Genetic Access on Combinatorial Problems

C Prasanna Ranjith
Al Jabal Al Gharbi University
Gharyan, Libya
pr74_99@yahoo.com

Taher Omran Ahmed
Al Jabal Al Gharbi University
Gharyan, Libya
fenneer@yahoo.com

Abstract – Genetic Algorithms are adaptive methods, which may be used to solve search and optimization problems. They are based on the genetic processes of biological organisms that evolve according to the principles of natural selection and "survival of the fittest". By mimicking this process, Genetic Algorithms are able to "evolve" solutions to combinatorial problems.

The Objective of this paper is to consider Shortest Path Routing Problem and solve it using Greedy and Exhaustive algorithms. The strengths and weaknesses of both the algorithms are studied. A simple algorithm is proposed using the concept of GA that aims to cure all the infeasible chromosomes with a simple repair function. Analysis shows that the proposed algorithm exhibits a much better quality of solution and a much higher rate of convergence.

INTRODUCTION

Many researchers have applied Genetic Algorithm to the Shortest Path Routing problem, multicasting routing problem, ATM bandwidth allocation problem, and the dynamic routing problem. It is noted that all these problems can be formulated as some sort of a combinatorial optimization problem. To solve large scale optimization problems, we need new computing infrastructure which enables us to easily access to computational resources including hardware and software library distributed across a wide area network like the Internet. For example, Applegate et al. [1] implemented the Danzig, Fulkerson and Johnson's cutting plane method for the large scale TSP (travelling salesman problem). They obtained the optimal solution of the TSP that has 15,112 cities (nodes) in Germany. The computation was executed on a network of 110 processors located at Rice and Princeton Universities. They estimated the total computation time was 22.6 years, scaled to a Compaq EV6 (21264) Alpha processor running at 500MHz.

There are many algorithmic techniques like Brute-Force Algorithms, Divide-and-Conquer Algorithms, Dynamic Programming, Greedy Algorithms, Genetic Algorithms, etc. Each of the techniques has special properties that make them appropriate for solving certain types of problems.

The objective of this paper is to prove that Genetic Algorithm works well on combinatorial problems than traditionally used algorithms. This is done taking Shortest Path Routing Problem as an example. The problem is solved using Exhaustive method, Greedy Method and Genetic Algorithm, which has been solved using practical examples. This paper also provides information about the merits and demerits of using the different algorithms and justifies Genetic Algorithm.

Problem Specification

The problem is primarily concerned with using Genetic Algorithms and other Heuristic algorithms to solve Shortest Path Routing Problem.

Shortest Path Routing Problem

The underlying topology of multihop networks can be specified by the directed graph G = {N, A}, where N is a set of nodes (vertices), and A is a set of its links (arcs or edges). There is a cost C_{ij} associated with each link {i,j}. The costs are specified by the cost matrix C = | C_{ij} |, where C_{ij} denotes a cost of transmitting a packet on link {i,j}. Each link has the link connection indicator denoted by, which plays the role of a chromosome map (masking) providing information on whether the link from node i to node j is included in a routing path or not.

Methods Adopted

Shortest Path problem is solved using the following methods:

Exhaustive Algorithm
Greedy Algorithm
Genetic Algorithm

Exhaustive Algorithm

The N-node Shortest Path Routing problem is solved by systematically considering all the permutations of the given n nodes. Then, the minimum cost network is selected by comparing the costs. Exhaustive Algorithm requires as input, the number of nodes (N) and the cost matrix (C).

Algorithm EA (Exhaustive Algorithm) To find the Shortest path route by systematically considering all the permutations on the first N-1 positive integers. In this way generate all possible routes and choose ROUTE with the least cost MIN. Exhaustive requires as input the number of nodes N and the cost matrix C.

Step 0. [Initialize] Set ROUTE = ∅;
and MIN = ∝

Step 1. [Generate all permutations] For I = 1 to (N-1)! do through step 4 od; and STOP.

Step 2. [Get new Permutation]
Set P=the Ith permutation of integers 1,2,..N-1

Step 3. [Construct new Route] Construct the route R(P) that corresponds to the permutation P; and compute the cost COST(R(P)).

Step 4. [Compare] If COST(R(P)) < MIN the set ROUTE = R(P) and MIN = COST(R(P)) fi.

