FINAL PROJECT FOR MODERN OPTIMIZATION METHODS: TRAVELER SALESMAN PROBLEM

SINGYUAN YEH

Abstract. This project introduces different method to optimize the traveler salesman problem (TSP). TSP is introduced in section 1. Section 2 presents how I collect the data, *i.e.* distance matrix. The method I used and how to modify these method to solve TSP are introduced in section 3. There are seven method, such as hill climbing, random walk, genetic algorithm (GA), simulated annealing (SA), tabu search (Tabu), particle swarm optimization (PSO) and ant colony optimization (ACO). Section 4 shows the results which are solved by different method. Section 5 is conclusion.

1. A Revision on Traveler Salesman Problem?

1.1. **What is TSP?.** The process of this traveler salesman is as follows: start by choosing any city in a given list, and next, find a tour that visits each of the cities exactly once and end up by returning to the departure one. Given the distances between cities, the traveled distance have to be minimized. Thus, we have to apply optimization method to decide the order of the cities, which the traveler pass by. In this report, the cities are replaced by the 7-Eleven convenient stores.

1.2. **What are the difficulties in TSP?.** Given n cities, there are $\frac{n!}{2n}$ permutations of order of cities. Hence, the time complicity of solve the global best solution is at least $O(n!)$ if calculating the length is linear time.

1.3. **What are the applications of TSP in real life?.** The TSP has many applications, for instance logistics, which is the management of the flow of things between the point of origin and the point of consumption to meet the requirements of customers or corporations. The TSP also can also be found in astronomy. Consider the situation that astronomers who observe many sources will hope to minimize the time spent moving the telescope between the sources.

2. Data Collection

The distance between two 7-Eleven was searched from "Google Map" and they are captured at 23:00 on Nev 30. The distance is the route path in the real world. I just capture the upper triangle of matrix. Thus, the following distance matrix is symmetric. Note that the distance unit is kilometer.

