**Solving Travelling Sales Man problem with Genetic Algorithm**

Xin Zheng

[Computer Engineer and Computer Science]

Speed School of Engineering

University of Louisville, USA

[x0zhen07@louisville.edu](mailto:x0zhen07@louisville.edu)

1. **Introduction**

This article provides a solution with Genetic Algorithm (GA) [1] to solve ‘Travelling salesman problem’ (TSP). As we know [2]TSP asks the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city once and returns to the origin city. There are 100 cities in this project, coordinates of x and y are supplied for each city. Base on above information, I need find out a good path with short distance.

1. **Approach**

**2.1 Tool and software in this paper**

* Programming language: Python3.7
* Excel Sheet for mac: 16.161.

**2.2 Algorithms**

**Genetic Algorithm**

The genetic algorithm is a method based on natural selection, which is one of the basic mechanisms of evolution, along with mutation, migration, and genetic drift. GA repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.

The steps for genetic algorithm are as below:

1. Initialize a Population with certain number of Individuals randomly, the certain number in this report is ‘Max\_city\_amount’ = 30.
2. Calculates fitness for each individual. Here the individual\_fitting\_rate = 1/ distance of the path.
3. Select individuals which meet the ‘natural selection rate’ as parents and put them in a new list.

‘natural\_selection\_rate’ = (1/max\_city\_amount)\* total\_fitting\_rate

‘total\_fitting\_rate’= sum(individual\_fitting\_rate) for the whole population.

1. Random pick up two parents from new list, then generate two children with crossover rule.
2. Run mutation rule on the two children and check whether the child is greater than both parents or not. The condition is if the distance of child is shorter then both of the parents, when the condition is met, the child can be add to a parent in a new population for the next generation.
3. Keep on seletion, crossover and mutation till there are 30 parents in the new population.
4. Check if ‘iteration\_number’ is meet ‘max\_iteration = 30’, if yes, iterations stop, otherwise, go back to step 2.

My idea is mainly from webpage below:

<https://blog.csdn.net/suitingwei/article/details/45767667>

**3. Data**

In this project, there is one city list with 100 cities.

The format is ‘.tsp’ which is a txt file. [Tap here for details about document format.](https://wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/)

Here is an example of list with 4 cities.

NAME: concorde4

TYPE: TSP

COMMENT: Generated by CCutil\_writetsplib

COMMENT: Write called for by Concorde GUI

DIMENSION: 4

EDGE\_WEIGHT\_TYPE: EUC\_2D

NODE\_COORD\_SECTION

1 87.951292 2.658162

2 33.466597 66.682943

3 91.778314 53.807184

4 20.526749 47.633290

**4.1 Result**

**4.1.1 Step1, analyze four datasets.**

According to the teacher’s request to analyze the results of two different settings for two different parameters. I set up crossover1 and crossover2, mutation1 and mutation2. And get 4 datasets as below.

|  |  |  |
| --- | --- | --- |
|  | Crossover 1 | Crossover 1 |
| Mutation 1 | Dataset 1A | Dataset 1B |
| Mutation 2 | Dataset 2A | Dataset 2B |

Here is the code for crossover and mutation.

def crossover(ind1,ind2,rate):  
 inde1 = random.randint(0, len(cities)-rate)  
 inde2 = inde1 + rate  
 segment1 = list(ind1[inde1:inde2])  
 segment2 = list(ind2[inde1:inde2]

#find out same element in the two segement  
 same\_element = list(set(segment1).intersection(segment2))

#remove same elements in two segment  
 seg1 = [elem for elem in segment1 if elem not in same\_element ]  
 seg2 = [elem for elem in segment2 if elem not in same\_element ]  
 mapindex1 = [list(ind1).index(i) for i in seg2]  
 mapindex2 = [list(ind2).index(j) for j in seg1]  
 ind1[inde1:inde2] = list(segment2)  
 ind2[inde1:inde2] = list(segment1)  
 for i in range(0, len(seg1)):  
 ind1[mapindex1[i]] = seg1[i]  
 ind2[mapindex2[i]] = seg2[i]  
 return ind1,ind2  
  
