Application of Computational Intelligence in Engineering

Final project - Michelin-starred restaurants

1. Introduction (background, problem definition)

Background

The Traveling Salesman Problem (TSP) is an NP-hard problem in combinatorial optimization. The content of the title is "Given a series of cities and the distance between each pair of cities, find the shortest path that visits each city once and returns to the starting city". In Taiwan, with the rise of the logistics and distribution industry, finding the shortest path between various sites has become an important key for operators to reduce distribution costs. Thus, we wanted to design a distribution route for ingredient distributors.

Problem definition

Our clients are mostly Michelin star restaurants, so we decided to find 16 Michelin star restaurants and find the shortest route for the delivery staff. We used the real routes from google to calculate distances between 16 restaurants to make the distance matrix. And when we calculated the total distances, we did not consider the distance from the last restaurant to the first delivery restaurant. Then we used 4 different algorithms to solve TSP and try to find the minimum distance delivery path.

2. Algorithmic design and parameter setting

First, we encoded the 16 restaurants to 0 to 15 and used 4 different algorithms to major their differences. The following are the algorithms we used and the parameter setting for each algorithm that we used full factor design to choose the best parameter:

- Genetic Algorithm (GA): iteration=500, mutation rate=0.9, crossover rate=0.1
- Ant Colony Optimization (ACO): iteration=500, evaporate rate=0.1, α=1, β=7, Q in heuristic=5
- Simulated Annealing (SA): $\text{starting temperature} = 1000, \alpha = 0.98$ $\text{final temperature} = 0.001$
- Particle Swarm Optimization (PSO): iteration=500, swarm size=100, we used the method of changing the restaurant order to update the velocity, so we did not use the inertia weight and acceleration constants.

3. Numerical example

We convert the 16 restaurants into numbers from 0 to 15 and calculate their distance matrix. The result is as follows:

	晶華軒 S	辰園｜台	香宮｜香	心潮飯店	欣葉台菜	雞家莊(	旬採 (中	日本橋玉	壽司芳	想想廚房	Thai & Th	Chope Ch	頤宮中餐	田園海鮮	真的好海	先進海產店
晶華軒 Silks House	0															
辰園｜台北喜來登	1.9	0														
香宮｜香格里拉台北遠東	5.9	5	0													
心潮飯店	6.4	5.5	3.6	0												
欣葉台菜(創始店)	2.2	2.9	7.3	7.3	0											
雞家莊(長春路)	0.8	1.4	5.4	6	1.5	0										
旬採 (中山)	1.3	1.3	7	7.7	1.5	1	0									
日本橋玉井	4.3	5.6	3.9	4.5	3.3	2.5	3	0								
壽司芳	4.9	4	1.8	2.2	6	4.1	5	3.9	0							
想想廚房	1.9	2.2	4	5.3	3.1	1.6	2.2	3.2	3.1	0						
Thai & Thai	3.2	3.9	4	4.5	3.8	1	1	1.4	3	2.3	0					
Chope Chope	6	5.2	3	0.5	7.5	5.9	6.5	4.8	1.7	4.8	4	0				
頤宮中餐廳	1.3	1.2	5.3	6	2.8	1.6	1.5	5.5	4.6	2.2	1.3	6.3	0			
田園海鮮	2.6	2.2	3.5	4.1	3.5	2	2.6	2.9	2.6	1.1	2	3.9	2.5	0		
真的好海鮮	3.9	3.2	2.1	3.3	5.1	3.3	4.1	3.5	2.1	2.2	2.9	3	3.5	1.7	0	
先進海產店	3.9	3.8	2.7	2.2	5.5	3.6	4.3	3.3	0.9	2.3	2.3	2.2	4	1.9	2.2	0

- c_{ij} is distance from restaurant i to restaurant j.
- Decision variables：
- $x_{ij} = \begin{cases} 1, the\ path\ goes\ from\ restaurant\ i\ to\ restaurant\ j \\ 0, otherwise \end{cases}$

$$\text{minimize} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{subject to} \sum_{j=1}^{n} x_{ij} = 1, i = 1, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, n$$

$$u_i - u_j + nx_{ij} \leq n - 1, i, j = 2, \ldots, n$$

$$x_{ij} \in \{0, 1\},$$

4. Optimization result (selection of tuning parameters, final solution and convergence history)

The parameter selection method of GA, ACO and SA are full factorial experiment and the algorithm used by PSO will randomly select the parameters and automatically find the best parameters.

The selected parameters and the best parameters are as follows:

Algorithm	method	Parameter	Parameter	Parameter
GA	Full factor	Mutation rate [0.1,0.5,0.9]	Crossover rate [0.1,0.5,0.9]	
ACO	Full factor	Evaporate rate [0.1,0.3,0.5]	Q in heuristic [3,5,10]	α =[1, 2, 3] β =[3, 5, 7]
SA	Full factor	Starting T [500,1000,2000]	End T [0.05,0.01,0.001]	α [0.95,0.98,0.99]
PSO	-	-	-	-

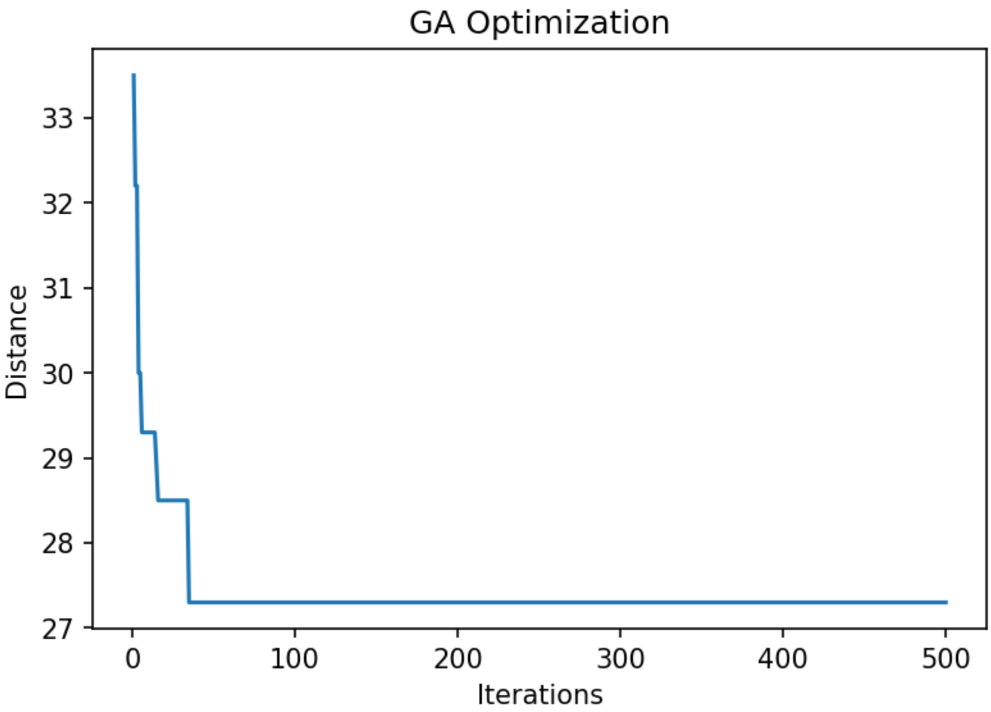
Final solution:

