CS5491 Artificial Intelligence Project 1	
WANG Yue	56359462

SUBQUESTION ONE : Basic TSP Problem

◆ Description:

The basic TSP problem is about a salesman who spends his time visiting n cities cyclically. In one tour, he must visit each city just once and should finish up where he started. Since each city is situated in different locations, the distance between every city will be different. The objective is to find the shortest round-trip route that visits each city once and then returns to the starting city. In this situation, TSPs are symmetric - which means The distance between the two cities is fixed. The distance from A to B is the same as the distance from B to A.

◆ Parameter Settings:

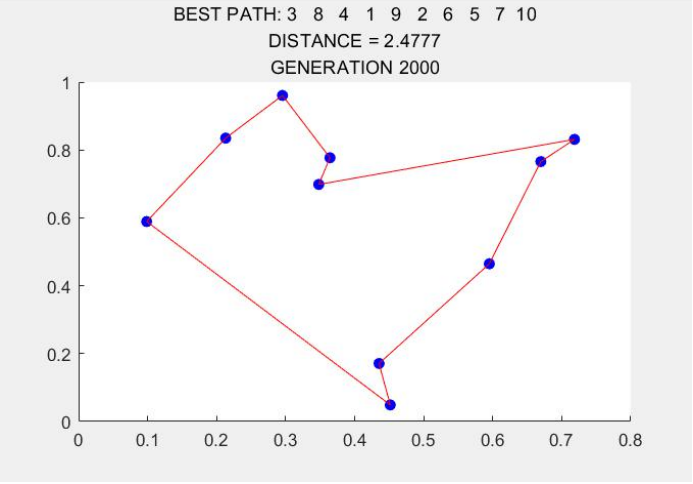
Parameter Description	Settings
Number of city	10
Initial population number	20
Crossover rate	0.8
Mutation rate 1 (Swap two random cities)	0.05
Mutation rate 2 (Exchange two parts of a path)	0.02
Generations / Iterations	2000
Fitness function	1 / (length of path)

◆ Some details about program:

[1]Selection: With the roulette wheel, the greater the fitness value the greater the area it occupies, the more likely it is to be selected and recorded. The smaller the value, the more likely it is to be eliminated. **[2]**Crossover: Gene recombination, the replacement of part of the structure of the parent individual. **[3]**Mutation: I did it in two different ways - the first is swap two random cities in a certain path, and second way is exchange two parts(fragments) of a path. The mutation rate of second way is much lower than the first one cause it can be seen as a combination of multiple single city mutations, and less likely to occur.

◆ Results figures and analysis:

Figure1.1 - The wire map of best path in 2000th generation



As shown in figure1.1, When the number of iterations reaches 2000, the best path is 3 -> 8 -> 4 -> 1 -> 9 -> 2 -> 6 -> 5 -> 7 -> 10 -> 3. And the shortest distance is 2.4777. The ten blue dots are the locations of cities, and the loop drawn in the red line is the best path.

Figure1.2 - Some properties of the chromosome population in the last iteration

	Cromosoma(Path)	Distance_path	fitness(xi)	Prob_Select	Nr_NumberOfSelect
1	3 8 4 1 9 2 6 5 7 10	2.4777	0.4036	0.0650	1.3007
2	3 8 4 1 9 2 6 5 7 10	2.4777	0.4036	0.0650	1.3007
3	3 8 4 1 9 5 6 2 10 7	3.0250	0.3306	0.0533	1.0654
4	3 8 4 1 9 5 6 2 10 7	3.0250	0.3306	0.0533	1.0654
5	3 8 5 9 6 2 1 4 7 10	3.5703	0.2801	0.0451	0.9027
6	8 3 5 9 6 2 1 4 7 10	3.7135	0.2693	0.0434	0.8679
7	2 8 3 7 6 4 9 5 1 10	4.4626	0.2241	0.0361	0.7222
8	3 8 2 1 9 4 6 5 7 10	3.2093	0.3116	0.0502	1.0042
9	1 8 4 3 9 2 6 5 7 10	2.8580	0.3499	0.0564	1.1277
10	3 1 4 8 9 10 6 5 7 2	4.1706	0.2398	0.0386	0.7728
11	3 9 2 1 8 4 6 5 7 10	3.0630	0.3265	0.0526	1.0522
12	3 8 4 1 7 2 6 5 9 10	3.6082	0.2771	0.0447	0.8932
13	3 8 5 1 9 2 6 4 10 7	3.8628	0.2589	0.0417	0.8343
14	3 8 4 1 9 5 6 2 10 7	3.0250	0.3306	0.0533	1.0654
15	3 9 2 1 8 4 6 5 7 10	3.0630	0.3265	0.0526	1.0522
16	3 8 2 1 7 4 6 5 9 10	4.5496	0.2198	0.0354	0.7084
17	3 8 4 1 9 2 6 5 7 10	2.4777	0.4036	0.0650	1.3007
18	3 8 5 1 9 2 6 4 7 10	3.6312	0.2754	0.0444	0.8876
19	1 8 7 3 9 2 6 5 4 10	4.1572	0.2405	0.0388	0.7753
20	3 8 4 1 9 2 6 5 7 10	2.4777	0.4036	0.0650	1.3007
	Cromosoma(Path)	Distance_path	fitness(xi)	Prob_Select	Nr_NumberOfSelect
21	SUM		6.2056	1	20
22	AVG		0.3103	0.0500	1
23	MAX		0.4036	0.0650	1.3007

In figure 1.2, The upper half is the property of each chromosome in the last iteration: the distance of path, fitness of path, the probability of being selected and number of certain path to be selected for a population size of N, respectively. We can see that the first path has the highest fitness(0.4036), so the best path is 3->8->4->1->9->2->6->5->7->10->3. It's logical. The bottom half is the sum, average and maximum of these parameters.

Figure1.3 - Convergence trajectory of The shortest path's distance

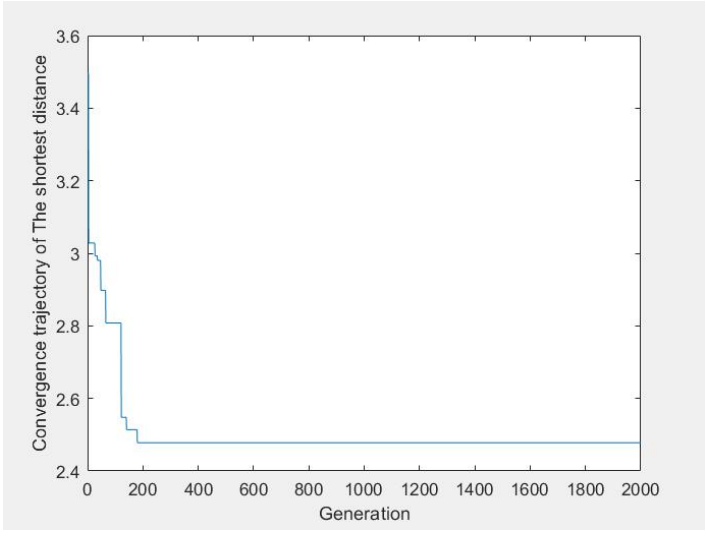


Figure 1.3 shows the convergence trajectory of The shortest distance. Its vertical axis is the distance of the shortest path in each round of iteration. We can see We can see that it stabilizes

around round 200, which means it has found the optimal solution with high probability.