	中坡	中研	中貿	玉成	玉德	向揚	庄研	佳樂	忠陽	昆陽	林坊	研究	胡適	重陽	香城	凱松	港泰	港高	港捷	港勝	港運	港德	港興	港環	港麗	華技	雄強	慈愛	新福	經貿	聯成	聯坊	馥樺	鵬馳	耀港	鑫貿
中坡	0	5.5	4.7	2.2	0.35	2.4	5.6	3.4	2	1.7	0.65	4.5	4.9	2.8	3.9	1	3.3	3.1	4.4	4.3	0.85	1	3.4	3.1	3	6.2	1.2	3	0.5	4	1.8	0.65	4.4	2	4.1	4.7
中研	5.5	0	3	4.6	5.5	4.5	1.3	2.9	4.1	4.7	5.2	1	0.65	4.2	2.4	5.7	2.8	3.2	2.2	2.1	5.7	5.7	3.4	3.2	3	1.9	5.1	4	5.7	2.5	4.2	5.2	2.7	4.8	1.7	2.5
中貿	4.7	3	0	3.1	4.5	3	2.6	1.5	3	3.6	4.2	1.4	1.8	2.3	0.75	4.5	1.8	1.8	0.45	0.4	4.3	4.7	1.9	1.8	2	3.2	4.1	2	4.7	0.65	3.1	4.2	0.25	3.3	0.9	0.13
玉成	2.2	4.6	3.1	0	2.3	0.4	4.4	1.6	0.45	0.5	1.9	3.3	3.7	0.75	2.2	1.3	2.1	1.8	3	2.9	1.7	2.5	1.6	1.8	1.8	5	1	1	2.5	2.2	0.65	2	2.7	0.26	2.9	3.3
玉德	0.35	5.5	4.5	2.3	0	2.2	5.3	3.1	1.8	1.4	0.4	4.2	4.6	2.5	3.7	1	3	2.9	4.1	4	0.55	1.2	3.2	2.9	2.7	5.9	1.2	2.7	0.65	3.7	1.4	0.5	4.2	1.7	3.8	4.4
向揚	2.4	4.5	3	0.4	2.2	0	4.8	2.3	0.85	0.85	2.3	3.7	4.1	0.55	2.9	2	2.7	2.2	3.4	3.6	2.2	2.9	1.9	2.2	2.2	5.4	2.3	0.75	2.9	2.9	1	2.4	3	0.65	3.4	3.1
庄研	5.6	1.3	2.6	4.4	5.3	4.8	0	2.9	4	4.6	5.2	1.2	0.75	4.2	2.4	5.7	2.8	3.2	2.2	2.1	5.7	5.7	3.3	3.2	3	0.6	5.1	3.9	5.7	2.5	4.1	5.2	2.7	4.8	1.7	2.5
佳樂	3.4	2.9	1.5	1.6	3.1	2.3	2.9	0	1.7	2.3	2.8	1.8	2.2	1.3	0.6	2.9	0.45	0.2	1.3	1.2	2.9	3.3	0.4	0.21	0.6	3.5	2.7	1	3.3	0.65	1.8	2.9	1.1	1.9	1.3	1.5
忠陽	2	4.1	3	0.45	1.8	0.85	4	1.7	0	0.026	1.8	3	3.4	1.1	2.3	1.6	1.8	1.5	2.7	2.6	1.6	2.1	1.8	1.5	1.5	4.7	0.55	1.3	2.1	2.3	0.19	1	2.8	0.75	2.6	3.2
昆陽	1.7	4.7	3.6	0.5	1.4	0.85	4.6	2.3	0.026	0	1.2	2.8	3.2	1.1	2.2	1.6	1.6	1.5	2.7	2.6	1.4	1.7	1.8	1.4	1.3	4.5	0.5	1.3	1.7	2.3	0.16	1	2.7	1	2.4	3
林坊	0.65	5.2	4.2	1.9	0.4	2.3	5.2	2.8	1.8	1.2	0	3.9	4.3	2.2	3.3	1.3	2.7	2.5	3.8	3.7	0.8	1.2	2.9	2.5	2.4	5.6	0.55	2.4	0.6	3.4	1.4	0.16	3.8	1.6	3.5	4.1
研究	4.5	1	1.4	3.3	4.2	3.7	1.2	1.8	3	2.8	3.9	0	0.4	3.6	1.8	5.1	2.2	2.6	1.6	1.5	5.1	5.1	2.7	2.6	2.4	1.7	4.5	3.3	5.1	1.9	3.5	4.6	2.1	4.2	0.55	1.9
胡適	4.9	0.65	1.8	3.7	4.6	4.1	0.75	2.2	3.4	3.2	4.3	0.4	0	3.7	1.9	5.2	2.3	2.7	1.7	1.6	5.2	5.2	2.9	2.7	2.5	1.3	4.6	3.5	5.2	2.1	3.7	4.7	2.2	4.3	1.2	2
重陽	2.8	4.2	2.3	0.75	2.5	0.55	4.2	1.3	1.1	1.1	2.2	3.6	3.7	0	1.8	2	1.7	1.3	2.5	2.6	2.3	2.7	0.9	1.3	1.2	4.8	2.1	0.23	2.7	2.1	1.2	2.2	2	1	2.6	2.1
香城	3.9	2.4	0.75	2.2	3.7	2.9	2.4	0.6	2.3	2.2	3.3	1.8	1.9	1.8	0	4	1.3	1	0.5	0.45	3.8	4.2	1.1	1	1.5	2.9	3.6	2	4.2	0.4	2.7	3.8	0.55	3.2	0.65	0.8
凱松	1	5.7	4.5	1.3	1	2	5.7	2.9	1.6	1.6	1.3	5.1	5.2	2	4	0	3.3	2.9	4.1	4	0.35	2.1	2.9	2.9	2.9	6.2	1.1	2.3	1.6	5.1	1.8	1.4	4.9	1	4.1	5.4
港泰	3.3	2.8	1.8	2.1	3	2.7	2.8	0.45	1.8	1.6	2.7	2.2	2.3	1.7	1.3	3.3	0	0.35	1.6	1.5	2.9	2.9	0.65	0.35	0.16	3.4	2.3	1.8	2.9	1.5	1.3	2.4	1.7	2	1.4	1.9
港高	3.1	3.2	1.8	1.8	2.9	2.2	3.2	0.2	1.5	1.5	2.5	2.6	2.7	1.3	1	2.9	0.35	0	1.2	1.1	3.2	3.2	0.45	0.014	0.4	3.3	2.6	1.5	3.2	1.2	1.7	2.8	1.3	2	1.4	1.5
港捷	4.4	2.2	0.45	3	4.1	3.4	2.2	1.3	2.7	2.7	3.8	1.6	1.7	2.5	0.5	4.1	1.6	1.2	0	0.24	4.2	4.6	1.9	1.7	1.9	3.1	4	2.2	4.6	0.45	3.1	4.1	0.6	3.4	0.5	0.35
港勝	4.3	2.1	0.4	2.9	4	3.6	2.1	1.2	2.6	2.6	3.7	1.5	1.6	2.6	0.45	4	1.5	1.1	0.24	0	3.6	4.1	1.2	1.2	1.4	2.7	3.5	1.8	4.1	0.3	2.5	3.6	0.45	2.7	0.6	0.35
港運	0.85	5.7	4.3	1.7	0.55	2.2	5.7	2.9	1.6	1.4	0.8	5.1	5.2	2.3	3.8	0.35	2.9	3.2	4.2	3.6	0	1.8	3	2.7	2.7	6	1.4	2.5	1.3	3.8	1.5	1.8	4	1.3	3.8	5.7
港德	1	5.7	4.7	2.5	1.2	2.9	5.7	3.3	2.1	1.7	1.2	5.1	5.2	2.7	4.2	2.1	2.9	3.2	4.6	4.1	1.8	0	3.3	3	2.8	6.1	1.8	2.9	0.55	4.2	1.5	1.2	4.3	2.7	4	4.6
港興	3.4	3.4	1.9	1.6	3.2	1.9	3.3	0.4	1.8	1.8	2.9	2.7	2.9	0.9	1.1	2.9	0.65	0.45	1.9	1.2	3	3.3	0	0.45	0.7	3.8	2.8	1.2	3.4	1.2	1.8	2.9	1.4	1.8	1.7	1.5
港環	3.1	3.2	1.8	1.8	2.9	2.2	3.2	0.21	1.5	1.4	2.5	2.6	2.7	1.3	1	2.9	0.35	0.014	1.7	1.2	2.7	3	0.45	0	0.4	3.3	2.6	1.5	3.2	1.2	1.7	2.8	1.3	2	1.5	1.6
港麗	3	3	2	1.8	2.7	2.2	3	0.6	1.5	1.3	2.4	2.4	2.5	1.2	1.5	2.9	0.16	0.4	1.9	1.4	2.7	2.8	0.7	0.4	0	3.7	2.1	1	2.7	1.8	1.2	2.2	1.9	1.8	1.6	2.2
華技	6.2	1.9	3.2	5	5.9	5.4	0.6	3.5	4.7	4.5	5.6	1.7	1.3	4.8	2.9	6.2	3.4	3.3	3.1	2.7	6	6.1	3.8	3.3	3.7	0	5.7	4.6	6.3	3.2	4.8	5.8	3.3	5.4	2.3	3.1
雄強	1.2	5.1	4.1	1	1.2	2.3	5.1	2.7	0.55	0.5	0.55	4.5	4.6	2.1	3.6	1.1	2.3	2.6	4	3.5	1.4	1.8	2.8	2.6	2.1	5.7	0	1.8	1.6	3.1	1	0.4	3.3	0.7	2.9	3.5
慈愛	3	4	2	1	2.7	0.75	3.9	1	1.3	1.3	2.4	3.3	3.5	0.23	2	2.3	1.8	1.5	2.2	1.8	2.5	2.9	1.2	1.5	1	4.6	1.8	0	2.9	1.9	1.4	2.5	1.7	1.2	2.3	1.8
新福	0.5	5.7	4.7	2.5	0.65	2.9	5.7	3.3	2.1	1.7	0.6	5.1	5.2	2.7	4.2	1.6	2.9	3.2	4.6	4.1	1.3	0.55	3.4	3.2	2.7	6.3	1.6	2.9	0	4.6	1.9	0.9	4.7	2.1	4.4	5
經貿	4	2.5	0.65	2.2	3.7	2.9	2.5	0.65	2.3	2.3	3.4	1.9	2.1	2.1	0.4	5.1	1.5	1.2	0.45	0.3	3.8	4.2	1.2	1.2	1.8	3.2	3.1	1.9	4.6	0	2.7	3.7	0.13	2.8	1.3	0.4
聯成	1.8	4.2	3.1	0.65	1.4	1	4.1	1.8	0.19	0.16	1.4	3.5	3.7	1.2	2.7	1.8	1.3	1.7	3.1	2.5	1.5	1.5	1.8	1.7	1.2	4.8	1	1.4	1.9	2.7	0	1.1	2.8	1.1	2.5	3.1
聯坊	0.65	5.2	4.2	2	0.5	2.4	5.2	2.9	1	1	0.16	4.6	4.7	2.2	3.8	1.4	2.4	2.8	4.1	3.6	1.8	1.2	2.9	2.8	2.2	5.8	0.4	2.5	0.9	3.7	1.1	0	3.7	1.1	3.4	4
馥樺	4.4	2.7	0.25	2.7	4.2	3	2.7	1.1	2.8	2.7	3.8	2.1	2.2	2	0.55	4.9	1.7	1.3	0.6	0.45	4	4.3	1.4	1.3	1.9	3.3	3.3	1.7	4.7	0.13	2.8	3.7	0	2.7	1.2	0.4
鵬馳	2	4.8	3.3	0.26	1.7	0.65	4.8	1.9	0.75	1	1.6	4.2	4.3	1	3.2	1	2	2	3.4	2.7	1.3	2.7	1.8	2	1.8	5.4	0.7	1.2	2.1	2.8	1.1	1.1	2.7	0	3.2	3.6
耀港	4.1	1.7	0.9	2.9	3.8	3.4	1.7	1.3	2.6	2.4	3.5	0.55	1.2	2.6	0.65	4.1	1.4	1.4	0.5	0.6	3.8	4	1.7	1.5	1.6	2.3	2.9	2.3	4.4	1.3	2.5	3.4	1.2	3.2	0	0.8
鑫貿	4.7	2.5	0.13	3.3	4.4	3.1	2.5	1.5	3.2	3	4.1	1.9	2	2.1	0.8	5.4	1.9	1.5	0.35	0.35	5.7	4.6	1.5	1.6	2.2	3.1	3.5	1.8	5	0.4	3.1	4	0.4	3.6	0.8	0

