MECS 4510 HW1

Name: Jiong Lin

UNI: jl6017

Instructor: Hod Lipson

Grace hour used: 0

Grace hour remained: 96

Columbia Engineering
10/05/2021

1 Result summary table

		Evaluations	Length
Tsp1	The shortest path	57441	14.011
	The longest path	43889	800.091

Table 1: Result summary

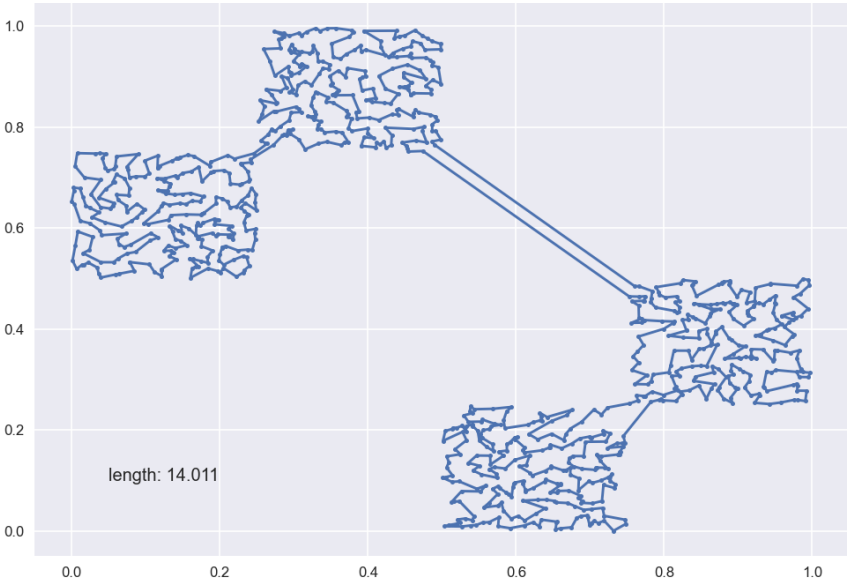


Figure 1: the shortest road

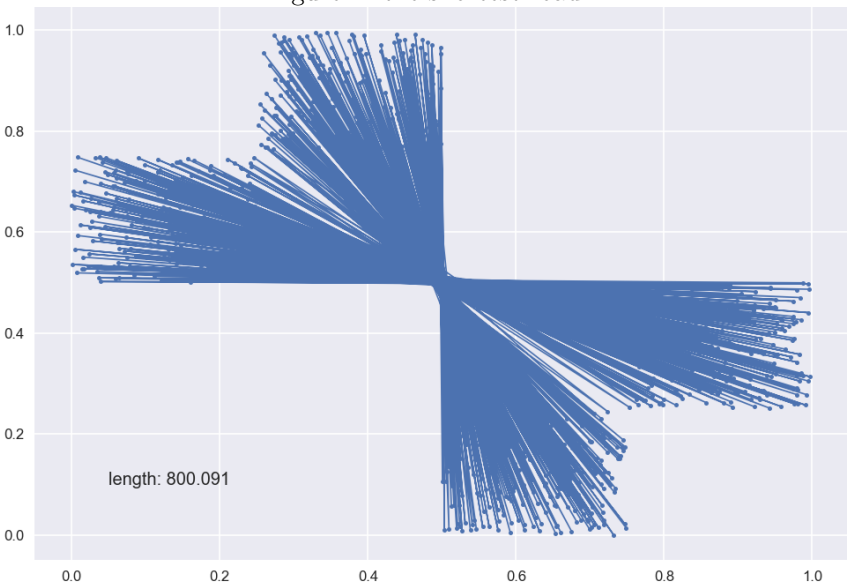


Figure 2: the Longest road

2 Methods

In this homework, I tried six kinds of methods, including random search, hill climb, and genetic algorithms.
I used c++ to calculate the evaluations and write the results into CSV files, and then used python to read files and plot them into graphics.

2.1 Representation

I used lists of integers to represent the genes. This is the most direct method but maybe not the best. In this tsp problem, the lists range from 0 to 999, and work as the indexes to get the orders of the cities.

2.2 Random search

1. Get a random list of integers as the order of the cities.
2. Calculate the length of this order.
3. If it is shorter than the last length, then keep it as the shortest length.
4. Back to the top and form the loop.

2.3 Random mutation hill climber

1. parent genes: get 400 random lists of integers as the orders of the cities.
2. Offspring genes: mutations and get 400 lists of integers.
3. Calculate the fitness of each genes and sort them.
4. Select top 50% as the next generation. Form the loop.

2.4 Genetic Algorithm

1. parent genes: get 400 random list of integers as the orders of the cities.
2. Offspring genes: crossovers and mutations and get 400 lists
3. Calculate the fitness of each genes and sort them.
4. Select top 50%(or use roulette method) as the next generation. Form the loop.

