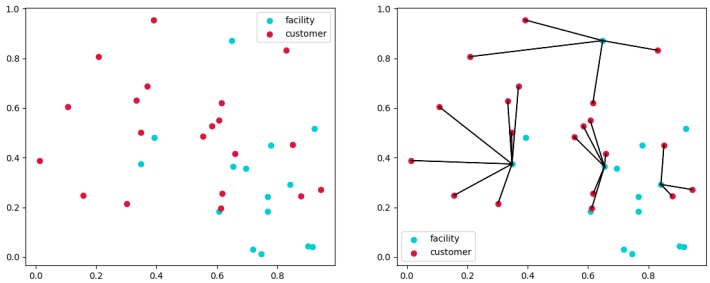

Figure 1: The visualized solution of FLP

Q(g): Solve the facility problem using FLP, AFL, FLP-LR and AFL-LR. Repeat this process for 100 times with different realization of all random variables.
Solution of (g) :

We repeat the Question(f) to all formulations for **100** times. Then we achieve the following diagram:

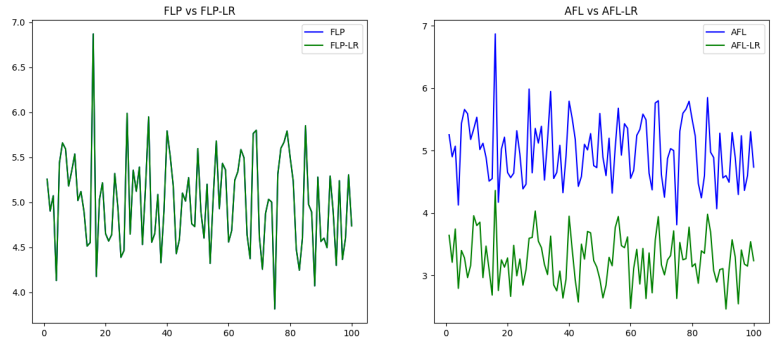

Figure 2: Outcomes of 4 formulations

From the diagram, there are three important things:

- As for FLP and FLP-LR, they **almost** share the same optimal value.
- As for AFL and AFL-LR, the optimal value to AFL-LR seems to be a **lower bound** of that to AFL strictly.
- As for FLP and AFL, they share the **same** optimal value, which confirms our argument in Question(b)

Q(h): Compare FLP-Val with FLP-LR-Val, AFL-Val with AFL-LR-Val respectively.
Solution of (h) :

Repeat above procedure for **300** times, we have:

- There are **297** cases in which FLP-Val = FLP-LR-Val.
- There are **0** cases in which AFl-Val = AFL-LR-Val

2 TSP with Stimulated Annealing Algorithm

2.1 Model and Background

TSP is a problem which seeks the shortest tour while visiting every location exactly once and returning to the starting location. We know that it can be formulated as Linear Program with node-arc matrix of directed graph as follows,

However, notice that the number of constraints is exponential in the problem dimension. It means that TSP is a HARD problem, which needs plenty of computational costs.
Thus, we try to use some heursitic algorithms, which can reach a blance between computational cost and accuracy. Among these algorithms, we implement Stimulated Annealing Algorithm for solving TSP.

2.2 Solutions of Stimulated Annealing Algorithm

Q(a): Create a function that takes a tour as input and outputs the total distance.
Solution of (a) :

Firstly, we construct a distance matrix, whose (i, j)-th entry represents the distance from i to j. In order to get the function, we take both a certain tour and the distance matrix as input. My main trick is to use slice of list to attain two sub-tours. Then, we can easily get the function from the two sub-tours.

Q(b): Create a function that takes a tour as input, and outputs a perturbed tour.
Solution of (b) :

Notice that what we do is to inverse part of the sequence, and position of the part is controlled by two random integers.
This can be achieved by two steps. Firstly, choose two unequal integers as indices. After that, inverse the subsequence between the two indices.
As far as I am concerned, this special Rroposal Rule resembles MUTATION in some way, giving us chances to escape from local optimum.

Q(c): Implement an instance of Stimulated Annealing Algorithm for TSP problem.
Solution of (c) :

I will show the framework of my code by a diagram below:
To illustrate, I will briefly go over this algorithm. Compared with other algorithms, our trick here is to accept some tours which may be not shorter than the current one with some probability. Moreover, the probability will decrease over time.

