International Journal of Innovative Technology & Adaptive Management (IJITAM)
ISSN: 2347-3622, Volume-1, Issue-10, July, 2014

A STUDY TO SOLVE TRAVELLING SALESMAN PROBLEM USING GENETIC ALGORITHMS.

[1]Bipasha Biswas Mallick

Assisstant Professor, Haldia Institute of Technology

Abstract: This paper presents the applications of Genetic Algorithm (GA) to solve Travelling Salesman problem (TSP). TSP is a simple to describe and mathematically well characterized problem but it is quite difficult to solve. This is a NP-hard type problem i.e. this problem is hard as the hardest problem in NP-complete space.

1. INTRODUCTION

The TSP is one of the oldest combinatorial/ optimization problem. TSP can be formulated as: A salesman visits n number of cities in such a way that a salesman starts journey by selecting one city among n cities, visits the other cities one by one and returns to the first city from where the journey was started. So the journey of the salesman provides a complete tour that consists of all the cities. Now the resultant TSP is to find the tour with minimum cost. The TSP can be represented by a complete weighted graph G=(V,E) with V being a set of vertices (cities) and E being a set of edges fully connected with the nodes. Each edge (i,j) εE is assigned to a weight d_{ij} which represents the distance between i and j. Formally the goal is to find a Hamiltonian tour of minimal length on a fully connected graph.The problem can have number of feasible solutions but the outcome that will give best result in terms of space and time will be represented as the optimal solution or prove that there is no feasible solution. If optimal solutions cannot be computed efficiently in practice the only possibility is to trade the guarantee of optimality for efficiency.

2. Used Methods

For the solution of optimization tasks it is possible to use various algorithms. Some algorithms give better result and some give worst ones. The tested algorithms are as follows: Exhaustive, Backtracking, Random search, Greedy, Hill climbing, Simulated annealing, Tabu search, Ant colony, Genetic search and Particle Swarms. The Genetic algorithm is relatively new type of optimization technique to solve the problems that are treated as NP-hard problem. This algorithm does not ensure an optimal solution but it gives a good approximation in a reasonable amount of time. This algorithm, therefore, can be a good algorithm to try on travelling salesman problem, which is a famous NP-hard problem.

3. GENETIC ALGORITHM: AN OVERVIEW

Genetic algorithm is based on natural evolution and uses a 'Survival of fittest' technique, where the best solution survives and varied until we get a good result. Genetic algorithm starts working on a randomly generated set of solutions, known as initial population. Fitness is associated with each solution. The fitness evaluation is based on the objective function. In this each string representing the solution is called the chromosome; each bit of string is called the gene. The set of string is called population.

Basic genetic algorithm:

1. **[START]** Generate random population of n chromosomes.

2. **[FITNESS]** Evaluate a fitness function f(x) of each chromosome of x in population.

3. **[NEW POPULATION]** Create a new population by repeating following steps until the new population is complete.

3.i. [Selection] Select two parent chromosomes from a population according to their fitness(the better fitness, the bigger chance to be selected).

3.ii. [Crossover] With a crossover probability cross over the parents to form new offspring. If no crossover is performed the child is the exact copy of parent.

3.iii. [Mutation] With a mutation probability mutate each offspring at each locus (position in chromosome).

3.iv. [Accepting] Place new offspring in a new population. Here is the implementation of above genetic algorithm on GEN() function.

4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**

Function GEN(Initial population) returns an individual

{Current=Initial population;

Repeat;

{ New-generation=Ø

For (i=1; i<=k; i++)

{First=Randomly selected individual

From current according to

Fitness function ;

Second= Randomly selected individual

From current according to

Fitness function ;

New_ind=Crossover (first, second);

If(fitness(New_ind<=threshold)
Mutation(New_ind);

New generation=(New generation U New_ind)}

Current=New generation; }

Until some individual from current fits enough;

Return(The best individual from current according to fitness); }

- Actually chromosome is a data structure, usually a string that represents a candidate solution.
- The population is a collection of chromosomes.