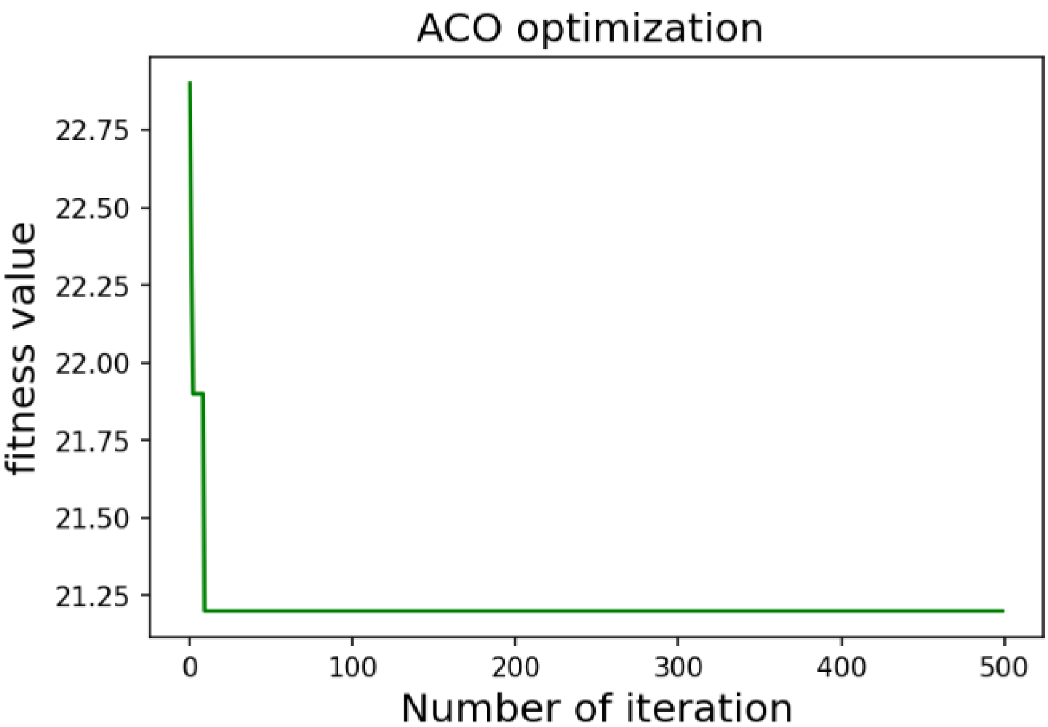
The following are the best results of each algorithm after 100 experiments.

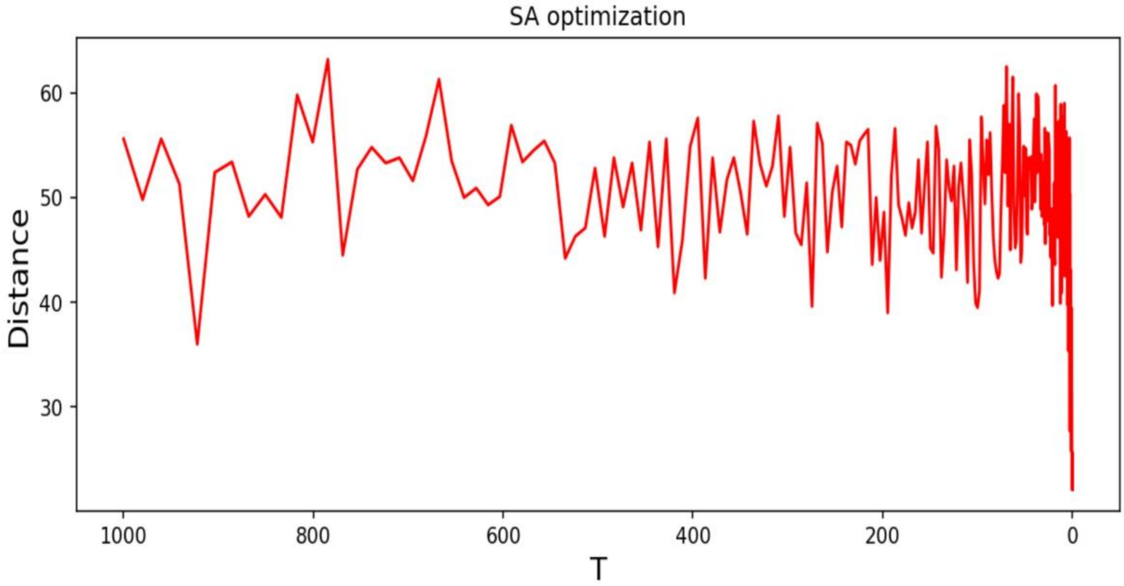
Algorithm	Route	Distance
GA	[4, 6, 0, 5, 12, 1, 13, 9, 10, 7, 15, 2, 3, 11, 8, 14]	25.3
ACO	[12, 1, 6, 7, 10, 5, 2, 8, 15, 3, 11, 13, 9, 14, 4, 0]	21.2
SA	[7, 10, 6, 4, 5, 0, 12, 1, 9, 13, 14, 2, 8, 15, 3, 11]	21.2
PSO	[3, 11, 15, 8, 2, 14, 13, 9, 5, 4, 0, 6, 1, 12, 10, 7]	21.6

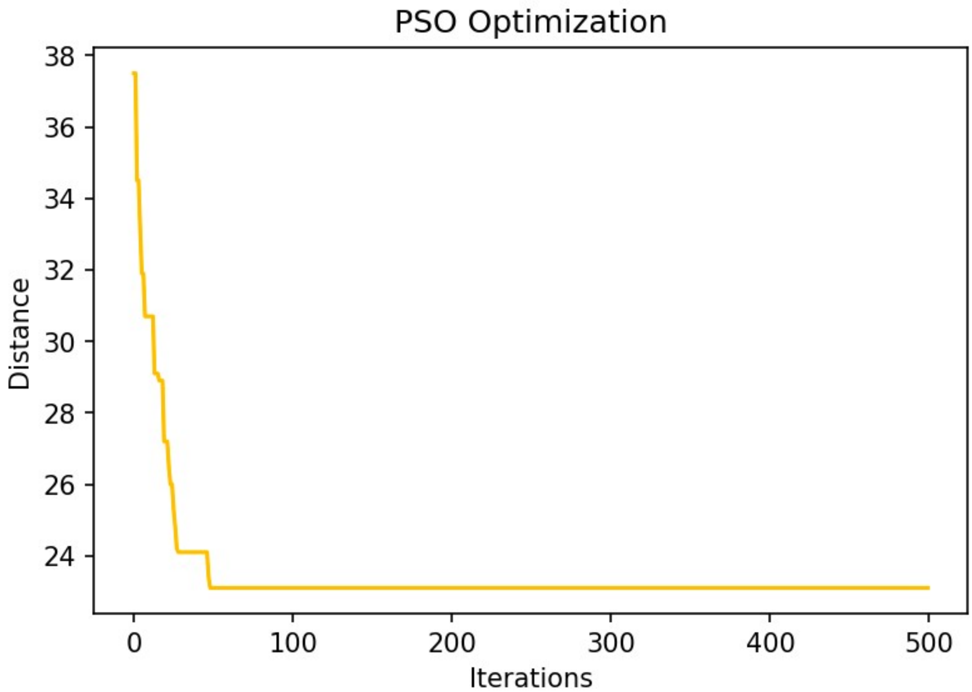
It can be seen that both SA and ACO can find the best solution, the result of PSO is the second best, and the result of GA is the worst.

Convergence history:






SA optimization
Distance
60
50
40
30
1000
800
600
400
200
0
T


PSO Optimization
Distance
38
36
34
32
30
28
26
24
0
100
200
300
400
500
Iterations

5. Validation of the performance (run the algorithm for multiple times and show the statistics: is the algorithm consistently effective and efficient?). A comparison with previous work or other metaheuristics would be advantageous.

Run 100 times

	Mean	Variance	Time
Genetic Algorithm(GA)	28.557	1.052	1694.83s
Ant Colony Optimization(ACO)	21.2	$1.07e^{-14}$	7832.915s
Simulated Annealing(SA)	23.602	1.674	1266.18s
Particle Swarm Optimization(PSO)	24.435	1.773	79.33s

After 100 experimental verifications, we found that ACO is a consistently effective algorithm because its variance and average distance are very small. ACO is also advantageous in efficiency, although it spend most time in all algorithm , from the convergence history, ACO always converge in 100 iteration .The convergence speed of ACO is very fast.

6. Conclusions

After many experimental verifications, we found that ACO has quite good performance in Effectiveness, Efficiency, and Consistency compared to other algorithms.

It is worth mentioning that in the results of running 100 times, ACO can find the best solution almost every time, and the variation is very small. We think this result is very reasonable, because in original ACO paper ,Dorigo used the similarity between ant colony searching for food path and the famous TSP, to solve the TSP by artificially simulated the process of ant searching for food.

Finally, we experimentally verify that ACO is a very suitable algorithm for solving TSP.