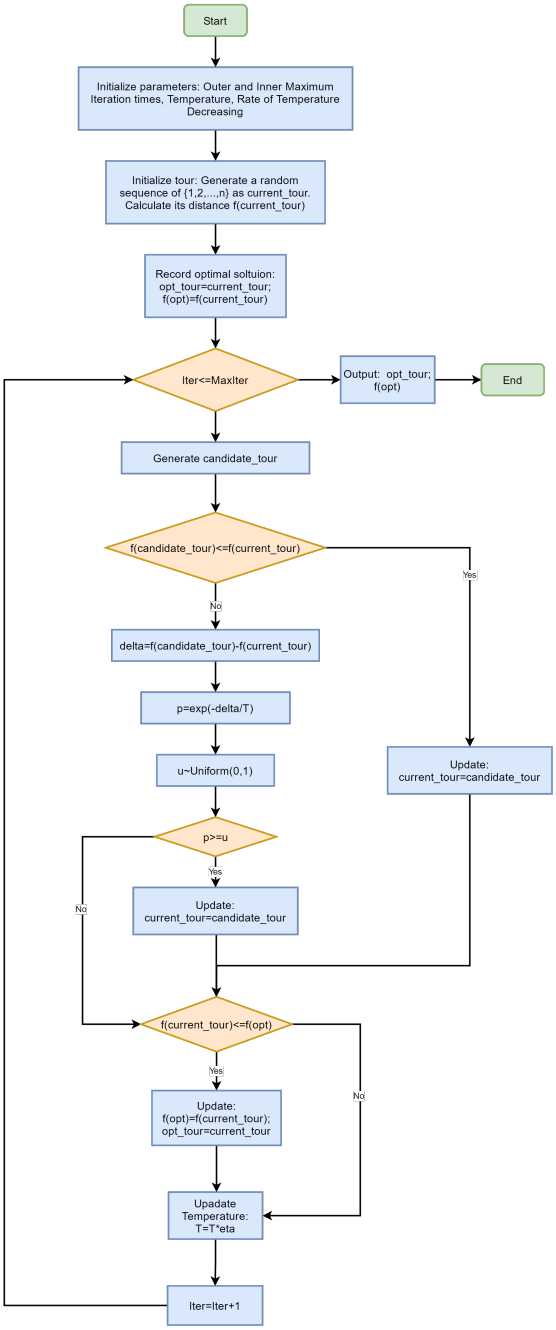

Figure 3: Stimulated Annealing Algorithm

That is to say, in the very beginning, we do not require that the updated tour must be better than the current one. This allows us to excape from local optimum. However, over time, we are stricter and stricter to the quality of updated tour, which will lead to the convergence of the algorithm.

Q(d): Generate a random instance to test your implementation. Pick n = 100, and generate n cities uniformly at random over the $[0,1]\times[0,1]$
Solution of (d) :

We fix $\eta = 0.99$, and select the appropriate iteration times and initial temperature by experiments.

- Iteration Times

Choose three different iteration times, denote as $n_1 = 10000,\ n_2 = 30000,\ n_3 = 50000$. We focus on the decrease of shortest tour if we take more iteration times. The results of experiment can be illustrated from the diagram below:

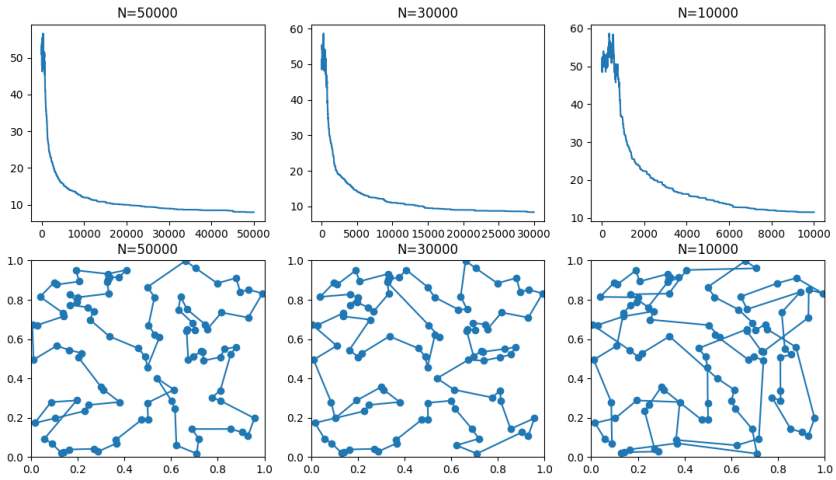

Figure 4: Different iteration times

We can also achieve that the shortest distance is **7.9452**, **8.3581** and **11.4812** from left to right.
Therefore, from both diagram and the shortest distance, we can find that when iteration increases from 10000 to 30000, the shortest distance improves dramatically. While iteration increases from 30000 to 50000, the shortest distance improves slightly.
Consequently, as for **100 cities**, we choose iteration times euqal to **30000**, which can reach a blance between accuracy and computational cost.

- Initial Temperature

Choose three different initial temperatures, with $T_1 = 500$, $T_2 = 1000$ and $T_3 = 2000$ respectively. In this step, we choose iteration times equal to 30000 from the above discussion. Then, we want to see the shortest tour in each case. The result is shown as follows:

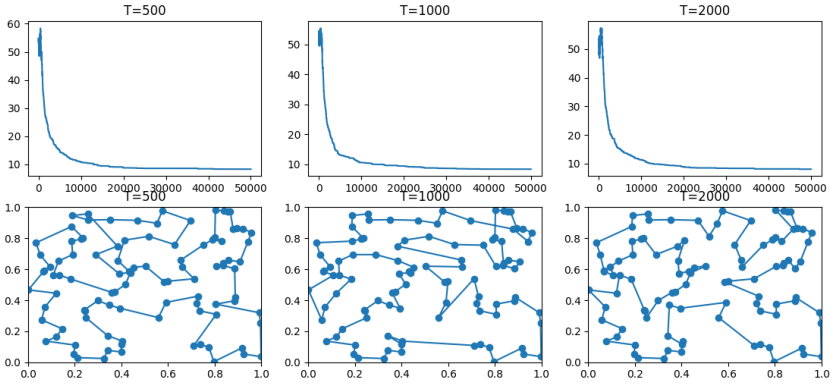

Figure 5: Different initial temperatures

We can also achieve that the shortest distance is **8.2934**, **8.3991** and **8.1898** from left to right.
Therefore, we can conclude that Stimulated Annealing Algorithm is not sensitive to initial temperature.
Additionally, we can see something intersting from the diagram. Notice that when initial temperature grows higher, it fluctuates more in the very beginning. This means it will accept more tours at first, which is a great way to escape from the local optimum. However, it also needs to take more iteration as a cost for the sake of convergence.

Q(e): Plot the cities with the tour to show your results.
Solution of (e) :
The diagram can be seen as follows:

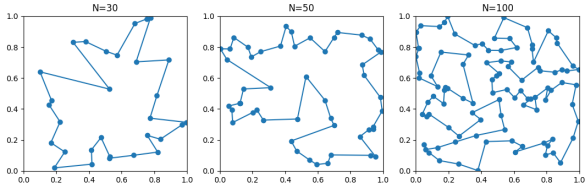

Figure 6: different number of cities

In the above figure, we choose different number of cities to show our answer, with 30, 50 and 100 cities respectively.
When N=30 and N=50, there are no edges that intersect. Therefore, they can be optimal tours. It is mainly because these two cases are actually simpler, and it can be possible for us to work it out.
When N=100, there exists some edges that intersect, which indicates this tour cannot be an optimal tour. Since there are 100 cities, the problem here is much more complicated. Therefore, it is difficult for us to attain the global optimal tour.