Each chromosome has a numerical fitness, which indicates the quality of chromosome's fitness. There is an example to make understand about how the Genetic algorithm works is shown below:

A. Start from the representation of chromosomes.

Encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution.

Encoding of chromosomes:

Chromosome 1	1101100100111100
Chromosome 2	1101111000011110

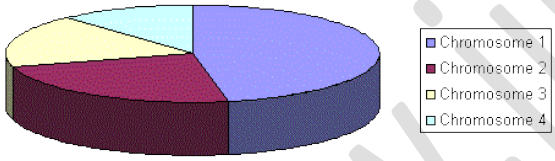
B. Fitness function

The GAs are used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as F(x)=1/f(x), where *f*(x) is the objective function.

C. Selection:

There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others.

C.(i) Roulette Wheel Selection: Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. Imagine a **roulette wheel** where are placed all chromosomes in the population, everyone has its place big accordingly to its fitness function, like on the following picture.



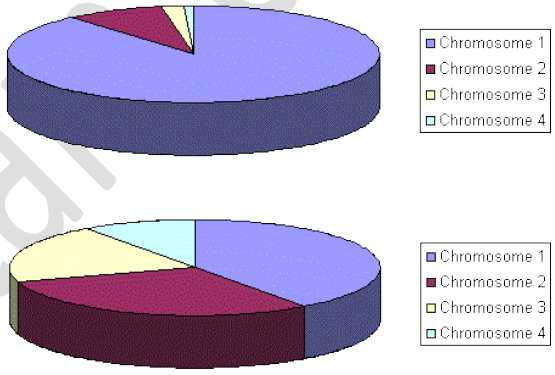
Chromosome with bigger fitness will be selected more times. This can be simulated by following algorithm.

1. **[Sum]** Calculate sum of all chromosome fitnesses in population - sum ***S***.
2. **[Select]** Generate random number from interval ***(0,S)*** - ***r***.
3. **[Loop]** Go through the population and sum fitnesses from ***0*** - sum ***s***. When the sum ***s*** is greater than ***r***, stop and return the chromosome where you are.

Of course, step **1** is performed only once for each population.

C. (ii) Rank selection:

Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking. The previous selection will have problems when the finesses differ very much. For example, if the best chromosome fitness is 90% of the entire roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness ***1***, second worst ***2*** etc. and the best will have fitness ***N*** (number of chromosomes in population). You can see in following picture, how the situation changes after changing fitness to order number.



C. (iii) Elitist selection:

The fit members of each generation are guaranteed to be selected.

D. Reproduction

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation.

1. **Methods of Reproduction:- Select parents for the mating pool**

   (Size of mating pool = population size)

2. **Shuffle the mating pool**

3. **For each consecutive pair apply crossover with probability p_c , otherwise copy parents**
4. **For each offspring apply mutation (bit-flip with probability p_m independently for each bit)**
5. **Replace the whole population with the resulting offspring**

Crossover and Mutation

a. Crossover operation:

Crossover can look like this (| is the crossover point)

Chromosome 1	1101100100111100	
Chromosome 2	1101111000011110	
Offspring 1	11011	11000011110
Offspring 2	11011	00100111100

There are other ways to make crossover; for example we can choose more crossover points. Specific crossover for specific problem can improve performance of the genetic algorithm.

b. Mutation operations:

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search as helps to prevent the population from stagnating at any local optima. Mutation occurs during evolution according to a user-definable mutation probability. This probability should usually be set fairly low (0.01 is a good first choice). If it is set to high, the search will turn into a primitive random search. There are different types of mutation.

