Assignment 1 – Traveling Salesman

Hongbo Zhu UNI: hz2629

Evolutionary Computation & Design Automation

(MECS E4510)

Instructor: Prof. Hod Lipson

Date Submitted: 9/17/2019

Grace Hours Used: 0 h

Grace Hours Remaining: 96 h

0. Introduction

This assignment proposes an example of using random search method, to solve Traveling Salesman Problem. This problem is about finding the shortest closed path that goes through all points exactly once, and returns to the starting point. In this assignment, the number of points is 1000.

Random search is kind of optimization method. By sample enough times, this method is able to find a part of solution in entire search space, and each sample is independent. By comparing the results of each sample, the best solution it finds is sorted out.

By using C++, the coordinates data of each point can be imported into random search script. After the operation of C++ script in several times of sampling, the fitness, which is the length of path the salesman takes, and the order of the point this salesman chooses are imported into two text files. MATLAB is used to generate the figures of fitness and the path.

In first chapter, the path figure and fitness of the original points in tsp.txt is shown, which can be used to compared to the figure of points after processing. The path and the fitness figure of the shortest path are in the second chapter. In third chapter, the figures of longest path are shown. In forth path, a summary is proposed. Finally, the C++ code is in appendix.

1. Original Path

The fitness of Original Path, i.e. the length of the path, is calculated by MATLAB (Fig.1). The result is 510.5282. The path is shown in Fig.2. The red dots are points in space; blue lines are paths.

```
1 -    x = xlsread('Point0.xlsx','A1:A1000');
2 -    y = xlsread('Point0.xlsx','B1:B1000');
3 -    fitness = 0;
4 -    for i = 1:999
5 -        xd = x(i+1) - x(i);
6 -        yd = y(i+1) - y(i);
7 -        fitness = fitness + sqrt(xd^2 + yd^2);
8 -    end
```