#crossover with rate 30  
def crossover1(ind1,ind2):  
 crossover\_rate = 30  
 ind1,ind2 = crossover(ind1,ind2,crossover\_rate)  
 return ind1,ind2  
  
# crossover with rate 10  
def crossover2(ind1, ind2):  
 crossover\_rate = 10  
 ind1,ind2 = crossover(ind1,ind2,crossover\_rate)  
 return ind1,ind2  
  
# reverse 3 numbers in individual  
def mutation1(ind1,ind2):  
 reverse\_rate = 3  
 inde1 = random.randint(0, len(cities) - reverse\_rate)  
 inde2 = inde1+ reverse\_rate  
 ind1[inde1:inde2] = ind1[inde1:inde2][::-1]  
 ind2[inde1:inde2] = ind2[inde1:inde2][::-1]  
 return ind1,ind2  
  
#random select 4 numbers to mutate  
def mutation2(ind1,ind2):  
 mutation\_rate = 4  
 index = []  
 index = random.sample(range(0, len(cities)), mutation\_rate)  
 ind1[index[0]],ind1[index[1]] = ind1[index[1]],ind1[index[0]]  
 ind1[index[2]],ind1[index[3]] = ind1[index[3]],ind1[index[2]]  
 ind2[index[0]],ind2[index[1]] = ind2[index[1]],ind2[index[0]]  
 ind2[index[2]],ind2[index[3]] = ind2[index[3]],ind2[index[2]]  
 return ind1,ind2

At first, I run four crossover and mutation rules to the first generation, 30 individules, and obtain 4 datasets with 120 individuals. After that, iterative run each dataset with corresponding crossover and mutation rule 30 times. That is one iteration. In this report, I the output are from 30 iteration. The average running time for each iteration is 443.2597806 seconds, and the cost of best path is 2174.176624.

GRAPH1

GRAPH1 is the output of best path for 30 iterations, the cost is between 2200 and 2800. The blue column, which is dataset2b, and green column, which is dataset1b, look shorter in most iterations. The purple column, which is dataset 2a, looks higher in most iterations. To approve my guess, I make a histogram according to the frequency of best path. In each iteration, I select dataset which the best path is in, and calculate the frequency.

GRAPH2

According to the histogram, dataset 1b and datset2b generate more best path, dataset1a and dataset 2a generate much less best paths. From this point of view, we can make a conclusion that ‘crossover1’ and ‘crossover2’ did not affect the output a much, even the ‘crossover\_rate’ in the two crossover rule has big gap. In the other hand, ‘mutation2’ influences the output significantly.

TABLE1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **item** | **dataset1a** | **dataset1b** | **datset2a** | **dataset2b** |
| **min** | 2178.72214 | 2256.74031 | 2315.26404 | 2174.17662 |
| **max** | 2801.77322 | 2654.16802 | 2748.47244 | 2598.89757 |
| **average** | 2482.43083 | 2406.82655 | 2569.91849 | 2421.87063 |
| **SD** | 129.657841 | 98.3687651 | 113.477844 | 116.271303 |

TABLE1 is the minimum, maximum, average, and standard deviation of best costs for each dataset in 30 iterations. The best path is in dataset 2b, which is with cost 2172.17662. However, the standard deviation of dataset1b is the smallest, which denotes the gap among all best cost for dataset1b is the smaller than other datasets. According to GRAPH1, 11 best paths are generated by dataset1b, 13 paths are from dataset 2b, the difference are not obviously. Therefore, I will choose dataset 1b, with crossover1 of crossover\_rate = 30 and mutation2 of random mutating 4 numbers, as the best solution.