2.4.1 Variation operators

The variation operator used in this homework is the two-points crossover.
The main problem for the list of integers to conduct the crossover is that the numbers would be repeated if we simply exchange two segments. My method to avoid this problem is to compare the number in two segments, get the two numbers to be changed, then search each list and change the numbers inside each list respectively.
However, there is still a flaw in this exchange method. Exchange inside means we have to change other numbers of the list (outside the segment). Using priority encoding can avoid this problem and do a better job in the crossover.
I also tried to compare the difference between one-point crossover(not plot here) and two-point crossover. I got similar learning curves. I think that might be because the genes in the TSP problem are head-tail connected. So the list is actually a circle. Therefore, the one-point crossover is equal to a special two-point crossover that always cuts the end of the list.
As for the mutation, I select two random numbers in the gene and flip the segment between them.

2.4.2 Selection process

I used two kinds of selection processes, which are truncation and stochastic universal sampling.
Truncation: First I sort the group of genes by their fitness. Then I select the top 50% of the genes as the next generation.
Stochastic universal sampling: The selection probability is related to fitness. For example, when the population size is 400, and selection pressure is 50%, I select 80 from the top 100, select 60 between 100 and 200, select 40 between 200 and 300, and select 20 between 300 and 400.

2.5 Results analysis

Based on the learning curves, GA with large population and diversity have good results, whose final lengths down to 14. GA with small population and low diversity have bad results.
As for the selection methods, Truncation and Stochastic universal sampling have similar outcomes. Truncation is slight quicker during the evaluation, while SUS has better diversity, and they both get the final result within 60000 generations.

1. What worked: Two-points crossover; big population; diversity; Truncation and SUS selection
2. What did not work: low probability crossover; low diversity; small population; random search and RMHC

Reason:
The basic reasons why GA works are mutation and recombination.
RMHC has a poor link between parent and offspring because it breaks the blocks of parent genes through random mutation. So RMHC does not have good recombination.
GA with two-point crossover works well. Because two-point crossover is a good variation operator that has a strong link between parent and offspring. This method has good recombination.
GA with low mutation probability has low diversity and will be easily stuck at local optimal solutions.
GA with small population evolves slower than those with big population. But if we have a suitable mutation-probability, we can still get a decent result after many generations.

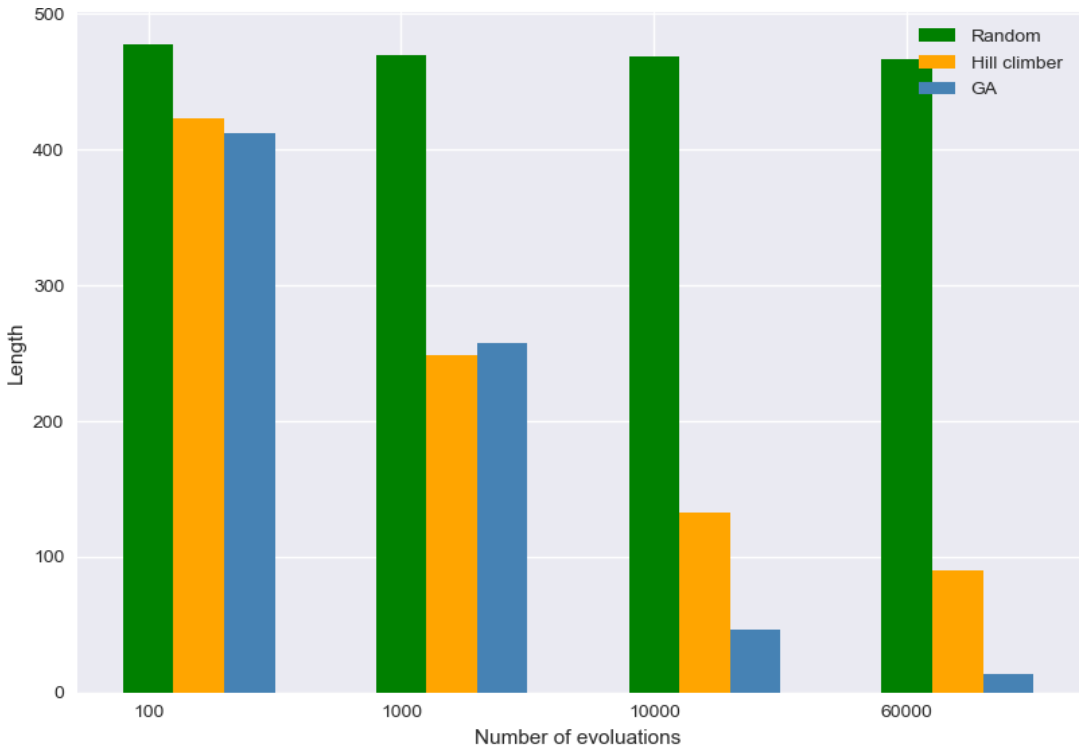


Figure 3: Results of three different methods

3 Performance plot

There are six different methods plot here, including random search, random mutation hill climber, and four Genetic algorithms. I also made the movie of optimizing path.