◆ Experiment with the various parameters:

[1] Change crossover rate

This part keeps the other parameters unchanged and sets the crossover probability to 0, 0.3, 0.6, 0.9, respectively, and observe the changes in the experimental results:

Figure1.4 - When crossover rate = 0

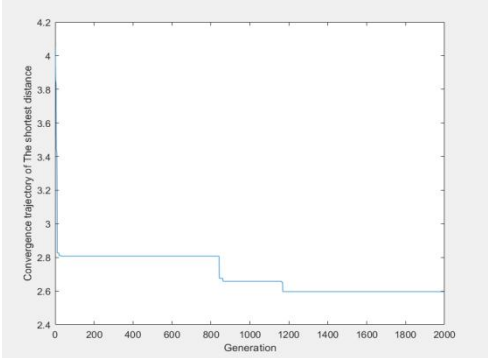


Figure1.5 - When crossover rate = 0.3

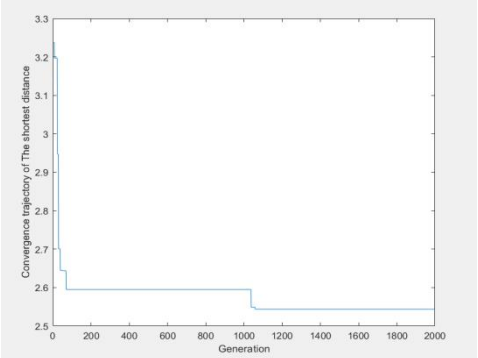


Figure1.6 - When crossover rate = 0.6

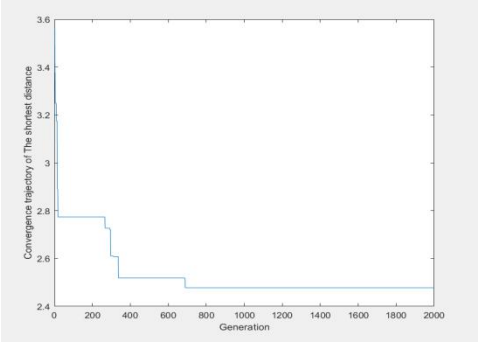
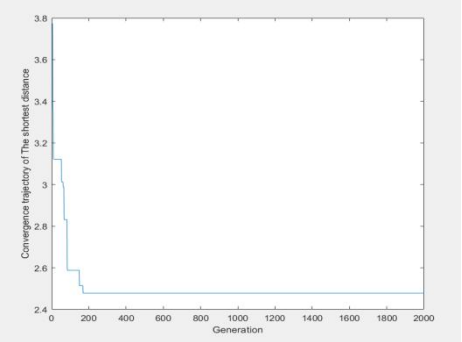


Figure1.7 - When crossover rate = 0.9



We can see Figure 1.4 to Figure 1.7 are the change in convergence rate because of the different crossover rate. When crossover rate is 0, the convergence speed is very slow, and sometimes the global optimal solution can not be found because of the local optimal solution (for example, Figure1.4 shows that in this simulation, it can't find the global best path. At the last iteration, it is around 2.6 rather than 2.4777). When crossover rate increasing, It converges faster and it can find the optimal solution in the earlier iteration times.

To conclude, If the crossover probability is too high, it loses its meaning and becomes a random algorithm;If it's too small, it converges too slowly. So it is important to choose an appropriate value.

[2] Change mutation rate

This part keeps the other parameters unchanged and sets the mutation rate1 and mutation rate2 to 0|0, 0.005|0.002, 0.05|0.02, 0.5|0.2, respectively, and observe the changes in the experimental results:

