Genetic Algorithm for Solving Traveling Salesman Problem

Dongdong Geng-183139

August 3, 2019

Abstract

The traveling salesman problem(TSP) is a combinatorial optimization problem with important practical significance. Genetic algorithm is one of the typical algorithms for solving traveling salesman problems. This paper first introduces the definition of traveling salesman problem and common algorithms, and elaborates the principle of genetic algorithm. In this paper, the reciprocal of the total length of the path is used as the fitness function to ensure that the understanding develops toward the optimization direction. Then select the sequential crossover operator to generate new individuals, which ensures the efficiency of the iteration. The mutation operator makes the algorithm simple and easy to use. Finally, two practical problems were solved when the number of cities was 10 and 20. Finally, the characteristics of the genetic algorithm are analyzed, and the direction of the improved algorithm is proposed.

1 Introduction

The traveling salesman problem is a famous and classic problem in the study of optimal combination problems, and its research value is self-evident. There are practical applications for the traveling salesman problem, such as in traffic management planning. The main purpose of traffic management is to optimize the route

of the vehicle in a complex geographic network structure to reduce transportation costs. Examples of application of the traveling salesman problem include: designing a safe, reasonable and efficient transportation network to reduce traffic congestion; how to plan better logistics routes, reduce the costs incurred by the operating industry, and reduce resource consumption.

Since the traveling salesman problem has such important practical significance, the algorithm for solving the traveling salesman problem will also develop. The earliest solution was linear programming, which later produced a variety of algorithms for solving traveling salesman problems. They can be roughly divided into precise algorithms, approximation algorithms, and intelligent algorithms. Precise algorithms: linear programming, dynamic programming, branch and bound; approximation algorithms include: interpolation, Clark & Wright algorithm, spanning tree algorithm, nearest neighbor algorithm, probability algorithm, etc.. Many new intelligent algorithms have emerged in recent years. Such as simulated annealing, ant colony algorithm and genetic algorithm.

This article describes what a traveling salesman problem is, and then briefly describes some of the algorithms currently used to solve the traveling salesman problem. After that, it introduces in detail what is the genetic algorithm and the implementation process of the genetic algorithm. Then it describes how to use the genetic algorithm to solve the traveling salesman problem and actually applied it.

2 Literature Review

In the 1960s, Professor J.H. Holland of the University of Michigan began to recognize the similar relationship between biological genetic and natural evolution phenomena and artificial adaptive systems, and introduced biological genetic mechanisms into the research of artificial adaptive systems. In 1967, his student Bagley J.D. first proposed the term ”genetic algorithm” and developed genetic operators such as replication, crossover, mutation, dominant, and inversion. In the 1970s, Professor Holland published the monograph ”Adaptation in Nature and Artificial Systems”, systematically expounded the basic theory and method of genetic algo-

rithm, and proposed the basic theorem of genetic algorithm - the pattern theorem, which laid a theoretical foundation for the development of genetic algorithm. Based on the idea of genetic algorithm, Dr. De. Jong of the United States conducted a large number of pure numerical function optimization calculation experiments on the computer and established the famous De.Jong five function test platform. In the 1980s, genetic algorithms began to be widely used in various fields. In 1980, Smith applied genetic algorithms to the field of machine learning; Bethke systematically studied the genetic algorithm of function optimization. In 1989, David Goldberg published "Genetic Algorithm in Search, Optimization and Machine Learning", which became the first monograph on genetic algorithms. In the 1990s, genetic algorithms entered a period of prosperity, and both theoretical research and applied research have become very hot topics. In particular, the application research of genetic algorithms is particularly active, and is widely used in the fields of combinatorial optimization, production scheduling, machine learning, image processing, artificial life, etc. Moreover, the ability to use genetic algorithms for optimization and rule learning is also significantly improved.

3 Traveling Salesman Problem

3.1 Definition of Traveling Salesman Problem

Traveling salesman problem(TSP) is an ancient problem in combinatorial mathematics. Although it is easy to describe, it has not been completely solved so far. It has been summarized as the NP complete problem. The classic formulation is as follows: the salesman wants to bring a pile of goods. Some places that you know sell goods. After starting from the starting place, you need to travel the rest of the place and then return to the starting place. Ask the salesman what route to choose to make the total distances the shortest (distance between cities is known).

