Genetic Algorithm for Solving Traveling Salesman Problem

ZHOU Zikang
56413041

1 Introduction

The Traveling Salesman Problem (TSP) is an NP-hard problem in combinatorial optimization. In this report, we try to use genetic algorithm (GA) to solve TSP and analyze the performance of GA. Figure 1 shows the procedure of a typical GA.

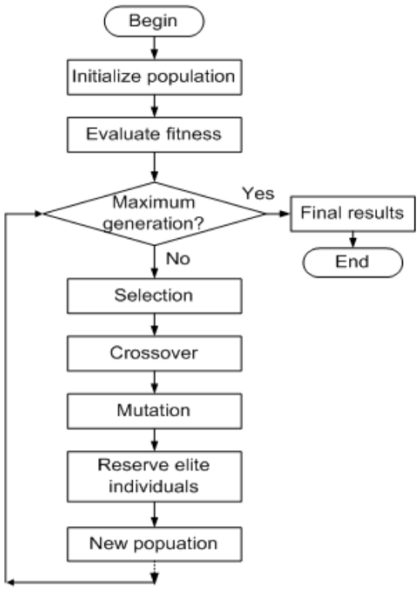

Fig.1: The procedure of a typical GA.

2 Traveling Salesman Problem

2.1 Problem Setup

TSP aims to figure out the shortest path for a salesman to visit n cities, given the constraints that the salesman must visit each city just once and should finish up where he started. In the original TSP, the distance from city A to city B is the same as that from city B to city A, i.e., the TSP is symmetric. We first consider the symmetric TSP. The objective function of symmetric TSP can be described as:

$$\min \sum_{i=1}^{n-1} d(i, i+1) + d(n, 1),$$

where $d(i, j)$ denotes the distance between city i and city j, and $d(i, j) = d(j, i)$.

2.2 Method

2.2.1 Representation

Suppose the IDs of the cities' range from 1 to n. Designing a representation for an individual in GA is quite straightforward in TSP: since each solution for TSP is a permutation of the cities, a chromosome can be represented as a permutation of the sequence ranging from 1 to n, where each element in the sequence denotes a gene.

2.2.2 Population Initialization

To initialize the population, we randomly permute the sequence $\{1, 2, \ldots, n\}$ to obtain an individual until reaching the population size.

2.2.3 Fitness Evaluation

In TSP, the objective is to minimize the total length of the path, so the fitness of a chromosome should be calculated as:

$$f = \frac{1}{c} ,$$

Where f is the fitness of a chromosome, and c is the total length of the path corresponding to the chromosome, which should be:

$$c = \sum_{i=1}^{n-1} d(g_i, g_{i+1}) + d(g_n, g_1),$$

Where g_i denotes the i^{th} gene in the chromosome.

2.2.4 Selection Strategy

We use the same strategy as that in the provided code. Firstly, two individuals are chosen randomly in the population. Then, we select the individual with a higher fitness and add it into the mating population. This procedure is performed repeatedly until the mating population is full.

2.2.5 Crossover Operator

Since each city can only be visited once, any pair of genes in a chromosome cannot be duplicate. However, the crossover operator may violate this constraint. To guarantee the individuals produced by crossover operator to be valid, we design an iterative strategy to resolve this problem.

To begin with, we shuffle the individuals in the population and get them paired off. We argue that this step is necessary in terms of the diversity of individuals in the next generation. After that, each pair of individuals has a probability of p_c to take crossover

operation. If a pair of individuals is going to take crossover, the gene segments to be exchanged are decided randomly. After exchanging their gene segments, we need to check both of these two individuals to see whether there are duplicate genes in the chromosomes. If there are conflicts, an iterative procedure needs to be executed. Specifically, after swapping some gene segments between chromosome A and chromosome B, it is likely that the i^{th} and the j^{th} gene in A, which is denoted as A_i and A_j respectively, are duplicate. Suppose that A_i locates inside the swapped segment and A_j locates outside the swapped segment, and we exchange A_j with B_i. In this way, we eliminate each pair of conflicts iteratively until both A and B are valid chromosomes. Figure 2 describes this procedure with an example.