FIGURE 1 – The distance (km) matrix of two 7-Eleven.

The following figure shows the position of each 7-Eleven. The indices near the nodes represent the order in the above table. For instance, "0" stands for "中坡".

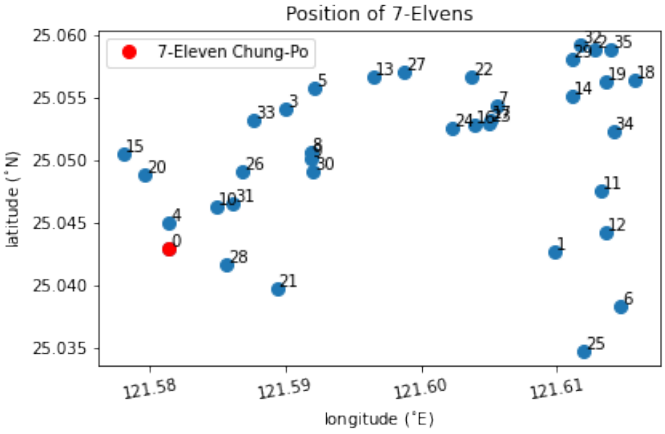

FIGURE 2 – The position of 36 7-Eleven, which is labels by the order of above table.

3. An Implementation of Known Metaheuristic Methods

Before this section, I have to introduce some fundamental information:

- **Particle:** Here, the particle is defined as a path, which passes through 36 7-Eleven. Hence, it is a vector with length 36, whose entries belong to $\{0, \cdots, 35\}$, which represents 36 7-Eleven. Thus, the entries of vector are distinct.
- **Route Length:** The function to calculate the route length, which the route visits through all 36 7-Eleven and go back to the first 7-Eleven.

```
1 def routeLength(tsp, solution):
```

```
    # tsp is distance matrix
    # solution is one path go through all seven
    pathLength = 0
    for i in range(len(solution)):
        # summation from solution[-1] means sum over a cycle
        pathLength += tsp[solution[i - 1]][solution[i]]
    return pathLength
```

CODE 1 – Calculation of route length.

- **Goal:** Minimize the length of the path through all 36 7-Eleven.
- **How to move to next step:** This is big questions to be solved in this project. I will state it carefully and precisely in the following paragraph.

3.1. **Hill climbing algorithm.** Please refer to the code for more details of my algorithm. Here, I specifically introduce how to move to next path. As following function, I will randomly choose two nodes to swap, which stands for moving to next step.

```
def SwapMove(solution, i ,j):
    # change the "index i, j" th of solution, instead of city-i and -j
    solution_test = solution.copy()
    solution_test[j] = solution[i]
    solution_test[i] = solution[j]
    return solution_test
```

CODE 2 – Move to next step.

Then, as usual hilling climbing algorithm, consider the route length of next step. If one less than current route length, accept this move. Otherwise, find new move.

3.2. **Random walk algorithm.** Please refer to the code for more details of my algorithm. Here, the method of how to move to next step is same as in CODE 2. Then, as usual random walk algorithm, consider the route length of next step. If one less than current route length, accept this move and record this length. Otherwise, still accept this move but not record.

3.3. **Genetic algorithm (GA).** Genetic algorithm is contained three main parts: "Selection", "Crossover" and "Mutation". Please refer to the code for the rest parts. Now, we introduce three parts as follows:

(1) **Selection:** The method I apply is as usual "roulette wheel selection" method. However, I convert the route length to fitness value by $\frac{1}{1+\texttt{route length}}$ such that the shorter the route length is, the higher probability it will be chosen.

```
# pop is population, fitness is fitness value and numof_matepool is how
    many parents you hope
def selection(pop, pathlen, numof_matepool):
    matepool = np.zeros((numof_matepool, pop.shape[1]), dtype=int)
    vecindex = np.array(range(len(pathlen))) # for choice which population

    # convert the route length to fitness value by 1/(1+len)
    fitness = 1/(1+pathlen)
```

```
    sum_fitness = np.sum(fitness)
    fitness = fitness/sum_fitness
    # construct the roulette wheel by cumulation
    roulette_wheel = fitness.cumsum()

    # choose numof_matepool parents from population
    for j in range(numof_matepool):
        # construct a dart
        dart = random.random()
        # shoot the dart
        choice_index = vecindex[np.where(roulette_wheel>dart)[0]][0] #
    choose the choice_index-th population
        matepool[j, :] = pop[choice_index, :]

    return matepool
```

Code 3 – Selection via roulette wheel method.

