A Genetic Algorithm for Solving Travelling Salesman Problem

Adewole Philip
Department of Computer Science
University of Agriculture,
Abeokuta, Nigeria
philipwole@yahoo.com

Akinwale Adio Taofiki
Department of Computer Science
University of Agriculture,
Abeokuta, Nigeria
aatakinwale@yahoo.com

Otunbanowo Kehinde
Department of Computer Science
University of Agriculture,
Abeokuta, Nigeria
kenny_csc@yahoo.com

***Abstract*— In this paper we present a Genetic Algorithm for solving the Travelling Salesman problem (TSP). Genetic Algorithm which is a very good local search algorithm is employed to solve the TSP by generating a preset number of random tours and then improving the population until a stop condition is satisfied and the best chromosome which is a tour is returned as the solution. Analysis of the algorithmic parameters (Population, Mutation Rate and Cut Length) was done so as to know how to tune the algorithm for various problem instances.**

Keywords- Genetic Algorithm, Generation, Mutation rate, Population, Travelling Salesman Problem

I. Introduction

The traveling salesman problem (TSP) is a well-known and important combinatorial optimization problem. The goal is to find the shortest tour that visits each city in a given list exactly once and then returns to the starting city. In contrast to its simple definition, solving the TSP is difficult since it is an NP-complete problem [4]. Apart from its theoretical approach, the TSP has many applications. Some typical applications of TSP include vehicle routing, computer wiring, cutting wallpaper and job sequencing. The main application in statistics is combinatorial data analysis, e.g., reordering rows and columns of data matrices or identifying clusters.

The NP-completeness of the TSP already makes it more time efficient for small-to-medium size TSP instances to rely on heuristics in case a good but not necessarily optimal solution is sufficient.

In this paper genetic algorithm is used to solve Travelling Salesman Problem. Genetic algorithm is a technique used for estimating computer models based on methods adapted from the field of genetics in biology. To use this technique, one encodes possible model behaviors into "genes". After each generation, the current models are rated and allowed to mate and breed based on their fitness. In the process of mating, the genes are exchanged, crossovers and mutations can occur. The current population is discarded and its offspring forms the next generation. Also, Genetic Algorithm describes a variety of modeling or optimization techniques that claim to mimic some aspects of biological modeling in choosing an optimum. Typically, the object being modeled is represented in a fashion that is easy to modify automatically. Then a large number of candidate models are generated and tested against the current data. Each model is scored and the "best" models are retained for the next generation. The retention can be deterministic (choose the best k models) or random (choose the k models with probability proportional to the score). These models are then randomly perturbed (as in asexual reproduction) and the process is repeated until it converges. If the model is constructed so that they have "genes," the winners can "mate" to produce the next generation.

II. Travelling Salesman Problem

The TSP is probably the most widely studied combinatorial optimization problem because it is a conceptually simple problem but hard to solve. It is an NP complete problem. A Classical Traveling Salesman Problem (TSP) can be defined as a problem where starting from a node is required to visit every other node only once in a way that the total distance covered is minimized. This can be mathematically stated as follows:

$$\text{Min} \quad \sum_{i,j} c_{ij}x_{ij} \qquad (1)$$

$$\text{s.t} \quad \sum_{j} x_{ij} = 1 \,\forall\, i \neq j \qquad (2)$$

$$\sum_{i} x_{ij} = 1 \forall \; j \neq i \qquad (3)$$

$$u_i = 1 \qquad (4)$$

$$2 \leq u_i \leq n \forall\, i \neq 1 \qquad (5)$$

$$u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) \; \forall\, i \neq j \forall\, j \neq 1 \qquad (6)$$

$$u_i \geq 0 \;\forall\, i \qquad (7)$$

$$x_{ij} \in \{0,1\} \forall i, j \qquad (8)$$

Constraints set (4), (5), (6) and (7), are used to eliminate any sub tour in the solution. Without the additional constraints for sub tour elimination, the problem reduces to a simple assignment problem which can be solved as an Linear Programming without binary constraints on $\mathbf{x_{ij}}$ and will still result in binary solution for $\mathbf{x_{ij}}$. Introduction of additional constraints for sub tour elimination, however, makes the problem a Mixed Integer Problem with $\mathbf{n^2}$ integer variables for a problem of size **n**, which may become very difficult to solve for a moderate size of problem [7].