Q(f): Experiment with different Proposal Rules to generate candidate tour.
Solution of (f) :

We give TWO other basic methods of generating candidate tour from the current tour. They are achieved by simple transformation, which can be viewed as the following diagram:

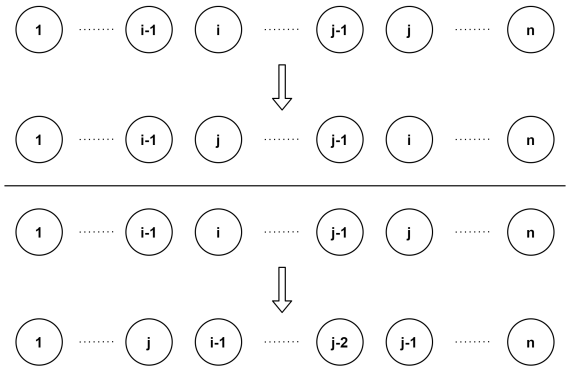

Figure 7: Two other generating methods

Compare this two Proposal Rules to the given one, it can be observed that these two rules change the sequence slightly, only to change few orders.
That is to say, these two Rules are softer compared to the given one, which may guarantee the sequence will not change dramatically.
Additionally, we can also take one special Proposal Rule, which combines the three basic rules together. That is, when we need to use Proposal Rules, we select one of the three rules with equal probability. This may give more randomness to our generated candidate tour.
The graph of comparison between different Proposal Rules will be given in the Modification Part.

2.3 Modification on Stimulated Annealing Algorithm

After our discussion above, we achieve SA Algorithm. Actually, we can see from those graphs that, among all the iterations, only few of them are used to accept some tours which may not be better. Therefore, some modification should be made.

The modification is mainly from three following parts:

- Choice of Proposal Rules
- Choice of Initial Temperature
- Structure of Iteration

2.3.1 Choice of Proposal Rules

As we mentioned, we can apply different Proposal Rules to Stimulated Annealing Algorithm. Then, we will make comparisons between Basic rule and Combined rule. The Basic rule is just given by swap two location, and the Combined rule is to choose one of the three Basic rules with equal probability. The comparison can be shown as follows:

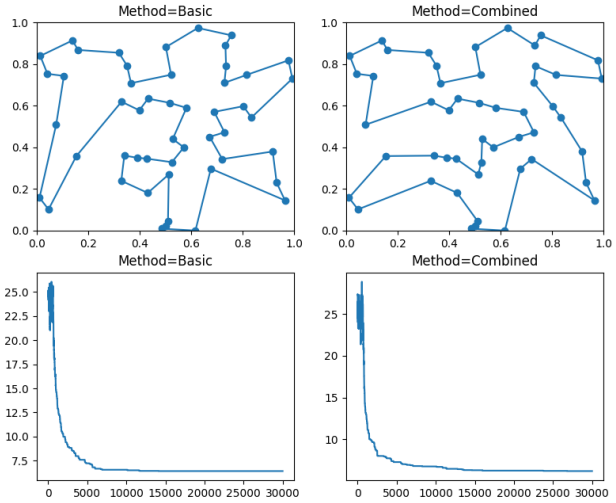

Figure 8: Different choice of Proposal Rules

From the graph, we can observe that, in both two pictures, there are no edges that intersect with each other. This means both two tours are relatively good. but actually from the total distance of two tours, we can draw the conclusion

that the tour obtained by Combined Proposal Rules is slightly better.
Therefore, we can see that using Combined Rule may achieve a better tour. Then, to confirm this, we repeat this procedure for 100 times, checking the frequency of Combined Rule is better than Basic Rule. And in our numerical stimulation, we randomly generate 20 cities to illustrate the superiority of Combined rule. The result can be viewed from the graph below:

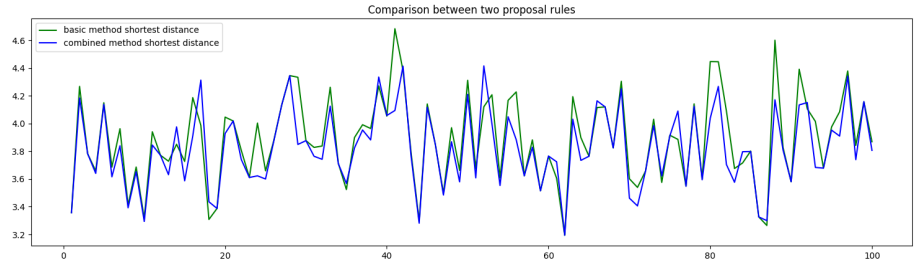

Figure 9: 100 times of comparison

To be more precise, among **100** experiments, there are only **17** cases in which using **Basic rules** leads to **shorter** tour distance. This shows that, at least in simple cases, applying **Combined Proposal Rules** will lead to a better consequence.

2.3.2 Choice of Initial Temperature

When we need to choose an appropriate temperature, we should focus on two things, the scale of given data and the decrease rate η.

As for the scale of given data, notice that our method of accepting some doubtful tours is to check the value of the expression below:

$$e^{-\frac{f_{cand}-f_{curr}}{T}} \geq u$$

In this expression, u is a uniform random variable in $[0, 1]$. Therefore, we should adjust the initial temperature according to the scale of $f_{cand} - f_{curr}$.

As for the decrease rate η, we want to use it to control the value of the whole exponential term. That is, we can choose the initial temperature propotional to $ln\eta$, such that the whole exponential term will not be too big to accept all possible tours. We still need the accepted tours to be somewhere good.

