Generic Algorithm for TSP Report

27-10-2021

1. Introduction:

Traveling Salesman Problem (TSP) is a classical problem. This problem is about a salesman who spends his time visiting n cities cyclically. The traditional way is to find all possible routes and then compare the distance. The computation time increases exponentially with increasing cities. In this project, we will use genetic algorithm (GA) to solve this classic TSP problem.

2. Methodology:

Generic algorithm (GA) is an imitation of natural evolution. In natural evolution, fittest animals are more likely to survive and produce offspring. This offspring inherits some characteristics of parents, having higher chance to survive. When this keeps processing, good genes will stay in that kind of animal and this is natural evolution. In GA for TSP, the condition is similar. A random population is generated. In each generation, routes with shorter distance are more likely to be chosen. Then, they will crossover (reproduction) and generate a new route. Some routes will also mutate as well (mutation). After some generation, the cost or distance of routes get smaller. If a route can satisfy the termination criteria, we can use it as a solution. However, we may get into a local minimum sometimes. We should repeat GA with different initial population to ensure a global minimum or a minimum distance which is satisfying enough.

**2.1. Basic assumptions for given 10-city symmetric TSP:**

1. This TSP is symmetric.
2. Each city is travelled once only.
3. Salesman returns to start city after last city for a round-trip.
4. No restrictions on start and end city.

Some assumptions may be changed in later part of this report.

**2.2. Evolution in GA:**

Start Population generation Termination criteria True End

False

Selection Crossover Mutation Elitism

After the generation of population, the population will evolute in generations.

In each generation, the population will be examined by termination criteria. If it is true, we can stop the evolution. If it is false, the population will undergo a set of processing as below.

To generate a child population, we need to use a for loop to generate child route until the number of routes in child population is equal to population size. In each iteration, a pair of parents is chosen in current population by rank weighting selection. A child route is generated by crossing over the pair of parents followed by mutation. Then, this child route is appended to child population. After the loop, we get the whole child population. We perform elitism to ensure convergence of solution.

Check the termination criteria again to decide whether next generation is needed.

**2.3. Functions used in GA:**

Below are functions used in GA. Every function will be explained in detail.

2.3.1. Population generation:

Read the data of 10 cities given. Define a simple class **City** with attributes **num**, **x** and **y**. Then, simply generate some numbers of routes randomly based on the population size given.

2.3.2. Termination criteria:

There are some criteria to stop our evolution.

1. Generation limit reached.
2. Route smaller than cost limit found.

Cost limit rather than fitness limit is used as we need to find shortest but not longest distance.

We will examine these two criteria in each generation. If one of the criteria is satisfied, we have found a solution. If not, it keeps looping with next generation.

2.3.3. Selection:

First, the population are sorted in descending order of cost. They are assigned with ranks. The lower the cost, the higher the rank assigned. Then, the probability of each route being chosen is:

Note that route with lower cost has higher chance to be chosen which is like natural evolution. Better routes are more likely to survive. In a result, two parent routes are chosen by rank weighting selection for breeding.

2.3.4. Crossover:

A random number in [0, 1] is generated. If it is smaller than the crossover rate, perform crossover as follow.

Generate two random numbers within number of cities. Copy the cities from parent route 1 with positions between these two numbers to child route in exact positions. Then, append the cities from parent route 2 which do not exist in child route yet to child route in order. Crossover can keep some characteristic of parent routes. This is similar to natural breeding. This can ensure some characteristic of parent routes are kept in child route.

Please refer to the following figure for clearer explanation of real operation of crossover.

A picture containing text, orange, screenshot

Description automatically generated

2.3.5. Mutation:

A random number in [0, 1] is generated. If it is smaller than the mutation rate, perform mutation as follow.

Generate two random numbers within number of cities. Simply swap the cities at positions of these two numbers. Mutation generates some random changes in the population to form some route that can never be formed by selection and crossover. However, be careful that mutation probability cannot be too high or else the evolution will become totally random.

2.3.6. Elitism:

Best routes in parent population replace worst routes in child population based on elitism size. For example, 1 route replacement for elitism size of 0.1 in 10 population size. After crossover and mutation, there is a chance that the performance of child population is worse than parent population. Elitism can ensure the solution converges in each generation.

3. Experimental results and analysis:

**3.1. Results and analysis without restrictions:**

There are several parameters that can be adjusted in GA. They are crossover rate, mutation rate, population size, number of generations, and cost limit.