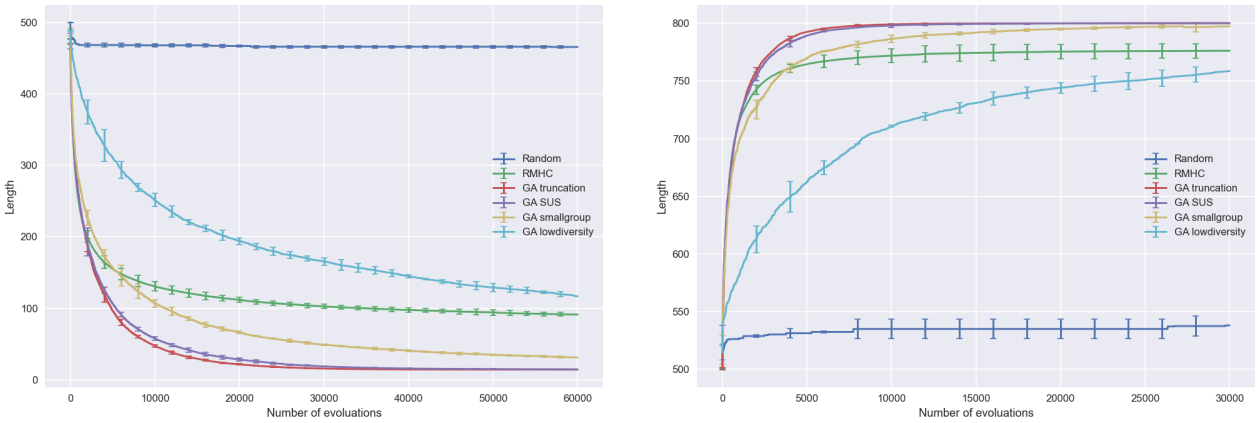
3.1 Here is the movie link:

GeneticAlgorithmforTSP

3.2 Parameters and graphs

	RMHC	GA truncation	GA SUS	GA small group	GA low diversity
Crossover	0%	100%	100%	100%	100%
Mutation	100%	30%	30%	30%	1%
Population	400	400	400	10	10