(2) **Crossover:** Here, I modify the "uniform crossover" method. From parent 1 and 2, I uniformly choose some gene in parent 1 and re-order these gene by parent 2. That is, record the order of these gene in parent 2 and re-order these gene in parent 1, which generates child 1. By the similar work, do the same thing to parent 2, which generates child 2.

```
# matepool is all parents, numof_offsprings is how many children you hope
    and crossover_rate is cross over rate
def crossover(matepool, numof_offsprings, crossover_rate):
    offsprings = np.zeros((numof_offsprings, matepool.shape[1]), dtype=int
    )
    j = 0

    # construct all offsprings
    while j < numof_offsprings:
        # avoid over the matepool
        parents1 = matepool[j%matepool.shape[0], :]
        parents2 = matepool[(j+1)%matepool.shape[0], :]
        # determine which genetic to cross over
        idxfilter = (np.random.rand(matepool.shape[1]) < crossover_rate)

        # two parents cross over and generates two kids
        # step 1: find these city in parent-1
        ordercity1 = parents1[idxfilter]
        ordercity2 = parents2[idxfilter]

        # step 2: find such city's  index of in parent-2
        crossidx2 = np.in1d(parents2, ordercity1).nonzero()[0]
        crossidx1 = np.in1d(parents1, ordercity2).nonzero()[0]
```

```
22
23          # step 3: reorder such city in parent-2 by parent-1
24          children1 = parents1.copy()
25          children2 = parents2.copy()
26          children2[crossidx2] = ordercity1
27          children1[crossidx1] = ordercity2
28
29
30          offsprings[j, :] = children1
31          offsprings[j+1, :] = children2
32          j = j+2
33      return offsprings
```

Code 4 – Crossover via uniform method

(3) **Mutation:** Here, I modify the "consecutive multi-bits" method. I randomly choose consecutive multi-bits, and re-order these bits by counterclockwise change.

```
1  # offsprings is new population, mutation_rate is mutation rate
2  # and numof_flip is determine how many gene you hope to mutation
3  def mutation(offsprings, mutation_rate, numof_flip):
4      mutants = offsprings.copy()
5
6      for j in range(mutants.shape[0]):
7          # mutation or not
8          if np.random.rand() < mutation_rate:
9              # random choose a ''first'' index to mutation
10             randidx = random.randint(0, offsprings.shape[1]-(1+numof_flip)
       )
11             # mutation index from randidx to (randidx + numof_flip)
12             # because of we hope to mutate consecutive multi-bit, avoid
       choose tail
13
14             # Counterclockwise mutation
15             # replace the first element by the last element (in this
       mutation region)
16             mutants[j, randidx] = offsprings[j, randidx+numof_flip-1]
17             for k in range(1, numof_flip):
18                 idx = (randidx+k)
19                 mutants[j, idx] =  offsprings[j, idx-1]
20     return mutants
```

Code 5 – Mutation via consecutive multi-bits method.

3.4. **Simulated annealing method (SA).** Please refer to the code for more details of my algorithm. Here, the method of how to move to next step is same as in Code 2. Then, as usual SA algorithm, I set ΔE by new route length minus current route length. Then, if $\Delta E < 0$,

accept new move. If $\Delta E > 0$, randomly accept the new move according to the probability $e^{-\frac{\Delta E}{K_B T}}$, where K_B is Boltzman constant and T is temperature, which as usual SA method.

3.5. **Tabu search method (Tabu).** The detail of my algorithm can refer to my code. Here, the method of how to move to next step is same as in Code 2. Thus, the "swap" is chosen as "*move*", *i.e.* there are $\binom{36}{2}$ of move in the candidate list (neighbor solutions).

Moreover, the "expectation improvement aspiration" is chosen as aspiration criteria so we will check if the route length of the new move is better than the current route length. This part is as usual tabu search algorithm.

3.6. **Particle swarm optimization method (PSO).** Please refer to the code for more details of my algorithm. Before this algorithm is introduced, the *velocity* of two particles should be defined first. Note that the position of a particle is a path. The difference of two positions are defined as permutation. Moreover, for any permutation can be decomposition as 2-cycle permutation. For instance, consider two particles $X_1 = [2, 0, 1]$ and $X_2 = [0, 1, 2]$. The difference of two particles is $X_2 - X_1 = (0, 2)(1, 2)$. Then, the velocity is defined as such 2-cycle permutations.

```
def find_move(best, x_cur):
    # v = best - x_cur
    x = x_cur.copy()
    numofcity = len(best)   # get length
    moveperm = []
    for i in range(numofcity):
        # check if the best[i] equal to x[i]
        idxinx = np.where(x == best[i])[0]

        # if not, swap
        if i != idxinx:
            move = [x[i], x[idxinx][0]]
            moveperm.append(move)
            x[idxinx], x[i] = x[i], x[idxinx]

    return moveperm
```

Code 6 – Compute the velocity.

Besides, the scalar multiplies *velocity* (permutations) should be defined. We only allow $c \cdot v$ where $c \in [0, 1]$ and v is velocity. For convenience, let $v = \tau_1 \cdots \tau_n$, where τ_i is 2-cycle permutation. Then, we can define $cv = \tau_1 \cdots \tau_k$ where $k = \lfloor c \cdot n \rfloor$ where $\lfloor \cdot \rfloor$ is floor function and n is number of 2-cycle permutation in v.

Finally, we have to define *position* (path) plus *velocity* (permutations). It just equal to permutations acts on such path, which is reorder the route path. Now, it's sufficient to deal with the main equation of PSO algorithm,

$$v_i \leftarrow w v_i + c_1 r_1 (p_i - X_i) + c_2 r_2 (p_g - X_i)$$
$$X_i \leftarrow X_i + v_i$$

where p_i is the best position of i-th particle and p_g is global best position. Note that according to mentioned above, we have to set $c_1 = c_2 = 1$ and $w, r_1, r_2 \in [0, 1]$.

```
def updatemove(w, cr1, cr2, moves_inertia, moves_loc, moves_global, maxv):
    # v = w*moves_inertia + cr1*moves_loc + cr2*moves_global

    # scalar times velocity
    wint = np.floor(w*len(moves_inertia)).astype(int)
    cr1int = np.floor(cr1*len(moves_loc)).astype(int)
    cr2int = np.floor(cr2*len(moves_global)).astype(int)
    # summation three part of velocity
    moves = moves_inertia[0:wint]
    moves.extend(moves_loc[0:cr1int])
    moves.extend(moves_global[0:cr2int])

    # avoid velocity too large
    if len(moves) > maxv:
        numofdel = len(moves)-maxv
        idx = np.random.permutation(len(moves))
        for i in range(numofdel):
            moves.pop()

    return moves
```

Code 7 – Update the velocity by three part.