**4.1.2 Generate the best path with dataset1b in 80 iterations.**

As I got the conclusion that dataset 1b works best, now I will run dataset1b solely with different parameters. In test 2, I will change below parameters:

max\_iteration = 80 # limit iteration number,it was 30 in last test

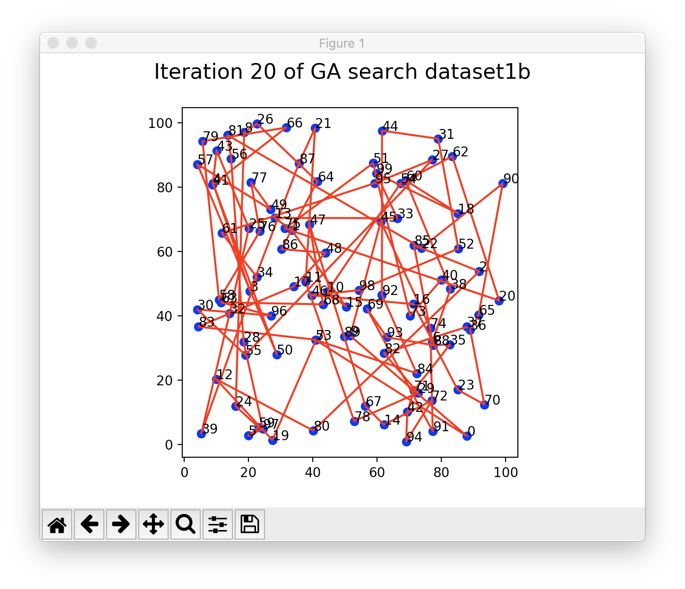
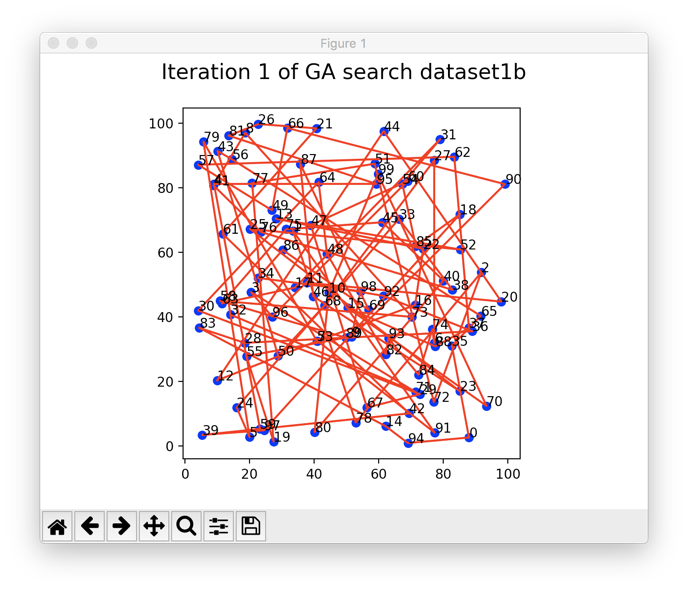
max\_city\_amount = 50 # number of individual in population,it was 30 in last test.