III. Methodology

Genetic algorithm is a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Genetic algorithm is inspired by Darwin's theory about evolution. It is not too hard to program or understand, since they are biological based. The general algorithm for a GA:

1) Create a Random Initial State:

An initial population is created from a random selection of solutions (which are analogous to chromosomes). This is unlike the situation for symbolic artificial intelligence systems where the initial state in a problem is already given instead.

2) Evaluate Fitness:

A value for fitness is assigned to each solution (chromosome) depending on how close it actually is to solving the problem (thus arriving to the answer of the desired problem). These "solutions" are not to be confused with "answers" to the problem, think of them as possible characteristics that the system would employ in order to reach the answer.

3) Reproduce (*ChildrenMutate*):

Those chromosomes with a higher fitness value are more likely to reproduce offspring which can mutate after reproduction. The offspring is a product of the father and mother, whose composition consists of a combination of genes from them (this process is known as "crossing over").

4) Next Generation:

If the new generation contains a solution that produces an output that is close enough or equal to the desired answer then the problem has been solved. If this is not the case, then the new generation will go through the same process as their parents did. This will continue until a solution is reached.

B. Algorithm

1. Initialization: Generate N random candidate routes and calculate fitness value for each route.

2. Repeat following steps Number of iteration times:

a) Selection: Select two best candidate routes.

b) Reproduction: Reproduce two routes from the best routes.

c) Generate new population: Replace the two worst routes with the new routes.

3. Return the best result

IV. Implementation

1) Program Details

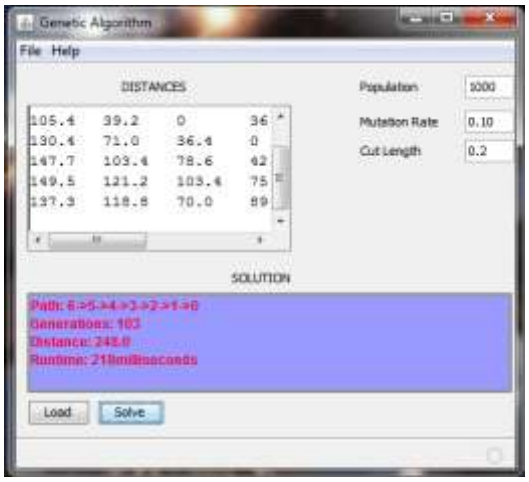
The program was written with Java. In genetic algorithm, a class "Chromosome" is needed. The Chromosome class generates random tours and makes them population members when its object is instantiated in the TSP class. The TSP class uses the Chromosomes "mate" method to reproduce new offspring from favoured Population of the previous generations. The TSP class in this case has two methods that use methods in Chromosome, the two methods are described below.

start(): This method initializes the cities and creates new chromosomes by creating an array of Chromosome objects. It also sorts the chromosomes by calling the method sortChromosomes() in Chromosomes then it sets the generation to 0

run(): Gets the favoured population from all the chromosomes created and mates them using mate() after this it sorts the chromosomes and then calculates the cost of the tour of the best chromosome. It repeats this procedure until the cost of the best tour can't be further improved.

In this program we have three algorithmic parameters that can be altered at each run of the program so as to vary the evolutionary strategies. The two parameters are Population and Mutation rate. The parameters go long way in determining the result of the algorithm. The program generates **n** random tours where **n** is the population size. These **n** tours are then sorted based on their fitness where the fitness function is basically the cost of tour. The best two tours gotten after sorting are mated to produce two new tours. And some randomly selected tours are also mutated. The worst tours are removed from the population and replaced with the new ones gotten from mating and mutation. This continues until best candidate can no longer be improved.