Note that I set the parameter maximum velocity as length of permutations, which avoids that the position crossing over the best solution. The rest part is as usual PSO algorithm.

3.7. **Ant colony optimization method (ACO).** Please refer to the code for more details of my algorithm. Here, we introduce the main two ideas. The first idea is how to update pheromone. I define the amount of pheromone at (i, j)-path as

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}$$

where $\rho \in [0, 1]$ is evaporate rate and $\Delta\tau_{ij}$ is the amount of pheromone which is dropped by ants at (i, j)-path. Note that I define $\Delta\tau_{ij} = \frac{Q}{L_{ij}}$ where pheromone intensity Q is constant and L_{ij} is length between node-i and node-j if an ant pass through (i, j).

```
def update_pheromone(pher_tab, distmat, pathtable, rho, Q):
    # evaporate pheromone all the path
    pheromone_tab = pher_tab.copy()
    pheromone_tab = (1-rho)*pheromone_tab

    #Add pheromone to the path of the ants
    num_cities = np.shape(pathtable)[1]
    for solution in pathtable:
        # update pheromone on each path (city i, city j) by such ant
        for j in range(num_cities-1):
```

```
            city1 = solution[j]
            city2 = solution[j+1]
            pheromone_drop_amount = Q/distmat[city1, city2] # delta tau
            pheromone_tab[city1, city2] = pheromone_tab[city1, city2] +
    pheromone_drop_amount

        # cycle: deal with last city
        city2 = solution[0]
        pheromone_drop_amount = Q/distmat[city1, city2] # delta tau
        pheromone_tab[city1, city2] = pheromone_tab[city1, city2] +
    pheromone_drop_amount

    return pheromone_tab
```

Code 8 – Update the pheromone

The second idea is transition probability. The probability of an ant move from i-th node to j-th node is defined as

$$P_{ij} = \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{j\in N(i)}\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}$$

where $N(i)$ is neighborhood of i, τ_{ij} is the amount of pheromone, η_{ij} is desirability values. Here, we set $\eta_{ij} = 1/L_{ij}$ where is reciprocal of (i, j)-path length. The parameter α, β are constants, which control the influence of pheromone and the influence of length of route path, respectively.

```
def Construct_a_Path(pheromone_tab, distmat, alpha, beta):
    num_cities = np.shape(pheromone_tab)[1]
    # total number of city
    a_path = [-1]*num_cities
    # the candidate cities
    candidates = list(range(num_cities))

    # random choose city as first city
    city_cur = random.choice(candidates)
    a_path[0] = city_cur
    candidates.remove(city_cur)

    # select best from candiate
    for t in range(1, num_cities-1):
        # construct the transition probability list of next step
        transprob_list = []
        for city_next in candidates:
            transprob = pow(pheromone_tab[city_cur, city_next], alpha)*\
                pow(1/distmat[city_cur, city_next], beta)
            transprob_list.append(transprob)

        # select best as next step by roulette wheel
```

```
        idx = do_roulette_wheel_selection(transprob_list) # output index instead
    of city-id
        next_city_id = candidates[idx] # convert index to city-id
        candidates.remove(next_city_id)
        a_path[t] = next_city_id
        city_cur = next_city_id # update current city

    a_path[-1] = candidates.pop()

    return np.array(a_path)
```

Code 9 – Construct a path

The rest part is as usual ACO algorithm. Please refer to my code.

4. Method Comparison

In this section, I divide into two parts: the first part show the parameters I choose and the results of the algorithm; the second part show the comparison of seven method.

4.1. **Information of seven method.** In this subsection, I just show the "order" of visiting 36 7-eleven by illustration of route diagram. That is, in the following figures, the line between 2 nodes is only show the order, **instead of "real" path**. Note that I compute the length of route path by the "real" route path, *i.e.* by Google Map. Moreover, the indices labeled near the nodes stand for the order that the traveler visits.

4.1.1. *Hill climbing method.* I choose maximum number of steps as 400. There is no step size, because every updating step is a swap, which is mentioned above.

The following left figure shows the current length of route path. After 200 iteration, the best length of route path is 40.82 (km) and the order of 36 7-Eleven is as following box:

中坡–> 新福玉–> 玉德–> 昆陽–> 胡適–> 研究–> 港興–> 港環球–> 中研–> 庄研–> 華技–> 港德–> 雄強–> 凱松–> 港運–> 玉成–> 向揚–> 重陽–> 慈愛–> 佳樂–> 港高鐵–> 港麗–> 港泰–> 聯成–> 忠陽–> 鵬馳–> 馥樺–> 香城–> 港捷–> 經貿–> 港勝–> 鑫貿–> 中貿–> 耀港–> 林坊–> 聯坊–> 中坡

According above order, the illustration of route path is shown in the following right figure.

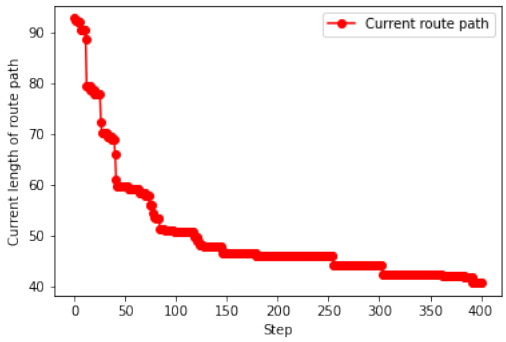

FIGURE 3 – Current length of route path of hill climbing method.