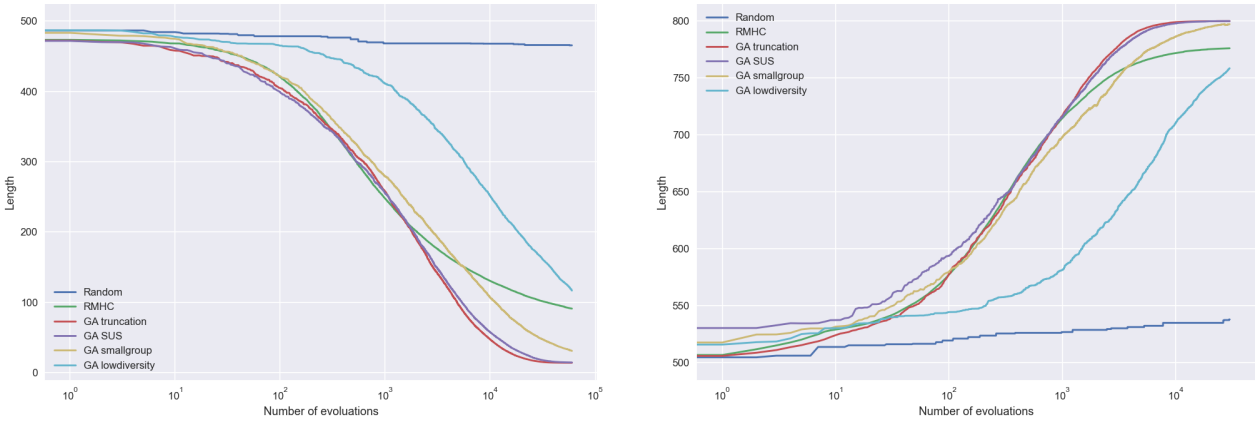
Table 2: Methods and parameters



(a) shortest road learning curve

(b) longest road learning curve

Figure 4: learning curves with error bars



(a) shortest road learning curve

(b) longest road learning curve

Figure 5: learning curves with X scale in log10

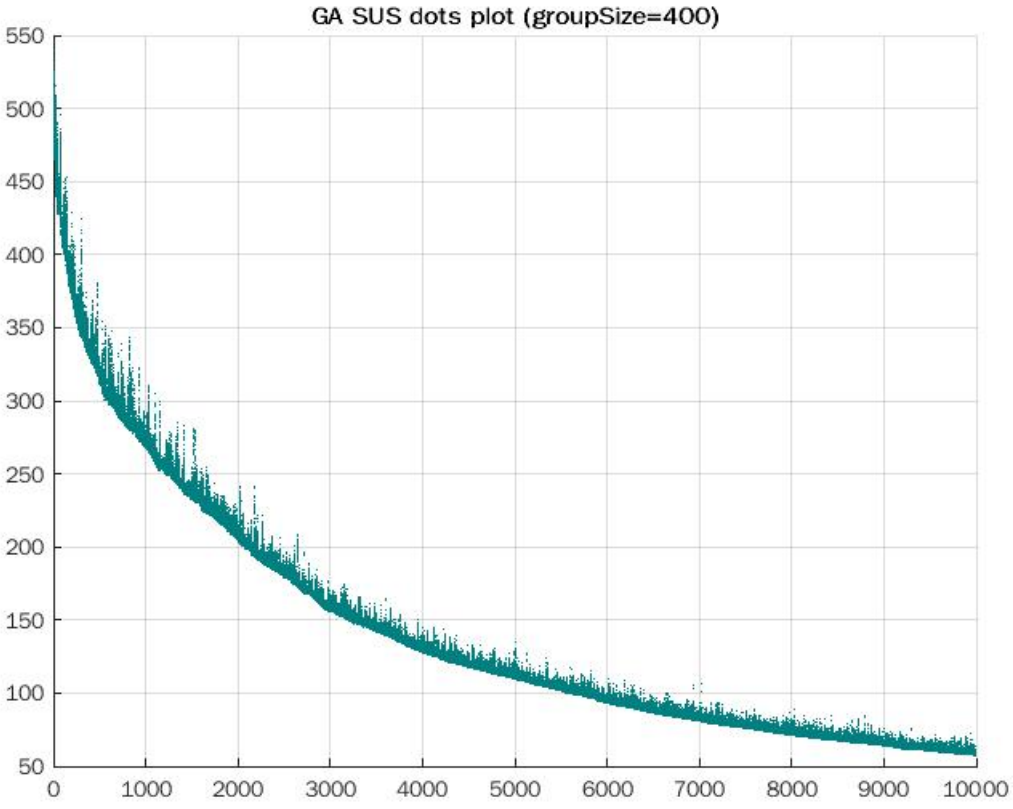


Figure 6: GA SUS dots plot

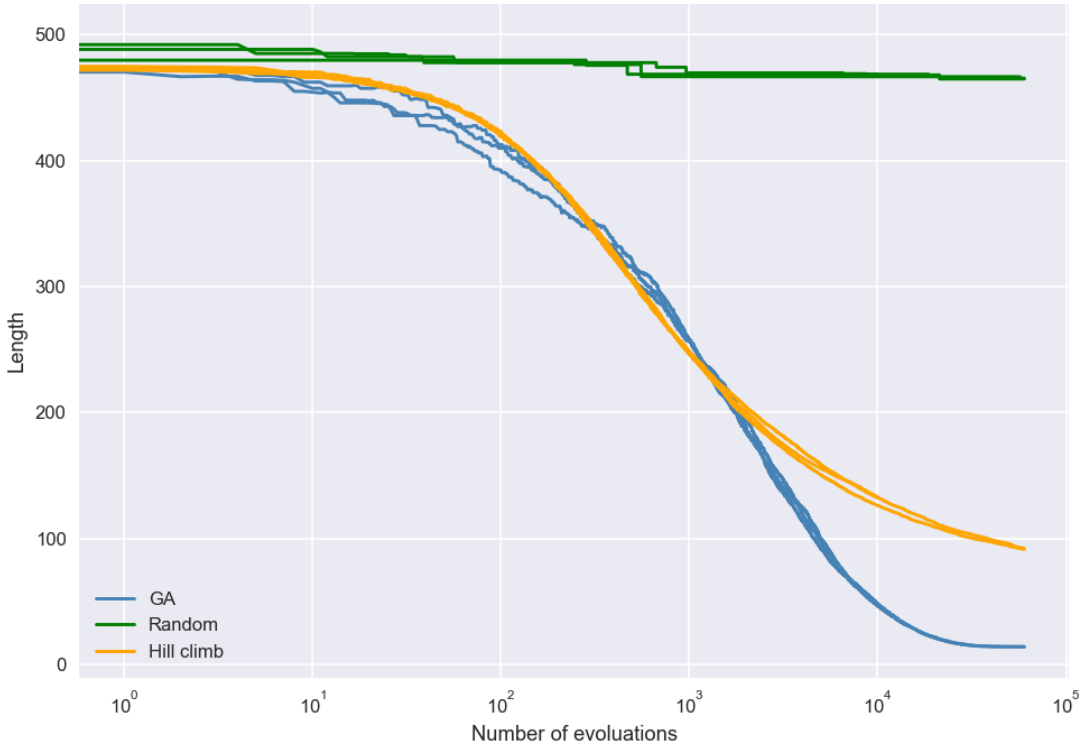


Figure 7: Convergence plot

4 Appendix

4.1 random search c++

```cpp
#include <iostream>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int pointNum = 1000;
int gen = 50000;

void readFile(float x[], float y[]) {
    ifstream inFile("tsp.txt", ios::in);
    if (!inFile)
    {
        cout << "fail to open " << endl;
        exit(1);
    }
    int i = 0;
    string line;
    string field;
    while (getline(inFile, line))
    {
        string field;
        istringstream sin(line);
        getline(sin, field, ',');
        x[i] = atof(field.c_str());
        getline(sin, field, ',');
        y[i] = atof(field.c_str());
        i++;
    }
    inFile.close();
}

void getMatrix(float matrix[][pointNum], float x[], float y[]) {
    for (int i = 0; i < pointNum; i++) {
        for (int j = 0; j < pointNum; j++) {
            matrix[i][j] = sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
        }
    }
}

float sumLen(int index[], float matrix[][pointNum]) {
    float sumLen = 0;
    for (int i = 1; i < pointNum; i++) {
        sumLen += matrix[index[i - 1]][index[i]];
    }
    return sumLen;
}

int main()
{
    //float x[pointNum], y[pointNum];
    float* x = new float[pointNum];
    float* y = new float[pointNum];
    int index[pointNum];
    float(*matrix)[pointNum] = new float[pointNum][pointNum];
    //float matrix[pointNum][pointNum];
    float shortLen = 500;

    for (int i = 0; i < pointNum; i++) {
        index[i] = i;
    }
    readFile(x, y);
    getMatrix(matrix, x, y);
    ofstream outFile("LDatars01.csv", ios::out);
    outFile << 0 << ",";
    outFile << shortLen << endl;
```

```
    for (int i = 0; i <= gen; i++) {
        shuffle(begin(index), end(index), std::mt19937(std::random_device()()));
        //LOG(index[0]);
        float l = sumLen(index, matrix);
        if (l > shortLen) {
            shortLen = l;
        }
        outFile << i << ",";
        outFile << shortLen << endl;
    }

    outFile.close();
    LOG(shortLen);
    return 0;
}
```

4.2 hill climb c++

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int pointNum = 1000;
const int gen = 50000;
const int groupSize = 400;
const int halfSize = 200;
const int chance = 200; // out of 1000

void readFile(float x[], float y[]) {
    ifstream inFile("tsp.txt", ios::in);
    if (!inFile)
    {
        cout << "fail to open " << endl;
        exit(1);
    }
    int i = 0;
    string line;
    string field;
    while (getline(inFile, line))
    {
        string field;
        istringstream stin(line);
        getline(stin, field, ',');
        x[i] = atof(field.c_str());
        getline(stin, field, ',');
        y[i] = atof(field.c_str());