To test the algorithm a geometric city with 7 nodes is used. The optimal tour of the geometric city is 6 -> 5 -> 4 -> 3 -> 2 ->1 ->0 or 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6, both have the same cost. Genetic Algorithm was tested and the result is shown on the screen capture figure 1 below.



```
105.4   39.2    0       36
130.4   71.0    36.4    0
147.7   103.4   78.6    42
149.5   121.2   103.4   75
137.3   118.8   70.0    89
```

Population 1000
Mutation Rate 0.10
Cut Length 0.2

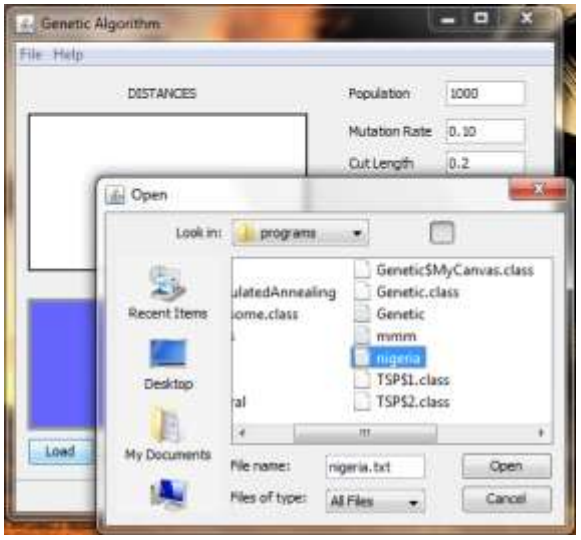
```
Path: 6->5->4->3->2->1->0
Generations: 103
Distance: 248.0
Runtime: 218milliseconds
```

Figure 1: Using the Algorithm to solve a Geometric City.

The program is used to solve a geometric city with clear optimal solution so as to be sure that the algorithm can arrive at optimal solution. As we can see in the figure 1 above, the path is optimal and the run time is fair at 218 milliseconds.

V. Results and Discussions

The data used in this paper is the distances between Nigerian major cities. The data was used because it is an average sized problem with 31 cities and its distances are moderate. The data was stored in a text file which can be imported into the program by clicking the load button on the GUI as shown in figure 2.



Population 1000
Mutation Rate 0.10
Cut Length 0.2
File name: nigeria.txt

Figure 2: Loading cities from text file

It is possible to alter the data in the file or add another different data. Table 1 shows the results of experiments carried out on the algorithm using different parameters. The table shows all the parameters used and the results. Performance of the result is based in the runtime and distance (cost).

TABLE 1 PARAMETERS AND RESULTS

Pop.	Mut. Rate	Cut Length	Run Time	Distance	Individuals
1000	0.1	0.2	546	5918.0	221000
1000	0.2	0.2	640	5896.0	226000
1000	0.3	0.2	748	5829.0	232000
1000	0.4	0.2	577	5886.0	192000
1000	0.5	0.2	577	5700.0	97000
1000	0.6	0.2	453	5981.0	190000
1000	0.7	0.2	577	5973.0	191000
1000	0.8	0.2	500	6100.0	195000
1000	0.9	0.2	608	5925.0	211000
1000	1	0.2	562	6010.0	209000
1000	0.01	0.2	532	9048.0	139000
100	0.1	0.2	31	10584.0	14400
100	0.2	0.2	47	10581.0	14600
100	0.3	0.2	31	11141.0	13600
100	0.4	0.2	31	12221.0	13500
100	0.5	0.2	32	10564.0	13900
100	0.6	0.2	32	9668.0	15200
100	0.7	0.2	31	9888.0	13900
100	0.8	0.2	31	10634.0	13700
100	0.9	0.2	31	11778.0	14000
100	1	0.2	31	10335.0	13000
100	0.01	0.2	46	9642.0	13600

It is important to note that change in the mutaton rate affects the runtime of the program. Figure 3 and 4 show the effect of mutation rate on runtime. Figure 3 is the plot using population size of 1000 while figure 4 illustrates the plot using a population size of 100.