Assuming that the number of cities is n, then the number of combined paths is $(n-1)!$. In the case of a small number of cities, it is obviously not difficult to find the best route. However, when the number of cities is large, the number of

combined routes will be exponentially long, which will be uncalculated, which is called the combined explosion problem. Now suppose that the number of cities is 20, so the number of combined paths is $(20-1)!$, the number of combinations is quite large. Even with computer calculations, it takes 386 years to calculate the speed of retrieving 10 million lines per second.

3.2 Mathematical Description

TSP is described in mathematical language as finding a path to travel and satisfies the following objective function:

$$f(V) = \min \sum_{i=1}^{n-1} d\left(v_i, v_{i+1}\right) + d\left(v_n, v_1\right) \tag{1}$$

Where v_i is the city number $v_i \in N, 1 \leq v_i \leq n$, $d(v_i, v_j)$ represents the weight between the city i and the city j (may represent: distance, time and other expenses), if it is a symmetric TSP, then there is $d(v_i, v_j) = d(v_j, v_i)$.

Let $G = (V, E)$ be the weighted complete graph, $V \in N$ denote the vertex set, and E denote the edge set.$(d_{ij} > 0, d_{ii} = 0, i, j \in N)$

Suppose x_{ij} is equal to 0 or 1.

$$x_{ij} = \begin{cases} 1, (i,j) \in L \\ 0, (i,j) \notin L \end{cases} \tag{2}$$

L is the solution sequence, then the mathematical model of TSP can be written as the following linear programming form:

$$\min D(L) = \sum_{i \neq j} d_{ij} x_{ij} \tag{3}$$

$$\sum_{i \in V, j \neq i} x_{ij} = 1 \tag{4}$$

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \tag{5}$$

$$\sum_{i,j \in V} x_{ij} = |L| \quad L \subset E \tag{6}$$

Where $|L|$ is the number of elements in the set L. Constraint (4) (5) constrains each vertex to have one and only traversal. Constraint (6) constrains Hamilton to

do not have any sub-loops, and finds the one with the smallest weight among all Hamiltons that satisfy the condition.

4 Genetic Algorithm

4.1 Genetic Algorithm Description

Genetic algorithm is a random parallel search algorithm based on natural selection and genetic genetics. It is an efficient global optimization algorithm that does not consider initial information. The algorithm considers the solution set as a population. Using a selection operation, a cross operation, a mutation operation, etc., an excellent solution is finally obtained. It has global optimization ability, strong adaptability, simple calculation process, strong robustness on nonlinear problems, no specific restrictions on problems, low requirements on search space, simple operation combined with other algorithms, etc.. It has a wide range of applications in optimization functions, image processing, automatic control, economic forecasting and engineering optimization, and has a wide range of applications in solving NP-hard problems.

4.2 Genetic Algorithm Steps

Goldberg summarizes a unified and basic genetic algorithm and becomes the Basic Genetic Algorithms (SGA). SGA only uses three kinds of genetic operators: selection operator, crossover operator and mutation operator. The structure is simple and easy to understand. It is the dimension and foundation of other genetic algorithms. The basic process of SGA is shown in Figure 1.

The specific algorithm steps are as follows:

Step1: Initialize to produce an initial population;

Step2: Individual evaluation, that is calculating the fitness of each individual in the population, and judging whether the optimization criteria are met. If it is met, output the optimal solution or the satisfactory solution of the best individual and its representative, stop; otherwise, go to Step3;