Figure1.8 - mutation rate1 = 0, mutation rate2 = 0

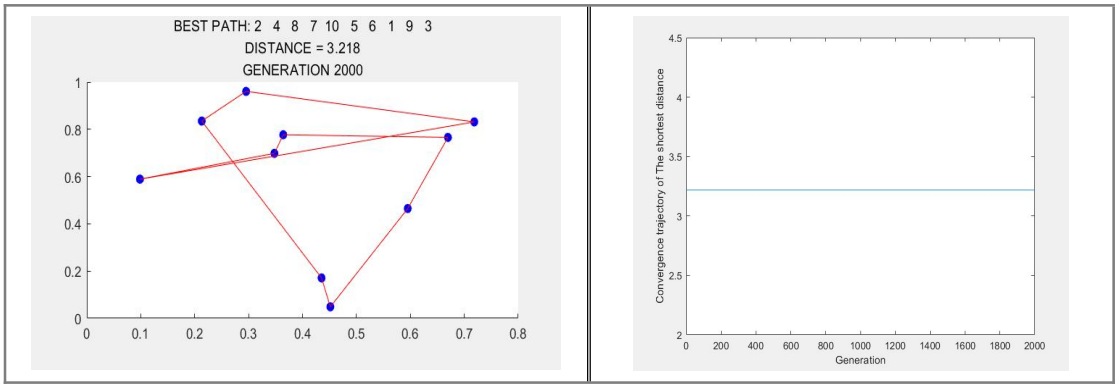


Figure1.9 - mutation rate1 = 0.005, mutation rate2 = 0.002

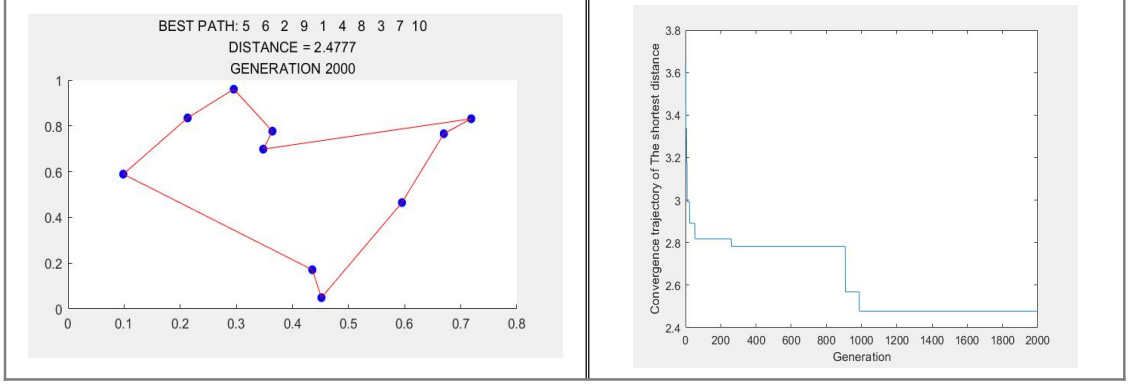


Figure1.10 - mutation rate1 = 0.05, mutation rate2 = 0.02

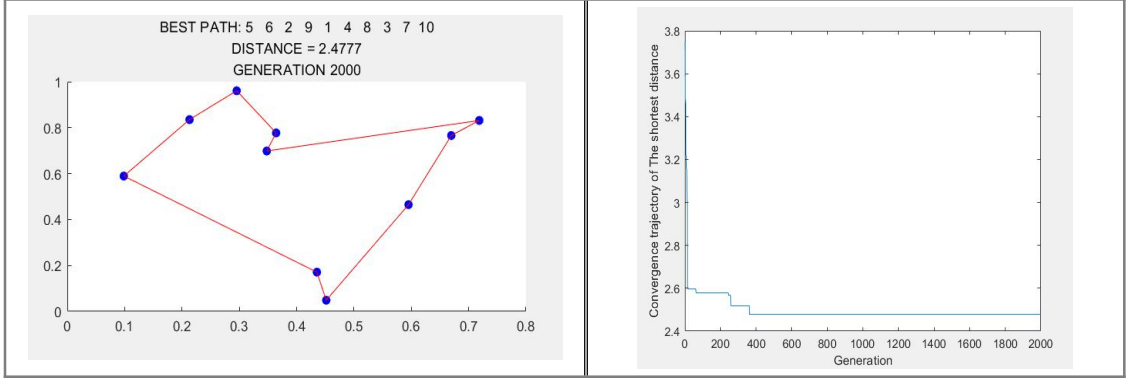


Figure1.11 - mutation rate1 = 0.5, mutation rate2 = 0.2

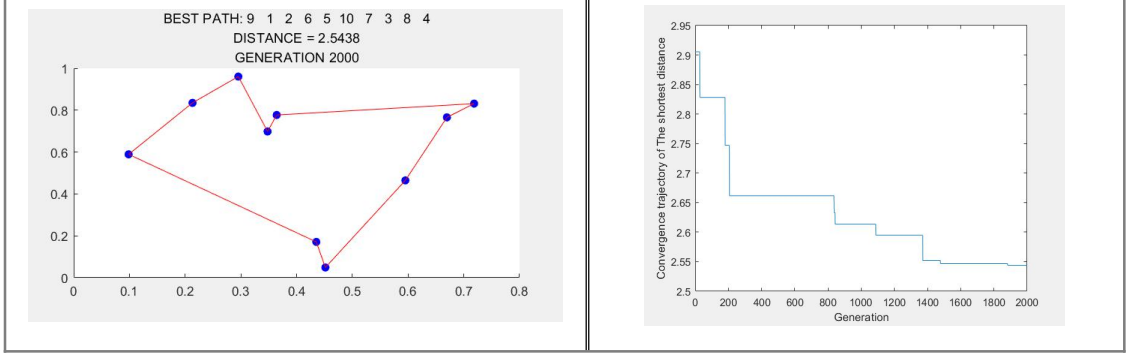


Figure 1.8 to Figure 1.11 are the results of different mutation rates, where mutation rate1 is the rate of swap two random cities(can also call it gene) and mutation rate2 is the rate of exchange

two parts of a path.

We can see in figure 1.8, both mutation rates are 0, which means the population can't take mutation. This may make it difficult for the population to produce new chromosomes. So without mutations, it's hard to find an optimal solution as shown in the figure. In figure 1.9, mutation rates are vary small, and we can see that the convergence speed is very slow, the optimal solution is found when the iteration reaches 1000 times. In figure 1.10, mutation rates are suitable, it can find an optimal solution quickly. In figure 1.11, mutation rates are too large, which results in a highly variable population and become hard to be stable in the optimal solution. And we can see that at the end of the iteration it still doesn't find an optimal solution.

In conclusion, mutation is to randomly change the value of some genes of individuals in a population with a small variation probability Pm. The basic process of variation operation is to generate a random number rand between [0,1]. If rand<Pm, then carry out mutation operation. Mutation operation itself is a kind of local random search, together with the selection, crossover operator, to avoid due to selection and crossover operator of some of the permanent loss of information, to ensure the effectiveness of the genetic algorithm, genetic algorithm has the local random search ability, makes the genetic algorithm can maintain the population diversity at the same time, to avoid premature convergence. In the variation operation, the variation probability should not be too large, if Pm>0.5, the genetic algorithm is degraded for random search.

[3] Change population size

This part keeps the other parameters unchanged and sets the population size to 10, 20, 50, 100, respectively, and observe the changes in the experimental results:

Figure1.12 - population size = 10

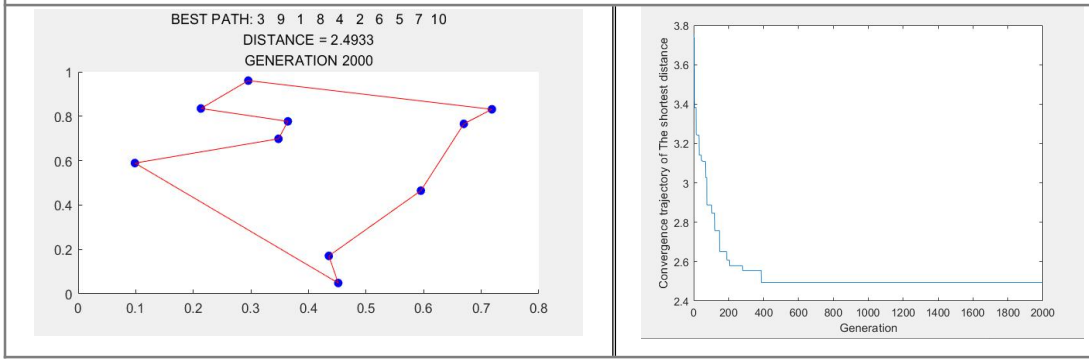


Figure1.13 - population size = 20

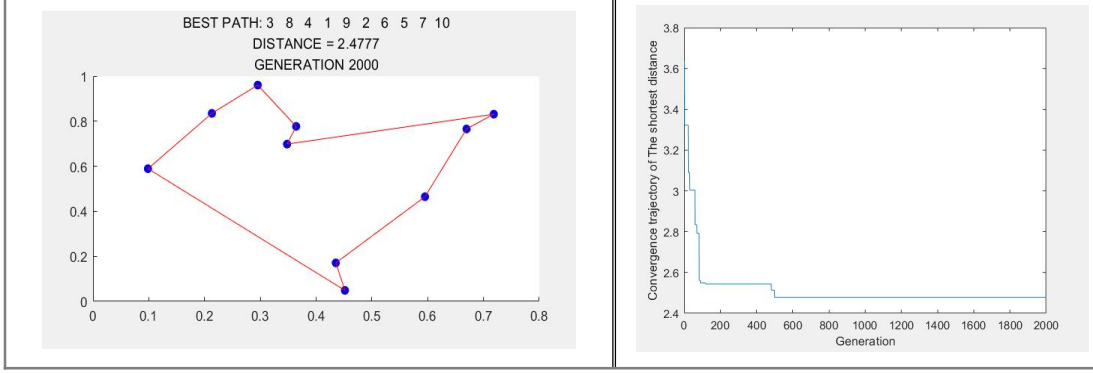


Figure1.14 - population size = 50

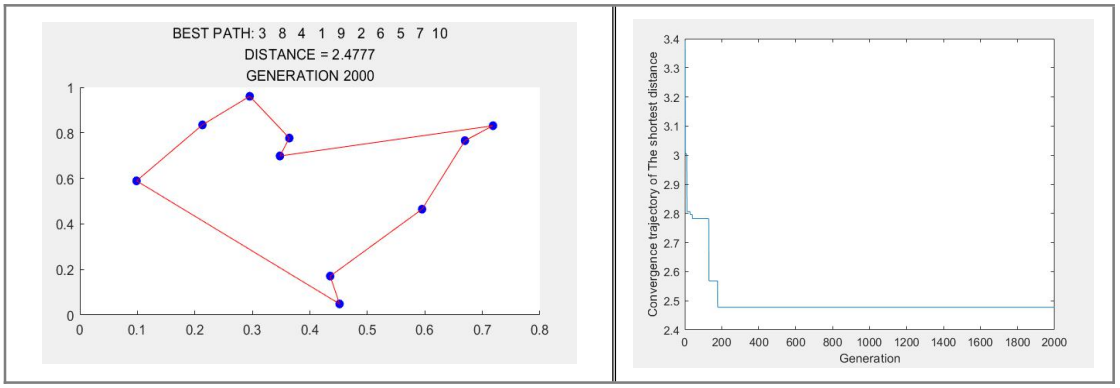


Figure1.15 - population size = 100

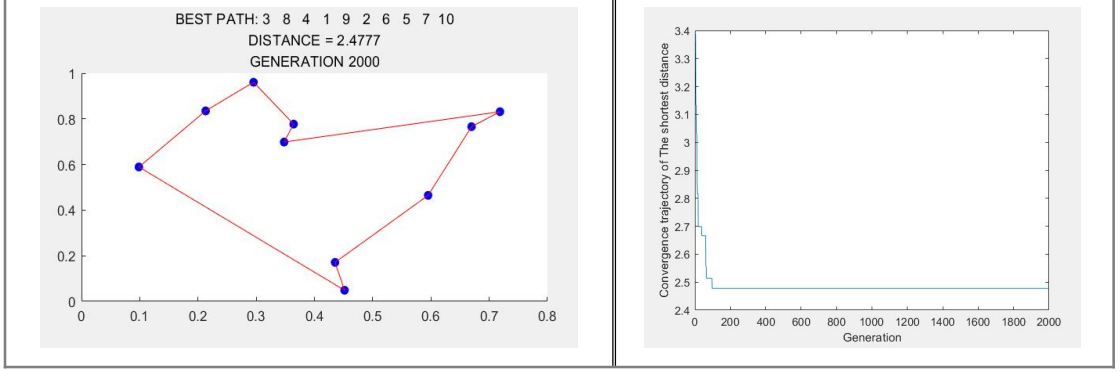


Figure 1.12 to Figure 1.15 are the results of using different population size.