GRAPH 3

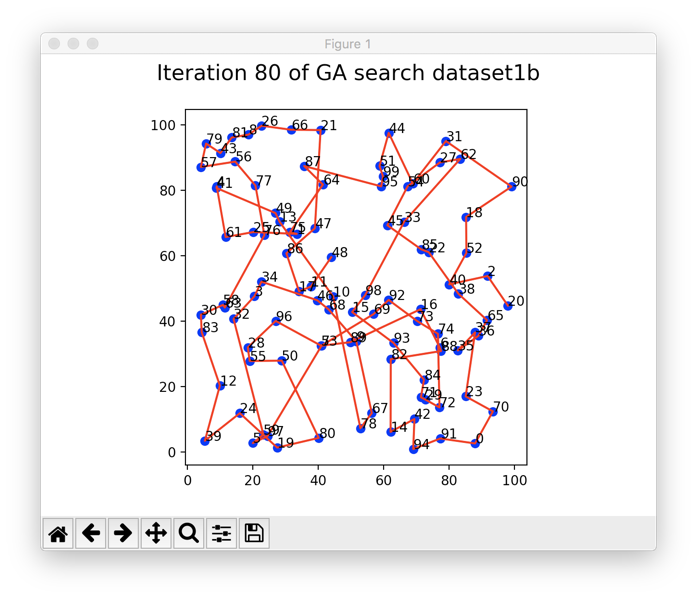
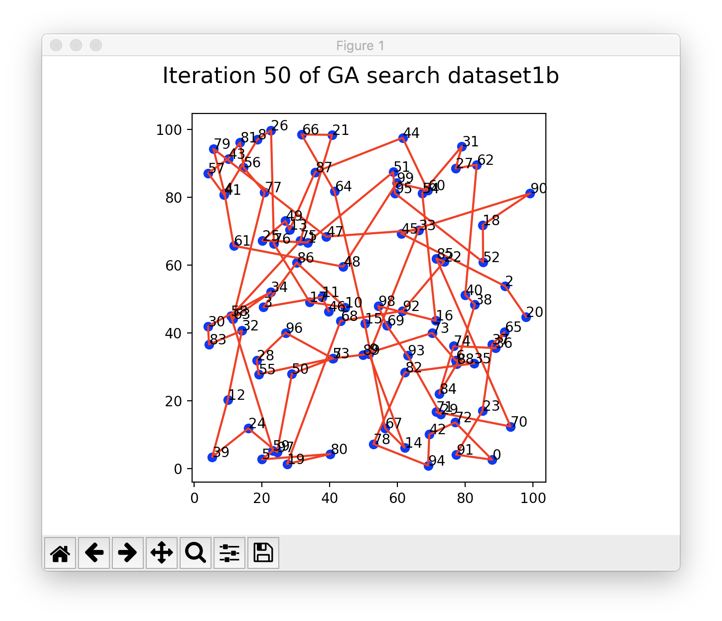
GRAPH 3 is the cost vs iteration number for test2, as the iteration number increase, the cost decrease.

Below are some outputs.

Iteration 1, distance =4351.116537386776. Iteration 20, distance = 2901.902339442244



Iteration 50, distance = 1770.1442451044722. Iteration 80, distance is 1232.0046860583263



Obviously, the figures visualize that as the iteration number increase, the cost is decreases.

**5. Discussion**

**5.1 Difficult part in GA**

The most difficult part of GA is how to make a crossover without duplicate elements. After picking the segments to crossover, we need do below steps:

1. Find out the same elements in the two segments, which is ‘same\_elements’ in this project.
2. Remove same elements in two segments, and get ‘seg1’, ‘seg2’
3. Map the location or index number for ‘seg1’ and ‘seg2’ from individual2 and individual1 then obtain mapindx1 and mapind2
4. Run crossover for each individual;
5. Put elements in ‘seg1’ to individual2 with index from mapindex2, and put ‘seg2’ to individual1 with mapindex1.

**5.2 Conclusion**

The concept of Genetic algorithm is very easy to understand; GA works well when there is a large numbers of parameters and the objective function is stochastic. GA is kindly of a last resort, I will use that only when I have a lot of CPU time in my hand. For solving travelling sales man’s problem, I prefer greedy algorithm when I am in rush, since it takes less than 30 seconds to generate a solution of cost around 2500. Meanwhile, it takes 6957.617886 seconds to obtain a solution of cost around 1200 with GA. I will choose GA when cost is more important, and I can wait for nearly 2 hours to get the solution.

Due to the time limit to run 4 datasets for a big number of times, the parameter of max iteration number and the number of individuals in a population of this project is only 30. I will change above parameters to 50 in the further to get more precise result.

**6.References**

1. What is Genetic Algorithm

<https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html>

1. What is travelling sales man problem?

<https://en.wikipedia.org/wiki/Travelling_salesman_problem>

**7.Appendix**

1. Raw data for two tests:

Project4\_rawdata.xlsx

1. Codes with genetic algorithm

project4\_Zheng.py