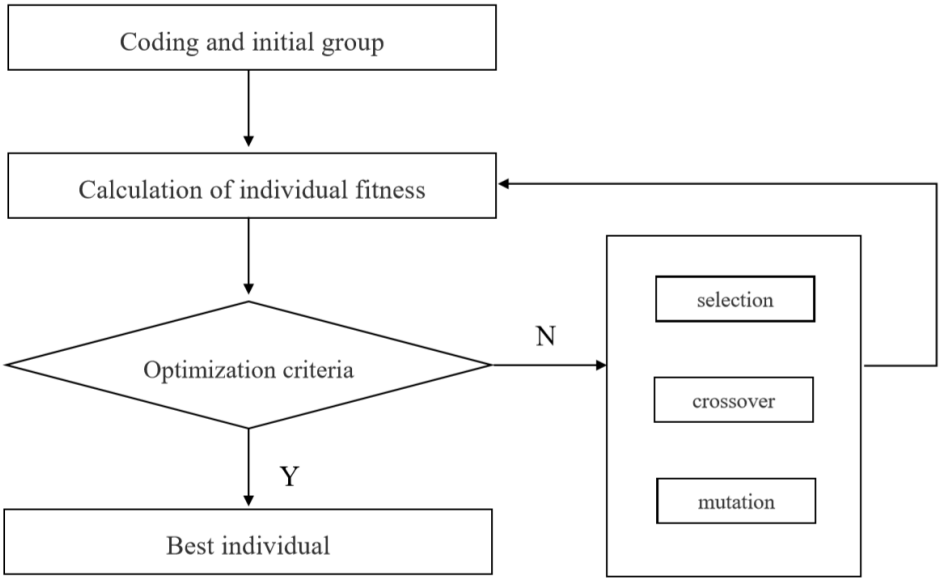


Figure 1: Genetic algorithm

Step3: Perform a selection operator according to the selection probability p_s, and select some individuals from the current population to enter the next generation population;

Step4: Perform a crossover operator according to the crossover probability p_c;

Step5: Perform a mutation operator according to the mutation probability p_m;

Step6: Generate a new generation of populations from crossovers and mutations, and turn to Step 2.

The optimization criteria of genetic algorithms generally have different determination methods depending on the problem. Generally, the following criteria can be used as the judgment conditions: (1)Achieved a predetermined evolutionary algebra; (2)The optimal individuals in the population have not improved after several consecutive generations; (3)The optimal individual reaches a predetermined satisfactory solution.

5 Genetic Algorithm for Solving TSP

5.1 Coding

Using natural number coding, each natural number represents a city. For example, an individual's chromosome can be represented by 1 2 3 4 5 6 7 , representing a route from city 1 through 2 3 4 5 6 7 and finally back to city 1.

In this paper, the optimization of the initial individual is based on the greedy algorithm, which uses its advantage of local optimization to generate new individuals. Select a city in a random way as the current city C_0, join the individual, search for the remaining cities, find the nearest city C_1 from the current city C_0, add C_1 to the individual and then use C_1 as the current city., repeated searches until all cities are added to the individual to form a complete route. Using this strategy to generate initial populations is also random, and it also improves the overall quality and speeds up the optimization.

5.2 Fitness Function

The individual's badness of the population is evaluated by fitness. The greater the fitness value, the better the individual. The fitness function is represented by the reciprocal of the total distance of the paths between cities in the traveling salesman problem. The fitness function expression is as follows:

$$f(x) = \frac{1}{\sum_{i=1}^{n-1} d\left(C_i, C_{i+1}\right) + d\left(C_n, C_1\right)} \tag{7}$$

5.3 Selection Operator

The selection operation selects individuals from a determined probability in the original population to generate a new population. In order to preserve the excellent individuals in the group, this paper designs a new selection operator and adds the elite strategy. when the selection of the generation of the population is generated, the proportional selection operator is used first (The probability that each individual is selected is P. The upper generation $\alpha \times M$ individuals are selected.) The parent

populations are arranged in order of fitness size, and the $(1-\alpha)\times M$ individuals with better fitness are selected to insert the sub-populations, and finally M individuals of the offspring are generated, wherein α is a selection factor. The probability P of an individual being selected is:

$$P = \frac{f(x)}{\sum f(x)} \tag{8}$$

5.4 Crossover Operator

This paper uses a sequential crossover operator, It first randomly selects two intersections in the parent, and then exchanges the intersections. The other positions are determined according to the relative order of the cities in the parent.

$$1 \quad 2 \quad |3 \quad 4| \quad 5 \quad 6 \quad \times \quad 1 \quad 3 \quad |5 \quad 2| \quad 4 \quad 6$$

$$\downarrow$$

$$1 \quad 5 \quad |3 \quad 4| \quad 2 \quad 6$$

5.5 Mutation Operator

Although individual crossover increases population diversity, some good information may be lost, thus introducing a variation in fitness. The genetic position of the chromosome is changed correspondingly according to a certain probability. The mutation methods used in this paper include exchange mutation operators and in-position mutation operators. The specific operations are as follows:

1. Exchange mutation operator. Two city numbers are randomly generated first, and the two city numbers are interchanged. If 3 and 5 are randomly generated from $[1, 6]$, determine the city number to be exchanged.

$$1 \quad 3 \quad 6 \quad 5 \quad 4 \quad 2 \longrightarrow 1 \quad 5 \quad 6 \quad 3 \quad 4 \quad 2$$