Bit Flip -A mutation operator that simply inverts the value of the chosen gene (0 goes to 1 and 1 goes to 0). This mutation operator can only be used for binary genes.There are other ways to make Mutation operation. Such as,

Original Offspring 1	1101111000011110
Original Offspring 2	1101100100110110
Mutated Offspring 1	1100111000011110
Mutated Offspring2	1101101100110110

Boundary(A mutation operator that replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly))

Non-Uniform (A mutation operator that increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution then allows the genetic algorithm to fine tune the solution in the later stages of evolution. This mutation operator can only be used for integer and float genes)

Uniform (A mutation operator that replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. This mutation operator can only be used for integer and float genes)

Gaussian (A mutation operator that adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. This mutation operator can only be used for integer and float genes).

Here an example of Bit-Flip operation is shown below.

E. Recommendations

Recommendations are often results of empiric studies of GAs that were often performed on binary encoding only.

• **Crossover rate**: Crossover rate should be high generally, about **80%-95%**. (However some results show that for some problems crossover rate about 60% is the best.)

• **Mutation rate**: On the other side, mutation rate should be very low. Best rates seems to be about **0.5%-1%**

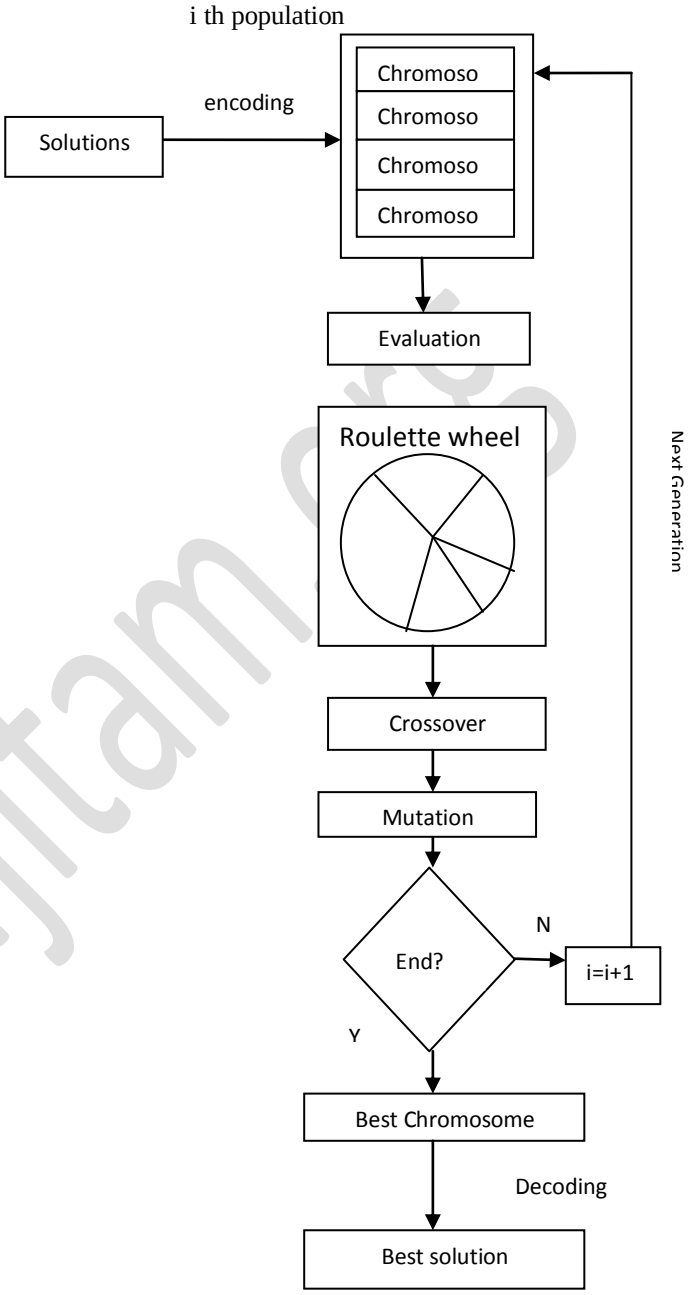
.• **Population size**: It may be surprising, that very big population size usually does not improve performance of GA (in the sense of speed of finding solution). Good population size is about **20-**

30,however sometimes sizes 50-100 are reported as the best. Some research also shows, that the best population size depends on the **size of encoded string** (chromosomes). It means that if you have chromosomes with 32 bits, the population should be higher than for chromosomes with 16 bits.