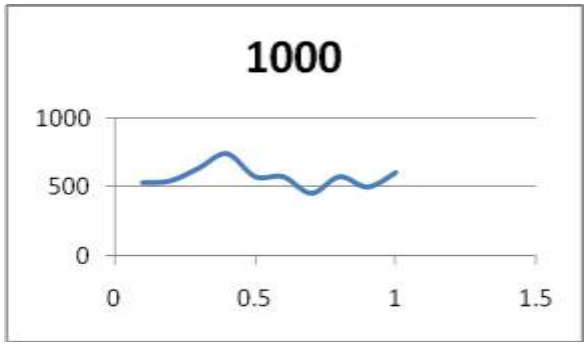


Figure 3: Plot of runtime against mutation rate for Population size of 1000

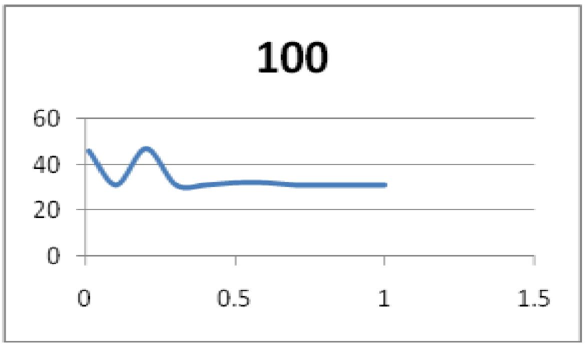


Figure 4: Plot of runtime against mutation rate Population size of 100

Another major thing to note about the algorithm is the number of Individuals which is the result of the population size and the ability to improve candidate solution. Also the more the number Individuals used the higher the likelihood of getting a better solution. Figure 5 shows the plot of Individuals against the distance of tour gotten.

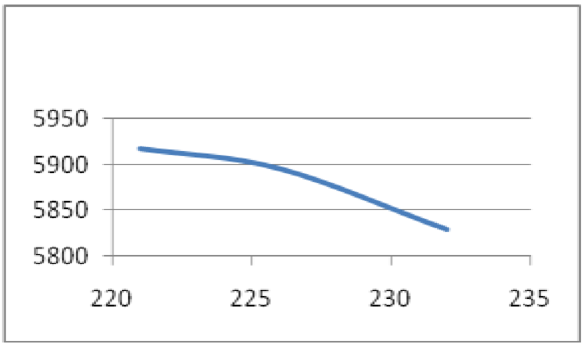


Figure 5: Plot of Individuals against distance

VI. CONCLUSION

We presented an efficient Genetic Algorithm program for solving the Travelling Salesman Problem. The program produced good results for various problem sizes but run time increases with increasing number of cities. We found that the population should be tuned to match the problem size (not arithmetically). To get very good solutions a tradeoff must be made between runtime and the solution quality.

REFERENCES

[1] R. Anthony and E. D. Reilly (1993), Encyclopedia of Computer Science, *Chapman & Hall*.

[2] U. Aybars, K. Serdar, C. Ali, Muhammed C. and Ali A (2009), Genetic Algorithm based solution of TSP on a sphere, *Mathematical and Computational Applications,* Vol. 14, No. 3, pp. 219-228.

[3] B. Korte (1988), Applications of combinatorial optimization, *talk at the 13th International Mathematical Programming Symposium, Tokyo*.

[4] E. L. Lawler, J. K. Lenstra, A. H. G. RinnooyKan, and D. B. Shmoys (1985), The Traveling Salesman Problem, *John Wiley & Sons, Chichester.*

[5] H. Holland (1992), Adaptation in natural and artificial systems, *Cambridge, MA, USA: MIT Press.*

[6] H. Nazif and L.S. Lee (2010), Optimized CrosSover Genetic Algorithm for Vehicle Routing Problem with Time Windows, *American Journal of Applied Sciences 7 (1*): pg. 95-101.

[7] R. Braune, S. Wagner and M. Affenzeller (2005), Applying Genetic Algorithms to the Optimization of Production Planning in a real world Manufacturing Environment, *Institute of Systems Theory and Simulation Johannes Kepler University.*

[8] Z. Ismail, W. Rohaizad and W. Ibrahim (2008), Travelling Salesman problem for solving Petrol Distribution using Simulated Annealing, *American Journal of Applied Sciences 5(11):* 1543-1546.