The basic setup of these parameters are as follows:

Text

Description automatically generated

values are just some typical values. Cost limit is set to 0 as we want to find the minimum possible cost. The program runs 10 times. Below are the results.

Text

Description automatically generated

There are several possible routes which give a cost 2.583248284332358. [6 2 9 1 4 8 3 7 10 5] can be one of the solutions. Other routes which have the same cost are alternative solutions.

In the following, the parameters will be changed and experimented one by one. Each of them will be tested with a bunch of values. Each value will then be run by 50. Both average generation number and average convergence time will be used for evaluating the performance. The smaller the generation number and convergence time, the better the performance.

3.2. Population size:

Population size will be tested with integers from 10 to 100. Cost limit is set to 2.6, which is a good standard of enough convergence. Other parameters are unchanged. Then, we can investigate number of generation and time needed for the solution to converge.

Chart, line chart

Description automatically generated

From the figure, we can see a general trend despite some random nature of GA. When population size is getting large, we can use less generation number to converge the solution. A larger population can find out the best route more easily. It requires only a few generations. However, the time of convergence also increases at the same time. A population size means longer computation time for each generation. We cannot say what size of population is the best. It depends on the preference on small number of generation or short convergence time, and the computation power you have.

3.3. Crossover rate:

Crossover rate will be tested with numbers from 0.1 to 1.0. Cost limit is set to 2.6. Other parameters are unchanged. Then, we can investigate number of generation and time needed for the solution to converge.

Chart, line chart

Description automatically generated

From the figure, the best crossover rate is around 0.8 to 1.0. When the crossover rate is getting larger, the faster it will converge. When the crossover rate is higher, parent routes can give birth to child route more easily. It can converge faster and give better performance.

3.4. Mutation rate:

Mutation rate will be tested with numbers from 0.000001 to 0.1. Cost limit is set to 2.6. Other parameters are unchanged. Then, we can investigate number of generation and time needed for the solution to converge.

Chart, line chart

Description automatically generated

From the figure, the best mutation is about 0.01. When mutation rate increases to 0.01, performance keeps improving as larger mutation rate can find out the best route which cannot be found by crossover only. However, performance drops after mutation rate 0.01. It is because too large mutation rate make it become random search, causing worse performance.

3.5. Elitism size:

Elitism size will be tested with numbers from 0.1 to 1. Cost limit is set to 2.6. Other parameters are unchanged. Then, we can investigate number of generation and time needed for the solution to converge.

Chart, line chart

Description automatically generated

From the figure, the best elitism rate is about 0.3 to 0.5 in this problem. With a small elitism rate, the best routes cannot be inherited to next population, lower the efficiency in convergence. On the other hand, with a large elitism rate keeps too many routes to next generation. This lowers evolution of routes, lowering the efficiency in convergence.

3.6. Generation limit:

Generation limit is the maximum number of generations set on GA. The smaller the generation limit, the worse solution we can get as there are not enough generations for GA to converge. To have a good enough solution, it is better to set a larger limit to see the rough number of generations for solution convergence. Then, run the algorithm again with a reasonable generation limit. Side effect of too large generation limit is waste of computation power and time.

4. Extensions

**4.1. More cities:**

When using traditional method with TSP problem, computation time and complexity will increase exponentially. We try to use add more cities with to the original 10 cities to see the performance of GA. More extra cities with coordinates [0, 1] are generated. We test the convergence time of solution for number of cities from 20 to 100.

Chart, line chart

Description automatically generated

We can see that the convergence time increases almost linearly but not exponentially using GA. Even for 100 cities, the average convergence time for each run is around 36s which is satisfactory.

**4.2. Given start and end cities:**

We need to adjust both crossover function and mutation function, setting up some restrictions on them.

* Population generation: start and end cities are fixed when every route generated
* Cost function: no need to compute the cost from end city to start city
* Crossover function: cannot change the position of start and end cities while crossing
* Mutation function: both start and end cities cannot be mutated
* Run evolution function: include start and end cities as parameters

Other functions remain unchanged. Please refer to **ga\_given\_start\_end.py** for the customized functions.

Then, the problem is run again with given example of start city 9 and end city 5 as an example.

The setup of parameters are as follows:

Text

Description automatically generated

Result:

Text

Description automatically generated

The best route is [9 1 6 2 4 8 3 7 10 5] with cost 2.4315533675724423.

As there is a restriction on start and end cities, there is only one solution. The travel route is not a loop. It cannot start and end in arbitrary city but give the same cost.

Please note that the cost is also an increase cost. It is because the start and end cities may be close from each other. With the restriction, the solution cannot find a route which travel these two cities consecutively, resulting in extra cost.