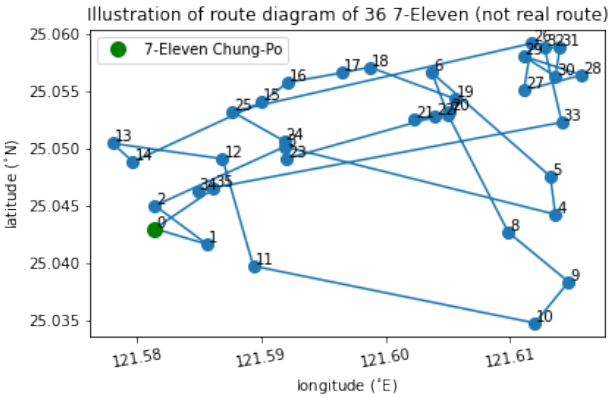

FIGURE 4 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.2. *Random walk method.* I choose maximum number of steps as 400. There is no step size, because every updating step is a swap, which is mentioned above.

The following left figure shows the current length of route path. After 200 iterations, the best length of route path is 70.01 (km) and the order of 36 7-Eleven is as following box:

中坡–> 經貿–> 港麗–> 港泰–> 港高鐵–> 胡適–> 港捷–> 忠陽–> 中研–> 研究–> 佳樂–> 庄研–> 新福玉–> 林坊–> 玉德–> 港德–> 鑫貿–> 港興–> 華技–> 港環球–> 耀港–> 馥樺–> 玉成–> 聯坊–> 香城–> 中貿–> 港勝–> 重陽–> 鵬馳–> 雄強–> 昆陽–> 凱松–> 港運–> 慈愛–> 向揚–> 聯成–> 中坡

According above order, the illustration of route path is shown in the following right figure.

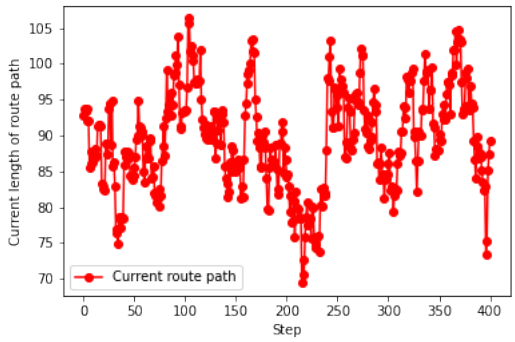

FIGURE 5 – Current length of route path of random walk mehotd.

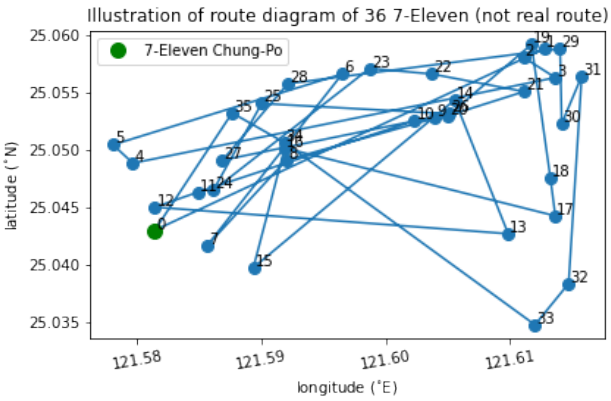

FIGURE 6 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.3. *Genetic algorithm method.* I choose maximum number of steps as 20 and the population size as 20. Moreover, The parameter `crossover rate` is chosen as 0.4, `mutation rate` is chosen as 0.3 and `number of bits flip` in mutation is chosen as 3.

The following left figure shows the mean of 20 length of current route path at each step. After 20 iterations, the best length of route path is 69.79 (km) and the order of 36 7-Eleven is as follows box:

中坡–> 新福玉–> 研究–> 港捷–> 港勝–> 香城–> 中研–> 港麗–> 港運–> 玉成–> 港泰–> 凱松–> 華技–> 港環球–> 林坊–> 玉德–> 聯坊–> 雄強–> 重陽–> 昆陽–> 忠陽–> 胡適–> 中貿–> 庄研–> 耀港–> 聯成–> 慈愛–> 鑫貿–> 港德–> 港興–> 港高鐵–> 佳樂–> 經貿–> 馥樺–> 向揚–> 鵬馳–> 中坡

According above order, the illustration of route path is shown in the following right figure.

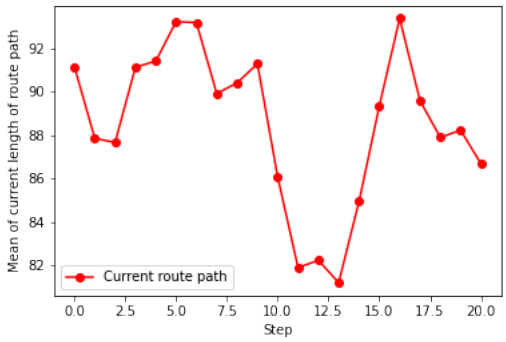

FIGURE 7 – Mean of current length of route path of genetic algorithm mehotd.

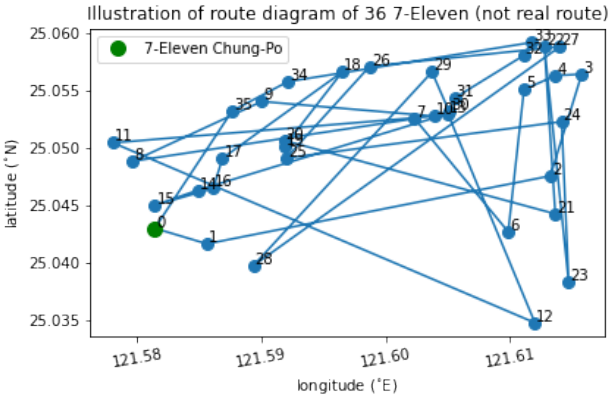

FIGURE 8 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.4. *Simulated annealing method.* I choose maximum number of steps as 400. The parameter `temperature reduction factor` is chosen as 0.8 and `Boltzmann constant` is chosen as 1. There is no step size, because every updating step is a swap, which is mentioned above.

The following left figure shows the current length of route path. After 200 iterations, the best length of route path is 41.23 (km) and the order of 36 7-Eleven is as following box:

中坡–> 聯坊–> 聯成–> 港泰–> 胡適–> 中研–> 中貿–> 慈愛–> 向揚–> 玉成–> 昆陽–> 港麗–> 港環球–> 香城–> 佳樂–> 港興–> 鵬馳–> 凱松–> 港運–> 忠陽–> 雄強–> 林坊–> 港德–> 重陽–> 耀港–> 華技–> 庄研–> 研究–> 經貿–> 馥樺–> 港勝–> 鑫貿–> 港捷–> 港高鐵–> 玉德–> 新福玉–> 中坡

According above order, the illustration of route path is shown in the following right figure.