Fig.1 MATLAB Code of calculating fitness

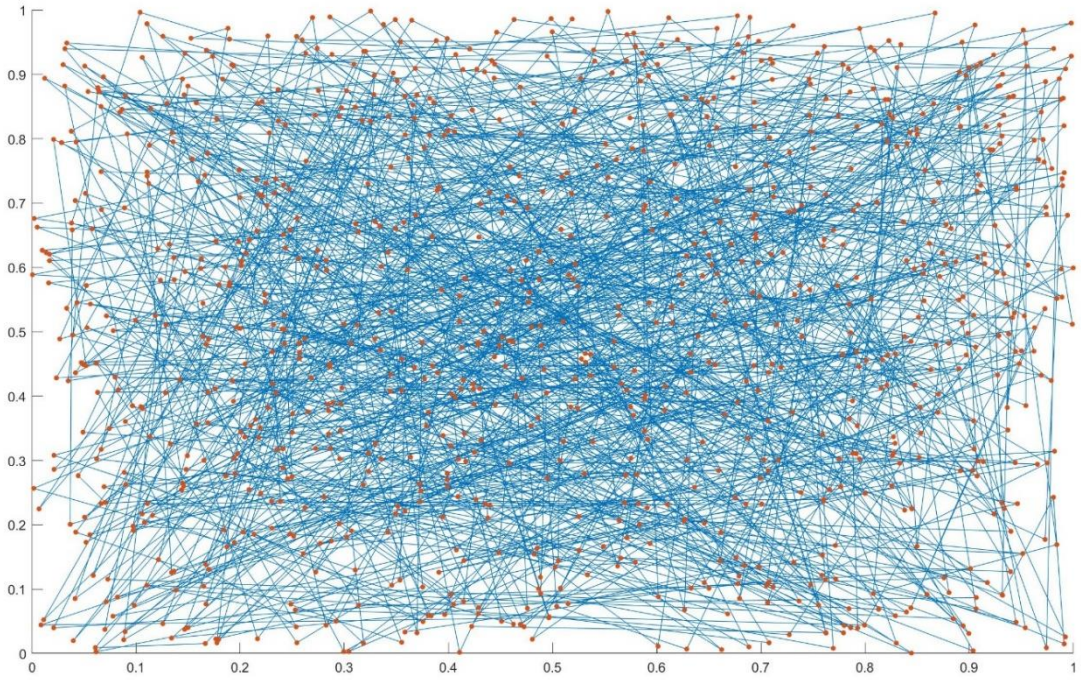

Fig.2 Shortest the Path of Original Points

2. Shortest Path

2.1 Sampling 200000 times

The learning curves is shown in Fig.3 which is averaged on five runs. The smallest fitness in figure is 490.554182, and the standard deviation in this point is 1.764974.