Future research：

- Use other algorithms. (e.g., Tabu search, artificial bee colony algorithm)
- Escalate to Multiple Travel Salesperson Problem(m-TSP)

Limitations：

- Simple path calculation.
- Add road conditions to modify the route in time

7. References (highlight the publications mostly related to your work)

- K Katayama, H Sakamoto, H Narihisa, The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem, Mathematical and Computer Modelling, Volume 31, Issues 10–12, 2000, Pages 197-203, ISSN 0895-7177, https://doi.org/10.1016/S0895-7177(00)00088-1.

8. Appendix:

SA code:

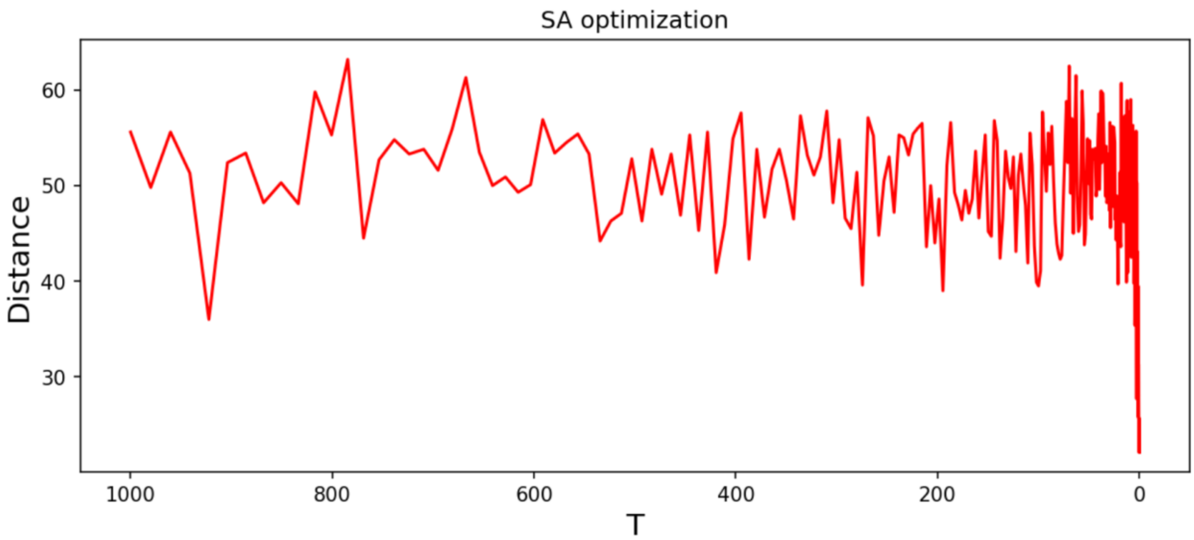
```
import pandas as pd
import math
import random
import matplotlib.pyplot as plt
import time
from pandas import Series, DataFrame
from itertools import permutations
from random import sample
import numpy as np

cf = pd.read_csv("MAPP0.csv")
```

```
def init_solv(): #get the unitial solution
    a = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
    random.shuffle(a)

    return a

def cal_dist(per): #calculate distance
        dis = 0
        for i in range(15):
            dis += cf.iloc[per[i]][per[i+1]]
        return dis
```

```
def plot(results): #draw convergence history

                X = []
                Y = []
                for i in range(len(T)):
                        X.append(T[i])
                        Y.append(results[i])
                plt.figure(figsize=(10,4),dpi = 150)
                plt.plot(X,Y,color='red')
                plt.gca().invert_xaxis()
                plt.xlabel('T',size = 15)
                plt.ylabel('Distance',size = 15)
                plt.title('SA optimization')
                plt.show()
```

```
exp_r = [] #紀錄實驗結果(路徑)
exp_d = [] #紀錄實驗結果(距離)
TIME = [] #紀錄實驗結果(時間)


for k in range(100): #次實驗

    start = time.time()
    start_T = 1000 #初始化溫度,起始解
    end_T = 0.001
    b = init_solv()
    results = [] #紀錄降溫結束後距離,用於繪圖
    T = [] #紀錄降溫


    while start_T > end_T: #outer loop

        for i in range(10): #inner loop
        # for i in range(int(100*math.exp(-start_T/500))):
            b_1 = b.copy()
            s = random.sample(range(0,16),2) #sample 2 nodes do Transposition
            b_1[s[0]], b_1[s[1]] = b_1[s[1]], b_1[s[0]] #Transposition
            delt = cal_dist(b_1)-cal_dist(b) #calculate delta

            if delt < 0: #replace process
                b = b_1

            elif random.random()< math.exp(-delt/start_T):
                b = b_1

        results.append(cal_dist(b))
        T.append(start_T)
        start_T = start_T*0.98 #cooling process

    end = time.time()
    Time = end - start
    exp_r.append(b)
    exp_d.append(cal_dist(b))
    TIME.append(Time)
```

```
plot(results)
```




```
print("100次實驗中最短距離為:",min(exp_d))
print("100次實驗中平均距離為:",sum(exp_d)/100)
print("100次實驗中距離標準差為:",np.std(exp_d))
print("100次實驗中最短路徑:",exp_r[exp_d.index(min(exp_d))])
print("1次實驗時間:",TIME[0])
print("1次實驗距離:",exp_d[4])
print("1次實驗路徑:",exp_r[4])
print("100次實驗平均時間,",sum(TIME)/100)
```

```
100次實驗中最短距離為: 21.199999999999996
100次實驗中平均距離為: 23.601999999999993
100次實驗中距離標準差為: 1.29367538432174
100次實驗中最短路徑: [7, 10, 6, 4, 5, 0, 12, 1, 9, 13, 14, 2, 8, 15, 3, 11]
1次實驗時間: 12.708637952804565
1次實驗距離: 23.699999999999996
1次實驗路徑: [4, 6, 0, 5, 9, 14, 2, 8, 11, 3, 15, 13, 1, 12, 10, 7]
100次實驗平均時間, 12.661840207576752
```

PSO code：

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import random
import statistics
import math
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import time
```

```
# the distance_matrix
df=pd.read_csv("MAPP.csv")
df=df.drop(['Unnamed: 0'],axis=1)
distance_matrix=np.array(df)
distance_matrix
```

```python
#caculate the toatal distance
def fitness_func(distance_matrix, x_i):
    total_distance = 0                          #x_i為PSO的一个解（路徑順序）
    for i in range(1, city_num):
        start_city = x_i[i - 1]
        end_city = x_i[i]
        total_distance += distance_matrix[start_city][end_city]
    #total_distance += distance_matrix[x_i[-1]][x_i[0]]         #從最後的站回到出發的站
    return total_distance

#update the velocity
def get_ss(x_best, x_i, r):                    #r指隨機產生的r1,r2
    velocity_ss = []
    for i in range(len(x_i)):
        if x_i[i] != x_best[i]:
            j = np.where(x_i == x_best[i])[0][0]
            so = (i, j, r)
            velocity_ss.append(so)
            x_i[i], x_i[j] = x_i[j], x_i[i]      #執行交換操作
    return velocity_ss


#update the location
def do_ss(x_i, ss):
    for i, j, r in ss:                  #ss用來算r1(pbest-xi),r2(gbest-xi))
        rand = np.random.random()
        if rand <= r:
            x_i[i], x_i[j] = x_i[j], x_i[i]
    return x_i
```

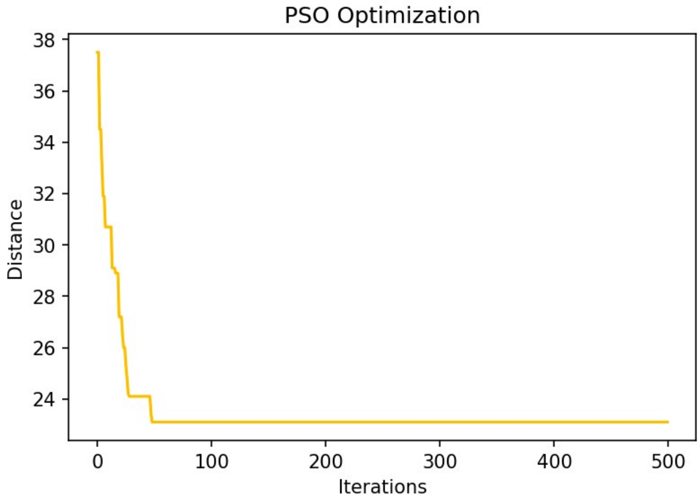
```python
#main algorithm
iter_max_num=500
size = 100
city_num = 16
def main(city_num ,size,iter_max_num ):
    r1 = np.random.rand()
    r2 = np.random.rand()
    start =time.time()