        i++;
    }
    inFile.close();
}

void getMatrix(float matrix[][pointNum], float x[], float y[]) {
    for (int i = 0; i < pointNum; i++) {
        for (int j = 0; j < pointNum; j++) {
            matrix[i][j] = sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
        }
    }
}

float sumLen(int index[], float matrix[][pointNum]) {
    float sumLen = 0;
    for (int i = 1; i < pointNum; i++) {
        sumLen += matrix[index[i - 1]][index[i]];
    }
    sumLen += matrix[index[0]][index[pointNum - 1]];
    return sumLen;
}
```

```
58
59  void evoGroup(int group[][pointNum], int bigGroup[][pointNum], float matrix[][pointNum]) {
60      //write fathers into big group
61      for (int i = 0; i < groupSize; i++) {
62          for (int j = 0; j < pointNum; j++) {
63              bigGroup[i][j] = group[i][j];
64          }
65      }
66
67      //each father change two points to get a son
68      for (int i = 0; i < groupSize; i++) {
69          int r1 = rand() % pointNum;
70          int r2 = rand() % pointNum;
71          if (r1 > r2) {
72              int t = 0;
73              t = r1;
74              r1 = r2;
75              r2 = t;
76          }
77          int temp = group[i][r1];
78          group[i][r1] = group[i][r2];
79          group[i][r2] = temp;
80      }
81
82      //write sons into big group
83      for (int i = 0; i < groupSize; i++) {
84          for (int j = 0; j < pointNum; j++) {
85              bigGroup[i + groupSize][j] = group[i][j];
86          }
87      }
88  }
89
90  float select(int bigGroup[][pointNum], int group[][pointNum], float matrix[][pointNum]) {
91      // get sum of each road
92      float mySum[2 * groupSize];
93      for (int i = 0; i < 2 * groupSize; i++) {
94          mySum[i] = sumLen(bigGroup[i], matrix);
95      }
96      // sort and get index
97      int idx[2 * groupSize] = { 0 };
98      float copy[2 * groupSize] = { 0 };
99      memcpy(copy, mySum, 2 * groupSize * sizeof(float));
100     sort(mySum, mySum + 2 * groupSize);
101     for (int i = 0; i < 2 * groupSize; i++) {
102         for (int j = 0; j < 2 * groupSize; j++) {
103             if (mySum[i] == copy[j]) { idx[i] = j; }
104         }
105     }
106     // write top half into group
107     for (int i = 0; i < groupSize; i++) {
108         for (int j = 0; j < pointNum; j++) {
109             group[i][j] = bigGroup[idx[i+groupSize]][j];
110         }
111     }
112     LOG(mySum[groupSize-1]);
113     return mySum[groupSize-1];
114 }
115
116 int main()
117 {
118     float* x = new float[pointNum];
119     float* y = new float[pointNum];
120     int index[pointNum];
121     float(*matrix)[pointNum] = new float[pointNum][pointNum];
122     int(*group)[pointNum] = new int[groupSize][pointNum];
123     int(*bigGroup)[pointNum] = new int[2 * groupSize][pointNum];
124     for (int i = 0; i < pointNum; i++) {
125         index[i] = i;
126     }
127
128     readFile(x, y);
129     getMatrix(matrix, x, y);
130
131     ofstream outFile("LDataHC00.csv", ios::out);
132
133     for (int i = 0; i < groupSize; i++) {
```