We can see that if population size is small, there is a risk that the optimal solution will not be found. As population size increases, it's going to find the optimal solution more quickly, or it's going to be more likely to find the optimal solution. However, it is worth noting that when population size is large, the time required for iteration will increase significantly.

In conclusion, with the increase of population size, the probability of convergence to the optimal solution will be increased, that is, the global search capability will be enhanced.At the same time, when searching in the solution space, the optimal solution can be found in relatively few algebras, so the evolutionary algebra also decreases with the increase of population size.Although the larger the population size, the more likely to find a global solution, but the running time is also relatively long, so moderate.

[4] Change number of generation

This part keeps the other parameters unchanged and sets the number of generation to 100, 500, 2000, respectively, and observe the changes in the experimental results:

Figure1.16 - number of generation = 100

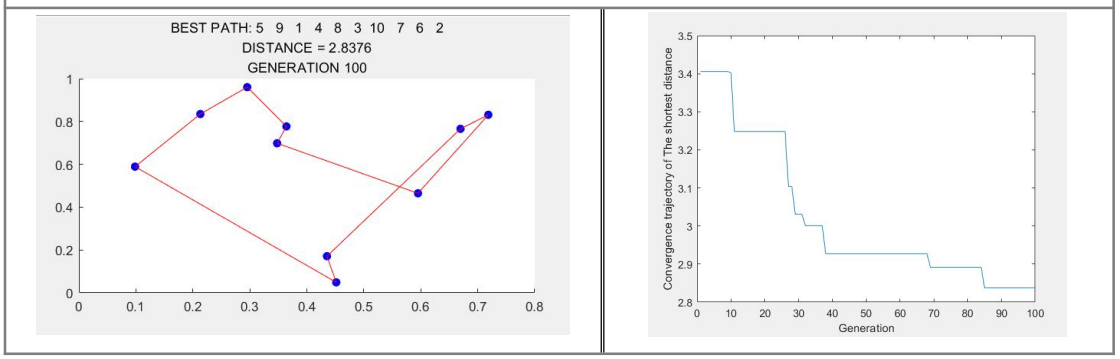


Figure1.17 - number of generation = 500

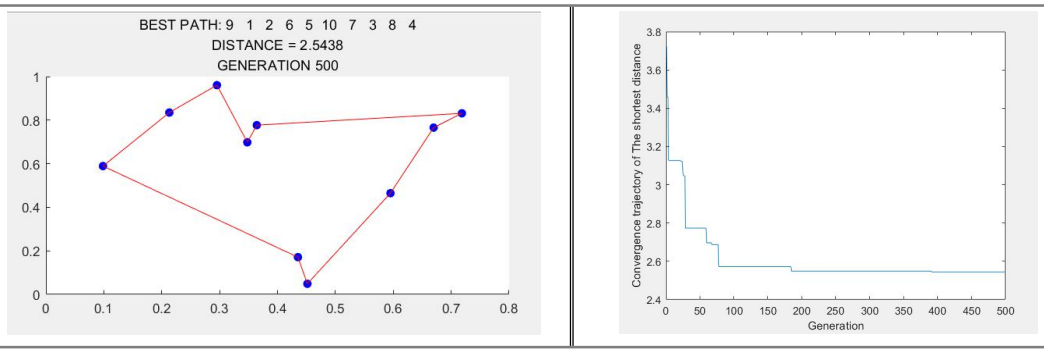


Figure1.18 - number of generation = 2000

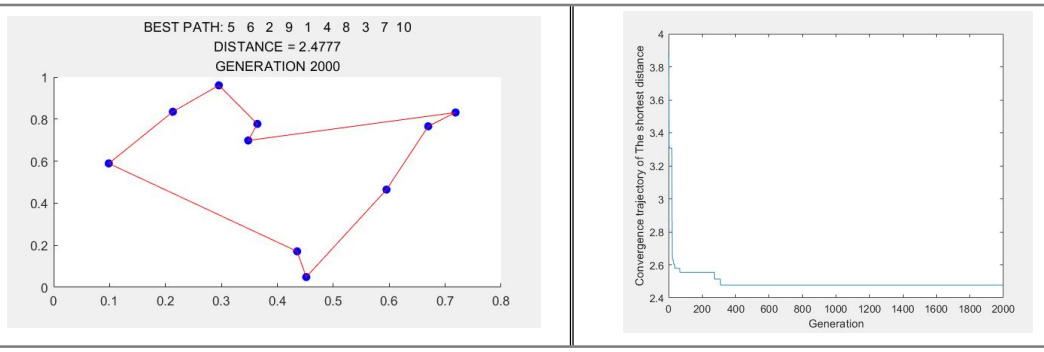


Figure 1.16 to Figure 1.18 are the results of using different number of generations.
We can see that if the number of generations is very small, there's a big risk that we won't find the optimal solution. As the number of generations goes up, this risk would be greatly reduced, but it would be replaced by a longer running time.

SUBQUESTION TWO : TSP Problem (with more cities)

◆ Description:

Based on the subquestion one, more cities are added in this part. The X-coordinate and Y-coordinate of the additional cities are limited to [0, 1]. So this part is designed to randomly generate multiple random points from 0 to 1, and then compare what happens to the result as the number of cities increases. To facilitate the experiment, all points will be randomly generated.

◆ Something new compared with the first program

[1] Elitism in GA

This part adds the technique of Elitism, which means Re-introduce in the population previous best-so-far (elitism). Keep the best 10% of parent population and re-introduce them in the next generation by replacing the worst 10% children population. In GA, elitism tends to improve on the final results.

[2] Reverse evolution

This is not available with standard genetic algorithms and is an operation I add to speed up evolution. Evolution here means that the reversal operation is unidirectional, that is, the individual will perform the reversal operation only after the reversal becomes better, otherwise the reversal is invalid.

The specific method is to randomly generate two random Numbers r1 and r2 between [1,10] (again, 10 cities are taken as examples here) (in fact, it is allowed to be the same, but when r1 and r2 are the same, it is not effective to reverse the nature, and setting cross variation is not effective, but this does not happen very often), and then reverse the sequence of genes between r1 and r2.