Greedy Algorithm

Greedy Algorithm are usually very fast and intuitively appealing. It is based on Hill-climbing idea. The goal is to find a minimum-cost network. It is one of the Heuristics Methods of solving a problem. Heuristic is defined as having the following two properties:

1. It will usually find good, although not necessarily, optimum solutions.

2. It is faster and easier to implement than any other known exact algorithms.

Algorithm GREEDY: To construct a candidate least-cost route ROUTE, which has a cost COST, for a N-node Shortest Path Problem with cost matrix C, starting at vertex U.

Step 0. [Initialize] Set ROUTE = ∅ ; COST = 0; V = U; label U "used" and all other vertices "unsed". (Vertex V is the present position in the network.)
Step 1. [Visit all cities] For K = 1 to N-1 do through step 2 od.

Step 2. [Choose next edge] Let (V,W) be the least costly edge from V to any unused vertex W;
Set ROUTE = ROUTE + (V,W);
COST = COST + C(V,W); label W "used"; and
set V = W;

Step 3. [Complete ROUTE] set ROUTE = ROUTE + (V,1); COST = COST + C(V,1);and STOP.

Genetic Algorithm

Genetic Algorithms are search algorithms based on the mechanics of Natural selection & natural genetic. They combine the fittest among string structures with structured yet randomized information exchange to form a search algorithm. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old. An occasional new part is tried for good measure[2].

Algorithm is started with a **set of solutions** (represented by **chromosomes**) called **population**. Solutions from one population are taken and used to form a new population. This is done in a hope, that the new population will be better than the old one.

Solutions which are selected to form new solutions (**offspring**) are selected according to their fitness - the more suitable they are the more chances they have to reproduce. This is repeated until some condition is satisfied.

Outline of Genetic Algorithm: [2]

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)

2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population

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

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

b. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.

c. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).

d. **[Accepting]** Place new offspring in a new population

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**

Encoding of a Chromosome

The chromosome should in some way contain information about solution. The mostly encoding is done with a binary string.
Example : The chromosome:

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution.

For direct integer or real numbers is also used to encode.

Crossover

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way how to do this is to choose randomly some crossover point and everything before this point copy from a first parent and then everything after a crossover point copy from the second parent.

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

Chromosome 1	**11011**| **00100110110**
Chromosome 2	11011 | 11000011110
Offspring 1	**11011** | 11000011110
Offspring 2	11011 | **00100110110**

There are other ways how to make crossover, for example we can choose more crossover points.

Specific crossover made for a specific problem can improve performance of the genetic algorithm.

Mutation

This is to prevent falling all solutions in population into a local optimum of solved problem.

Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1.

Mutation can then be following:

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

The mutation depends on the encoding as well as the crossover. For example when we are encoding permutations, mutation could be exchanging two genes.

FINDINGS

Exhaustive Algorithm

This algorithm is exponential. In an n-node problem, Algorithm ETS requires exhaustive permutation of the first n-1 positive integers. There are around (n-1)! of these permutations. Once a

permutation is presented, it is possible to find the corresponding route and its cost. The route and cost are stored. As all the permutations are presented with all the corresponding routes and costs, the shortest path route is generated on comparison.

For example if N=5, it generates 120 routes, its costs and by comparing the cost of 120 routes, it gives all the routes with minimum cost. If N = 6, then, 720 Routes are generated. If N = 10, then, 36,28,800 ROUTEs are to be generated.

As an example Consider a 7 node problem, whose cost matrix is defined I n table-1:

Nodes	0	1	2	3	4	5	6
0	0	6	12	6	4	8	1
1	6	0	10	5	4	3	3
2	8	7	0	11	3	11	8
3	5	4	11	0	5	8	6
4	5	2	7	8	0	3	7
5	6	3	11	5	4	0	2
6	2	3	9	7	4	3	0

Table-1: Cost Matrix

When the above problem is solved using this algorithm, it suggests 6 routes with a Minimum cost of 28 out of 5040 routes that were generated.

Greedy Algorithm

Greedy Algorithms are usually very fast and intuitively appealing, but they do not always work. Algorithm GTS is certainly easy to program but they are not that fast. To solve a Shortest path Problem it takes O (n^2) operations [4].

Algorithm may prove to be good for a reasonable large value of n. For example if N=5, then the algorithm finds 5 different ROUTEs having each node as starting node and its corresponding costs. Then by comparison, the ROUTE with least cost is produced as output. The total number of possible ROUTEs that can be generated for a 5-node problem is 120. But, Greedy generates only 5 routes and selects one out of it. Hence, there are possibilities of better solutions, which are ignored.