```
        shuffle(begin(index), end(index), std::mt19937(std::random_device()()));
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = index[j];
        }
    }
    srand(time(NULL));
    for (int g = 0; g < gen; g++) {
        outFile << g << ",";
        evoGroup(group, bigGroup, matrix);
        float theLength = select(bigGroup, group, matrix);
        outFile << theLength << "\n";
    }

    outFile.close();
    return 0;
}
```

4.3 genetic algorithm c++

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
#include <ctime>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int pointNum = 1000;
const int gen = 100;
const int groupSize = 400;
const int halfSize = 200;
const int chance = 300; // out of 1000

void readFile(float x[], float y[]) {
    ifstream inFile("tsp.txt", ios::in);
    if (!inFile)
    {
        cout << "fail to open " << endl;
        exit(1);
    }
    int i = 0;
    string line;
    string field;
    while (getline(inFile, line))
    {
        string field;
        istringstream stin(line);
        getline(stin, field, ',');
        x[i] = atof(field.c_str());
        getline(stin, field, ',');
        y[i] = atof(field.c_str());
        i++;
    }
    inFile.close();
}

void getMatrix(float matrix[][pointNum], float x[], float y[]) {
    for (int i = 0; i < pointNum; i++) {
        for (int j = 0; j < pointNum; j++) {
            matrix[i][j] = sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
        }
    }
}

float sumLen(int index[], float matrix[][pointNum]) {
    float sumLen = 0;
    for (int i = 1; i < pointNum; i++) {
        sumLen += matrix[index[i - 1]][index[i]];
    }
    sumLen += matrix[index[0]][index[pointNum - 1]];
    return sumLen;
```

```
}

void evoGroup(int group[][pointNum], int bigGroup[][pointNum], float matrix[][pointNum]) {
    //get two random number
    int r1 = rand() % pointNum;
    int r2 = rand() % pointNum;
    if (r1 > r2) {
        int t = 0;
        t = r1;
        r1 = r2;
        r2 = t;
    }

    //write fathers into big group
    for (int i = 0; i < groupSize; i++) {
        for (int j = 0; j < pointNum; j++) {
            bigGroup[i][j] = group[i][j];
        }
    }

    //cross to get sons
    for (int i = 0; i < groupSize; i = i + 2){
        for (int j = r1; j <= r2; j++) {
            int a = group[i][j];
            int b = group[i + 1][j];
            for (int e = 0; e < pointNum; e++) {
                if (group[i][e] == b) {
                    int t = 0;
                    t = group[i][j];
                    group[i][j] = group[i][e];
                    group[i][e] = t;
                }
            }
            for (int k = 0; k < pointNum; k++) {
                if (group[i+1][k] == a) {
                    int t = 0;
                    t = group[i+1][j];
                    group[i+1][j] = group[i+1][k];
                    group[i+1][k] = t;
                }
            }
        }
    }

    //write sons into big group
    for (int i = 0; i < groupSize; i++) {
        for (int j = 0; j < pointNum; j++) {
            bigGroup[i + groupSize][j] = group[i][j];
        }
    }

    //mutation
    for (int i = 0; i < 2 * groupSize; i++) {
        int r3 = rand() % pointNum;
        if (r3 < chance) {
            int r4 = rand() % pointNum;
            int r5 = rand() % pointNum;
            if (r4 > r5) {
                int t = 0;
                t = r4;
                r4 = r5;
                r5 = t;
            }
            int temp = 0;
            for (int j = r4; j < 1 + (r4 + r5) / 2; j++) {
                temp = bigGroup[i][j];
                bigGroup[i][j] = bigGroup[i][r4 + r5 - j];
                bigGroup[i][r4 + r5 - j] = temp;
            }
        }
    }
}

float select(int bigGroup[][pointNum], int group[][pointNum], float matrix[][pointNum]) {
    // get sum of each road
    float mySum[2 * groupSize];
```

```
    for (int i = 0; i < 2 * groupSize; i++) {
        mySum[i] = sumLen(bigGroup[i], matrix);
    }
    // sort and get index
    int idx[2 * groupSize] = { 0 };
    float copy[2 * groupSize] = { 0 };
    memcpy(copy, mySum, 2 * groupSize * sizeof(float));
    sort(mySum, mySum + 2 * groupSize);
    for (int i = 0; i < 2 * groupSize; i++) {
        for (int j = 0; j < 2 * groupSize; j++) {
            if (mySum[i] == copy[j]) { idx[i] = j; }
        }
    }
    // write top half into group
    for (int i = 0; i < groupSize; i++) {
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = bigGroup[idx[i]][j];
        }
    }
    LOG(mySum[0]);
    return mySum[0];
}

int main()
{
    float* x = new float[pointNum];
    float* y = new float[pointNum];
    int index[pointNum];
    float(*matrix)[pointNum] = new float[pointNum][pointNum];
    int(*group)[pointNum] = new int[groupSize][pointNum];
    int(*bigGroup)[pointNum] = new int[2 * groupSize][pointNum];
    for (int i = 0; i < pointNum; i++) {
        index[i] = i;
    }

    readFile(x, y);
    getMatrix(matrix, x, y);

    ofstream outFile("Data.csv", ios::out);
    ofstream anotherFile("road.csv", ios::out);

    for (int i = 0; i < groupSize; i++) {
        shuffle(begin(index), end(index), std::mt19937(std::random_device()()));
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = index[j];
        }
    }
    srand(time(NULL));
    for (int g = 0; g < gen; g++) {
        outFile << g << ",";
        evoGroup(group, bigGroup, matrix);
        float theLength = select(bigGroup, group, matrix);
        outFile << theLength << "\n";
        for (int i = 0; i < pointNum; i++) {
            anotherFile << group[0][i] << ",";
        }

    }

    outFile.close();
    anotherFile.close();

    return 0;
}
```

