Solving Travelling Santa with genetic algorithom

Maruf Sakib
Center for Geospatial and Visualization Computing
Old Dominion University
Norfolk, Virginia
msakib@odu.edu
www.cs.odu.edu/~msakib

Yu Zhang
Department of Computer Science
Old Dominion University
Norfolk, Virginia
yzhan007@odu.edu

***Abstract*— This paper discusses the use of genetic algorithm to determine the shortest path for travelling 150000 point representing cities Santa needs to visit. Thus, implementing a solution like the travelling salesmen problem where certain conditions were imposed to make this project more academic friendly. The first section discusses the problem, the tasks implemented, some details regarding the provided data and the basics of genetic algorithm used to solve of the problem. Second section provides more details regarding genetic algorithm and terminology related to it as well as the step by test workflow to implement provided tasks. The third section provides results and difference performance metrics while implementing the genetic algorithm on the provided and test data. The last section brings a conclusion of this paper by providing some observations based on the results and possible way to improve the solution further.**

Keywords— Artificial Intelligence, Genetic Algorithm, least cost path detection, travelling Santa problem

I. INTRODUCTION

Genetic algorithm can be used to find the least cost path connecting many points. As a probabilistic optimization algorithm, genetic algorithm uses concepts of natural selection and genetic inheritance. Inspired by the biological evolution process, genetic algorithm was originally developed by John Holland. [1]

This project is basically an academic rework of the Kaggle competition of the "Travelling Santa problem" with slight modifications. Although genetic algorithm is slow, it is well suited for hard local search problems like this. Our team's approach included well defined fitness function with PMX crossover and mutation percentage of 1%.

I.A Problem Definition:

In short, the problem in this project is termed as the "Traveling Santa Problem".

I.B Tasks:

This report is written considering the following two tasks:

Task 1: Find a tour yielding the minimum path cost ignoring the requirement of two disjoint paths and just treating it as a common traveling salesman's problem.

Task 2. Find a second tour yielding the minimum path cost avoiding two disjoint paths from first tour.

All the discussion from this point to forward will directly reference to task 1 and task 2 to point to the details mentioned on the task above respectively.

I.C Data Description:

The data is provided in for this project is consist of three columns and 150000 rows. The columns are id, X and Y representing the name, latitude, and longitude of 150000 cities.

figure no I.C.1: visual location representation of the full data

When all those cities are represented on a scatter diagram using the provided data, a visual representation of Santa is generated (figure no I.C.1). The attached image is retrieved from the mentioned problem of the Kaggle site.

To make the testing part of the developed genetic algorithm more convenient, the first 50 cities data was taken as a test data. This was a very useful approach as doing all the calculation such as population generation, crossover, mutation, distance calculation, and scoring on 50 cities was significantly faster than performing all these calculation on 150000 cities. From this point in the report, the first 50 cities data will be referred as test data and all 150000 cities data will be referred as full data.

Creation of test data heavily benefitted the rapid development and prototyping of the genetic algorithm.

II. Methods

As mentioned above, this report focuses on implementation of genetic algorithm to solve the travelling Santa problem. A genetic algorithm maintains a population of candidate solutions for the problem at hand, and makes it evolve by iteratively applying a set of stochastic operators. Starting with a subset of n randomly chosen solutions from the search space. This is the initial population. This population is used to produce a next generation of individuals by reproduction. Individuals with a higher fitness have more chance to reproduce (i.e. natural selection) [1].

II.A Basic concept of GA:

A general genetic algorithm process includes the following steps:

Selection: replicates the most successful solutions found in a population at a rate proportional to their relative quality.