    #初始化群各个粒子的位置，作為個體的歷史最佳pbest
    #每行都是1~16(站)的不重復隨機數，來表示站的先後順序
    pbest_init = np.zeros((size, city_num), dtype=np.int64)
    for i in range(size):
        pbest_init[i] = np.random.choice(list(range(city_num)), size=city_num, replace=False)

    #caculate gbest,gbest fitness
    pbest = pbest_init
    pbest_fitness = np.zeros((size, 1))
    for i in range(size):
        pbest_fitness[i] = fitness_func(distance_matrix, x_i=pbest_init[i])

    #caculate gbest,gbest fitness
    gbest = pbest_init[pbest_fitness.argmin()]
    gbest_fitness = pbest_fitness.min()

    #進行迭代
```

```python
    #進行迭代
    all_best=[]
    all_bestinter=[]
    for k in range(iter_max_num):
        for j in range(size):

            pbest_i = pbest[j].copy()
            x_i = pbest_init[j].copy()   #define initial x_i

            #計算交換序列，r1(pbest-xi),r2(gbest-xi)
            ss1 = get_ss(pbest_i, x_i, r1)
            ss2 = get_ss(gbest, x_i, r2)
            ss = ss1 + ss2
            #print(f'{ss1}+{ss2}={ss}')
            x_i = do_ss(x_i, ss)           # 進行交換，new location=old location+velocity

            #update pbest
            fitness_new = fitness_func(distance_matrix, x_i)
            fitness_old = pbest_fitness[j]
            if fitness_new < fitness_old:
                pbest_fitness[j] = fitness_new
                pbest[j] = x_i

           #update gbest
            gbest_fitness_new = pbest_fitness.min()
            gbest_new = pbest[pbest_fitness.argmin()]
            if gbest_fitness_new < gbest_fitness:
                gbest_fitness = gbest_fitness_new
                gbest = gbest_new


        all_bestinter.append(k)              #紀錄迭代次數
        all_best.append(gbest_fitness)       #記錄每次迭代的gbest_fitness
        all_mean= statistics.mean(all_best)


    end=time.time()                     #caculate tatal running time
    tt=end-start
```

```python
    #迭代结果
    print(f'迭代最優路徑為:{gbest}')       #沒f,會print 出{gbest},有{}
    print(f'迭代最優值為:{gbest_fitness}')
    print('time: %s Seconds'%(end-start))
    #print(f'迭代平均為:{all_mean}')
    #print(f'迭代變異數為:{(statistics.variance(all_best))}')

    return all_best , all_bestinter ,all_mean,tt, gbest_fitness
```

```python
#graph
import math
a,b,c,d,e = main(city_num = 16,size = 100,iter_max_num=500)
plt.figure(figsize=(4,1))
plt.figure(dpi=150)
plt.plot(b,a,'#fac205')
plt.title("PSO Optimization")
plt.xlabel("Interations")
plt.ylabel("Distance")
```

```
迭代最優路徑為:[ 4  5  0  6  1 12 10  7 14  2  8 11  3 15 13  9]
迭代最優值為:23.099999999999998
time: 0.8290970325469971 Seconds
```




```python
#run algorithm 100 times to varify the result
i,j,k,l,m=main(city_num=16 ,size=100,iter_max_num=500)
Best=[]
start = time.time()
for h in range(100):
    Best.append(m)

end = time.time()
tot = end-start
#print(Best)
print("100 times time:",tot)
print("100 times average:",np.mean(Best))
print("100 times variance:",np.var(Best))
```

```
100 times time: 79.33401203155518
100 times average: 24.435
100 times variance: 1.7730749999999993
```

ACO code：

Import necessary package and problem file

```python
from google.colab import drive
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
from itertools import permutations

drive.mount('/content/drive/')
distance = pd.read_csv('/content/drive/My Drive/colab/計算智慧MAPP.csv')
```

Distance Matrix

```
[ ] distance = distance.iloc[:16,1:17]
    distance.set_axis(['0','1','2','3','4','5','6','7','8','9','10','11','12','13','14','15'] , axis='columns', inplace=True)

    for i in range(len(distance)): #fill the NA
        distance.iloc[i,i+1:len(distance)] = distance.iloc[i+1:len(distance),i]

    distance
```

Caculate the total distance for route

```
[ ] def objective_value(routes):
        total_distance = 0
        for i in range(len(routes)-1):
            city1 = routes[i]
            city2 = routes[i+1]
            total_distance += distance.iloc[city1,city2]
        return total_distance
```

Construst the initial solution

```
[ ] def initial(pop_size,num_city):
        for i in range(pop_size): #Randomly generate a set of initial solutions
            solutions[i] = random.sample(range(16),16)

        for from_ in range(num_city): #Calculate visibility (reciprocal distance)
            for to_ in range(num_city):
                if from_ == to_:continue
                visibility[from_,to_] = 1/distance.iloc[from_,to_]

        return solutions
```

Using roulette to choose the next city

```
[ ] def roullete_wheel(fitness_list_):
        sum_fitness = sum(fitness_list_)
        transition_probability = []

        for fit in fitness_list_:
            transition_probability.append(fit/sum_fitness)

        #caculate cummulative probility
        for i in range(len(transition_probability)-1,-1,-1): # range(start,stop,[step])
            total = 0.0
            j=0
            while(j<=i):
                total += transition_probability[j]
                j += 1
            transition_probability[i] = total

        # roullete wheel selection
        for i in range(len(fitness_list_)):
            if transition_probability[i] > random.random():
                return i
```

Update pheromone

```
[ ] def update_pheromone1(pheromone_map, evaporate_rate, Q):
        pheromone_map *= (1-evaporate_rate) #Pheromone evaporation
        for solution in solutions:
            for j in range(num_city-1):
                city1 = solution[j]
                city2 = solution[j+1]
                    #Pheromone intensification
                pheromone_map[city1, city2] += Q/distance.iloc[city1, city2]
        return pheromone_map
```

Construct one solution

```
[ ] def one_solution_construction(alpha, beta):
        candidates = []
        one_solution = np.arange(num_city, dtype=int)

        for i in range(num_city): #Create candidate cities
            candidates.append(i)

        current_city = random.choice(candidates) #create starting city
        one_solution[0] = current_city #starting city
        candidates.remove(current_city)

        for i in range(1, num_city-1): #The first city has been selected
            fitness_list = []
            for city in candidates: #Calculate the fitness of all candidate cities
                fitness = pow(pheromone_map[current_city][city], alpha)*\
                          pow(visibility[current_city][city], beta)
                fitness_list.append(fitness)
            #Use the roulette method to choose the next city
            #The higher the fitness, the easier it is to be selected
            next_city = candidates[ roullete_wheel(fitness_list) ]
            candidates.remove(next_city)
            one_solution[i] = next_city

            current_city = next_city #move to next city
        one_solution[-1] = candidates.pop() #last city
        return one_solution
```

Construst the solutions by all ant

```
[ ] def each_ant_construct_solution(alpha, beta):
        for i in range(pop_size):
            solutions[i] = one_solution_construction(alpha, beta)
            fitness_value[i] = objective_value(solutions[i])

        return solutions, fitness_value
```

Plot the convergency history

```
[ ] def plot(results):
        X = []
        Y = []
        for i in range(iter):
            X.append(i)
            Y.append(results[i])
        plt.plot(X, Y, color='green')
        plt.plot(X, Y, color='green')
        plt.xlabel('Number of iteration', size = 15)
        plt.ylabel('fitness value', size = 15)
        plt.title('ACO optimization ', size = 15)
        plt.show()
```

Main algorithm