**4.3. Asymmetric traveling salesman problem (ATSP):**

To solve an asymmetric TSP problem, we need a matrix with all the distances among each pair of cities in two directions. Given the total number of cities is n, we need a total of n^2 distances for the algorithm. In this project, we will use the 10 cities given for demonstration.

A set of distances have been generated with random number [0, 1]. The dataset is stored in **“Dataset/ Atsp\_distance\_dataset.txt”**

Here is a capture of first ten distances in the file:

Text

Description automatically generated

The distance data will be read and stored in a dictionary for further cost evaluation.

In ATSP, we need to customize the cost function. The total cost of a route is calculated by adding up the distances in each city pair. The program will extensively make use of this new cost function in finding the best route. All the customized functions can be found in **ga\_atsp.py**.

The set up is almost the same but an extra distance\_dict for cost evaluation of ATSP.

Text

Description automatically generated

Result:

Text

Description automatically generated

The best route is [7 4 2 3 10 1 9 6 8 5] with cost 1.7157684599999998.

With asymmetric cost, some cities are likely to be visited in one direction but not the opposite. For example, if cost of A-> B is much larger than B -> A, the route is more likely to include B -> A. This kind of logic restricts the travelling route, resulting in one best route.

**4.4. Sequential ordering problem (SOP):**

A sequential order of some cities. Is given. For example, some cities must be visited before others. Several functions are customized for this restriction:

* Population generation: the sequence of cities is fulfilled in all routes in initial population
* Crossover function: checking of sequence before return the child route. If the cities are not in right order, perform crossover again
* Mutation function: checking of sequence before return the child route. If the cities are not in right order, perform mutation again
* Run evolution function: include a sequence of cities as one of the parameters

Other functions remain unchanged. Please refer to **ga\_sop.py** for the customized functions

Then, the problem is run with sequence 9 -> 5-> 3 as an example. City 9 needs to be visited before 5 and 3 while city 5 needs to be visited before 3.

Text

Description automatically generated

Result:

Text

Description automatically generated

The best route is [8 4 1 9 2 6 5 10 7 3] with cost 2.5832482843323583. The route can fulfil the requirement of sequence [9, 5, 3]. Any other sequences can be put in the sequence for requirement as well.

Please note that there is an increase in cost. It is because the given sequence may not be an efficient way in travelling. For example, maybe city 9 is near to city 3 and the cost between these two cities is very small. However, we must travel city 5 before 3. This kind of scenarios may contribute to extra cost.

**4.5 GATSP with clustering:**

For large-scale data, it is normal for us to divide the cities into several regions. To be more precise, the salesman has to visit all the cities in a region before visiting another. In the following clustering is applied to divide regions for region TSP.

In this project, 50 cities are given with x, y coordinates in data file **“Cluster\_dataset.txt”**. Here is the distribution of cities:

Chart, scatter chart

Description automatically generated

Then, KMeans from sklearn.cluster library is used for clustering job. 3 clusters are set for clustering job. After fitting, the result clustered cities distribution are as follows:

Chart, scatter chart

Description automatically generated

The output clustering result is stored in **“Dataset/clustering\_result.csv”**. Each city is assigned a number in [0, 1, 2], indicating its cluster number. Detailed coding of clustering job can be found in **clustering.ipynb**.

In each cluster, we perform GA on finding the best route for 10 runs. Then, we have three list of best routes of 3 clusters. Three routes selected from three different clustered are combined to form result routes. All these result routes are compared and the result route with lowest cost is chosen to be the solution. Detailed coding of some essential functions of clustering GATSP can be found in **ga\_clustering.py**.

The main function run\_evolution\_clustering in ga\_clustering.py is called with the following setup:

Text

Description automatically generated

Results:

Table

Description automatically generated

The final best route gives a cost 45.357688053321894.

The route is [40 26 37 29 36 27 28 31 32 34 35 33 19 22 25 30 20 17 13 8 12 38 5 4 1 2 3 9 6 23 15 21 18 16 24 7 10 11 14 49 50 48 47 46 44 42 41 39 43 45].

5. References:

Kie Codes. (2020, July 21). GitHub. <https://github.com/kiecodes/genetic-algorithms>

Thakkar, D. (2021, March 17). Using a Genetic Algorithm for Traveling Salesman Problem in Python – Cresco. Retrieved November 7, 2021, from https://crescointl.com/2021/03/17/using-a-genetic-algorithm-for-traveling-salesman-problem-in-python/