Therefore, our final Adaptive choice of initial temperature can be concluded as follows:

- Pick 30 routes randomly and calculate the maximal and minimal distance

- Choose $T = -\frac{f_{max}-f_{min}}{ln\eta}$

2.3.3 Structure of Iteration

In our Primal SA Algorithm, it can be easily noticed from those graphs that, among all iterations, only few of them are used to accept some doubtful tours, which is not good for us to attain a global optimum tour. Therefore, the modification on iteration structure should be made.
For one thing, we show the modified algorithm as follow:

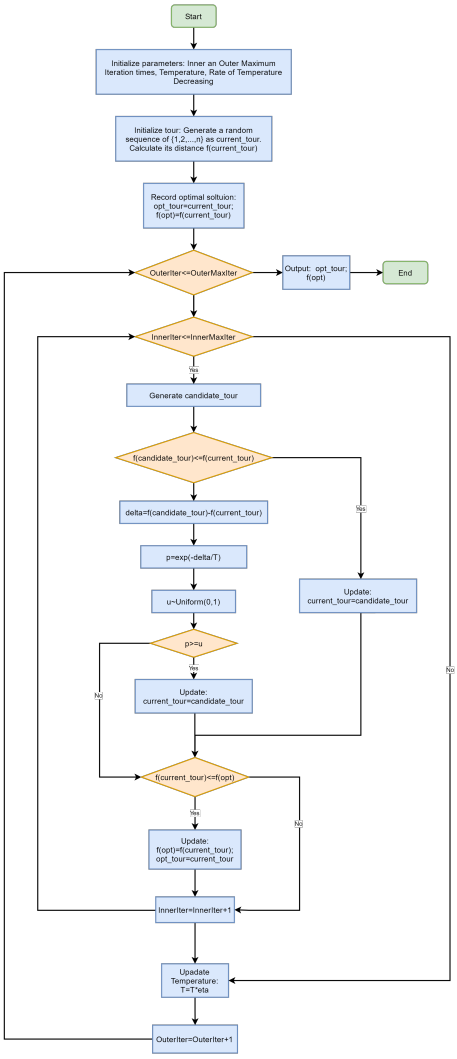

Figure 10: Modified Stimulated Annealing Algorithm

As we mentioned, the decrease of temperature in the Primal SA Algorithm might too fast. To compensate for this, we change our structure of iteration. Our aim is to offer tours more chances to get out of the local optimum, and to achieve this, we set two layers of iteration. In the outer iteration, we choose to decrease the temperature, and in the inner iteration, we let tours iterate sufficiently in each temperature.

For another thing, we want to figure out the Power of this Modified Algorithm. To see whether this algorithm is powerful or not, we compare it with Primal Stimulated Annealing Algorithm as below:

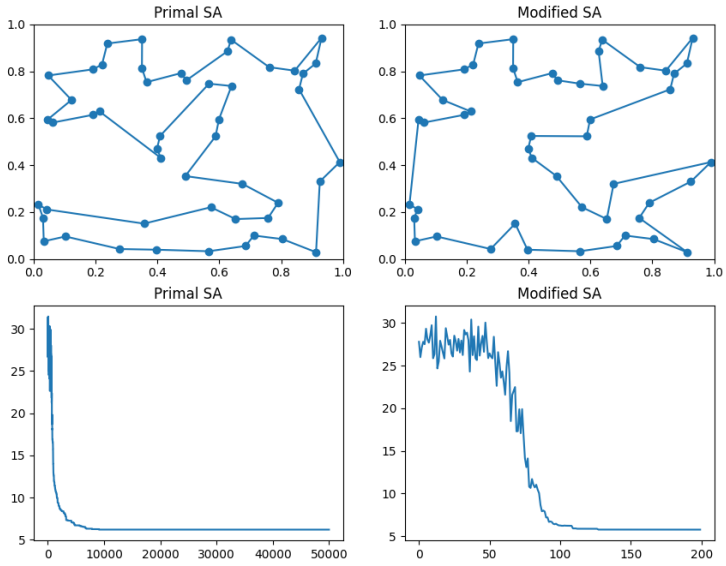

Figure 11: Modified SA vs Primal SA

In the above comparison, there are all **50** cities. The optimal distance of Primal SA is **6.2023**, and that of Modified SA is **5.7772**. This shows that our Modified SA is able to attain a shorter distance. The reason of this can also be seen from lower right corner of the figure above. That is, this Modified algorithm can assign more chances to candidate tour, even if it may not be a better tour, which can haul our current tour out of the local optimum.
Therefore, from the comparison above, we see that the Modified algorithm behaves much better than the Primal on simple dataset. The modification of iteration structure can be of benefit to search for the global optimal solution.

3 Challenging Question

In this question, we apply our modified SA Algorithm to solve this question in order to seek for better solution. And the final result is as follows:

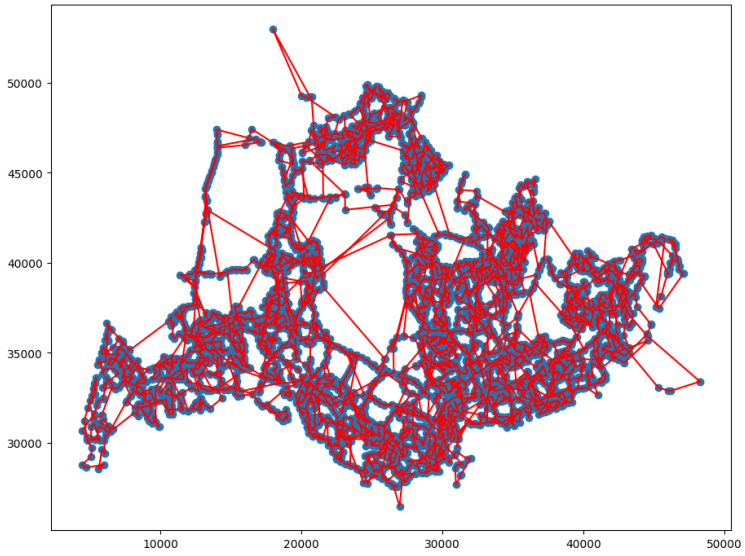

Figure 12: Challenging question solution

It can be seen from the graph that this solution is GOOD in some sense because there do not exist so many intersections in our final result.
In the end, our solution is attained from over **150,000** iterations and the shortest distance is around **1,837,316**, and if we randomly choose one path, the distance is over **10,000,000**. Thererfore, the decrease in the total distance is dramatical.