```
[ ] def  ACO(iter, pop_size, num_city, evaporate_rate, Q, alpha, beta):
        best_obj_value  =  100
        best_solution  =  np.arange(num_city)
        results_solution, results_fitness  =  [], []
        initial(pop_size, num_city)

        for  i  in  range(iter):
            one_solution_construction(alpha, beta)
            each_ant_construct_solution(alpha, beta)
            update_pheromone1(pheromone_map, evaporate_rate, Q)

            #Update  the  best  solution
            for  j  in  range(pop_size):
                if  fitness_value[j]  <  best_obj_value:
                    best_obj_value  =  fitness_value[j]
                    best_solution  =  solutions[j]

            print('iteration  is  :', i, 'Best  solution', best_solution, 'Best  fitness', best_obj_value)

            results_solution.append(best_solution)
            results_fitness.append(best_obj_value)
        print('final  solution  :', results_solution[-1], 'final  distance  :', results_fitness[-1])
        plot(results_fitness)
        return  results_fitness[-1]
```

Using full factor designt to tuning parameter

Each parameter have 3 level

```
iter  =  500
evaporate_rate  =  [0.1,0.3,0.5]  #rho
Q  =  [3,5,10]
alpha  =  [1,2,3]  #pheromone_factor
beta  =  [3,5,7]  #visibility_factor
pop_size  =  100
num_city  =  len(distance)

parameter  =  []
ACO_result3  =  []
Experiment_num  =  1

for  rho  in  evaporate_rate:
    for  q  in  Q:
        for  a  in  alpha:
            for  b  in  beta:
                parameter  =  []
                parameter.append(rho)
                parameter.append(q)
                parameter.append(a)
                parameter.append(b)
                print('Experiment:', Experiment_num, 'parameter  :', parameter, 'objective  value  :', ACO2(iter, pop_size, num_city, rho, q, a, b))
                Experiment_num  +=  1
```

```
Experiment: 1 parameter : [0.1, 3, 1, 3] objective value : 21.9
Experiment: 2 parameter : [0.1, 3, 1, 5] objective value : 21.9
Experiment: 3 parameter : [0.1, 3, 1, 7] objective value : 21.199999999999996
Experiment: 4 parameter : [0.1, 3, 2, 3] objective value : 21.799999999999997
Experiment: 5 parameter : [0.1, 3, 2, 5] objective value : 22.299999999999997
Experiment: 6 parameter : [0.1, 3, 2, 7] objective value : 22.299999999999997
Experiment: 7 parameter : [0.1, 3, 3, 3] objective value : 22.299999999999997
Experiment: 8 parameter : [0.1, 3, 3, 5] objective value : 22.299999999999997
Experiment: 9 parameter : [0.1, 3, 3, 7] objective value : 22.299999999999997
Experiment: 10 parameter : [0.1, 5, 1, 3] objective value : 21.9
Experiment: 11 parameter : [0.1, 5, 1, 5] objective value : 21.9
Experiment: 12 parameter : [0.1, 5, 1, 7] objective value : 21.199999999999996
Experiment: 13 parameter : [0.1, 5, 2, 3] objective value : 21.199999999999996
Experiment: 14 parameter : [0.1, 5, 2, 5] objective value : 22.7
Experiment: 15 parameter : [0.1, 5, 2, 7] objective value : 22.7
Experiment: 16 parameter : [0.1, 5, 3, 3] objective value : 22.7
Experiment: 17 parameter : [0.1, 5, 3, 5] objective value : 22.7
Experiment: 18 parameter : [0.1, 5, 3, 7] objective value : 22.7
Experiment: 19 parameter : [0.1, 10, 1, 3] objective value : 21.9
Experiment: 20 parameter : [0.1, 10, 1, 5] objective value : 21.199999999999996
Experiment: 21 parameter : [0.1, 10, 1, 7] objective value : 21.199999999999996
Experiment: 22 parameter : [0.1, 10, 2, 3] objective value : 21.199999999999996
Experiment: 23 parameter : [0.1, 10, 2, 5] objective value : 22.5
Experiment: 24 parameter : [0.1, 10, 2, 7] objective value : 22.5
Experiment: 25 parameter : [0.1, 10, 3, 3] objective value : 22.5
Experiment: 26 parameter : [0.1, 10, 3, 5] objective value : 22.5
Experiment: 27 parameter : [0.1, 10, 3, 7] objective value : 22.5
Experiment: 28 parameter : [0.3, 3, 1, 3] objective value : 21.9
Experiment: 29 parameter : [0.3, 3, 1, 5] objective value : 22.2
Experiment: 30 parameter : [0.3, 3, 1, 7] objective value : 22.2
Experiment: 31 parameter : [0.3, 3, 2, 3] objective value : 22.299999999999997
Experiment: 32 parameter : [0.3, 3, 2, 5] objective value : 23.2
Experiment: 33 parameter : [0.3, 3, 2, 7] objective value : 24.500000000000004
Experiment: 34 parameter : [0.3, 3, 3, 3] objective value : 24.500000000000004
Experiment: 35 parameter : [0.3, 3, 3, 5] objective value : 24.500000000000004
Experiment: 36 parameter : [0.3, 3, 3, 7] objective value : 24.500000000000004
Experiment: 37 parameter : [0.3, 5, 1, 3] objective value : 21.9
Experiment: 38 parameter : [0.3, 5, 1, 5] objective value : 21.199999999999996
Experiment: 39 parameter : [0.3, 5, 1, 7] objective value : 21.199999999999996
Experiment: 40 parameter : [0.3, 5, 2, 3] objective value : 21.9
```

```
Experiment: 43 parameter : [0.3, 5, 3, 3] objective value : 21.9
Experiment: 44 parameter : [0.3, 5, 3, 5] objective value : 21.9
Experiment: 45 parameter : [0.3, 5, 3, 7] objective value : 21.9
Experiment: 46 parameter : [0.3, 10, 1, 3] objective value : 21.199999999999996
Experiment: 47 parameter : [0.3, 10, 1, 5] objective value : 21.199999999999996
Experiment: 48 parameter : [0.3, 10, 1, 7] objective value : 21.199999999999996
Experiment: 49 parameter : [0.3, 10, 2, 3] objective value : 21.199999999999996
Experiment: 50 parameter : [0.3, 10, 2, 5] objective value : 21.199999999999996
Experiment: 51 parameter : [0.3, 10, 2, 7] objective value : 21.9
Experiment: 52 parameter : [0.3, 10, 3, 3] objective value : 21.9
Experiment: 53 parameter : [0.3, 10, 3, 5] objective value : 21.9
Experiment: 54 parameter : [0.3, 10, 3, 7] objective value : 21.9
Experiment: 55 parameter : [0.5, 3, 1, 3] objective value : 21.199999999999996
Experiment: 56 parameter : [0.5, 3, 1, 5] objective value : 21.199999999999996
Experiment: 57 parameter : [0.5, 3, 1, 7] objective value : 21.199999999999996
Experiment: 58 parameter : [0.5, 3, 2, 3] objective value : 21.9
Experiment: 59 parameter : [0.5, 3, 2, 5] objective value : 21.9
Experiment: 60 parameter : [0.5, 3, 2, 7] objective value : 21.9
Experiment: 61 parameter : [0.5, 3, 3, 3] objective value : 21.9
Experiment: 62 parameter : [0.5, 3, 3, 5] objective value : 21.9
Experiment: 63 parameter : [0.5, 3, 3, 7] objective value : 21.9
Experiment: 64 parameter : [0.5, 5, 1, 3] objective value : 21.199999999999996
Experiment: 65 parameter : [0.5, 5, 1, 5] objective value : 21.199999999999996
Experiment: 66 parameter : [0.5, 5, 1, 7] objective value : 21.199999999999996
Experiment: 67 parameter : [0.5, 5, 2, 3] objective value : 21.199999999999996
Experiment: 68 parameter : [0.5, 5, 2, 5] objective value : 22.499999999999996
Experiment: 69 parameter : [0.5, 5, 2, 7] objective value : 22.499999999999996
Experiment: 70 parameter : [0.5, 5, 3, 3] objective value : 22.499999999999996
Experiment: 71 parameter : [0.5, 5, 3, 5] objective value : 22.499999999999996
Experiment: 72 parameter : [0.5, 5, 3, 7] objective value : 22.499999999999996
Experiment: 73 parameter : [0.5, 10, 1, 3] objective value : 21.199999999999996
Experiment: 74 parameter : [0.5, 10, 1, 5] objective value : 21.199999999999996
Experiment: 75 parameter : [0.5, 10, 1, 7] objective value : 21.199999999999996
Experiment: 76 parameter : [0.5, 10, 2, 3] objective value : 21.9
Experiment: 77 parameter : [0.5, 10, 2, 5] objective value : 21.9
Experiment: 78 parameter : [0.5, 10, 2, 7] objective value : 21.9
Experiment: 79 parameter : [0.5, 10, 3, 3] objective value : 21.9
Experiment: 80 parameter : [0.5, 10, 3, 5] objective value : 21.9
Experiment: 81 parameter : [0.5, 10, 3, 7] objective value : 21.9
```

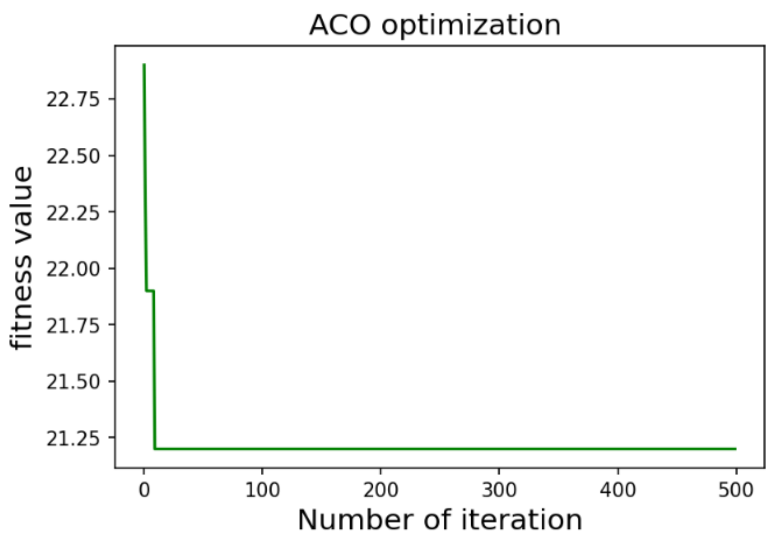
Run algorithm

[12]