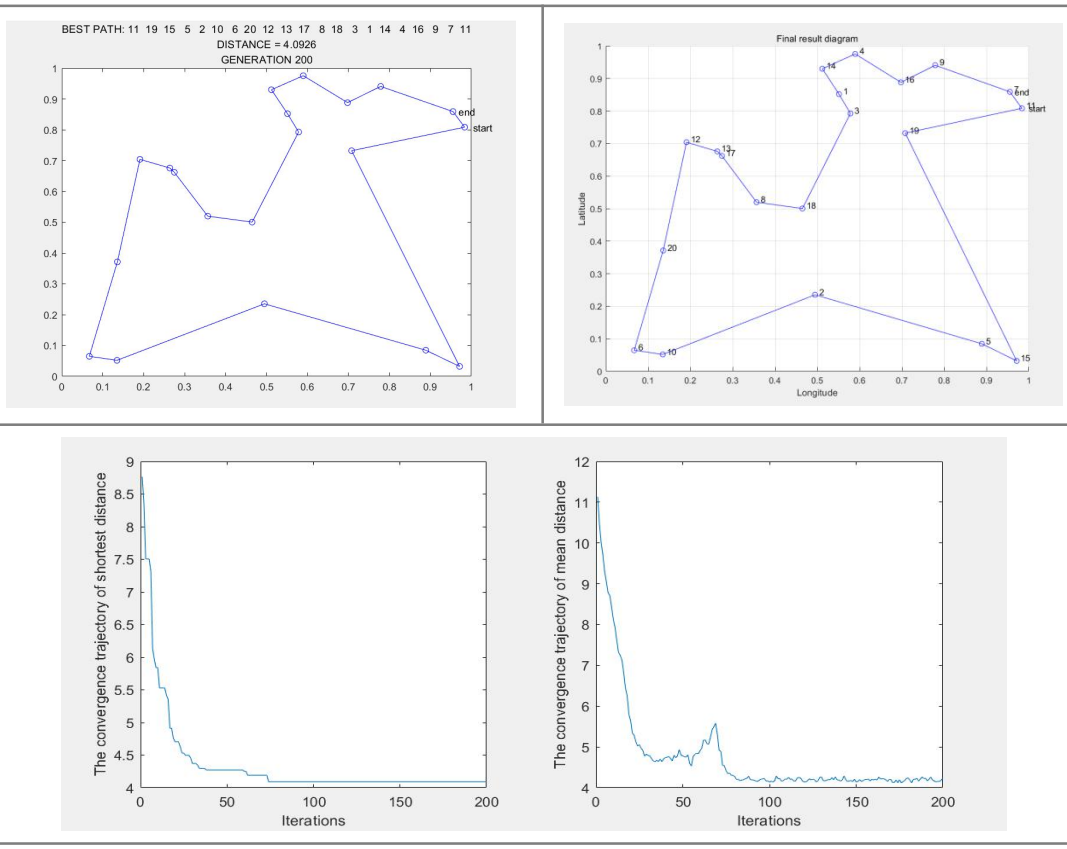
◆ Parameter Settings:

Parameter Description	Settings 1	Settings 2	Settings 3
Number of city	20	30	50
Initial population number	100	100	100
Crossover rate	0.8	0.8	0.8
Mutation rate	0.06	0.06	0.06
Generations / Iterations	200	300	400
Elitism rate	10%	10%	10%

◆ Results figures and analysis :

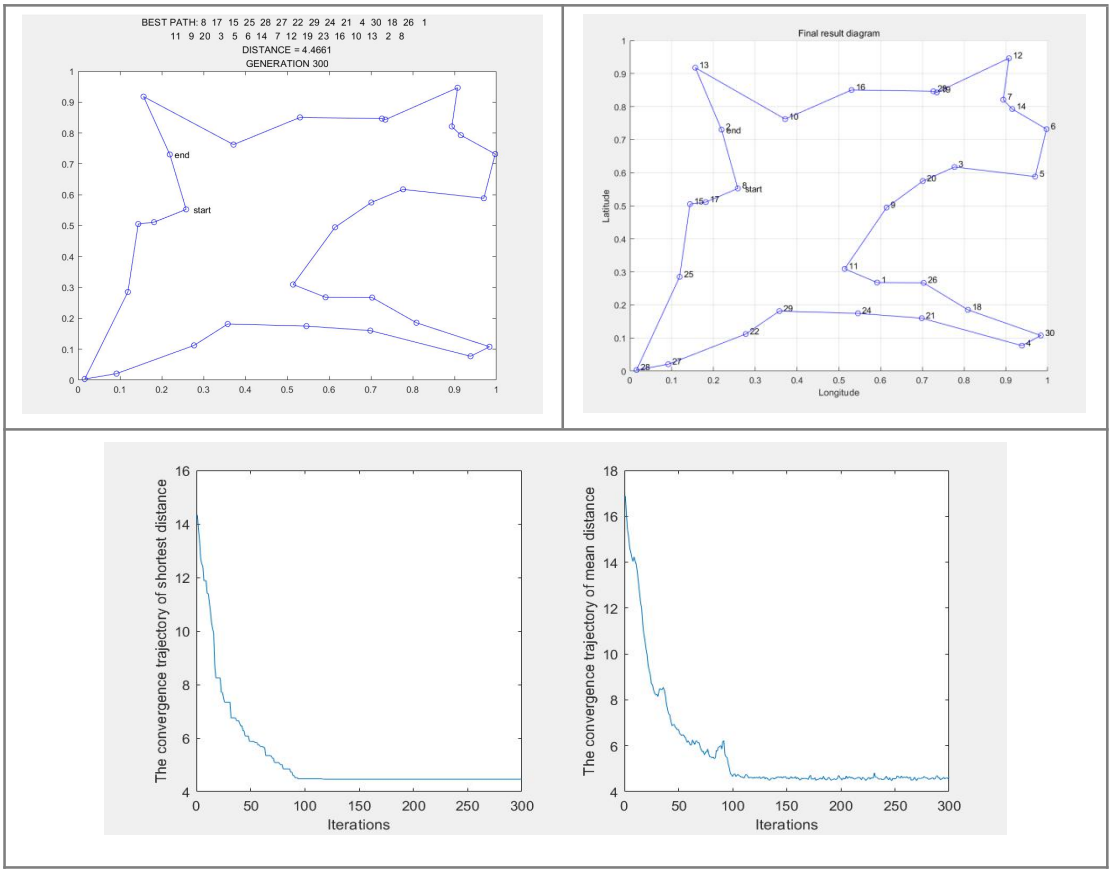
[1] when number of cities is 20 (Settings 1)

Figure set 2.1 : Optimal path diagram and convergence analysis when the number of cities is 20



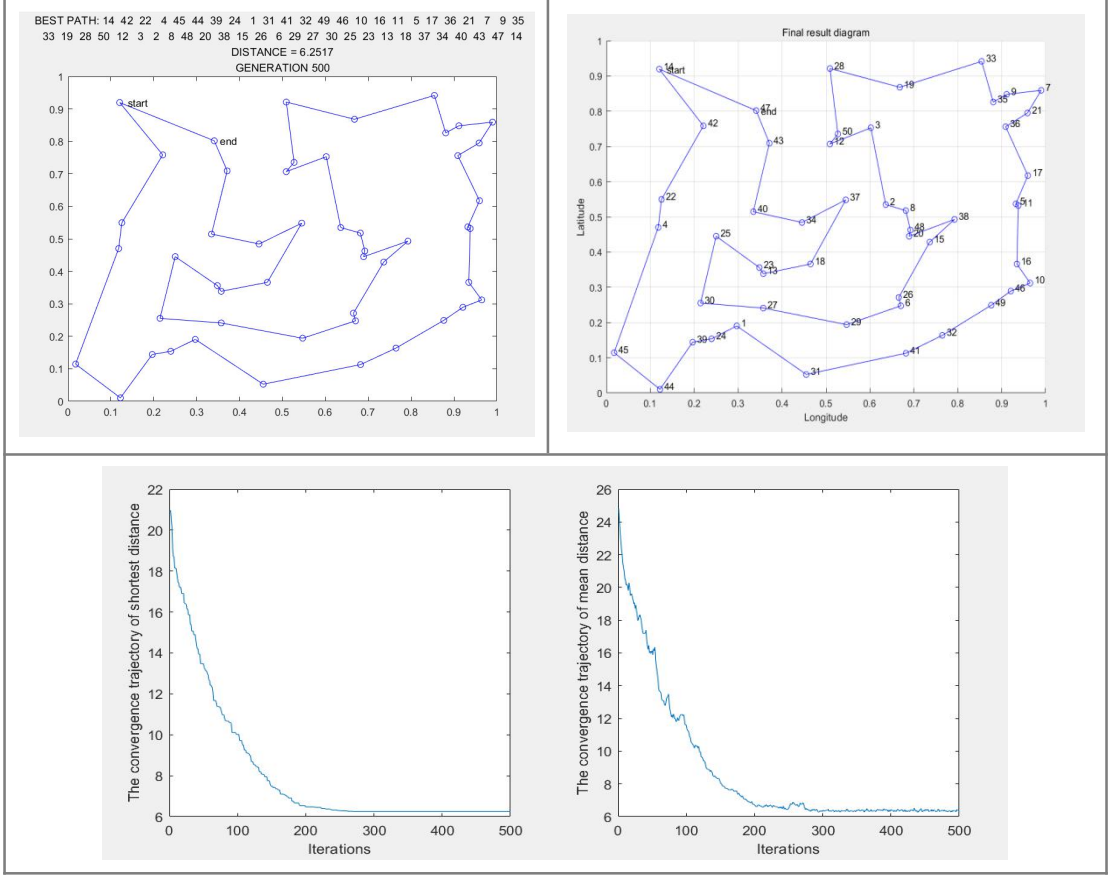
[2] when number of cities is 30 (Settings 2)