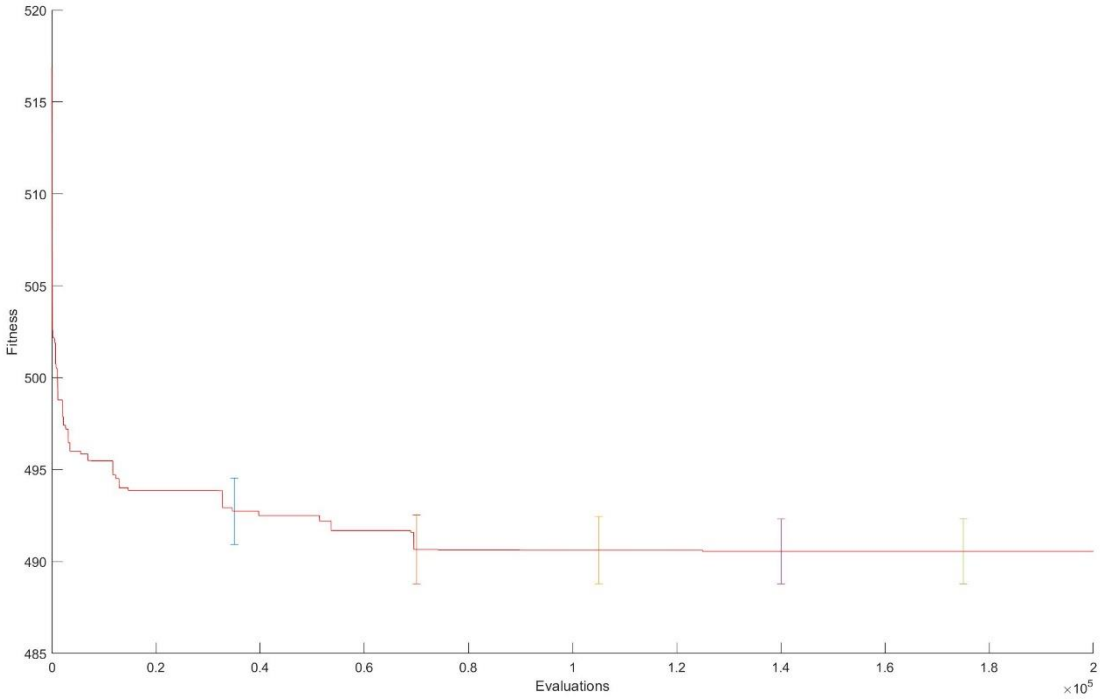

Fig.3 Fitness of Shortest Path after 200000 Times Sampling

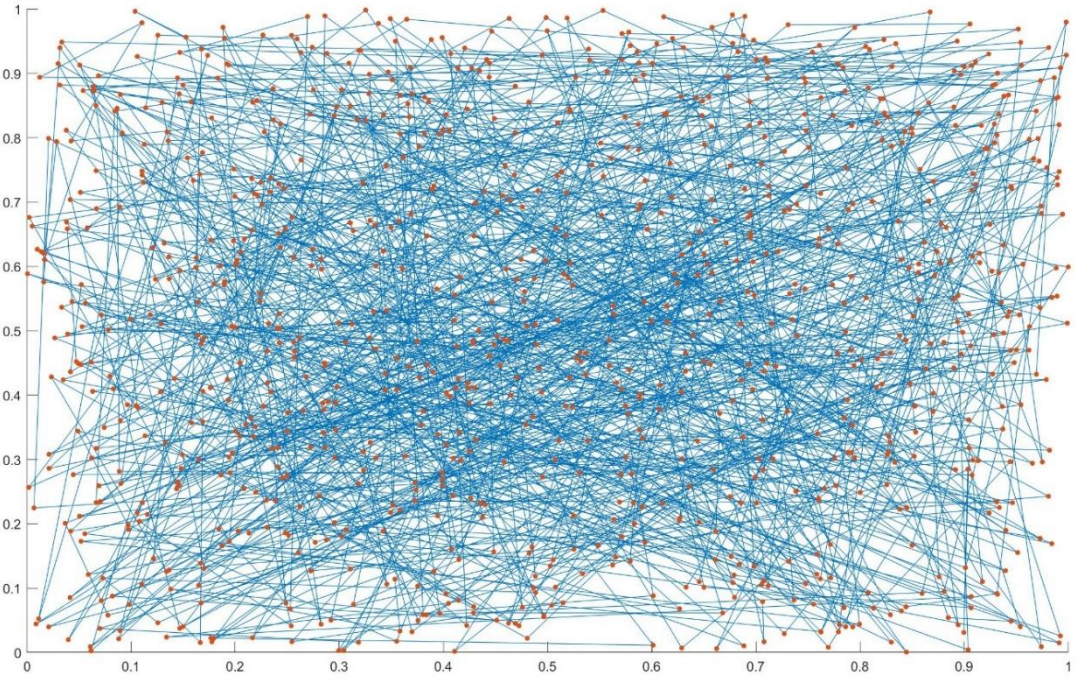

Fig.4 Shortest Path of Traveling after 200000 Times Sampling

2.2 Sampling 1000000 times

The learning curves is shown in Fig.5 which is averaged on five runs. The smallest fitness in figure is 486.455159, and the standard deviation in this point is 2.227728.