```python
iter = 500
evaporate_rate = 0.1 #rho
Q = 5
alpha = 1 #pheromone_factor
beta = 7 #visibility_factor
pop_size = 100
num_city = len(distance)

#initialize
pheromone_map = np.ones((num_city,num_city)) #tau
visibility = np.zeros((num_city,num_city)) #eta
solutions = np.zeros((pop_size,num_city),dtype=int)
fitness_value = np.zeros(pop_size)
obj_value = np.zeros(pop_size)

ACO(iter,pop_size,num_city,evaporate_rate,Q,alpha,beta)
```

```
iteration is : 0 Best solution [ 6  4  5  0  9 13 11  3 15  8  2 14  1 12 10  7] Best fitness 25.299999999999997
iteration is : 1 Best solution [ 0  5  6  4  7 10 12  1  9 13 14  2  8 15  3 11] Best fitness 22.999999999999996
iteration is : 2 Best solution [ 7 10  6  1 12  0  5  4  9 13 14  2  8 15  3 11] Best fitness 21.9
iteration is : 3 Best solution [ 7 10  6  4  5  0 12  1  9 13 14  2  8 15  3 11] Best fitness 21.199999999999996
iteration is : 4 Best solution [12  1  6  7 10  5  0  9 13 14  2  8 15  4  3 11] Best fitness 21.199999999999996
iteration is : 5 Best solution [12  1  6  5  0  9 13 14  2  8 15 10  7  4  3 11] Best fitness 21.199999999999996
iteration is : 6 Best solution [ 8 15  5  0 12  1  6 10  7  9 13 14  2 11  3  4] Best fitness 21.199999999999996
iteration is : 7 Best solution [ 6  5  0 12  1  4  7 10  9 13 14  2  8 15  3 11] Best fitness 21.199999999999996
iteration is : 8 Best solution [13  9  5  0  7 10  6  1 12  4 14  2  8 15 11  3] Best fitness 21.199999999999996
iteration is : 9 Best solution [12  1  6  5  0  9 13 14  2  8 15  3 11 10  7  4] Best fitness 21.199999999999996
iteration is : 10 Best solution [15  8  2 14 13  9  5  0  6 10  7  4  1 12 11  3] Best fitness 21.199999999999996
```

```
iteration is : 490 Best solution [10  5  0 12  1  6  4  9 13 14  2  8 15  3 11  7] Best fitness 21.199999999999996
iteration is : 491 Best solution [ 9 13 14  2  8 15  3 11 10  6  4  5  0 12  1  7] Best fitness 21.199999999999996
iteration is : 492 Best solution [11  3  8 15 13  9  5  0 12  1  6 10  7  4 14  2] Best fitness 21.199999999999996
iteration is : 493 Best solution [ 4  6  2  8 15  7 10  5  0 12  1  9 13 14 11  3] Best fitness 21.199999999999996
iteration is : 494 Best solution [13  9  5  0 12  1  6 10  7  4 14  2  8 15  3 11] Best fitness 21.199999999999996
iteration is : 495 Best solution [ 9 13 14  2  8 15  3 11  7 10  5  0 12  1  6  4] Best fitness 21.199999999999996
iteration is : 496 Best solution [ 5  0 11  3  8 15 13  9  6 10  7  4 12  1 14  2] Best fitness 21.199999999999996
iteration is : 497 Best solution [14  3 11  8 15 13  9  5  0 12  1  6 10  7  4  2] Best fitness 21.199999999999996
iteration is : 498 Best solution [13  9  5  0 12  1  6 10  7  4 14  2  8 15  3 11] Best fitness 21.199999999999996
iteration is : 499 Best solution [13  9  5  0 12  1  6 10  7  4 14  2  8 15  3 11] Best fitness 21.199999999999996
final solution : [13  9  5  0 12  1  6 10  7  4 14  2  8 15  3 11] final distance : 21.199999999999996
```



Run algorithm 100 times to varify the result

```
iter = 500
evaporate_rate = 0.1 #rho
Q = 5
alpha = 1 #pheromone_factor
beta = 7 #visibility_factor
pop_size = 100
num_city = len(distance)

ACO_result2=[]
for i in range(100):
    result = ACO2(iter,pop_size,num_city,evaporate_rate,Q,alpha,beta)
    ACO_result2.append(result)
    print('ACO round :',i+1,'final objective value :',result)

averge_value_2 = np.mean(ACO_result2)
std_2 = np.std(ACO_result2)
print('averge =',averge_value_2 ,'standard deviation',std_2)
```