Figure set 2.2 : Optimal path diagram and convergence analysis when the number of cities is 30



[3] when number of cities is 50 (Settings 3)

Figure set 2.3 : Optimal path diagram and convergence analysis when the number of cities is 50



The figure sets 2.1 to 2.3 show the best path and convergence trajectory with different number of cities. We can clearly see that as the number of cities going up, more iterations are needed to find the best path.

[4] Program execution time (generations are all 500)

Number of cities	Command Window screenshot	Program execution time
20	program execution time:29.513seconds	29.513 s
30	program execution time:29.799seconds	29.799 s
50	program execution time:30.097seconds	30.097 s

We can see from the table that program execution time increases as the number of cities increases(in this part, the settings of generations are all 500, so these three experiments just different from the number of cities).

SUBQUESTION THREE : TSP (The end city is given - not start city)

◆ Description:

Different from the first two parts, In this part, the ending city is no longer the beginning city. In other words, this means that the salesman must visit each city just once, but instead of finishing at where he started, his last location should be the given end city.

The coordinates of all the cities are randomly generated from 0 to 1.

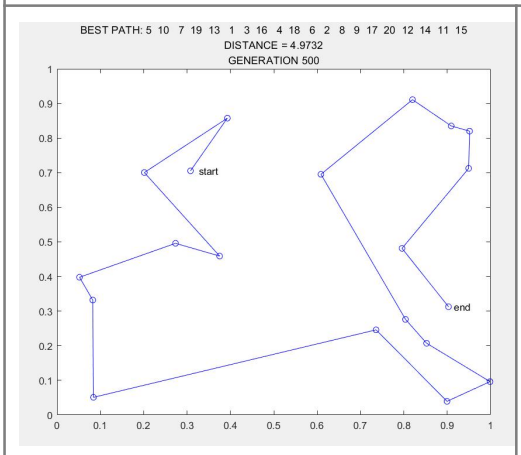
◆ Parameter Settings:

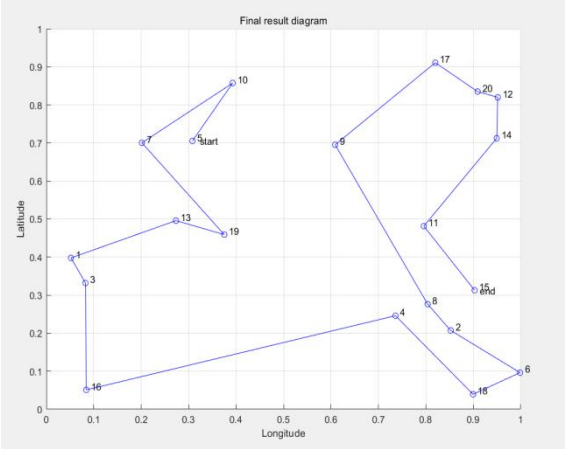
Parameter Description	Settings 1	Settings 2
Number of city	20	20
Crossover rate	0.8	0.8
Mutation rate	0.06	0.06
Generations / Iterations	500	500
Start city (serial number)	5	2
End city (serial number)	15	10

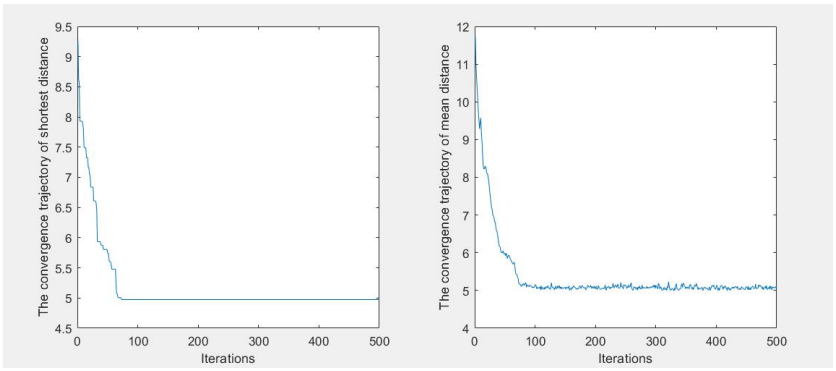
◆ Results figures and analysis :

[1] when start city is 5 and end city is 15

Figure set 3.1 : Optimal path diagram and convergence analysis when start city is 5 and end city is 15

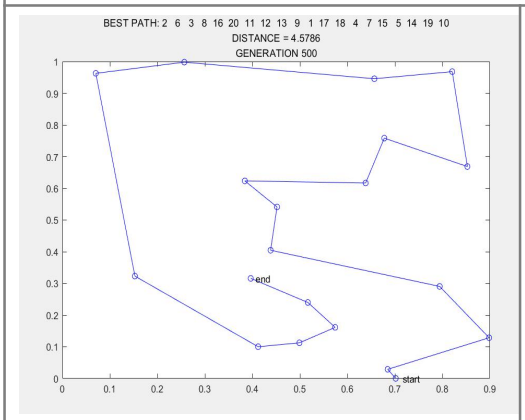


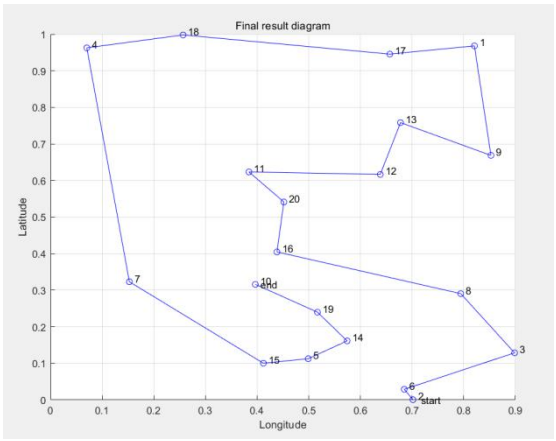


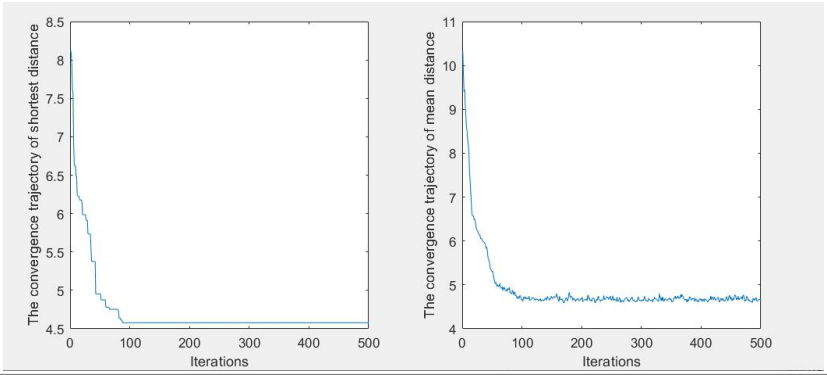


[2] when start city is 2 and end city is 10

Figure set 3.2 : Optimal path diagram and convergence analysis when start city is 2 and end city is 10







We can see that the result in Figure 3.1 conforms to the previous presupposition -- Go to city 5 first and city end up with 15. The two points in the figure are marked as start and end.
Figure 3.2 starts with city 2 and ends with city 10.