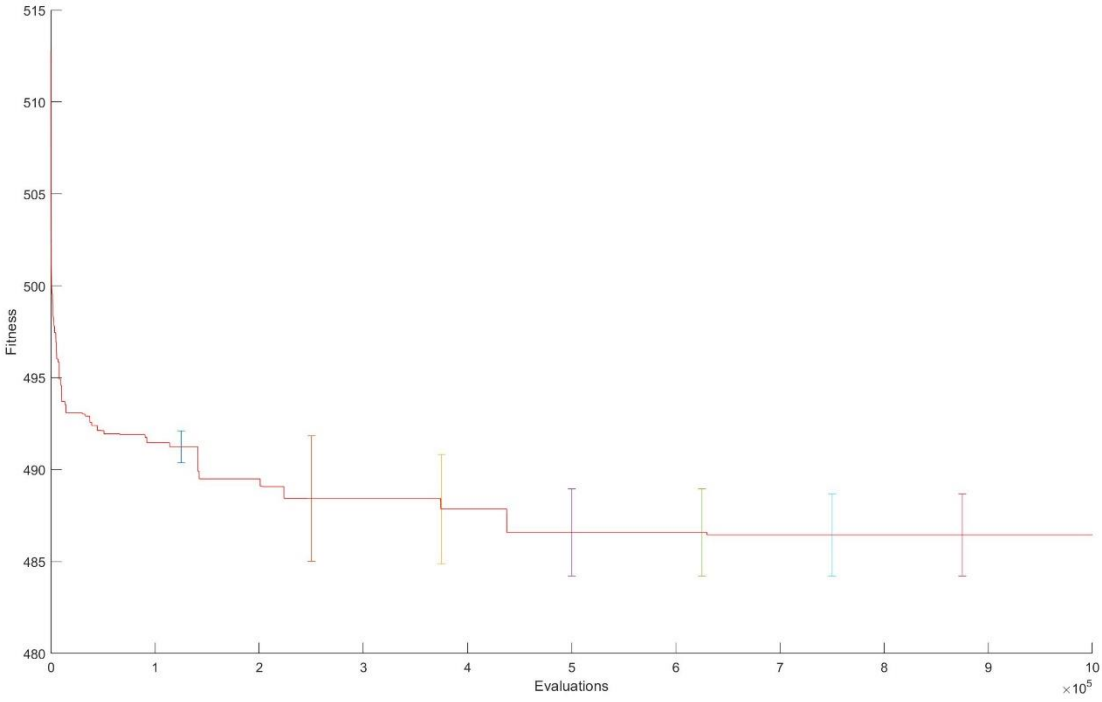

Fig.5 Fitness of Shortest Path after 1000000 Times Sampling

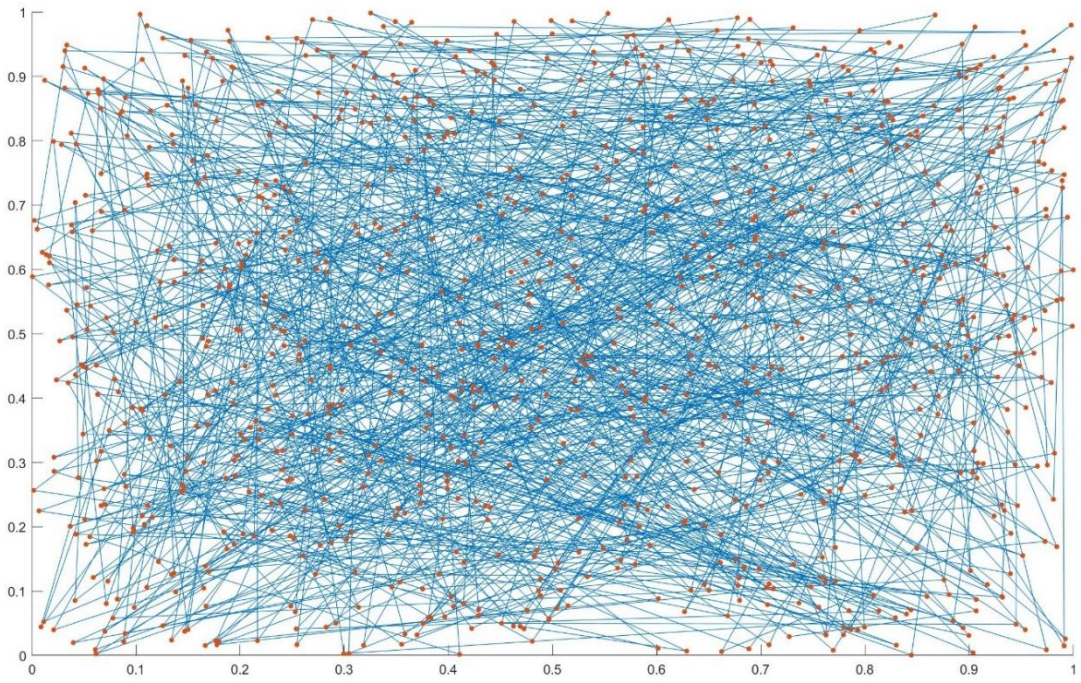

Fig.6 Shortest Path of Traveling after 200000 Times Sampling

3. Longest Path

3.1 Sampling 200000 times

The learning curves is shown in Fig.7 which is averaged on five runs. The biggest fitness in figure is 552.746073, and the standard deviation in this point is 1.754542.