Another problem with this Greedy Algorithm is the greedy way of selecting the next node from a visited node. By doing so, there may be a possibility to select node with larger costs during the completion of the ROUTE [3].

For Example Consider the same 7 node problem, whose cost matrix is defined in table-1

The solution for this problem is

4 **à** 1 **à** 5 **à** 6 **à** 0 **à** 3 **à** 2 **à** 4

Total Min.Cost = 29

Genetic Algorithm

ALGORITHM SPGA (SHORTEST PATH GA)

Considering the strengths and weaknesses of both the algorithms, a new algorithm is developed. This algorithm produces an optimum solution in a fewer steps.

Algorithm SPGA (Shortest Path Genetic Algorithm): To construct a candidate least –cost ROUTE ROUTE, which has a cost COST, for an N-node shortest path routing problem with cost matrix C starting at vertex U

Step0: [Initialize] set ROUTE **ß** ∅; COST **ß** 0; Label U "used" and all other vertices "unused" (Vertex V is the present position in the network)

Step1: [Visit all nodes] For K **ß** 1 to N-1 do step2 od;

Step2: [Choose next edge] Let(v,w) be the least cost edge from V to any unused vertex W. Set ROUTE **ß** ROUTE + (V,W) COST **ß** COST+C(V,W); label W "used" and set V **ß** W.

Step3: [Complete route] Set ROUTE **ß** ROUTE+(V,1); COST **ß** COST+C(V,1);

Step4: [Initialize] COST1 **ß** 0; COST2 **ß** 0

Step5: [Assign ROUTE to S] S=ROUTE;

Step6: [Divide S] Set X = S/2;

Step7:[Visit nodes] For I=1 to X+1 do COST1 **ß** COST1+COST(I)

Step8:[Visit nodes] For J=X+1 to N do COST2 **ß** COST2+COST(J)

Step9:[Compare] If COST1>COST2 then call ETS(2,X+1) Else call ETS(X+1,N-1) fi

On the whole the algorithm can be viewed as two phases, whose functionality are explained below:

Phase-I (SELECTION of the fittest PARENT)

The above algorithm initially selects a *Parent Route* by applying a simple *Fitness function*. Here, the fittest of the Route is selected by using the Greedy method.

Phase-II (Applying Genetic Operator on the fittest PARENT)

Now the Parent Route is divided into two parts. The cost of each parts are calculated. The costs are compared and the part that has the maximum cost is considered for repair. Applying Algorithm EA on this part of the route above basically does the Repair. As a result the best route found by Algorithm EA is replaced in the original parent, which is now the offspring. Cost of the offspring is found out.

Here the Algorithm **SPGA** reduces the complexity and gives the optimum. Considering the example problem, the algorithm produces the following result:

Min.Total Cost = 28 with 13 iterations

COMPARISON

From the examples above, it is clear that Algorithm EA (Exhaustive) gives the Shortest Path route out of 5040 routes, Algorithm Greedy gives the Shortest Path route out of 7 routes and Algorithm SPGA gives the Shortest Path route out of 13 routes as shown in the following table-3.

Method	No. of Nodes	No. Routes Generated	Cost of Best Route
Exhaustive	7	5040	28
Greedy	7	7	29
SPGA	7	13	28

Table-3: Comparison of the Methods

CONCLUSION

On observation, it is clear that Genetic Algorithm works better on Combinatorial Problems. Shortest Path Routing Problem being a good example for Combinatorial Problems work well when solved using Genetic Algorithm than the other two methods.

This is only a simple attempt towards optimum solution in a fewer iterations by fusing both Greedy and Exhaustive algorithms which is designed and tested. Even better results can be achieved by developing more sophisticated algorithm. Shortest Path Routing problem works well when solved using SPGA Algorithm. This innovative algorithm also needs less processing time and storage.

REFERENCES

[1] Applegate, D., R. Bixby, V. Chvatal and W. Cook, "Implementing the Dantzig-Fulkerson-Johnson Algorithm for Large Traveling Salesman Problems", *Mathematical Programming, vol.* 97, 2003.

[2] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston: Addison –Wesley, 1989.

[3] Goodman, S.E. & S.T. Hedetniemi, *Introduction to the Design and Analysis of Algorithm*. New York: McGraw-Hill, 1977.

[4] Heilmen, Gregory L., *Data Structures, Algorithms, and Object-Oriented Programming.* New Delhi: Tata McGraw-Hill, 2002.