```
ACO round : 1 final objective value : 21.199999999999996
ACO round : 2 final objective value : 21.199999999999996
ACO round : 3 final objective value : 21.199999999999996
ACO round : 4 final objective value : 21.199999999999996
ACO round : 5 final objective value : 21.199999999999996
ACO round : 6 final objective value : 21.199999999999996
ACO round : 7 final objective value : 21.199999999999996
ACO round : 8 final objective value : 21.199999999999996
ACO round : 9 final objective value : 21.199999999999996
ACO round : 10 final objective value : 21.199999999999996
ACO round : 11 final objective value : 21.199999999999996
ACO round : 12 final objective value : 21.199999999999996
ACO round : 13 final objective value : 21.199999999999996
ACO round : 14 final objective value : 21.199999999999996
ACO round : 15 final objective value : 21.199999999999996
ACO round : 16 final objective value : 21.199999999999996
ACO round : 17 final objective value : 21.199999999999996
ACO round : 18 final objective value : 21.199999999999996
ACO round : 19 final objective value : 21.199999999999996
ACO round : 20 final objective value : 21.199999999999996
ACO round : 21 final objective value : 21.199999999999996
ACO round : 22 final objective value : 21.199999999999996
ACO round : 23 final objective value : 21.199999999999996
ACO round : 24 final objective value : 21.199999999999996
ACO round : 25 final objective value : 21.199999999999996
ACO round : 26 final objective value : 21.199999999999996
ACO round : 27 final objective value : 21.199999999999996
ACO round : 28 final objective value : 21.199999999999996
ACO round : 29 final objective value : 21.199999999999996
ACO round : 30 final objective value : 21.199999999999996
ACO round : 31 final objective value : 21.199999999999996
ACO round : 32 final objective value : 21.199999999999996
ACO round : 33 final objective value : 21.199999999999996
ACO round : 34 final objective value : 21.199999999999996
ACO round : 35 final objective value : 21.199999999999996
ACO round : 36 final objective value : 21.199999999999996
ACO round : 37 final objective value : 21.199999999999996
ACO round : 38 final objective value : 21.199999999999996
ACO round : 39 final objective value : 21.199999999999996
ACO round : 40 final objective value : 21.199999999999996
ACO round : 41 final objective value : 21.199999999999996
ACO round : 42 final objective value : 21.199999999999996
ACO round : 43 final objective value : 21.199999999999996
ACO round : 44 final objective value : 21.199999999999996
ACO round : 45 final objective value : 21.199999999999996
ACO round : 46 final objective value : 21.199999999999996
ACO round : 47 final objective value : 21.199999999999996
ACO round : 48 final objective value : 21.199999999999996
ACO round : 49 final objective value : 21.199999999999996
ACO round : 50 final objective value : 21.199999999999996
ACO round : 51 final objective value : 21.199999999999996
ACO round : 52 final objective value : 21.199999999999996
ACO round : 53 final objective value : 21.199999999999996
ACO round : 54 final objective value : 21.199999999999996
ACO round : 55 final objective value : 21.199999999999996
ACO round : 56 final objective value : 21.199999999999996
ACO round : 57 final objective value : 21.199999999999996
ACO round : 58 final objective value : 21.199999999999996
ACO round : 59 final objective value : 21.199999999999996
ACO round : 60 final objective value : 21.199999999999996
ACO round : 61 final objective value : 21.199999999999996
ACO round : 62 final objective value : 21.199999999999996
ACO round : 63 final objective value : 21.199999999999996
ACO round : 64 final objective value : 21.199999999999996
ACO round : 65 final objective value : 21.199999999999996
ACO round : 66 final objective value : 21.199999999999996
ACO round : 67 final objective value : 21.199999999999996
ACO round : 68 final objective value : 21.199999999999996
ACO round : 69 final objective value : 21.199999999999996
ACO round : 70 final objective value : 21.199999999999996
ACO round : 71 final objective value : 21.199999999999996
ACO round : 72 final objective value : 21.199999999999996
ACO round : 73 final objective value : 21.199999999999996
ACO round : 74 final objective value : 21.199999999999996
ACO round : 75 final objective value : 21.199999999999996
ACO round : 76 final objective value : 21.199999999999996
ACO round : 77 final objective value : 21.199999999999996
ACO round : 78 final objective value : 21.199999999999996
ACO round : 79 final objective value : 21.199999999999996
ACO round : 80 final objective value : 21.199999999999996
ACO round : 81 final objective value : 21.199999999999996
ACO round : 82 final objective value : 21.199999999999996
ACO round : 83 final objective value : 21.199999999999996
ACO round : 84 final objective value : 21.199999999999996
ACO round : 85 final objective value : 21.199999999999996
ACO round : 86 final objective value : 21.199999999999996
ACO round : 87 final objective value : 21.199999999999996
ACO round : 88 final objective value : 21.199999999999996
ACO round : 89 final objective value : 21.199999999999996
ACO round : 90 final objective value : 21.199999999999996
ACO round : 91 final objective value : 21.199999999999996
ACO round : 92 final objective value : 21.199999999999996
ACO round : 93 final objective value : 21.199999999999996
ACO round : 94 final objective value : 21.199999999999996
ACO round : 95 final objective value : 21.199999999999996
ACO round : 96 final objective value : 21.199999999999996
ACO round : 97 final objective value : 21.199999999999996
ACO round : 98 final objective value : 21.199999999999996
ACO round : 99 final objective value : 21.199999999999996
ACO round : 100 final objective value : 21.199999999999996
averge = 21.199999999999985 standard deviation 1.0658141036401503e-14
```

GA code：

In [1]:
```python
import math
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
```

In [2]:
```python
#distances matrix
cities = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
distances = [[0.0, 1.9, 5.9, 6.4, 2.2, 0.8, 0.8, 4.3, 4.9, 1.9, 3.2, 6.0, 1.3, 2.6, 3.9, 3.9],
             [1.9, 0.0, 5.0, 5.5, 2.9, 1.4, 1.4, 5.6, 4.0, 2.2, 3.9, 5.2, 1.2, 2.2, 3.2, 3.8],
             [5.9, 5.0, 0.0, 3.6, 7.3, 5.4, 5.4, 3.9, 1.8, 4.0, 4.0, 3.0, 5.3, 3.5, 2.1, 2.7],
             [6.4, 5.5, 3.6, 0.0, 7.3, 6.0, 6.0, 4.5, 2.2, 5.3, 4.5, 0.5, 6.0, 4.1, 3.3, 2.2],
             [2.2, 2.9, 7.3, 7.3, 0.0, 1.5, 1.5, 3.3, 6.0, 3.1, 3.8, 7.5, 2.8, 3.5, 5.1, 5.5],
             [0.8, 1.4, 5.4, 6.0, 1.5, 0.0, 1.0, 2.5, 4.1, 1.6, 1.0, 5.9, 1.6, 2.0, 3.3, 3.6],
             [1.3, 1.3, 7.0, 7.7, 1.5, 1.0, 0.0, 3.0, 5.0, 2.2, 1.0, 6.5, 1.5, 2.6, 4.1, 4.3],
             [4.3, 5.6, 3.9, 4.5, 3.3, 2.5, 3.0, 0.0, 3.9, 3.2, 1.4, 4.8, 5.5, 2.9, 3.5, 3.3],
             [4.9, 4.0, 1.8, 2.2, 6.0, 4.1, 5.0, 3.9, 0.0, 3.1, 3.0, 1.7, 4.6, 2.6, 2.1, 0.9],
             [1.9, 2.2, 4.0, 5.3, 3.1, 1.6, 2.2, 3.2, 3.1, 0.0, 2.3, 4.8, 2.2, 1.1, 2.2, 2.3],
             [3.2, 3.9, 4.0, 4.5, 3.8, 1.0, 1.0, 1.4, 3.0, 2.3, 0.0, 4.0, 1.3, 2.0, 2.9, 2.3],
             [6.0, 5.2, 3.0, 0.5, 7.5, 5.9, 6.5, 4.8, 1.7, 4.8, 4.0, 0.0, 6.3, 3.9, 3.0, 2.2],
             [1.3, 1.2, 5.3, 6.0, 2.8, 1.6, 1.5, 5.5, 4.6, 2.2, 1.3, 6.3, 0.0, 2.5, 3.5, 4.0],
             [2.6, 2.2, 3.5, 4.1, 3.5, 2.0, 2.6, 2.9, 2.6, 1.1, 2.0, 3.9, 2.5, 0.0, 1.7, 1.9],
             [3.9, 3.2, 2.1, 3.3, 5.1, 3.3, 4.1, 3.5, 2.1, 2.2, 2.9, 3.0, 3.5, 1.7, 0.0, 2.2],
             [3.9, 3.8, 2.7, 2.2, 5.5, 3.6,  4.3, 3.3, 0.9, 2.3, 2.3, 2.2, 4.0, 1.9, 2.2, 0.0]]
```

In [3]:
```python
# TSP path Length 計算路線的長度#
def calTourMileage(tourGiven, nCities, distMat):
    q = []
    for j in range(len(tourGiven)):
        #mileageTour = distMat[tourGiven[j][nCities-1]][tourGiven[j][0]]
        mileageTour = 0
        for i in range(nCities-1):
            mileageTour = mileageTour + distMat[tourGiven[j][i]][tourGiven[j][i+1]]    #calculat total path Leagth
        q.append(mileageTour)
    return q
```

In [4]:
```python
#錦標賽法 找到 ini parents
def tournament(population,sol_num):
    final = []
    for j in range(sol_num):
        #random choise 20 parents in population
        parents = random.choices(population, k = sol_num)
        fit = calTourMileage(parents,10,distances)
        best_fitness = 1000*1000
        best1_x_y = []
        #replace parents if current fitness more better
        for i in range(len(parents)):
            if fit[i] < best_fitness :
                best_fitness = fit[i]
                best1_x_y = parents[i]
        final.append(best1_x_y)
    return final
#丟一個
def calTourMileage1(tourGiven, nCities, distMat):
    mileageTour = 0
    for i in range(nCities-1):
        mileageTour = mileageTour + distMat[tourGiven[i]][tourGiven[i+1]]    #calculat total path Leagth
    return mileageTour
```

In [5]:
```
#one mutation
def mutateSwap(tourGiven, nCities):
    i = np.random.randint(nCities)                          #random i and j
    while True:
        j = np.random.randint(nCities)                      # random value in 0 to 10
        if i!=j: break                                      # i not equal j