Recombination (crossover): decomposes two distinct solutions and then randomly mixes their parts to form novel solutions

Mutation: randomly perturbs a candidate solution

[1]

II.B Workflow:

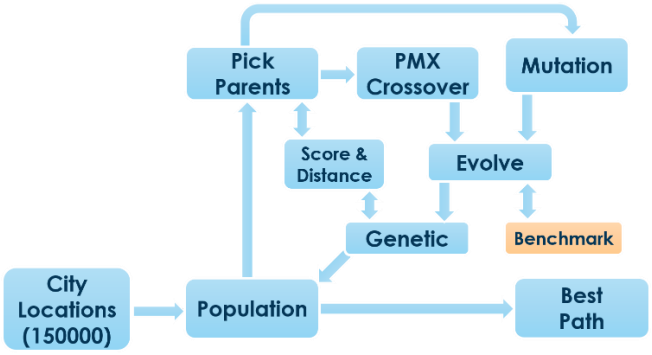

Figure no II.B.1: workflow of the genetic algorithm

The workflow of the genetic algorithm developed in this project are divided into several python function activities (Figure no II.B.1). The details of those functions are as the following:

Read_data: the first part of the genetic algorithm reads the data from input data sources such as test data or full data and prepares the data to be used by other functions.

Population: This function creates random paths from the reading the data and creates population samples to work with. The size of the population depends on the degree of freedom. To save time and increase convenience, in this project the population size was kept 100

Pick_parents: Here, two parents path/tour were randomly selected for genetic crossover and mutation. But their initial total distance and fitness function score is also called in this function.

Distance: This calculated the total distance of a tour (parent). If, A(x1,y1), B(x2,y2), and C(x3,y3) are three cities in a tour, then there total distance for population sample ABC is as the following:

$$\text{Total distance (ABC)} = ((x1-x2)^2+(y1-y2)^2)^{0.5}+ ((x2-x3)^2+(y2-y3)^2)^{0.5}+((x3-x1)^2+(y3-y1)^2)^{0.5}$$

Score: Score performs the fitness function for the genetic algorithm. In this project, the fitness function score was measured as 1/total_distance.

Crossover: Partially mapped crossover (PMX) was the chosen crossover method in this project. PMX ensured that no same city populated twice in a child when crossover done between two parents. This approach ensures that Santa doesn't have to visit the same city in a single tour.

Mutation: For mutation 1% cities of the child sample will randomly exchange place within themselves. In case of test data, it is 5 cities and 150 cities for the full data

Benchmark: The benchmark or check_duplicate function is only used for task to compare an evolved child with the result of task 1. If, there is match between two disjoint paths in the new child with two disjoint paths of the output of task 1, benchmark function will remove that newly evolved child. Thus for ABC path as an output of task 1, the benchmark function will filter out any newly evolved child with AB, BA, BC, CB, CA, or AC paths in them.

Evolve: the evolve function calls pick_parents, crossover and mutations functions to perform genetic evaluation for one single generation. For task 2, evolve also calls the benchmark function to figure out if it will keep the newly evolved child or not.

Genetic: Finally, the genetic function calls the evolve function multiple time to perform multigeneration genetic evolution. In this project, the genetic function also creates the output file for the best path and different statistics metrics for generation evolution.

III. Results

The genetic algorithm developed in this project has a lot of random influences in different functions developed such as crossover and mutation stages. As a result, the output of the test or full data were almost never the same. Although each time the output was different, the final distance and percentage improvement started to show some similarity.

As the execution of the developed genetic algorithm was significantly faster on the test data, 500 generations of genetic evolution were done on the test data. On the other hand, the full data went through 70 generations of genetic evolution as the execution of genetic algorithm on the full data was very slow.

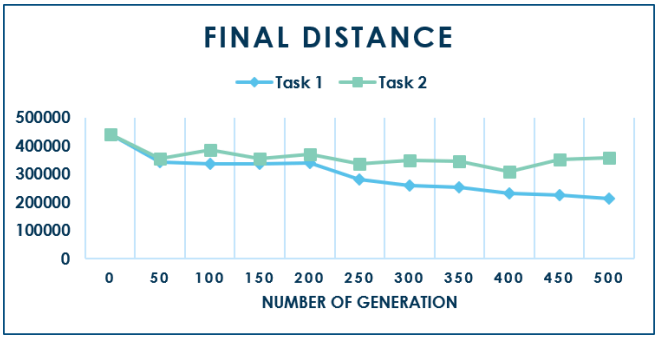

Figure no III.1: final distance using test data for task 1 and task 2

In case of the test data, the total distance at the generation first generation was 440552.98 distance unit and by the end of the task 1, it was 214059.62 distance unit (Figure no III.1). But, as task 2 cannot use any paths used in task 1, the final distance in task to came out as 357872.14 distance unit.