4 Appendix

4.1 Code for Q1

Function Code:

```
from pulp import *
import numpy as np


def FLP_Solver(c, dd, M, N, output=False):

#2(A). Solve FLP(use PuLP)
PB_FLP = LpProblem (name = 'FLP', sense = LpMinimize)

x_FLP = [LpVariable(f'x{i,j}', lowBound = 0, upBound = 1, cat ='
                                continuous') for i in range(1, M+
                                1) for j in range(1, N+1)]
y_FLP = [LpVariable(f'y{j}', cat = 'Binary') for j in range(1, N+
                                1)]

#objective
PB_FLP += lpDot(x_FLP, dd) + lpDot(y_FLP, c)
#constraints
vector_one1 = np.ones(N)
#equality
for i in range(1, M+1):
PB_FLP += (lpDot(vector_one1, x_FLP[(i-1)*N : i*N]) == 1)
#ineuqality
for j in range(1, N+1):
for i in range(1, M+1):
PB_FLP += (x_FLP[(i-1)*N + j-1] - y_FLP[j-1] <= 0)

# solve

PB_FLP.solve()
if output == True:
print("Status:", LpStatus[PB_FLP.status])
for v in PB_FLP.variables():
if v.varValue > 0:
print(v.name, "=", v.varValue)

print('============================================================
                                FLP
                                ============================================================
                                ')
print('============================================================
                                FLP
                                ============================================================
                                ')
return value(PB_FLP.objective), PB_FLP.variables()


def AFL_Solver(c, dd, M, N, output=False):
# 2(B). Solve AFL(use PuLP)
PB_AFL = LpProblem(name='AFL', sense=LpMinimize)
```

```
x_AFL = [LpVariable(f'x{i, j}', lowBound=0, upBound=1, cat='
                                continuous') for i in range(1, M+
                                1) for j in range(1, N+1)]
y_AFL = [LpVariable(f'y{j}', cat='Binary') for j in range(1, N+1)
                                ]

# objective
PB_AFL += lpDot(x_AFL, dd) + lpDot(y_AFL, c)
# constraints
vector_one1 = np.ones(N)
# equality
for i in range(1, M+1):
PB_AFL += (lpDot(vector_one1, x_AFL[(i-1) * N: i * N]) == 1)
# ineuqality
for j in range(1, N+1):
temp = 0
for i in range(1, M+1):
temp += x_AFL[j-1 + (i-1) * N]
PB_AFL += (temp - M * y_AFL[j-1] <= 0)

# solve

PB_AFL.solve()
if output==True:
print("Status:", LpStatus[PB_AFL.status])
for v in PB_AFL.variables():
if v.varValue > 0:
print(v.name, "=", v.varValue)

print('============================================================
                                AFL
                                ============================================================
                                ')
print('============================================================
                                AFL
                                ============================================================
                                ')
return value(PB_AFL.objective)


def FLP_LR_Solver(c, dd, M, N, output=False):
# 2(C). Solve FLP_LR(use PuLP)

PB_FLP_LR = LpProblem(name='FLP_LR', sense=LpMinimize)

x_FLP_LR = [LpVariable(f'x{i, j}', lowBound=0, upBound=1, cat='
                                LpContinuous') for i in range(1,
                                M+1) for j in range(1, N+1)]
y_FLP_LR = [LpVariable(f'y{j}', lowBound=0, upBound=1, cat='
                                LpContinuous') for j in range(1,
                                N+1)]

# objective
PB_FLP_LR += lpDot(x_FLP_LR, dd) + lpDot(y_FLP_LR, c)
# constraints
vector_one1 = np.ones(N)
# equality
```

```
for i in range(1, M+1):
PB_FLP_LR += (lpDot(vector_one1, x_FLP_LR[(i-1) * N: i * N]) == 1
                                )
# ineuqality
for j in range(1, N+1):
for i in range(1, M+1):
PB_FLP_LR += (x_FLP_LR[(i-1) * N + j-1] - y_FLP_LR[j-1] <= 0)

# solve

PB_FLP_LR.solve()
if output == True:
print("Status:", LpStatus[PB_FLP_LR.status])
for v in PB_FLP_LR.variables():
if v.varValue > 0:
print(v.name, "=", v.varValue)

print('=========================================================
                                FLP_LR
                                =========================================================
                                ')
print('=========================================================
                                FLP_LR
                                =========================================================
                                ')
return value(PB_FLP_LR.objective)


def AFL_LR_Solver(c, dd, M, N, output=False):
# 2(D). Solve AFL_LR(use PuLP)
PB_AFL_LR = LpProblem(name='AFL_LR', sense=LpMinimize)

x_AFL_LR = [LpVariable(f'x{i,j}', lowBound=0, upBound=1, cat='
                                continuous') for i in range(1, M+
                                1) for j in range(1, N+1)]
y_AFL_LR = [LpVariable(f'y{j}', lowBound=0, upBound=1, cat='
                                continuous') for j in range(1, N+
                                1)]
# objective
PB_AFL_LR += lpDot(x_AFL_LR, dd) + lpDot(y_AFL_LR, c)
# constraints
# equality
for i in range(1, M+1):
PB_AFL_LR += (lpSum(x_AFL_LR[(i - 1) * N: i * N]) == 1)
# ineuqality
for j in range(1, N+1):
temp=0
for i in range(1, M+1):
temp += x_AFL_LR[j-1 + (i-1)*N]
PB_AFL_LR += (temp - M*y_AFL_LR[j-1] <= 0)
# solve

PB_AFL_LR.solve()
if output == True:
print("Status:", LpStatus[PB_AFL_LR.status])
for v in PB_AFL_LR.variables():
if v.varValue > 0:
```

```
print(v.name, "=", v.varValue)

print('============================================================
                                AFL_LR
                                ============================================================
                                ')
print('============================================================
                                AFL_LR
                                ============================================================
                                ')
return value(PB_AFL_LR.objective)
```