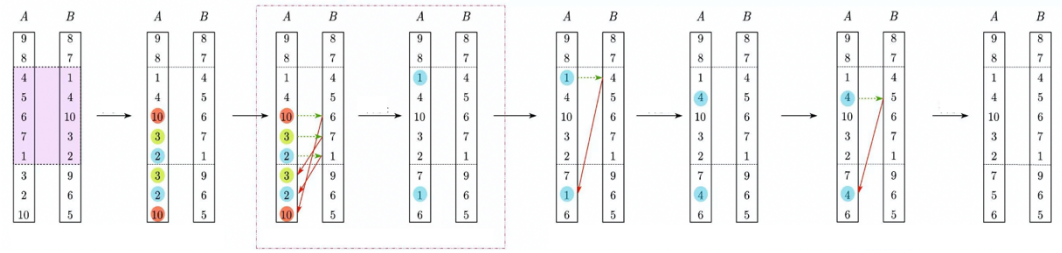

Fig.2: An example of crossover between A and B followed by transforming A into a valid one. The procedure of eliminating the conflicts in B is similar.

2.2.6 Mutation Operator

Considering the constraint that any pair of genes in a chromosome cannot be duplicate, we use the swap strategy for mutation operator. Firstly, each gene in the chromosome has a probability of p_m to mutate. Secondly, if a gene is going to mutate, then we randomly choose another gene in the chromosome and swap these two genes. Figure 3 is an example of our mutation operation.

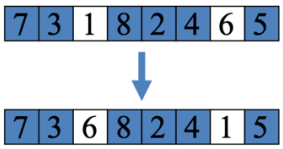

Fig.3: An example of the swap strategy used in mutation.

2.2.7 Elitism

The elitism strategy in our code is the same as that of the provided code. Specifically, the best individual in the last generation is added to the next generation directly.

2.3 Experiments

We perform all our experiments using Python 3.7. The logic of our code is similar to

that of the provided MATLAB code. All main modifications have been described in Section 2.2.

We first use GA to solve a 10-city symmetric TSP. The coordinates of the 10 cities are given in table 1.

Table1: Coordinates of the 10 cities

City	X-coordinate	Y-coordinate
1	0.3642	0.7770
2	0.7185	0.8312
3	0.0986	0.5891
4	0.2954	0.9606
5	0.5951	0.4647
6	0.6697	0.7657
7	0.4353	0.1709
8	0.2131	0.8349
9	0.2479	0.6984
10	0.4516	0.0488

Before running our algorithm, some parameters need to be determined, including population size, crossover rate, mutation rate, and the number of generations. The goodness of these parameters affects the performance of our algorithm, i.e., whether or not it is able to reach the optimal solution and the number of generations it takes to reach the optimal solution. Therefore, we perform grid search to compare different sets of parameters. The maximum number of generations is limited at 500. The candidate values of population size are 20 and 100. The candidate values of crossover rate are 0.1, 0.4, 0.7, and 1.0. The candidate values of mutation rate are 0.02, 0.05, and 0.08. Given a specific set of parameters, we will see how many generations it takes to reach the optimal solution on average by using the same set of parameters to run the algorithm for 10 times independently. If the algorithm cannot converge to the optimal solution during 500 generations, we regard the number of generations it takes to converge to the optimal solution as 500. We analyze the effects of changing population size, crossover rate, and mutation rate according to the results obtained in table 2.

Table 2: The average number of generations it takes to find the optimal solution given different sets of parameters.

population size	crossover rate	mutation rate	Average number of generations it takes to find the optimal solution
20	0.1	0.02	31.67
20	0.1	0.05	23.53
20	0.1	0.08	26.87
20	0.4	0.02	23.54

20	0.4	0.05	16.44
20	0.4	0.08	20.41
20	0.7	0.02	24.08
20	0.7	0.05	19.98
20	0.7	0.08	25.77
20	1.0	0.02	27.53
20	1.0	0.05	10.31
20	1.0	0.08	33.00
100	0.1	0.02	21.94
100	0.1	0.05	5.78
100	0.1	0.08	8.09
100	0.4	0.02	7.39
100	0.4	0.05	5.09
100	0.4	0.08	7.16
100	0.7	0.02	17.44
100	0.7	0.05	4.79
100	0.7	0.08	11.15
100	1.0	0.02	12.11
100	1.0	0.05	4.34
100	1.0	0.08	11.88

2.3.1 Analysis of Population Size

Given the same crossover rate and the same mutation rate, a larger population size always outperforms a smaller population size in terms of the average number of generations it takes to find the optimal solution. If the population size is too small, it is obvious that inbreeding will occur and pathological genes will be produced. Even if using a larger mutation rate, the possibility of generating competitive high-order schemas is still cancelled, let along the fact that a higher mutation rate might destroy the existing schemas greatly. At the same time, there are random errors (pattern sampling errors) in genetic operators, which hinder the correct propagation of effective schemas in small populations and make the evolution unable to produce the expected number of schemas according to the Schemata Theorem.

However, using a too large population size may be unwise: the larger population size it uses, the more computing resources it requires, and hence it takes more time for convergence. On the other hand, it might be a waste of computing resources to set a too large population size, since a relatively small population size may be good enough to yield good performance. Therefore, it would be better choosing a relatively large population size considering the available computing resources.