SUBQUESTION FOUR : Asymmetric traveling salesman problem (ATSP)

◆ Description:

The TSP problem is considered as an asymmetric one, i.e., for any two cities A and B, the distance from A to B is different from that from B to A.
The coordinates of all the cities are randomly generated from 0 to 1.

◆ Parameter Settings:

Parameter Description	Settings 1

Number of city	10
Initial population number	100
Generations / Iterations	100

Distance matrix

D

10x10 double

	1	2	3	4	5	6	7	8	9	10
1	0	0.8000	0.5000	0.1000	0.5000	0.5000	0.6000	0.5000	0.5000	0.9000
2	0.6000	0	0.5000	0.5000	0.5000	0.1000	0.5000	0.5000	0.8000	0.5000
3	0.7000	0.7000	0	0.5000	0.1000	0.5000	0.5000	0.5000	0.5000	0.5000
4	0.6000	0.5000	0.1000	0	0.5000	0.5000	0.5000	0.5000	0.5000	0.5000
5	0.5000	0.7000	0.5000	0.9000	0	0.8000	0.1000	0.5000	0.7000	0.5000
6	0.5000	0.5000	0.6000	0.5000	0.5000	0	0.6000	0.6000	0.5000	0.1000
7	0.6000	0.5000	0.7000	0.5000	0.5000	0.5000	0	0.5000	0.1000	0.6000
8	0.5000	0.1000	0.7000	0.6000	0.5000	0.5000	0.6000	0	0.5000	0.5000
9	0.5000	0.5000	0.8000	0.5000	0.5000	0.5000	0.5000	0.1000	0	0.9000
10	0.1000	0.8000	0.5000	0.9000	0.5000	0.5000	0.5000	0.5000	0.5000	0
11										

◆ Tips for verifying results:

Because the distance between cities is no longer just determined by location, in other words, the distance from A to B is different from that from B to A.

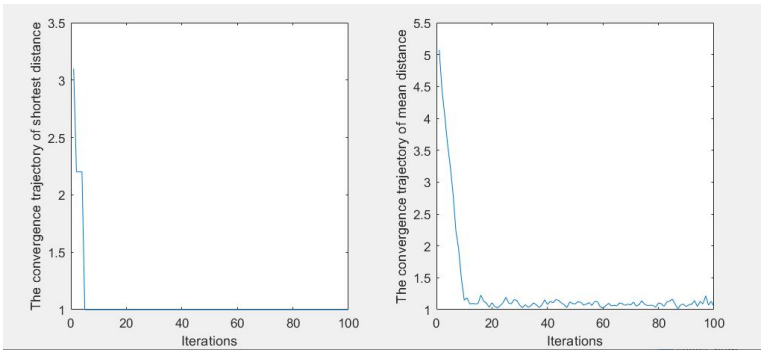
So here i use a trick to set the distance between cities: As shown c, the distance between cities enclosed by the orange box is set to 0.1 (small), and this path is the best path. On the other hand, i made the distance between the other two cities very large(>=0.5). Using this technique makes the final verification result more convenient and intuitive. We can clearly see that the best path in Distance matrix above is [1-4-3-5-7-9-8-2-6-10]. And the shortest path length is 1(0.1*10=1).

◆ Results figures and analysis :

Figure set 4.1 : Optimal path and convergence analysis

命令行窗口

```
The shortest distance around the city:1
The shortest route is shown below:
4 ---> 3
3 ---> 5
5 ---> 7
7 ---> 9
9 ---> 8
8 ---> 2
2 ---> 6
6 ---> 10
10 ---> 1
1 ---> 4
program execution time:5.833seconds
fx >>
```



The figure set 4.1 shows that the shortest path' length is 1 (as we expected). The best path is [4-3-5-7-9-8-2-6-10-1] (as we expected). So the experiment turned out to be correct.

SUBQUESTION FIVE : Sequential ordering problem (SOP)

◆ Description:

Sequence constraint is required in real world problem. The salesman is asked to visit certain cities in a required sequence. A particular city has to be visited before some other cities.

◆ Parameter Settings:

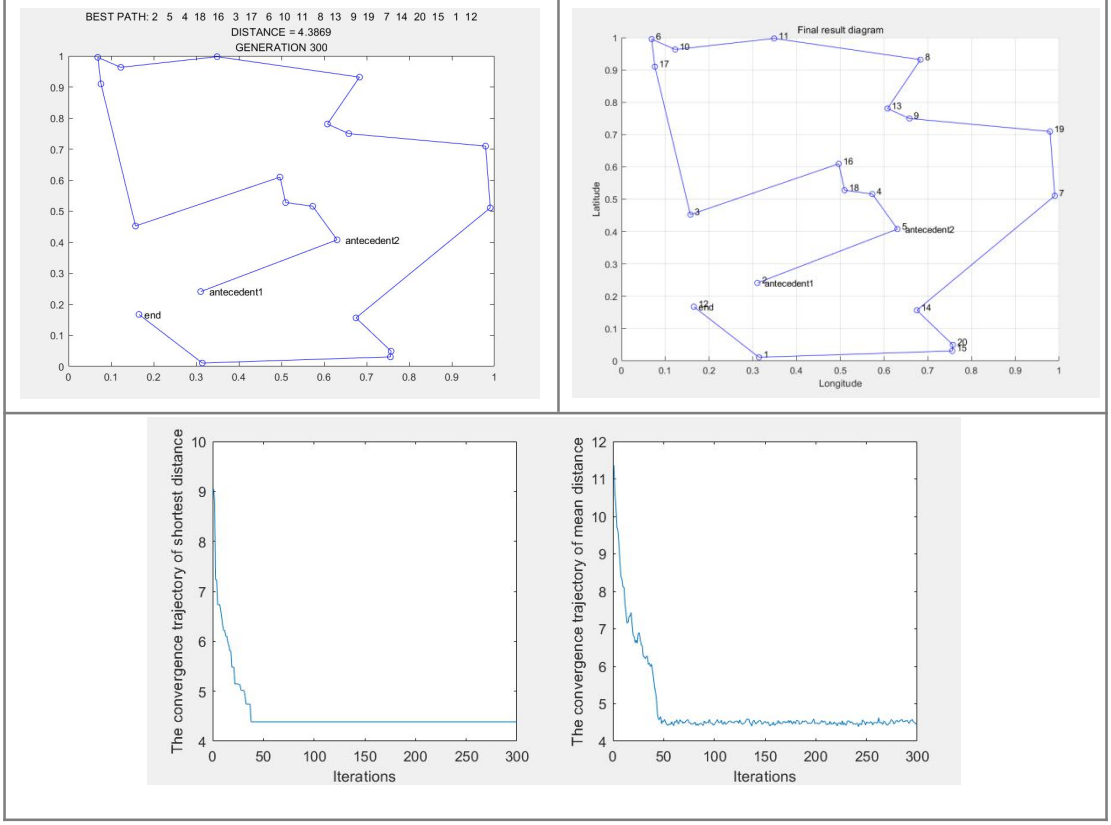
Parameter Description	Settings
Number of city	20
Generations / Iterations	300
Antecedence city 1 (serial number)	2
Antecedence city 2 (serial number)	5

◆ Sequence constraint:

Sequence constraint in this part is Antecedence city1 = 2 and Antecedence city2 = 5, which means that the salesman is asked to visit city2 and city5 preliminary before other cities.