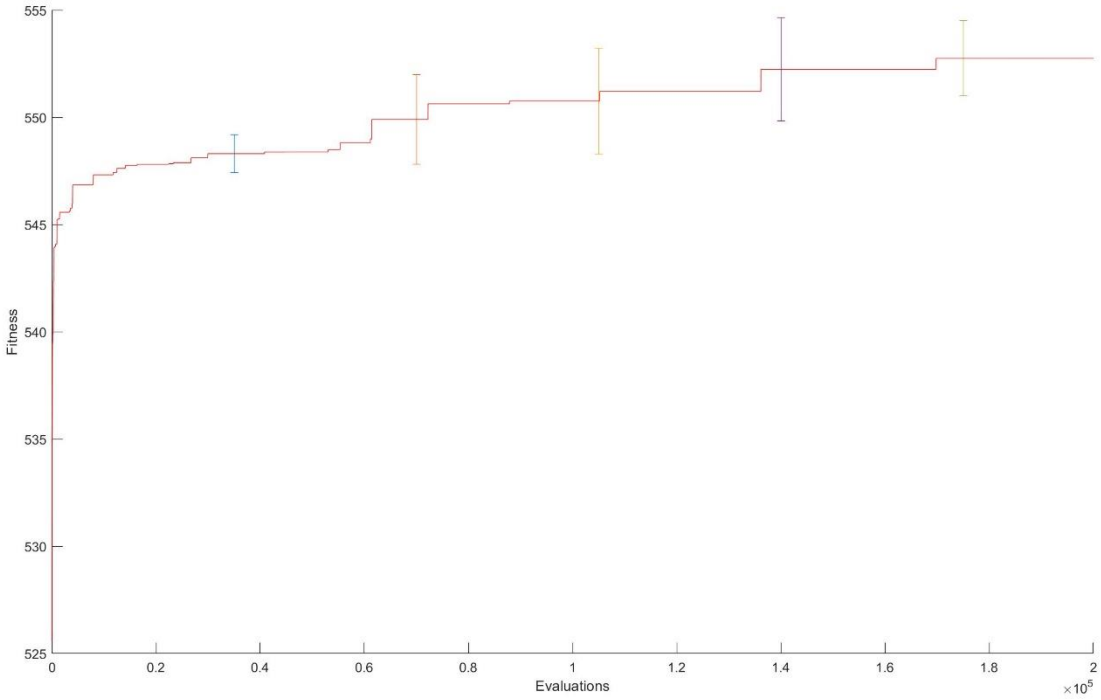

Fig.7 Fitness of Longest Path after 200000 Times Sampling

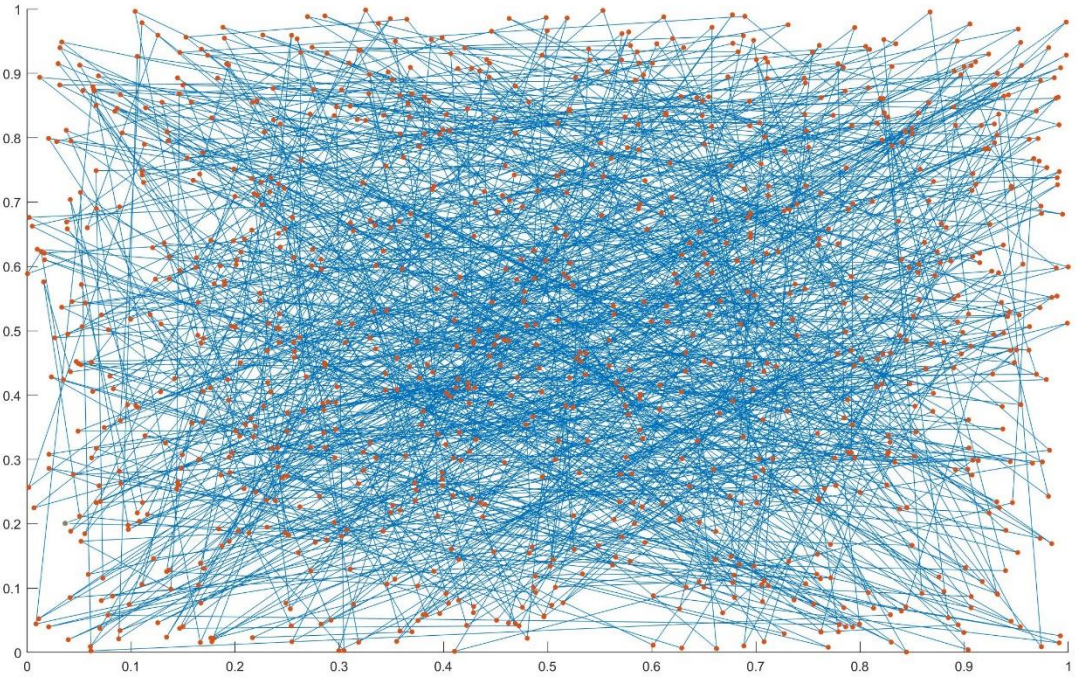

Fig.8 Longest Path of Traveling after 200000 Times Sampling

3.2 Sampling 1000000 times

The learning curves is shown in Fig.9 which is averaged on five runs. The biggest fitness in figure is 557.264124, and the standard deviation in this point is 2.300446.