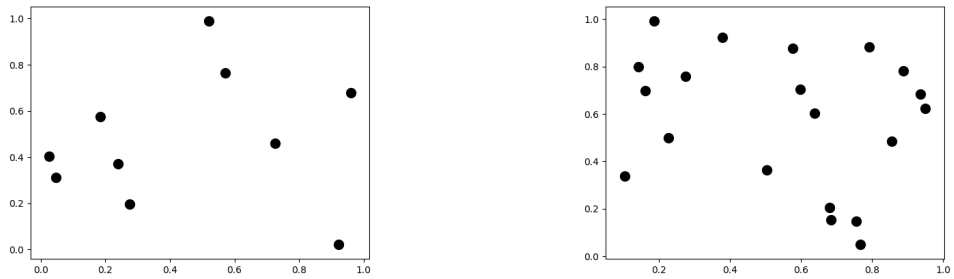
2. In-position mutation operator. Randomly determine the two positions, and then reverse the order of the city numbers between the two positions. If positions 3 and 5 are generated from [1, 6].

$$1 \quad 3 \quad 6 \quad 5 \quad 4 \quad 2 \longrightarrow 1 \quad 5 \quad 4 \quad 3 \quad 6 \quad 2$$

6 Case Study

6.1 Problem Statement

In this paper, the genetic algorithm is used to solve the traveling salesman problem with the number of cities 10 and 20. The specific location of the city is shown in Figure 2.

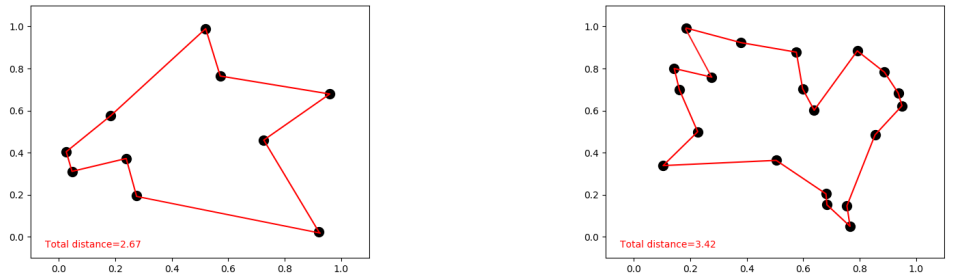


(a) The number of cities is 10 (b) The number of cities is 20

Figure 2: Specific location of the city

6.2 Result and Discussion

Use genetic algorithm to find the path as shown in Figure 3. The total distance of walking and the speed of iterative convergence of genetic algorithm are shown in Table 1.



(a) The number of cities is 10 (b) The number of cities is 20

Figure 3: Traveler walking path

Table 1: Calculation results

Number of cities	Total distance	Number of iterations
10	2.67	11
20	3.42	53

It can be seen from the table that the genetic algorithm solves the traveling salesman problem quickly. When the number of cities is 10, 11 iterations can find the solution to the problem. When the number of cities is 20, the number of iterations of the algorithm is 53. It is conceivable that when the number of cities increases, the number of times the algorithm needs to converge will increase sharply.

7 Conclusion

This paper briefly summarizes the methods to solve the traveling salesman problem and introduces the steps of the genetic algorithm. The coding, fitness function setting and various operator design when using the genetic algorithm to solve the traveling salesman problem are introduced in detail. Finally, the experiment shows that the genetic algorithm can solve the traveling salesman problem quickly. However, genetic algorithm tends to converge to local optimal solution, how to improve crossover operator, mutation operator and how to improve fitness function to improve genetic algorithm to make it iteratively to global optimal solution is the research direction of genetic algorithm to solve traveling salesman problem.

References

[1] Sun Hailei. *An Improved Genetic Algorithm for TSP*[D]. Chongqing University. 2007

[2] Wang Na. *Improved Genetic Algorithem for TSP*[D]. Chongqing University. 2007

[3] Zhang Zhanyun. *Research on Improved Genetic Algorithm Based on Traveling Salesman Problem*[J]. 2017(07):19-21.

[4] Yuan Hao. *Research and application of traveling salesman problem*[D]. Nanjing University of Posts and Telecommunications. 2017.

[5] Jiang Rong. *Genetic Algorithm and its application in Travelling Salesman Problem*[D]. Hefei University of Technology. 2009.