Main Code:

```
import Q1
import numpy.random
import numpy as np
import matplotlib.pyplot as plt
import math
import re

#Repeat times=100
iter = 100
n, m = 15, 20
#record the opt
opt_val_FLP = np.zeros(iter)
opt_val_AFL = np.zeros(iter)
opt_val_FLP_LR = np.zeros(iter)
opt_val_AFL_LR = np.zeros(iter)

for k in range(iter):
# 1. generate the COEFFICIENT
# step 1: generate N facility locations and M  customer locations
                                 uniformly
fac_x = numpy.random.rand(n)
fac_y = numpy.random.rand(n)

cust_x = numpy.random.rand(m)
cust_y = numpy.random.rand(m)

# step 2: use the generated location to calculate distance d_{i,j
                                }
d = np.zeros((m, n))
for i in range(m):
for j in range(n):
d[i,j] = math.sqrt((cust_x[i] - fac_x[j]) ** 2 + (cust_y[i] -
                                fac_y[j]) ** 2)

#adjust to the objective
distance = d.flatten()

# step 3: generate the cost c_{j} from uniformly distribution
cost = numpy.random.rand(n)

opt_val_FLP[k], variables = Q1.FLP_Solver(c=cost, dd=distance, M=
                                m, N=n)
opt_val_FLP_LR[k] = Q1.FLP_LR_Solver(c=cost, dd=distance, M=m, N=
```

```
                                    n)
opt_val_AFL[k] = Q1.AFL_Solver(c=cost, dd=distance, M=m, N=n)
opt_val_AFL_LR[k] = Q1.AFL_LR_Solver(c=cost, dd=distance, M=m, N=
                                    n)

if k==-1:
#visualize
arr_index_cus=[]
arr_index_fac=[]
arr_customer_x=[]
arr_facility_x=[]
arr_customer_y=[]
arr_facility_y=[]

plt.figure(1)
plt.subplot(121)
colors1 = '#00CED1'
colors2 = '#DC143C'
plt.scatter(fac_x, fac_y, c=colors1, label='facility')
plt.scatter(cust_x, cust_y, c=colors2, label='customer')
plt.legend()
plt.subplot(122)
colors1 = '#00CED1'
colors2 = '#DC143C'
plt.scatter(fac_x, fac_y, c=colors1, label='facility')
plt.scatter(cust_x, cust_y, c=colors2, label='customer')
plt.legend()
for v in variables:
if v.varValue>0:
matchObj = re.match(r'x.(\d{1,2}),_(\d{1,2})',v.name)
if matchObj:
arr_index_cus.append(int(matchObj.group(1)))
arr_index_fac.append(int(matchObj.group(2)))
for item in arr_index_cus:
arr_customer_x.append(cust_x[item-1])
arr_customer_y.append(cust_y[item-1])
for item in arr_index_fac:
arr_facility_x.append(fac_x[item-1])
arr_facility_y.append(fac_y[item-1])
for i in range(m):
plt.arrow(arr_customer_x[i],arr_customer_y[i],arr_facility_x[i]-
                                    arr_customer_x[i],arr_facility_y[
                                    i]-arr_customer_y[i])



# calculate FLP-LR tightness
count_FLP = 0
for i in range(iter):
if opt_val_FLP[i] == opt_val_FLP_LR[i]:
count_FLP+=1

# calculate AFL-LR tightness
count_AFL = 0
for i in range(iter):
if opt_val_AFL[i] == opt_val_AFL_LR[i]:
count_AFL+=1
```

```
print("Among 100 generated data, the number that FLP-Val equal to
                                FLP-LR-Val:", count_FLP)
print("Among 100 generated data, the number that AFL-Val equal to
                                AFL-LR-Val:", count_AFL)

print(opt_val_FLP_LR)
'''
# visualize the generated data(only one time)
if i == 1:
colors1 = '#00CED1'
colors2 = '#DC143C'
plt.scatter(fac_x, fac_y, c=colors1, label='facility')
plt.scatter(cust_x, cust_y, c=colors2, label='customer')
plt.legend()
'''
x_axis=list(range(1,iter+1))
plt.figure(2)
plt.subplot(121)
plt.plot(x_axis, opt_val_FLP, 'b', label='FLP')
plt.plot(x_axis, opt_val_FLP_LR, 'g', label='FLP-LR')
plt.title('FLP vs FLP-LR')
plt.legend()
plt.subplot(122)
plt.plot(x_axis, opt_val_AFL, 'b', label='AFL')
plt.plot(x_axis, opt_val_AFL_LR, 'g', label='AFL-LR')
plt.title('AFL vs AFL-LR')
plt.legend()
plt.show()
```