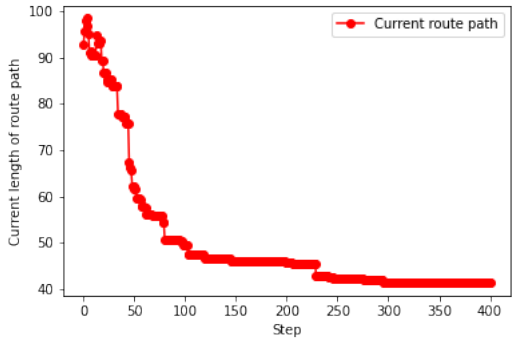

FIGURE 9 – Current length of route path of simulated annealing mehotd.

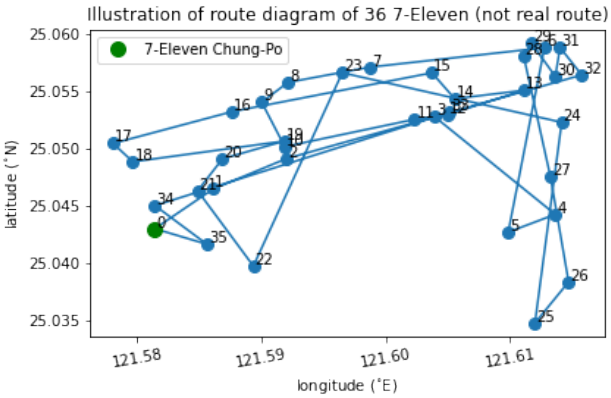

FIGURE 10 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.5. *Tabu search method.* I choose maximum number of steps is 400. The parameter `length of tabu list` is chosen as 50.

The following left figure shows the current length of route path. After 200 iterations, the best length of route path is 28.32 (km) and the order of 36 7-Eleven is as follows box:

中坡–> 新福玉–> 港德–> 重陽–> 港興–> 經貿–> 馥樺–> 中貿–> 鑫貿–> 港捷–> 港勝–> 香城–> 佳樂–> 港高鐵–> 港環球–> 港泰–> 港麗–> 聯成–> 昆陽–> 忠陽–> 玉成–> 向揚–> 鵬馳–> 雄強–> 聯坊–> 林坊–> 港運–> 凱松–> 慈愛–> 耀港–> 研究–> 中研–> 庄研–> 華技–> 胡適–> 玉德–> 中坡

According above order, the illustration of route path is shown in the following right figure.

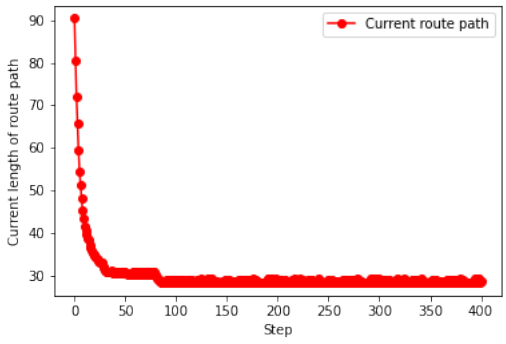

FIGURE 11 – Current length of route path of tabu search mehotd.

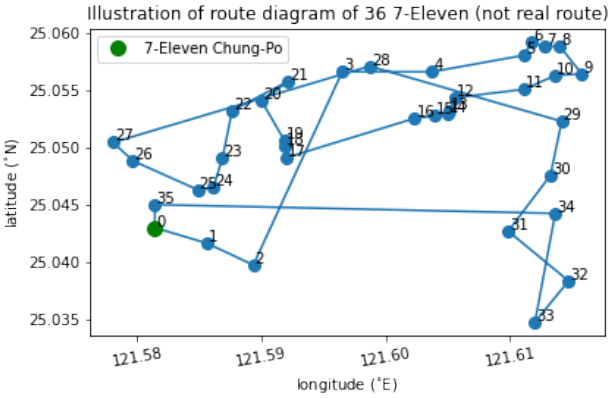

FIGURE 12 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.6. *Particle swarm optimization method.* I choose maximum number of step as 20 and the number of particles as 20. Moreover, The parameter `max velocity` is chosen as 15. The parameter `cognition factor`, c_1, and `social factor`, c_2, can only be 1.0.

The following left figure shows the mean of 20 length of current route path at each steps. After 20 iterations, the best length of route path is 68.65 (km) and the order of 36 7-Eleven is as follows box:

中坡–> 港德–> 中研–> 港勝–> 馥樺–> 港捷–> 香城–> 港高鐵–> 鑫貿–> 經貿–> 佳樂–> 華技–> 重陽–> 玉成–> 凱松–> 忠陽–> 胡適–> 耀港–> 庄研–> 雄強–> 港興–> 港麗–> 昆陽–> 玉德–> 林坊–> 鵬馳–> 新福玉–> 研究–> 中貿–> 向揚–> 聯成–> 慈愛–> 港運–> 港泰–> 港環球–> 聯坊–> 中坡

According above order, the illustration of route path is shown in the following right figure.

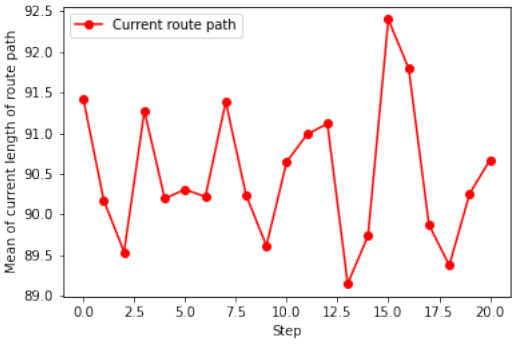

FIGURE 13 – Mean of current length of route path of particle swarm optimization mehotd.