    tourSwap = tourGiven.copy()
    tourSwap[i],tourSwap[j] = tourGiven[j],tourGiven[i]  #change city i and j location

    return tourSwap
```

In [6]:
```
# crossover one crossover
def crossover(ind1, ind2, r_cross):
    if  random.uniform(0,1) < r_cross:
        size = len(cities)
        p1, p2 = [0] * size, [0] * size
        # Initialize the position of each indices in the individuals
        for k in range(size):
            p1[ind1[k]] = k
            p2[ind2[k]] = k
        # Choose crossover points
        cxpoint1 = random.randint(0, size)
        cxpoint2 = random.randint(0, size - 1)
        if cxpoint2 >= cxpoint1:
            cxpoint2 += 1
        else:  # Swap the two cx points
            cxpoint1, cxpoint2 = cxpoint2, cxpoint1
    # Apply crossover between cx points
        for k in range(cxpoint1, cxpoint2):
        # Keep track of the selected values
            temp1 = ind1[k]
            temp2 = ind2[k]
        # Swap the matched value
            ind1[k], ind1[p1[temp2]] = temp2, temp1
            ind2[k], ind2[p2[temp1]] = temp1, temp2
        # Position bookkeeping
            p1[temp1], p1[temp2] = p1[temp2], p1[temp1]
            p2[temp1], p2[temp2] = p2[temp2], p2[temp1]
    return ind1, ind2
```

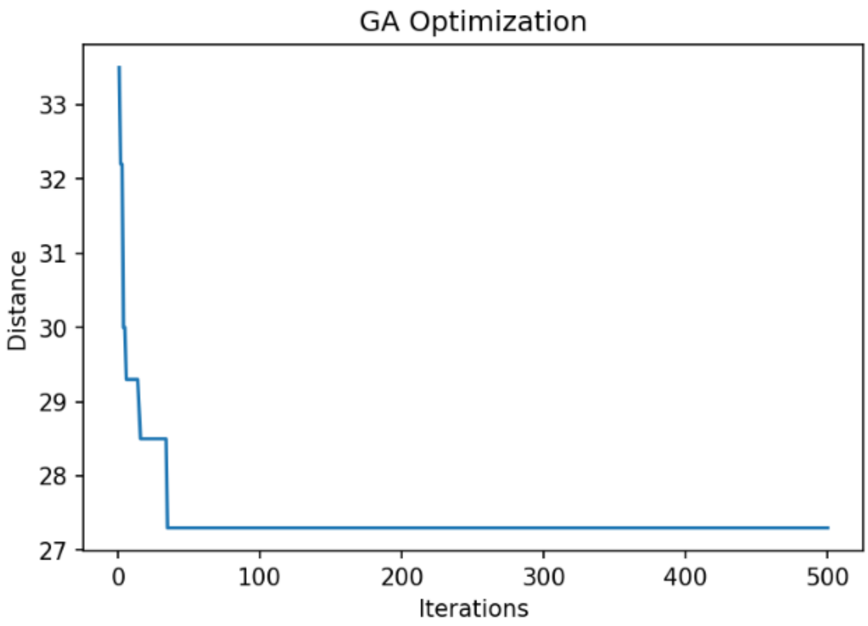
In [7]:
```
#Find the best solution and replace it in each interaction
def best(pop):
    bpath = []
    blength = 1000*1000
    for i in range(len(pop)):
        if calTourMileage1(pop[i], len(cities), distances) < blength:
            blength = calTourMileage1(pop[i], len(cities), distances)
            bpath = pop[i]
    return(bpath,blength)
```

In [8]:
```python
def main():
    start=time.time()
    population = []           # list that holds paths
    population_size = 10000   # max 120 combinations
    n_generations = 500
    m_prob = 0.9              #need to select
    r_cross = 0.1             #need to select
    bl = 100**100
    bp = []
    result_fitness = []
    result_index = []
    #creat 1000 popu (inipop)
    for i in range(population_size):
        population.append(random.sample(cities, len(cities)))

    #interaction 500 times
    for j in range(n_generations):
        # selection 20 parents use tournament
        parents = tournament(population, 80)

        # two by two to crossover then to be children
        co_children = []
        for k in range(0,len(parents),2):
            p1, p2 = parents[k], parents[k+1]
            co_children.append(crossover(p1, p2, r_cross))
        w = []
        for f in range(len(co_children)):
            for u in range(2):
                w.append(co_children[f][u])
        co_children = w

        # mutation by crossover result and replace parents
        co_children1 = co_children.copy()
        for p in range(len(co_children)):
            if random.uniform(0,1) < m_prob:
                co_children1[p] = mutateSwap(co_children[p],len(cities))
        parents = co_children1




        # choose each interaction best length and replace current best path and best length
        if best(parents)[1] <= bl:
            bl = best(parents)[1]
            bp = best(parents)[0]
        result_fitness.append(bl)
        result_index.append(j+1)
    end=time.time()
    t=end-start
    return bl,bp,result_fitness,result_index,t
```

In [13]:
```python
#result
rsult = main()
print("最優路徑",rsult[1])
print("最優路徑距離",rsult[0])
print("執行時間",rsult[4])

#plot the evolution history
from matplotlib import pyplot as plt
x = rsult[3]
y = rsult[2]
#plt.figure(figsize=(4,1))
plt.figure(dpi=150)
plt.plot(x,y)
plt.title("GA Optimization")
plt.xlabel("Iterations")
plt.ylabel("Distance")
```

```
最優路徑 [4, 6, 0, 5, 12, 1, 13, 9, 10, 7, 15, 2, 3, 11, 8, 14]
最優路徑距離 27.3
執行時間 5.227640867233276
```

Out[13]: `Text(0, 0.5, 'Distance')`



In [14]:
```python
ga_distance = []
gtime = []
ga_path = []
t = 100
start=time.time()
for p in range(t):
    result = main()
    ga_distance.append(result[0])
    ga_path.append(result[1])
    gtime.append(result[4])
end=time.time()
tot=end-start

s = 1000
q = []
for i in range(len(ga_distance)):
    if  ga_distance[i] < s:
        s = ga_distance[i]
        q = ga_path[i]
    else:
        s = s
        q = q
print("100 次實驗中平均距離 ",np.mean(ga_distance))
print("100 次實驗中平均執行時間 :", np.mean(gtime))
print("100 次實驗中變異數 :", np.var(ga_distance))
print("100 次實驗中總執行時間:", tot)
print("100 次實驗中最短距離 :",s)
print("100 次實驗中最短路徑 :",q)
#print("100 times distance : ", ga_distance)
#print("100 times interaction time :",gtime)
#print(ga_path)
```

```
100 次實驗中平均距離  28.526999999999997
100 次實驗中平均執行時間 : 5.309551360607148
100 次實驗中變異數 : 0.808771
100 次實驗中總執行時間: 531.0006713867188
100 次實驗中最短距離 : 25.399999999999995
100 次實驗中最短路徑 : [1, 12, 0, 5, 4, 6, 9, 13, 14, 2, 15, 11, 3, 8, 10, 7]
```