4.2 Code for Q2

Function Code:

```
import numpy as np
import random
import math
import matplotlib.pyplot as plt
import copy
# Aim: construct the stimulate annealing algorithm for TSP

#(a) Distance function
def TotalDist(tourlist, distance):
'''
:param tourlist: n-dimensional list(permutation of [1,2,...,n])
:param distance: n*n list
:return: totaldist
'''
totaldist = 0
for start, end in zip(tourlist[:-1], tourlist[1:]):
totaldist += distance[start-1][end-1]

totaldist += distance[tourlist[-1]-1][tourlist[0]-1]
return totaldist

#(b) Candidate-created function
```

```
def Create_candidatetour1(currenttour):
'''
:param currenttour:
:return: candidate tour
'''
# choose the position of the elements which need to change
i,j = random.sample(currenttour, 2)
temp=currenttour[i-1:j]
currenttour[i-1:j] = temp[::-1]
return currenttour

def Create_candidatetour2(currenttour):
'''
:param currenttour:
:return: candidate tour
'''
# choose the position of the elements which need to change
i,j = random.sample(currenttour, 2)
currenttour[i-1], currenttour[j-1] = currenttour[j-1],
                                     currenttour[i-1]
return currenttour

def Create_candidatetour3(currenttour):
'''
:param currenttour:
:return: candidate tour
'''
# choose the position of the elements which need to change
i,j = random.sample(currenttour, 2)
currenttour.insert(i-1, currenttour[j-1])
if i > j:
del currenttour[j-1]
else:
del currenttour[j]
return currenttour

def Create_candidatetour4(currenttour):
u = np.random.rand()
if 0 <= u < 1 / 3:
return Create_candidatetour1(currenttour)
elif 1 / 3 <= u < 2 / 3:
return Create_candidatetour2(currenttour)
else:
return Create_candidatetour3(currenttour)



def SA(N, distance, eta, N_iteration, T, Improvement = True,
                                     Method=1, Augument=False):
'''
:param N: number of city
:param distance: N*N matrix of distance between two cities
:param eta: speed of temperature decresing
:param N_iteration: the maximum iteration
:param T: temperature parameter
:return: the solution of TSP(tourlist)
'''
```

```
#initialization(By randomly choosing a sequence)
tourlist = random.sample(list(range(1,N+1)),N)
opt_tourlist = tourlist[:]
opt_len = TotalDist(opt_tourlist,distance)

# record all the candidate distance
all_opt = []
all_len = []

for k in range(N_iteration):
len = TotalDist(tourlist, distance)
temple_path= copy.deepcopy(tourlist)
#choose candidate
if Method == 1:
tourlist_candidate = Create_candidatetour1(tourlist)
elif Method == 2:
tourlist_candidate = Create_candidatetour2(tourlist)
elif Method == 3:
tourlist_candidate = Create_candidatetour3(tourlist)
else:
tourlist_candidate = Create_candidatetour4(tourlist)

len_candidate = TotalDist(tourlist_candidate, distance)

# if decrease, update; if not decrease, update with probability p
if not Augument:
if len_candidate < len:
tourlist = tourlist_candidate[:]
elif math.exp(-(len_candidate - len) / T) >= np.random.rand():
tourlist = tourlist_candidate[:]
else:
tourlist = temple_path
else:
if k<=10000:
if len_candidate < len:
tourlist = tourlist_candidate[:]
elif math.exp(-(len_candidate - len) / T) >= np.random.rand():
tourlist = tourlist_candidate[:]
else:
tourlist = temple_path
else:
if len_candidate < len:
tourlist = tourlist_candidate[:]
else:
tourlist=temple_path

# additional part
if Improvement == True:
if TotalDist(tourlist, distance) < opt_len:
opt_tourlist = tourlist[:]
opt_len = TotalDist(tourlist, distance)

T = T * eta

all_opt.append(opt_len)
all_len.append(TotalDist(tourlist, distance))
```

```
if Improvement == True:
print(opt_len)
print(opt_tourlist)
return opt_tourlist, all_opt, all_len, opt_len
else:
print(len)
print(tourlist)
return tourlist, all_opt, all_len,len

def SA_modify(N, distance, eta, N_iteration, T, tourlist):
# initialization(By randomly choosing a sequence)
#tourlist = random.sample(list(range(1, N + 1)), N)
all_opt=[]
temp_path=[]
temp_len = 0
for out_iter in range(N_iteration):

for in_iter in range(N):
temple_path=copy.deepcopy(tourlist)
len = TotalDist(tourlist, distance)

tourlist_candidate = Create_candidatetour4(tourlist)
len_candidate = TotalDist(tourlist_candidate, distance)

if len_candidate < len:
tourlist = tourlist_candidate[:]
#print('good')
elif math.exp(-(len_candidate - len) / T) >= np.random.rand():
tourlist = tourlist_candidate[:]
#print('yes')
else:
tourlist = temple_path
#print('no')


if in_iter == N-1:
temp_len = TotalDist(tourlist,distance)
temp_path = tourlist

all_opt.append(temp_len)
print(out_iter)
T=T*eta
print(temp_len)
print(temp_path)
return temp_path, all_opt

def initial_temp(N, distance, eta):
dist_list=[]
temp=list(range(1,N+1))
for _ in range(30):
temp = random.sample(temp,N)
dist = TotalDist(temp, distance)
dist_list.append(dist)

t0=-(max(dist_list)-min(dist_list))/math.log(eta)
return t0
def plot_scatter(x_city, y_city):
```

```
plt.scatter(x_city, y_city)
return 0

def plot_route(N,x_city, y_city, sol, color='b'):
x_modify = np.zeros(N + 1)
y_modify = np.zeros(N + 1)
for i in range(N):
x_modify[i] = x_city[sol[i] - 1]
y_modify[i] = y_city[sol[i] - 1]

x_modify[N] = x_modify[0]
y_modify[N] = y_modify[0]
plt.plot(x_modify, y_modify,color=color)
return 0

def plot_iter(iter, all_dist):
plt.plot(range(iter), all_dist)
return 0
```