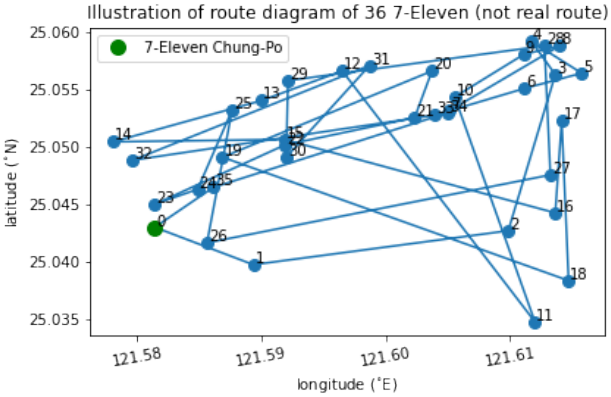

FIGURE 14 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.1.7. *Ant colony optimization method.* I choose maximum number of step as 20 and the number of ants as 20. Moreover, the parameter `pheromone factor`, α, is chosen as 1.0 and `Visibility facor`, β, is chosen as 2.0, the `evaporate rate`, ρ, is chosen as 0.8 and the `pheromone intensity`, Q, is chosen as 10.

The following left figure shows the mean of 20 length of current route path at each step. After 20 iterations, the best length of route path is 21.64 (km) and the order of 36 7-Eleven is as follows box:

中坡–> 新福玉–> 港德–> 港運–> 凱松–> 昆陽–> 忠陽–> 聯成–> 港麗–> 港泰–> 香城–> 經貿–> 馥樺–> 中貿–> 鑫貿–> 港勝–> 港捷–> 耀港–> 研究–> 胡適–> 中研–> 庄研–> 華技–> 港環球–> 港高鐵–> 佳樂–> 港興–> 重陽–> 慈愛–> 向揚–> 玉成–> 鵬馳–> 雄強–> 聯坊–> 林坊–> 玉德–> 中坡

According above order, the illustration of route path is shown in the following right figure.

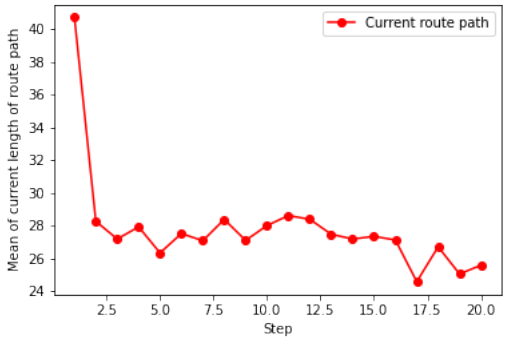

FIGURE 15 – Mean of current length of route path of ants colony optimization mehotd.

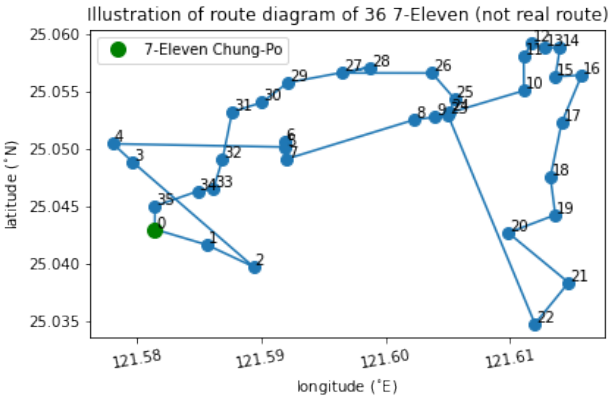

FIGURE 16 – Illustration of route path (instead of real route). Each node is a 7-Eleven.

4.2. **Comparison of 7 optimization method.** The following figure show the progress of seven optimization method.

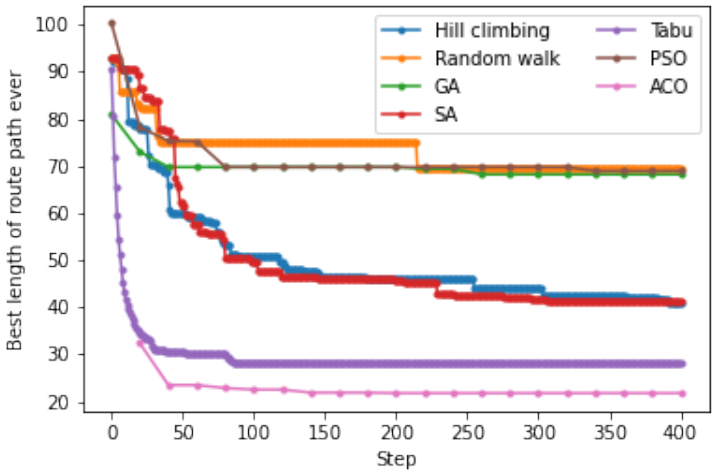

FIGURE 17 – Comparison of seven method.

Note that consider genetic algorithm (GA), particle swarm optimization (PSO) and ant colony optimization (ACO), there are 20 particles in one step. Hence, they plot every 20 steps. Besides, there is no initial gauss in ACO method, so the first step in ACO method is 1 instead of 0.

TABLE 1 – Title

Optimization method	HCO	RWO	GA	SA	Tabu	PSO	ACO
Best length	40.82	70.01	69.79	41.23	28.32	68.65	21.64

Above table shows the route length of best solution which is found by different methods. Note that HCO is hill climbing optimization method and RWO is random walk optimization method.

5. Conclusion

According to Figure 17, it found that GA, random walk and PSO is worst than other methods. Since there are too many randomly choice in GA method, *e.g.* randomly crossover, randomly mutation, the performance of GA method is similar to the performance of random walk method. Although they could avoid trapping at local optimization point, they are not efficient at searching the global optimization point.

Besides, the best solution of PSO and GA method is similar. It might be because the way in two algorithms are similar. That is, the way in GA to approach better solution is called "selection"; the way in PSO is called "social information". However, at the first few steps, GA is better than PSO, which might be because GA is applicable to discrete problems.

Next, we see that at the first few steps, random walk method is similar to SA method, but finally SA method is better than random walk method. This might be because during the cooling process, the move of solution can randomly search, which leads to skip the local optimization. However, it finally trap at local optimization, which is similar to hill climbing method.

Tabu search method will search the best solution in the neighborhood. It is systematic search, not like other methods. Hence, it takes much time to compute the list of objective value of the neighbor move. On the other hands, although it use tabu list to avoid getting a cycle, it seems that it still got a cycle in my reports. Refer to Figure 11. Hence, the solution is still trapped in local optimization point.

Finally, I thought the ACO method is the best method to TSP according to Figure 17. Since every ants in different cycle is independent. Thus, they can find different optimization even if it is local optimization. Moreover, the pheromone depends on length of path between two nodes, so it can find the short path directly. Besides, the pheromone can be evaporated, which leads to avoid the solution trapping at the local solution.

6. Reference

Please refer to my jupyter notebook.