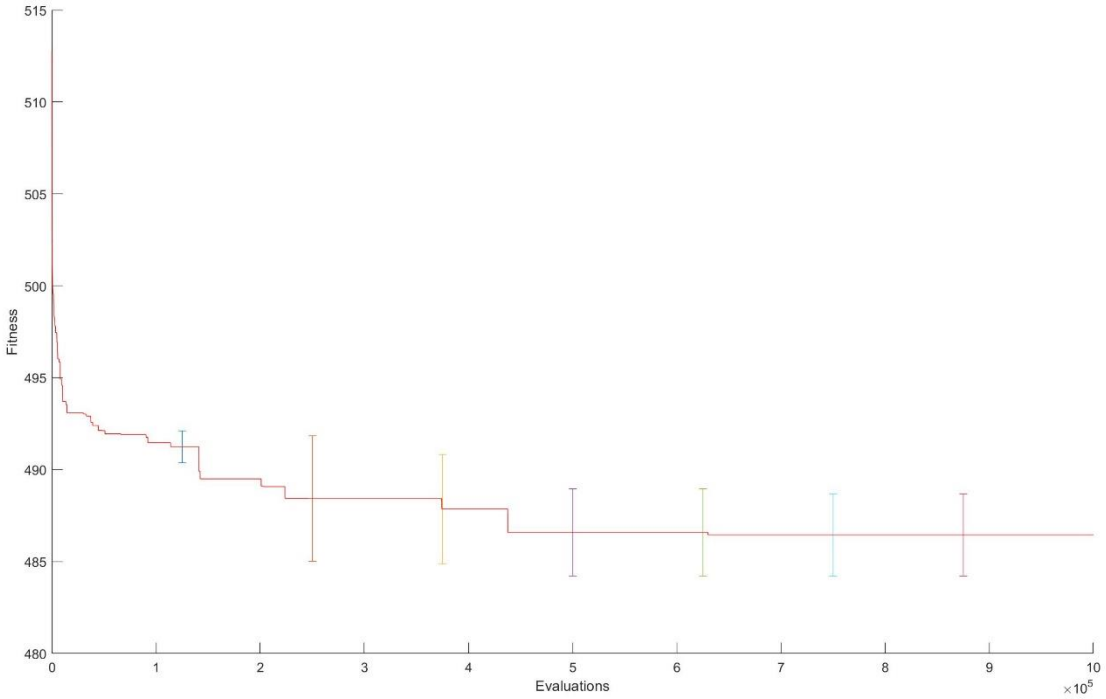

Fig.9 Fitness of Longest Path after 1000000 Times Sampling

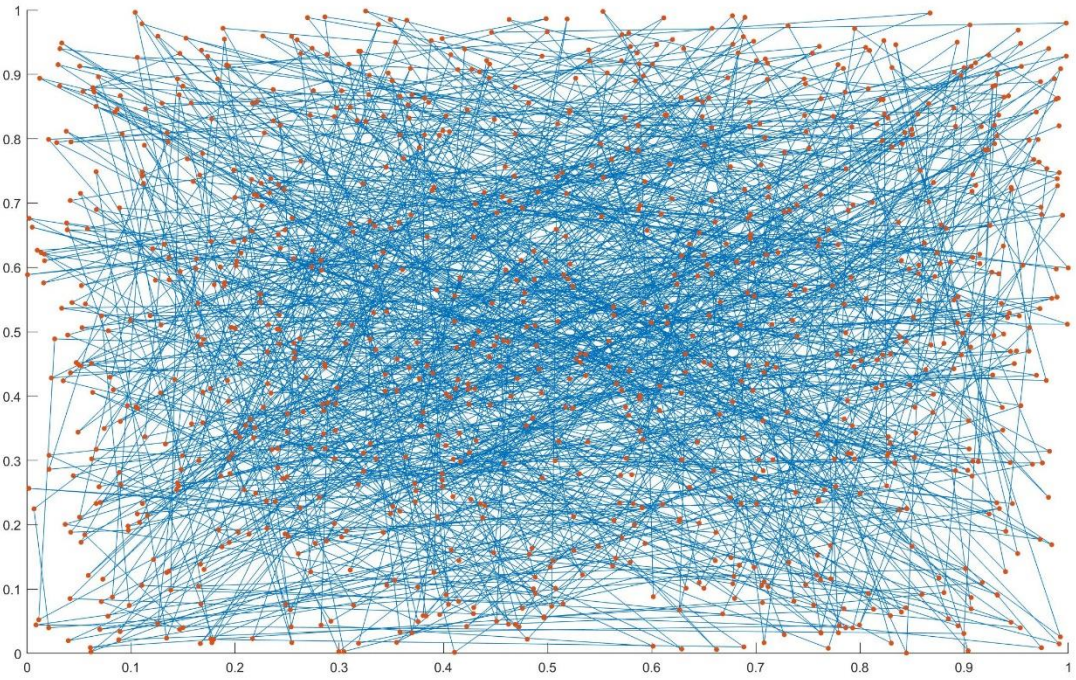

Fig.10 Longest Path of Traveling after 1000000 Times Sampling

4. Summary

After 1 million times processing, the shortest path which this method is able to find is 483.357921, the mean value of the shortest path is 486.455159 in five time runs, and the longest one is 556.3131, the mean value of the longest path is 486.455159 in five time runs.

In the Fig.3 and Fig.7, it is known that the best fitness is decreasing or increasing with the growth of sampling times. It means the random search method is able to find a better fitness. In former times, the fitness increases fast. It indicates that to find a fitness better than the original one randomly is quite easy. However, with the growth of sampling times, it is much harder to find a better path, sometimes it takes thousands of times to appear a new fitness. This phenomenon illustrates that this method is not "**evolving**". The effect of this method is declining with time. To find a much better path, it requires a great number of sampling to process which takes a lot of time. Comparing Fig.3, Fig.5, the best fitness in Fig.5 is lower than the former one. The more times this method loops, the better the fitness.

Besides, the standard deviation of best fitness in each figure is less than 2.5. In Fig.5 the mean value of fitness is 486.455159, and the standard deviation is 2.227728, which is only 0.475558% of mean value. It means the outcomes are stable in different process of running.

The path figures are all similar, which means the results of this method are not obvious and outstanding. However, comparing Fig.10 and Fig.6, the middle of Fig.10 is looks denser than the Fig.6's.