2.3.2 Analysis of Crossover Rate

In our experiments, a crossover rate of 0.4 is the most robust choice. Compared with

the crossover rates of 0.1, 0.7, and 1.0, the crossover rate of 0.4 makes the algorithm less sensitive to the changes of the mutation rate. In other words, if we set the crossover rate to be 0.4, the algorithm can find the optimal solution with relatively small efforts even when the choice of mutation rate is awful. Intuitively, it is easy to destroy the existing favorable schemas, increase the randomness, and miss the optimal individual if the crossover rate is too high. On the other hand, if the crossover rate is too low, it is not able to update the population effectively. Hence, it is not desirable to set a too high crossover rate or a too low crossover rate.

2.3.3 Analysis of Mutation Rate

Given the same population size and the same crossover rate, the mutation rate of 0.05 always outperforms the mutation rates of 0.02 and 0.08. According to this result, we draw a conclusion that the mutation rate can neither be too low nor too high. If the mutation rate is too low, it will harm the diversity of the population. Also, it will easily lead to the rapid loss of effective genes and is not easy to repair. Nevertheless, a too high mutation rate is not effective either. Although in this case the diversity of the population can be guaranteed, the probability of high-order schemas being destroyed will also increase. As a result, a medium level of mutation rate is preferred.

3 Asymmetric Traveling Salesman Problem

3.1 Problem Setup

In the asymmetric TSP, the distance from city A to city B can be different from that from city B to city A. The objective function of asymmetric TSP is the same as that of symmetric TSP, except that $d(i,j) \neq d(j,i)$.

3.2 Method

The only difference between symmetric TSP and asymmetric TSP lies in the distance matrix d. Since d is a non-symmetric matrix in this problem, we decide each entry in d randomly and independently. Concretely, we draw a value from $[0,1)$ uniformly for each entry in d.

3.3 Experiments

In this experiment, we explore the effects of the scale of the problem on the convergence speed. We vary the number of cities in the asymmetric TSP to be 10, 20, 30, 40, and 50. The population size equals to 100, the crossover rate equals to 0.4, the mutation rate equals to 0.05, and the maximum generation is 1000. The results are shown in figure 4.

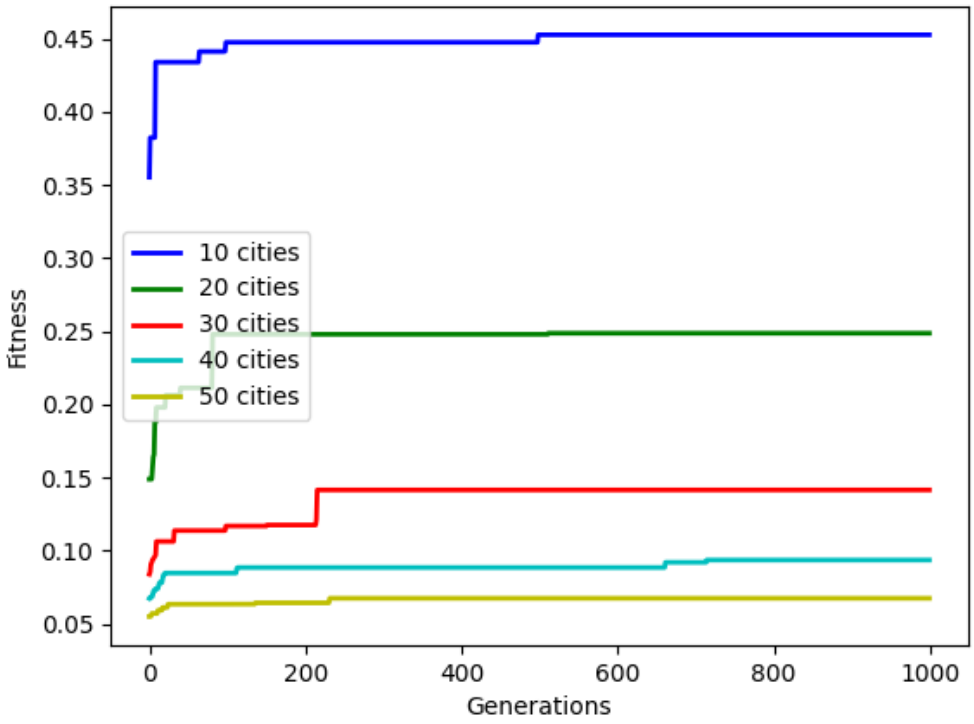

Fig.4: Fitness Trace of GA on asymmetric TSP with different number of cities

As the number of cities grows linearly, the solution space of TSP expands exponentially, which results in the increasing difficulty of solving the problem. As is shown in figure 4, however, GA does not suffer from the "curse of dimensionality". Even when the number of cities is relatively large, our algorithm can converge without too many generations. Although we are not able to figure out whether the algorithm has converge to the optimal solution, we believe that GA still has good performance even when the solution space is exponentially large.