• **Selection**: Basic **roulette wheel selection** can be used, but sometimes rank selection can be better. There are also some more sophisticated methods that change parameters of selection during the run of GA. Basically, these behave similarly like simulated annealing.

Elitism should be used for sure if you do not use other method for saving the best found solution.

- **Encoding**: Encoding **depends on the problem** and also on the size of instance of the problem.**Crossover and mutation type.** Operators depend on the chosen encoding and on the problem.



Flowchart of Genetic Algorithm

4. GENETIC ALGORITHM AS A METHOD TO SOLVE TRAVELLING SALESMAN PROBLEM: The traveling salesman starts at one node, visits all other nodes successively only one time each, and finally returns to the starting node.

i.e., given n cities, named {c1, c2, ……., cn}, and permutations, σ1, ..., σn!, the objective is to choose σi such that the sum of all Euclidean distances between each node and its successor is minimized. The successor of the last node in the permutation is the first one. The Euclidean distance d, between any two cities with coordinate (x1, y1) and (x2, y2) is calculated by:

$$d= ((x_1 – x_2)^2+ (y_1-y_2)^2)^{1/2}$$

An implicit way of solving the TSP is simply to list all the feasible solutions, evaluate their objective function values and pick out the best. However it is obvious that this "exhaustive search" is grossly inefficient and impracticable because of vast number of possible solutions to the TSP even for problem of moderate size. Since practical applications require solving larger problems, hence emphasis has shifted from the aim of finding exactly optimal solutions to TSP, to the aim of getting, heuristically, "good solutions‟ in reasonable time and "establishing the degree of goodness‟ . Genetic algorithm (GA) is one of the best heuristic algorithms that have been used widely to solve the TSP instances. A genetic algorithm can be used to find a solution in much less time. Although it might not find the best solution, it can find a near perfect solution for a 100 city tour in less than a minute.

To solve the traveling salesman problem, we need a list of city locations and distances, or cost, between each of them. The following basic steps are used to solve the traveling salesman problem using a GA.

a. • First, create a group of many random tours in what is called a population. This algorithm uses a greedy initial population that gives preference to linking cities that are close to each other.
b. • Second, pick 2 of the better (shorter) tours parents in the population and combine them to make 2 new child tours. Hopefully, these children tour will be better than either parent.
c. • A small percentage of the time, the child tours is mutated. This is done to prevent all tours in the population from looking identical.
d. • The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same.
e. • New children tours are repeatedly created until the desired goal is reached.

To apply GA for any optimization problem, one has to think a way for encoding solutions as feasible chromosomes so that the crossovers of feasible chromosomes result in feasible chromosomes. The techniques for encoding solutions vary by problem and, involve a certain amount of art. For the TSP, solution is typically represented by chromosome of length as the number of nodes in the problem.

4. a) Representation:

Representation is an ordered list of city numbers known as an *order-based* GA.

1) London 3) Dunedin 5) Beijing 7) Tokyo

2) Venice 4) Singapore 6) Phoenix 8) Victoria

CityList1 (3 5 7 2 1 6 4 8)

CityList2 (2 5 7 6 8 1 3 4)

4. b) Crossover: Crossover combines inversion and recombination:

* *

Parent1	3	5	7 2 1 6	4	8
Parent2	2	5	7 6 8 1	3	4
Child	5	8	7 2 1 6	3	4

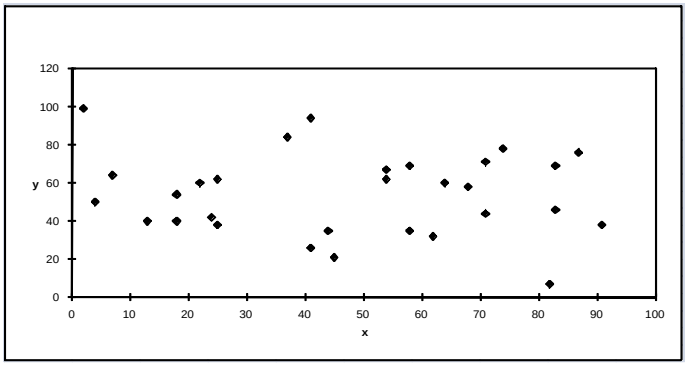
This operator is called the *Order1* crossover.

4. c) Mutation : Mutation involves reordering of the list:

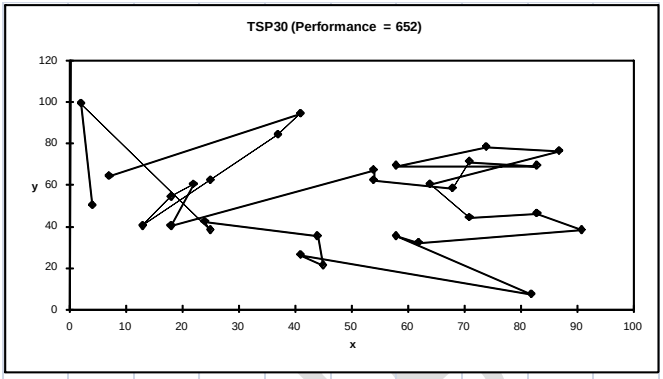
* *

Before	5	8	7	2	1	6	3	4
After	5	8	6	2	1	7	3	4

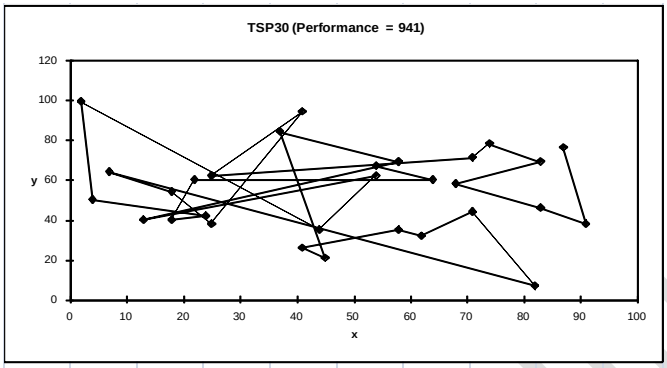
TSP Example: 30 Cities



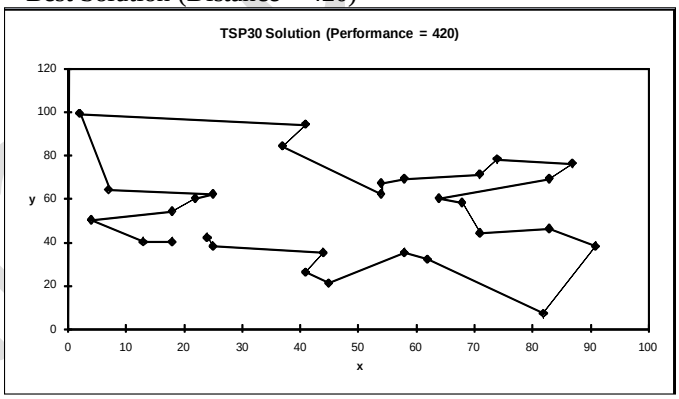
Solution $_k$(Distance = 652)