4.4 genetic algorithm SUS c++

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include <math.h>
#include <vector>
#include <random>
```

```
#include <ctime>
#define LOG(x) std::cout<<x<<std::endl
using namespace std;

const int pointNum = 1000;
const int gen = 100;
const int groupSize = 400;
const int chance = 300; // out of 1000

void readFile(float x[], float y[]) {
    ifstream inFile("tsp.txt", ios::in);
    if (!inFile)
    {
        cout << "fail to open " << endl;
        exit(1);
    }
    int i = 0;
    string line;
    string field;
    while (getline(inFile, line))
    {
        string field;
        istringstream stin(line);
        getline(stin, field, ',');
        x[i] = atof(field.c_str());
        getline(stin, field, ',');
        y[i] = atof(field.c_str());
        i++;
    }
    inFile.close();
}

void getMatrix(float matrix[][pointNum], float x[], float y[]) {
    for (int i = 0; i < pointNum; i++) {
        for (int j = 0; j < pointNum; j++) {
            matrix[i][j] = sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
        }
    }
}

float sumLen(int index[], float matrix[][pointNum]) {
    float sumLen = 0;
    for (int i = 1; i < pointNum; i++) {
        sumLen += matrix[index[i - 1]][index[i]];
    }
    sumLen += matrix[index[0]][index[pointNum - 1]];
    return sumLen;
}

void evoGroup(int group[][pointNum], int bigGroup[][pointNum], float matrix[][pointNum]) {

    //get two random number
    int r1 = rand() % pointNum;
    int r2 = rand() % pointNum;
    if (r1 > r2) {
        int t = 0;
        t = r1;
        r1 = r2;
        r2 = t;
    }

    //write fathers into big group
    for (int i = 0; i < groupSize; i++) {
        for (int j = 0; j < pointNum; j++) {
            bigGroup[i][j] = group[i][j];
        }
    }

    //cross to get sons
    for (int i = 0; i < groupSize; i = i + 2) {
        for (int j = r1; j <= r2; j++) {
            int a = group[i][j];
            int b = group[i + 1][j];
            for (int e = 0; e < pointNum; e++) {
                if (group[i][e] == b) {
                    int t = 0;
```

```
                    t = group[i][j];
                    group[i][j] = group[i][e];
                    group[i][e] = t;
                }
            }
            for (int k = 0; k < pointNum; k++) {
                if (group[i + 1][k] == a) {
                    int t = 0;
                    t = group[i + 1][j];
                    group[i + 1][j] = group[i + 1][k];
                    group[i + 1][k] = t;
                }
            }
        }
    }

    //write sons into big group
    for (int i = 0; i < groupSize; i++) {
        for (int j = 0; j < pointNum; j++) {
            bigGroup[i + groupSize][j] = group[i][j];
        }
    }