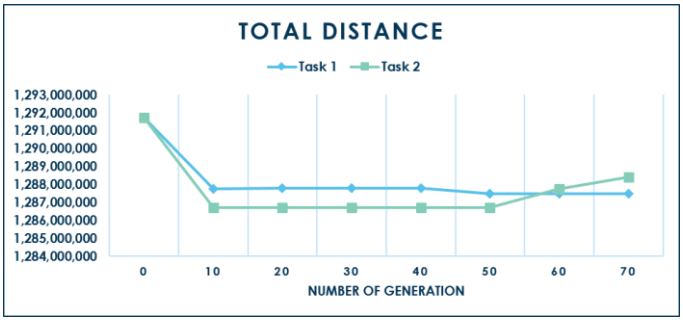

Figure no III.2: final distance using the full data for task 1 and task 2

Similarly, for the full data, the total distance at the generation first generation was 1291.1 million distance unit and by the end of the task 1, it was 1287.4 million distance unit (Figure no III.2). Again, as task 2 cannot use any paths used in task 1, the final distance in task to came out as 357872.14 distance unit.

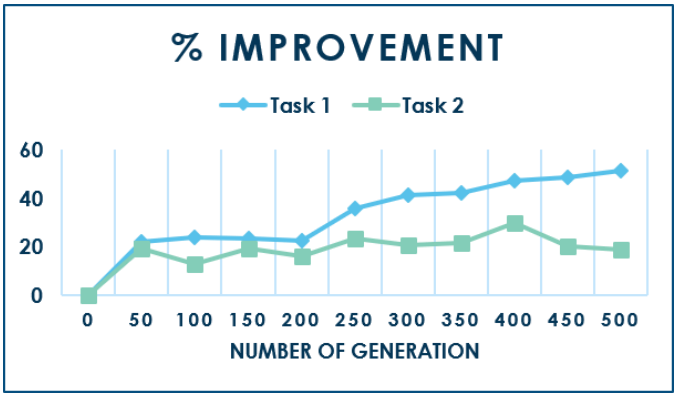

Figure no III.3: percentage of distance improvement using test data for task 1 and task 2

While considering total distance improvement in test data, by the end of the task 1, there was around 51.41 percent of the total distance improvement and task 2 had a distance improvement of 18.22 percent (Figure no III.3). In case of full data, task 1 had a distance improvement of only 0.33 percent and task 2 had a distance improvement of 0.26 percent (Figure no III.4).

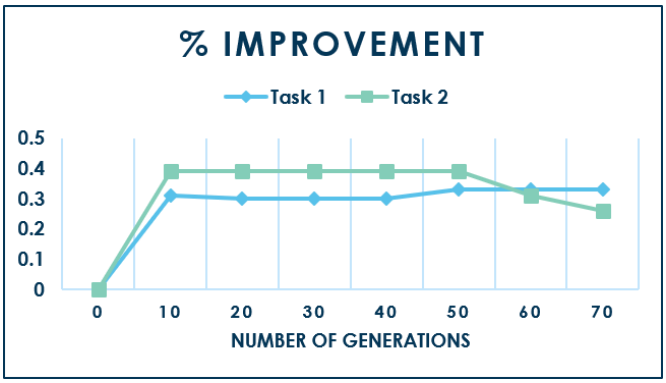

Figure no III.4: percentage of distance improvement using full data for task 1 and task 2

Although, in the first sight it is can be said that percentage of distance improvement is low in full data in comparison to test data. But the total and per generation distance reduction in full data is much higher.

Table III.1: Total and per generation distance improvement

Data source	Task	Total distance Improvement	Distance improvement per generation
Test Data	Task 1	226493.36	452.98
	Task 2	82680.84	165.36
Full data	Task 1	4.26 million	60941.89
	Task 2	3.31 million	47379.28

From table no III.1, in case of test data it can see that in average per generation the genetic algorithm reduced 452.98 distance unit in task 1 and 165.36 in task 2. But, in case of full data, the genetic it reduced 60941.89 distance unit in task 1 and 47379.28 in task 2 per generation. Thus, the amount of distance reduction seems to be much higher in case of full data for each generation.