◆ Results figures and analysis :

Figure set 5.1 : Optimal path diagram and convergence analysis
when Antecedence city1 = 2 and Antecedence city2 = 5



We can see that the result in Figure 5.1 conforms to the previous presupposition -- Go to city 2 and city 5 before other cities. The two points in the figure are marked as antecedent1 and antecedent2.

SUBQUESTION SIX : TSP (the cities are divided into several regions)

◆ Description:

For large-scale data, the cities can be divided into several regions , the salesman must finish visiting all the cities within the region before traveling to any other city in other regions.

◆ Parameter Settings:

Parameter Description	Settings 1	Settings 2
Number of city	50	50
Generations / Iterations	1000	600
Crossover rate || Mutation rate	0.8 || 0.08	0.8 || 0.08

◆ Pre-set several regions:

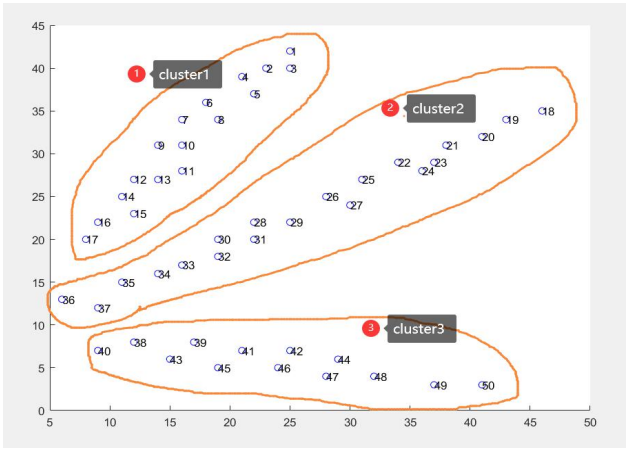


Figure 6.1 : The cities can be divided into 3 regions

I reorder the cities and divide them into 3 regions as the figure 6.1 shown:

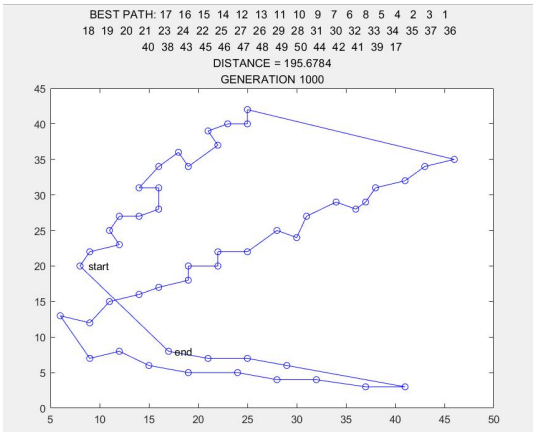
Cluster 1 includes city1 to city17;

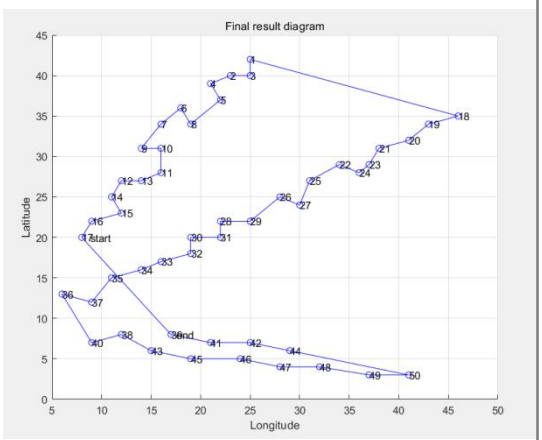
Cluster 2 includes city18 to city37;

Cluster 3 includes city38 to city40.

◆ Results figures and analysis :

Figure set 6.2 : Optimal path diagram and convergence analysis when have 3 regions





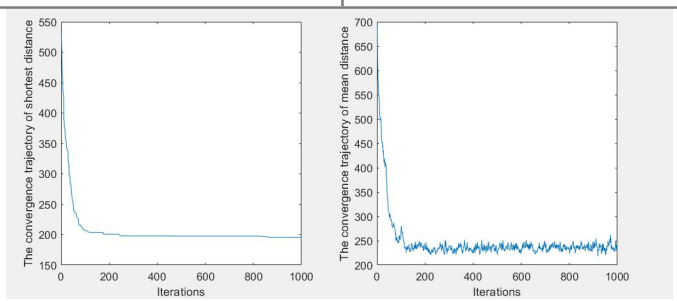
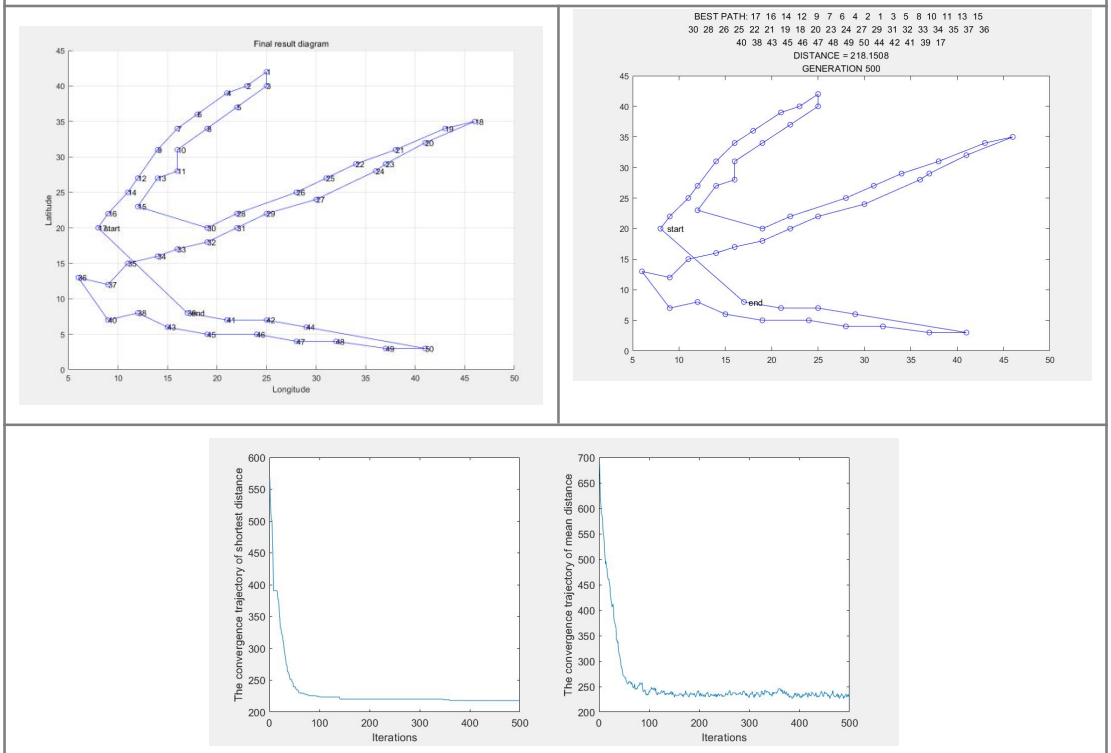


Figure set 6.3 : Optimal path diagram and convergence analysis when have 3 regions



We can see that the results of Figure set 6.2 and Figure set 6.3 both satisfy our conventional presupposition: First go to city 1 - 17, then go to city 18 - 37, and finally go to city 38 - 50 (These numbers are reordered number as figure 6.1 shown) . and we can see a better performance in Figure 6.2, so there may be some randomness in the results

Code description and operation method

These six files are the code for each of the six subproblems.

There is a main file in each file called **main.m**. Run the main file to get the results.