4 Traveling Salesman Problem Given the Start and the End City

4.1 Problem Setup

In this problem, a start city and an end city are given. As a result, a valid route must start from the start city and end at the end city. Now the new objective function can be described as:

$$\min\ d(s,1) + \sum_{i=1}^{n-3} d(i,i+1) + d(n-2,e),$$

where s denotes the start city and e denotes the end city.

4.2 Method

4.2.1 Representation

In this problem, the start city s and the end city e are specified, so the length of a chromosome should be $n-2$. Let sequence $L=\{1,2,\dots,n\}-\{s,e\}$, then each chromosome should be a permutation of L.

4.2.2 Fitness Evaluation

The calculation of fitness in this problem is the same as that in normal TSP, except that a different way to calculate the total length of the path corresponding to the chromosome should be used, which is written as follows:

$$c = d(s, g_1) + \sum_{i=1}^{n-3} d(g_i, g_{i+1}) + d(g_{n-2}, e).$$

4.3 Experiments

We use the same data used in the 10-city Symmetric TSP, and set city 1 as the start city and city 10 as the end city. The population size equals to 100, the crossover rate equals to 0.4, the mutation rate equals to 0.05, and the maximum generation is 50. The results are shown in figure 5.

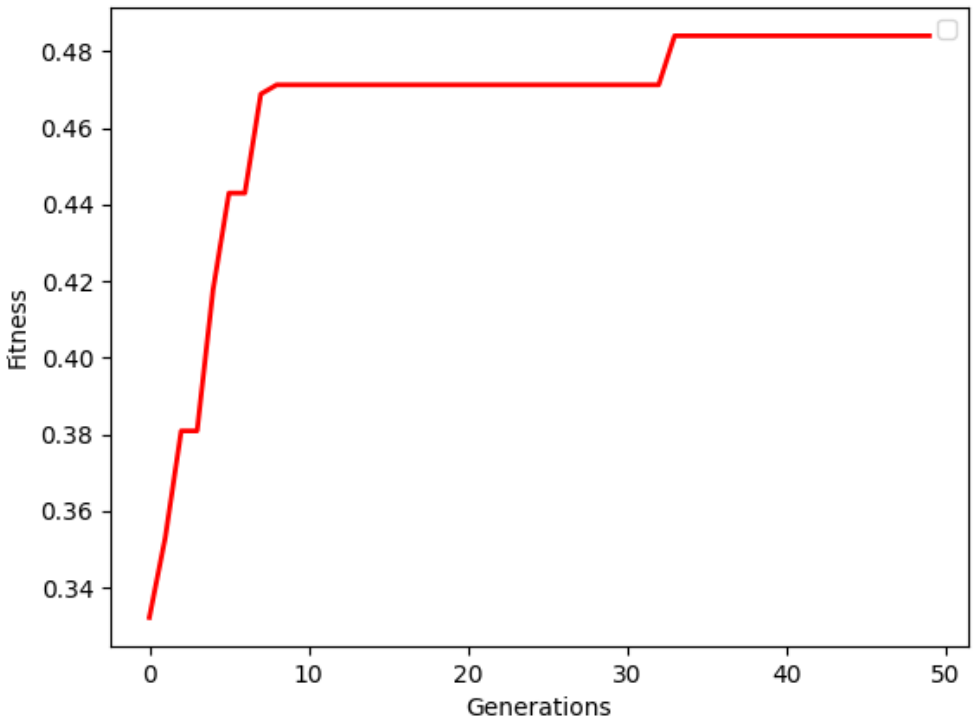

Fig.5: 10-city Symmetric TSP with specified start and end city

In this experiment, the algorithm converges to the optimal solution in the 33^{rd} generation, where the fitness equals to 0.484.

5 Conclusion

In this report, we solve TSP and its variants using genetic algorithm. We notice that the selection of parameters has high impact on the performance of the algorithm. Specifically, neither a too large value nor a too small value should be used for the parameters (including population size, crossover rate, and mutation rate) in order to achieve good performance. We also study the effects of the scale of the problem on the convergence speed. As the number of cities grows linearly, the solution space of TSP expands exponentially, which results in the increasing difficulty of solving the problem. Our experiments show that genetic algorithm can overcome the "curse of dimensionality" and converge to good solutions without too many generations.