Main Code:

```
import numpy as np
import math
import Q2
import matplotlib.pyplot as plt

# generate n cities in [0,1]*[0,1]
n1=10
n2=50
n=50

x_city = np.random.rand(n)
y_city = np.random.rand(n)

x_city1 = np.random.rand(n1)
y_city1 = np.random.rand(n1)

x_city2 = np.random.rand(n2)
y_city2 = np.random.rand(n2)

#iter times
N_iter1=50000
N_iter11=14000
N_iter111=10000
N_iter2=200

#eta
eta1=0.99
eta2=0.9

distance=np.zeros((n, n))
distance1=np.zeros((n1,n1))
distance2=np.zeros((n2,n2))
for i in range(n):
for j in range(n):
distance[i,j] = math.sqrt((x_city[i]-x_city[j])**2+(y_city[i]-
                                y_city[j])**2)
```

```
for l in range(n1):
for m in range(n1):
distance1[l,m] = math.sqrt((x_city1[l]-x_city1[m])**2+(y_city1[l]
                                -y_city1[m])**2)

for i in range(n2):
for j in range(n2):
distance2[i,j] = math.sqrt((x_city2[i]-x_city2[j])**2+(y_city2[i]
                                -y_city2[j])**2)


t01=Q2.initial_temp(N=n,distance=distance,eta=eta1)
t02=Q2.initial_temp(N=n,distance=distance,eta=eta2)

sol1, all_distance1, all_len1, len1 = Q2.SA(N=n,distance=distance
                                , eta=eta1, N_iteration=N_iter1,
                                T=t01, Method=4)
sol11, all_distance11 = Q2.SA_modify(N=n,distance=distance, eta=
                                eta2, N_iteration=N_iter2, T=t01)
#sol111, all_distance11, all_len111, len111 = Q2.SA(N=n,distance=
                                distance, eta=eta1, N_iteration=
                                N_iter111, T=t01, Method=4)
#t1=500
#t2=1000
#t3=5000
#sol1, all_distance1, all_len1, len1 = Q2.SA(N=n,distance=
                                distance, eta=eta1, N_iteration=
                                N_iter11, T=t1, Method=1)
#sol11, all_distance11, all_len11, len11 = Q2.SA(N=n,distance=
                                distance, eta=eta1, N_iteration=
                                N_iter11, T=t2, Method=1)
#sol111, all_distance111, all_len111, len111 = Q2.SA(N=n,distance
                                =distance, eta=eta1, N_iteration=
                                N_iter11, T=t3, Method=1)
#sol2, all_distance2= Q2.SA_modify(N=n,distance=distance,eta=eta2
                                ,N_iteration=N_iter2, T=t02)

'''
count=0
method_basic=[]
method_combined=[]
x_axis=list(range(1,101))
for _ in range(100):
x_city = np.random.rand(n)
y_city = np.random.rand(n)
for i in range(n):
for j in range(n):
distance[i, j] = math.sqrt((x_city[i] - x_city[j]) ** 2 + (y_city
                                [i] - y_city[j]) ** 2)
sol1, all_distance1, all_len1, len1 = Q2.SA(N=n, distance=
                                distance, eta=eta1, N_iteration=
                                N_iter11, T=t01, Method=1)
sol11, all_distance11, all_len11, len11 = Q2.SA(N=n, distance=
                                distance, eta=eta1, N_iteration=
                                N_iter11, T=t01,Method=4)
method_combined.append(len11)
method_basic.append(len1)
```

```
if(len11<=len1):
count+=1
print(count)


plt.figure(1)
plt.plot(x_axis, method_basic, color='g', label='basic method
                                shortest distance')
plt.plot(x_axis, method_combined, color='b', label='combined
                                method shortest distance')
plt.title('Comparison between two proposal rules')
plt.legend()
'''
#print(sol1)
#print(distance)

## check iteration\temperature

plt.figure(1)
plt.subplot(221)
Q2.plot_scatter(x_city=x_city, y_city=y_city)
Q2.plot_route(N=n, x_city=x_city, y_city=y_city, sol=sol1)
plt.title('Primal SA')
plt.xlim(0,1)
plt.ylim(0,1)

plt.subplot(222)
Q2.plot_scatter(x_city=x_city, y_city=y_city)
Q2.plot_route(N=n, x_city=x_city, y_city=y_city, sol=sol11)
plt.title('Modified SA')
plt.xlim(0,1)
plt.ylim(0,1)

'''
plt.subplot(133)
Q2.plot_scatter(x_city=x_city, y_city=y_city)
Q2.plot_route(N=n, x_city=x_city, y_city=y_city, sol=sol111)
plt.title('N=100')
plt.xlim(0,1)
plt.ylim(0,1)
'''
#check iteration\temperature

#plt.figure(2)
#Q2.plot_iter(iter=N_iter1, all_dist=all_distance1)
plt.figure(1)
plt.subplot(223)
Q2.plot_iter(iter=N_iter1, all_dist=all_len1)
plt.title('Primal SA')
plt.subplot(224)
Q2.plot_iter(iter=N_iter2, all_dist=all_distance11)
plt.title('Modified SA')
#plt.subplot(233)
#Q2.plot_iter(iter=N_iter11, all_dist=all_len111)
#plt.title('T=2000')

#check initial temperature
```

```
'''

plt.figure(4)
Q2.plot_scatter(x_city=x_city, y_city=y_city)
Q2.plot_route(N=n, x_city=x_city, y_city=y_city, sol=sol2)
plt.xlim(0,1)
plt.ylim(0,1)

plt.figure(5)
Q2.plot_iter(iter=N_iter2, all_dist=all_distance2)

'''
#plt.figure(6)
#Q2.plot_scatter(x_city=x_city, y_city=y_city)
#plt.title('The diagram of generated cities')


plt.show()
```

4.3 Code for Challenging Question

```
import numpy as np
import math
import Q2
import matplotlib.pyplot as plt
from csv import reader

filename=r'C:\Users\Faust\Desktop\busstops.csv'
bus_x=[]
bus_y=[]
# dict search
bus_id=[]
with open(filename, 'rt', encoding='UTF-8') as raw_data:
readers=reader(raw_data, delimiter=',')
x=list(readers)
data=np.array(x)
for item in data:
bus_x.append(float(item[2]))
bus_y.append(float(item[3]))
bus_id.append(item[0])

N=len(bus_x)
distance=np.zeros((N, N))
for i in range(N):
for j in range(N):
distance[i,j] = math.sqrt((bus_x[i]-bus_x[j])**2+(bus_y[i]-bus_y[
                                    j])**2)

eta1=0.99
eta2=0.9
#t01=Q2.initial_temp(N=N,distance=distance,eta=eta1)
#t02=Q2.initial_temp(N=N,distance=distance,eta=eta2)
```

```
N_iter1=150000
N_iter2=2
initial_tour=np.arange(5137)
#print(t01)
#sol1, all_distance1, all_len1, len1 = Q2.SA(N=N,distance=
                                distance, eta=eta1, N_iteration=
                                N_iter1, T=t01, Method=4,
                                Augument=True)
sol11, all_distance11 = Q2.SA_modify(N=N,distance=distance, eta=
                                eta2, N_iteration=N_iter2, T=1,
                                tourlist=initial_tour)


#visualize the bus stop
plt.figure(1)
Q2.plot_scatter(x_city=bus_x, y_city=bus_y)
Q2.plot_route(N=N, x_city=bus_x, y_city=bus_y, sol=sol11, color='
                                r')

#visualize the procedure
plt.figure(2)
Q2.plot_iter(iter=N_iter2, all_dist=all_distance11)


plt.show()
```