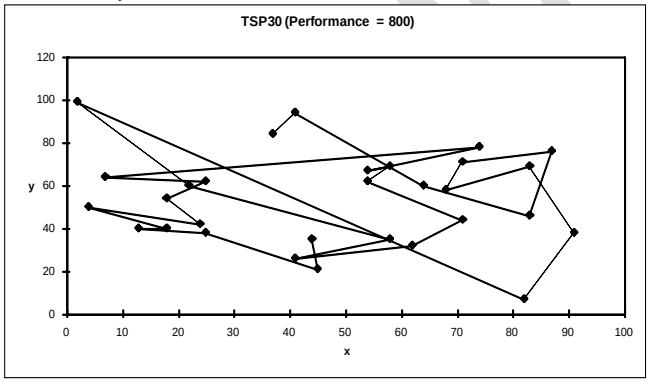
Solution $_i$(Distance=941)



Best Solution (Distance = 420)



Solution $_j$(Distance = 800)



5. CONTROL PARAMETERS: These are the parameters that govern the GA search process. Some of them are: (a) Population Size: - It determines how many chromosomes and thereafter, how much genetic material is available for use during the search. If there is too little, the search has no chance to adequately cover the space. If there is too much, the GA wastes time evaluating chromosomes. (b) Crossover probability: - It specifies the probability of crossover occurring between two chromosomes. (c) Mutation probability: - It specifies the probability of doing bit-wise mutation. (d) Termination criteria: - It specifies when to terminate the genetic search.

6. CONCLUSION: In this paper we have discussed the travelling salesman problem using Genetic Algorithm. Various techniques of genetic algorithm have been discussed in this paper to study travelling salesman problem which is a permutation problem in which goal is to find the shortest path between cities traversing each city at least once. This paper gives a solution to find an optimum route for traveling salesman problem using Genetic algorithm technique, in which cities are selected randomly as initial population. The new generations are then created repeatedly until the proper path is reached upon reaching the stopping criteria.

7. REFERENCES:

- L. Davis. "Job-shop Scheduling with Genetic Algorithms". Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985. Zakir H. Ahmed International Journal of Biometrics & Bioinformatics (IJBB) Volume (3): Issue (6) 105
- I.M. Oliver, D. J. Smith and J.R.C. Holland. "A Study of Permutation Crossover Operators on the Travelling Salesman Problem". In J.J. Grefenstette (ed.). Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hilladale, NJ, 1987.
- http://en.wikipedia.org/wiki/Genetic_algorithm
- http://www.obitko.com/tutorials/genetic-algorithms/index.php http://www.talkorigins.org/faqs/genalg/genalg.html#examples:systems
- D.E.Goldberg, (1989) "Genetic Algorithms in search,Optimization and Machine Learning", AddisonWesley Publishing Company, Ind. U.S.A.
- Melanie Mitche,(1998) "An introduction to genetic Algorithm", MIT press.
- Namita Khurana, Anju Rathi,Akshatha P S,(2011) "Genetic Algorithm: A search of Complex, IJCA.
- http://en.wikipedia.org/wiki/Genetic_algorithm.
- Emanuel Falkenauer(1998) Genetic Algorithms and Grouping Problems. JohnWiley andSons.
- Kangshun Li,, Lanlan Kang, Wensheng Zhang, Bing Li, Comparative Analysis of Genetic Algorithm and Ant Colony Algorithm on Solving
- Traveling Salesman Problem, IEEE International Workshop on Semantic Computing and Systems
- "Yasuhiro TSUJIMURA and Mitsuo GEN",(21-23 April 1998) Entropy-based Genetic Algorithm for Solving TSP, 1998 Second International
- Conference on Knowledge-Based Intelligent Electronic Systems, Adelaide, Australia. Editors, L.C. Jain and R.K.Jain
- "D. KAUR, M. M. MURUGAPPAN" Performance Enhancement in solving Traveling Salesman Problem using Hybrid Genetic Algorithm,
- Vijendra Singh and Simran Choudhary (2009) , Genetic Algorithm for Traveling Salesman Problem: Using Modified Partially-Mapped Crossover
- Operator, IMPACT.
- Gen, 31. and R. Cheng(1997) Genetic Algorithms and Engineering Design. John Wiely & Sons.
- http://www.darwins-theory-of-evolution .com/
- http://www.obitko.com/tutorials/genetic-algorithms/encoding.ph