[6] Yu Fengrui. *Improved Genetic Algorithm fer TSP*[D]. Inner Mongolia University Of Technology. 2016.

[7] Cheng Rong. *Genetic algorithm for traveling salesman problem*[J]. 2017(16):40-51.

[8] Deng Lifang. *Application Research of Genetic Algorithm for the Travelling Salesman Problem*[D]. Guangdong University of Technology. 2012.

[9] Tan Jinhua. *Eyolutionary Algorithms for Traveling Salesman Problems*[D]. Xidian University of Electronic Technology. 2008.

[10] Wu Jun, Yan Lina. *Solving TSP Problem Based on the Improved Genetic Algorithm*[J]. 2017(03):96-98.

Please excuse me!![1]

[1]The reference format of the references is wrong because the use of LaTeXis unskilled

A Code

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  N_CITIES = 20  # DNA size
4  CROSS_RATE = 0.1
5  MUTATE_RATE = 0.02
6  POP_SIZE = 500
7  N_GENERATIONS = 500
8  class GA(object):
9      def __init__(self, DNA_size, cross_rate, mutation_rate, pop_size, ):
10         self.DNA_size = DNA_size
11         self.cross_rate = cross_rate
12         self.mutate_rate = mutation_rate
13         self.pop_size = pop_size
14
15         self.pop = np.vstack([np.random.permutation(DNA_size) for _ in
               range(pop_size)])
16
17     def translateDNA(self, DNA, city_position):     # get cities' coord
           in order
18         line_x = np.empty_like(DNA, dtype=np.float64)
19         line_y = np.empty_like(DNA, dtype=np.float64)
20         for i, d in enumerate(DNA):
21             city_coord = city_position[d]
22             line_x[i, :] = city_coord[:, 0]
23             line_y[i, :] = city_coord[:, 1]
24         return line_x, line_y
25
26     def get_fitness(self, line_x, line_y):
27         total_distance = np.empty((line_x.shape[0],), dtype=np.float64)
```

```
28          for i, (xs, ys) in enumerate(zip(line_x, line_y)):
29              total_distance[i] = np.sum(np.sqrt(np.square(np.diff(xs)) +
                    np.square(np.diff(ys))))
30          fitness = np.exp(self.DNA_size * 2 / total_distance)
31          return fitness, total_distance
32
33      def select(self, fitness):
34          idx = np.random.choice(np.arange(self.pop_size), size=self.
                pop_size, replace=True, p=fitness / fitness.sum())
35          return self.pop[idx]
36
37      def crossover(self, parent, pop):
38          if np.random.rand() < self.cross_rate:
39              i_ = np.random.randint(0, self.pop_size, size=1)
                                        # select another individual from
                    pop
40              cross_points = np.random.randint(0, 2, self.DNA_size).astype
                    (np.bool)   # choose crossover points
41              keep_city = parent[~cross_points]
                                                    # find the city
                    number
42              swap_city = pop[i_, np.isin(pop[i_].ravel(), keep_city,
                    invert=True)]
43              parent[:] = np.concatenate((keep_city, swap_city))
44          return parent
45
46      def mutate(self, child):
47          for point in range(self.DNA_size):
48              if np.random.rand() < self.mutate_rate:
49                  swap_point = np.random.randint(0, self.DNA_size)
```

```
                swapA, swapB = child[point], child[swap_point]
                child[point], child[swap_point] = swapB, swapA
        return child

    def evolve(self, fitness):
        pop = self.select(fitness)
        pop_copy = pop.copy()
        for parent in pop:  # for every parent
            child = self.crossover(parent, pop_copy)
            child = self.mutate(child)
            parent[:] = child
        self.pop = pop
class TravelSalesPerson(object):
    def __init__(self, n_cities):
        self.city_position = np.random.rand(n_cities, 2)
        plt.ion()

    def plotting(self, lx, ly, total_d):
        plt.cla()
        plt.scatter(self.city_position[:, 0].T, self.city_position[:,
            1].T, s=100, c='k')
        plt.plot(lx.T, ly.T, 'r-')
        plt.text(-0.05, -0.05, "Total distance=%.2f" % total_d, fontdict
            ={'size': 20, 'color': 'red'})
        plt.xlim((-0.1, 1.1))
        plt.ylim((-0.1, 1.1))
        plt.pause(0.01)
```