    //mutation
    for (int i = 0; i < 2 * groupSize; i++) {
        int r3 = rand() % pointNum;
        if (r3 < chance) {
            int r4 = rand() % (pointNum-40);
            int r5 = rand() % 20;

            int temp = 0;
            for (int j = r4; j < 1 + r4 + r5; j++) {
                temp = bigGroup[i][j];
                bigGroup[i][j] = bigGroup[i][2 * r4 + 2 * r5 - j];
                bigGroup[i][2 * r4 + 2 * r5 - j] = temp;
            }
        }
    }
}

float select(int bigGroup[][pointNum], int group[][pointNum], float matrix[][pointNum]) {
    // get sum of each road
    float mySum[2 * groupSize];
    for (int i = 0; i < 2 * groupSize; i++) {
        mySum[i] = sumLen(bigGroup[i], matrix);
    }
    // sort and get index
    int idx[2 * groupSize] = { 0 };
    float copy[2 * groupSize] = { 0 };
    memcpy(copy, mySum, 2 * groupSize * sizeof(float));
    sort(mySum, mySum + 2 * groupSize);
    for (int i = 0; i < 2 * groupSize; i++) {
        for (int j = 0; j < 2 * groupSize; j++) {
            if (mySum[i] == copy[j]) { idx[i] = j; }
        }
    }
    // write top half into group
    for (int i = 0; i < 80; i++) {
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = bigGroup[idx[i]][j];
        }
    }
    for (int i = 100; i < 160; i++) {
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = bigGroup[idx[i]][j];
        }
    }
    for (int i = 200; i < 240; i++) {
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = bigGroup[idx[i]][j];
        }
    }
    for (int i = 300; i < 320; i++) {
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = bigGroup[idx[i]][j];
        }
```

```
    }
    LOG(mySum[0]);
    return mySum[0];
}

int main()
{
    float* x = new float[pointNum];
    float* y = new float[pointNum];
    int index[pointNum];
    float(*matrix)[pointNum] = new float[pointNum][pointNum];
    int(*bigGroup)[pointNum] = new int[2 * groupSize][pointNum];
    int(*group)[pointNum] = new int[groupSize][pointNum];

    for (int i = 0; i < pointNum; i++) {
        index[i] = i;
    }

    readFile(x, y);
    getMatrix(matrix, x, y);

    ofstream outFile("Data.csv", ios::out);
    ofstream anotherFile("road.csv", ios::out);

    for (int i = 0; i < groupSize; i++) {
        shuffle(begin(index), end(index), std::mt19937(std::random_device()()));
        for (int j = 0; j < pointNum; j++) {
            group[i][j] = index[j];
        }
    }
    srand(time(NULL));
    for (int g = 0; g < gen; g++) {
        outFile << g << ",";
        evoGroup(group, bigGroup, matrix);
        float theLength = select(bigGroup, group, matrix);
        outFile << theLength << "\n";
        for (int i = 0; i < pointNum; i++) {
            anotherFile << group[0][i] << ",";
        }

    }

    outFile.close();
    anotherFile.close();

    return 0;
}
```

4.5 plot with python

```
import matplotlib.pyplot as plt
def plotFile(file1, file2, file3, label, color):
    f1 = open(file1, "r")
    f2 = open(file2, "r")
    f3 = open(file3, "r")
    line1 = f1.readlines()
    line2 = f2.readlines()
    line3 = f3.readlines()
    x, y, yerr = [], [], []
    ymax, ymin = [], []
    for i in range(60000):
        x.append(i)
        y1 = eval(line1[i])[1]
        y2 = eval(line2[i])[1]
        y3 = eval(line3[i])[1]
        li = [y1,y2,y3]
        l = max(li)-min(li)
        ymax.append(max(li)+l)
        ymin.append(min(li)-l)
        yerr.append((max(li)-min(li)))
        y.append((y1+y2+y3)/3)
    plt.xscale('log')
    # plt.errorbar(x,y,yerr=yerr,errorevery=2000,capsize=3,capthick=1,label=label)
    plt.plot(x,y,label=label)
    # plt.fill_between(x,ymax,ymin,alpha=0.3)
```

```
plt.style.use('seaborn')
plt.figure(dpi=120)
plotFile("Datars00.csv","Datars01.csv","Datars02.csv", 'Random', 'steelblue')
plotFile("DataHC00.csv","DataHC01.csv","DataHC02.csv", 'RMHC', 'chocolate')
plotFile("DataGA00.csv","DataGA01.csv","DataGA02.csv", 'GA truncation', 'mediumpurple')
plotFile("dataSUS01.csv","dataSUS02.csv","dataSUS03.csv", 'GA SUS', 'teal')
plotFile("data_small_group.csv","data_small_group02.csv","data_small_group03.csv", 'GA smallgroup', '
    skyblue')
plotFile("datalowdiv01.csv","datalowdiv02.csv","datalowdiv03.csv", 'GA lowdiversity', 'yellowgreen')
plt.title("learning curves")
plt.xlabel("Number of evoluations")
plt.ylabel("Length")
plt.legend()
plt.show()
```