III.A Performance Analysis

In case of performance measurement, it needs to be kept in mind that performance of the same algorithm can be different in different computers based on each of their configuration. But, all this calculation for test data and full data regarding task 1 and task 2 are dome on the same hardware to keep the comparison value fair. In this project, total execution time and average execution time is use as a performance metric.

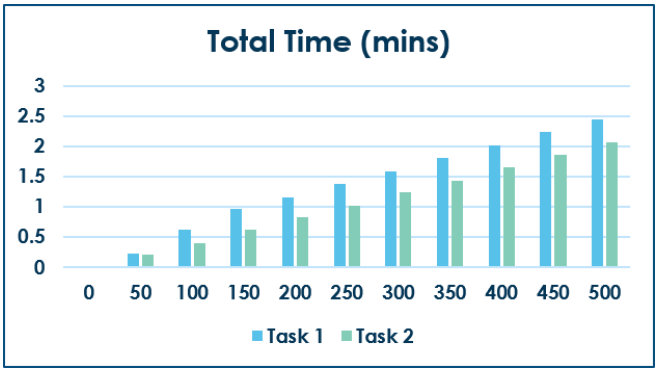

Figure no III.A.1: Time required to achieve different generations using test data for task 1 and task 2

The 500 generation of test data took 2.45 mins for task 1 and 2.07 mins for the task 2. Where each generation in average took 0.29 seconds for task 1 and 0.24 seconds for task 2 to be complete. Although the fair assumption given us the impression that average time in task 2 should be higher than average time in task 1, the random nature in crossover and mutation stages in task 2 were much more beneficial to task 2 when these performance values were noted (Figure no III.A.1).

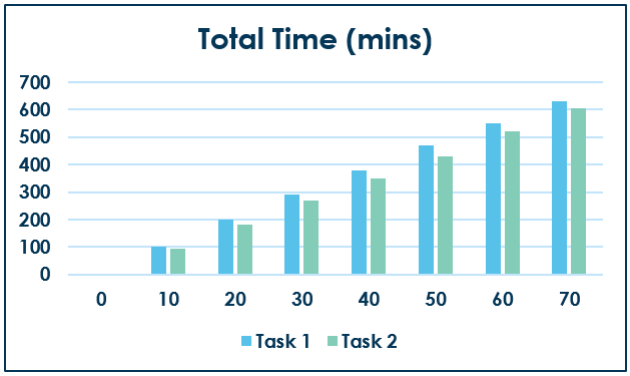

Figure no III.A.2: time required to achieve different generations using full data for task 1 and task 2

Both the total execution time and average execution time for each generation of task 1 and task 2 for the full data is much higher that the test data. In case full data, the genetic algorithm took 632.36 mins for task 1 and 604.55 mins for task 2. Again, the average per generation execution time for full data was 9.03 mins for task 1 and 8.63 for task 2.

IV. Conclusion

No matter how many cities are added in the source data, in all cases the execution of the genetic algorithm seems to be reducing the total distance of the tour path. Adding more cities seems to be positively influence the reduction of the total distance in each generation in average. But, adding more cities also added more execution time in the genetic algorithm. Also, from the previous figures, it can be seen that after some generation of execution, the amount of distance reduction definitely starts to go low. To develop a more effective genetic algorithm, the following step should be considered:

- Development of a better fitness function (currently 1/distance)
- Inclusion of crossover and mutation rate to increase AI robustness
- Worthy child check after crossover and mutation is done (currently unavailable to improve performance)

Acknowledgment

This report is done to fulfill the term project requirement for the CS796: Advanced artificial Intelligence graduate course at Old dominion University. They project team provides their sincere gratitude to course instructor and teaching assistance for their cooperation and time.

References

[1] Y. Li, *Applications of Genetic Algorithms: Artificial Intelligence Graduate Class Lecture 8*, Norfolk, Virginia: Old Dominion University, 2018.

[2] Y. Li, *Genetic Algorithms: Artificial Intelligence Graduate Class Lecture 7*, Norfolk, Virginia: Old Dominion University, 2018.