Appendix

```
#include <iostream>
#include <fstream>
#include <vector>
#include <chrono>
#include <random>
#include <ctime>
#include <iomanip>
using namespace std;

#define POINT_NUM 1000          //1000 random points
#define GENERATION 1000000      //the number of sampling

//declear sub function:
void read();
vector<int> Random_list();
double Fitness(vector<int> Ind, double x_coord[], double y_coord[]);
vector<int> Operator();

// global variable:
double xlist[POINT_NUM] = { 0 };
double ylist[POINT_NUM] = { 0 };
double fitnesslist[GENERATION] = { 0 };

int main()
{
        vector<int> Index;
        vector<int> BestOrder;


        Index.clear();
        BestOrder.clear();

        read();                     //read data from tsp.txt
        BestOrder = Operator();   //output the order of points which has best fitness

        ofstream outputoder("order_in_1000000_big.txt");     //import order into a txt file

        for (int i = 0; i < POINT_NUM; i++)
                outputoder << setprecision(9) << BestOrder[i] << endl;

        outputoder.close();

        ofstream outputfitness("fitness_in_1000000_big.txt");  //import fitness into a txt file

        for (int i = 0; i < GENERATION; i++)
                outputfitness << setprecision(9) << fitnesslist[i] << endl;

        outputfitness.close();

        ofstream outputpoint("point_in_1000000_big.txt");      //import points into a txt file

        for (int i = 0; i < POINT_NUM; i++)
        {
                //make sure the precision:
                outputpoint << setprecision(9) << xlist[BestOrder[i]] << "   ";
                outputpoint << setprecision(9) << ylist[BestOrder[i]] << endl;
        }
        outputpoint.close();

        return 0;
}

//sub function :
//read data from tsp.txt
void read()
{
        int i = 0;
        ifstream input("tsp.txt");

        double a, b;
        while (input >> a >> b) {
                xlist[i] = a;
                ylist[i] = b;
                i++;
        }
        input.close();
```

```
}

//shuffle a list from 0 to 999 randomly
vector<int> Random_list()
{
	vector<int> randomlist;
	for (int i = 0;i < POINT_NUM; i++)
		randomlist.push_back(i);		//a list from 0 to 999

	//shuffle the list:
	unsigned seed = (unsigned)chrono::system_clock::now().time_since_epoch().count();
	shuffle(randomlist.begin(), randomlist.end(), default_random_engine(seed));

	return randomlist;
}

//calculate the fitness of different order
double Fitness(vector<int> Ind, double x_coord[], double y_coord[])
{
	double fitness = 0;
	double X_distance;
	double Y_distance;

	//the fitness is the distance between each point in order
	for (int i = 1; i < POINT_NUM; i++)
	{
		X_distance = double(x_coord[Ind[i]]) - double(x_coord[Ind[i - 1]]);
		Y_distance = double(y_coord[Ind[i]]) - double(y_coord[Ind[i - 1]]);
		fitness += sqrt(X_distance * X_distance + Y_distance * Y_distance);
	}
	X_distance = double(x_coord[Ind[0]]) - double(x_coord[Ind[Ind.size() - 1]]);
	Y_distance = double(y_coord[Ind[0]]) - double(y_coord[Ind[Ind.size() - 1]]);
	fitness += sqrt(X_distance * X_distance + Y_distance * Y_distance);
	return fitness;
}

//choose the best order and its fitness
vector<int> Operator()
{
	double fitness;
	double BestFitness;
	vector<int> Ind, BestInd;

	Ind = Random_list();
	BestFitness = Fitness(Ind, xlist, ylist);
	BestInd = Ind;

	for (int i = 0; i < GENERATION; i++)
	{
		Ind = Random_list();
		fitness = Fitness(Ind, xlist, ylist);

		if (fitness >= BestFitness)
		{
			BestFitness = fitness;
			BestInd.assign(Ind.begin(), Ind.end());
		}
		fitnesslist[i] = BestFitness;

	}
	return BestInd;
}
```