International Journal of Soft Computing 5 (4): 171-176, 2010
ISSN: 1816-9503
© Medwell Journals, 2010

Network Design Problem Using Genetic Algorithm-an Empirical Study on Mutation Operator

[1]Anand Kumar and [2]N.N. Jani
[1]Department of Master of Computer Applications, AMC Engineering College, Bangalore, India
[2]Faculty of Computer Studies, Kadi Sarva Vishwavidyalya, Gandhinagar

Abstract: This study presents an influence of mutation operator in genetic algorithm for small to large network design problem. A network design problem for this study falls under the network topology category which is a minimum spanning tree with various types of constraint which makes it NP-hard problem. Mutation operator plays an important role in genetic algorithm approach. Since many researchers have tried to solve this problem for small to mid size, we have explored the use of genetic algorithm with various mutation functions with modification but without changing the nature of genetic algorithm. Various mutation functions have been developed here as per the requirement of the problem and applied with the various size of network. In this study we have tried to show that how mutation functions affects the performance of genetic algorithm and also shown that GA is an alternative solution for this NP-hard problem.

Key words: Genetic algorithm, network design, mutation operator minimum spanning tree, performance, approch, constraints

INTRODUCTION

In genetic algorithms of computing, mutation is a genetic operator used to maintain genetic diversity from one generation of a population of algorithm chromosomes to the next. It is analogous to biological mutation. The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence.

This random variable tells whether or not a particular bit will be modified. This mutation procedure, based on the biological point mutation is called single point mutation. Other types are inversion and floating point mutation. When the gene encoding is restrictive as in permutation problems, mutations are swaps, inversions and scrambles.

The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fitness of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.

There are many mutation schemes for Genetic Algorithms (Gas) each with different characteristics. Since the nature of genetic algorithm is very uncertain, various mutation operators can be used to derive optimal result. This study presents the influence of various types of mutation operators with various size of network and it is the extension of the research work Network design problem (Kumar and Jani, 2010). This problem is one of the hardest problems in NP-hard category.

There are no traditional methods available to solve this problem. A genetic algorithm approach to design the network is one of the ultimate solutions because traditional heuristics has the limited success. Researchers in operation research have examined this problem under the broad category of minimum cost flow problem (Taha, 2007). A simple GA approach is applied by many researchers (Basu, 2005; Melanie, 1998; Vose, 1999) but in this study the influence of mutation function in genetic algorithm is shown.

Genetic algorithms are being used extensively in optimization problem as an alternative to traditional heuristics. It is an appealing idea that the natural concepts of evolution may be borrowed for use as a computational optimization technique which is based on the principle Survival of the fittest given by Darvin. We have tried to show that the influence of mutation function and the little variation in genetic algorithm approach is very effective.

Corresponding Author: Anand Kumar, Department of Master of Computer Applications, AMC Engineering College, Bangalore, India

Network design: In this study network design is considered as network topology which is a spanning tree consists of various nodes considered as vertex. A tree is a connected graph containing no cycles. A tree of a general undirected graph G = (V, E) with a node (or vertex) set V and edge set E is a connected subgraph T = (V, E) containing no cycles with (n-1) edges where n is total no of node.

In this study undirected networks are considered with the weight (distance) associated with each node. For a given connected, undirected graph G with n nodes, a minimum spanning tree T is a sub graph of a G that connects all of G's nodes and contains no cycles (Deo, 2000). When every edge (i, j) is associated with a distance cij , a minimum spanning tree is a spanning tree of the smallest possible total edge cost:

$$C = \Sigma c_{ij}$$

Where (i, j). $\in$ T

Genetic algorithm: Genetic Algorithms (GA) is a powerful, robust search and optimization tool which work on the natural concept of evolution, based on natural genetics and natural selection:

Work flow of GA:

- Initialisation of parent population
- Evaluation
  - Self loop check
  - Isolated node or edge check
  - Cycle check
  - Store the best result
- Selection of child population
- Apply crossover/Recombination
- Evaluation
- Replace the result if it is better than previously stored
- Apply mutation
- Evaluation
- Replace the result if it is better than previously stored
- Go to step 3 until termination criteria satisfies

NETWORK DESIGN PROBLEM PRESENTATION AND ITS SOLUTION USING GENETIC ALGORITHM APPROACH

The Network design problem is considered as a unidirectional graph and represented with the help of adjacency matrix.

Parent population in the form of chromosome is generated randomly according to the size of network. Number of gene in a chromosome is equal to number of node in a network.

The total number of chromosome may vary and it is based on user input. Here a chromosome is generated for a 10 node network. The association between nodes is considered between positions to position.

Node: 1 2 3 4 5 6 7 8 9 10
Chromosome: 2 10 4 9 6 7 5 9 8 3

The logic behind association is that the node (Taha, 2007) is connected with node 2; node (Basu, 2005) is connected with 10 and so on. From Fig. 1 it is clear that this is not a spanning tree because of the isolated circle. Similarly with some other randomly generated chromosome, some other problems have been observed. By observing these problems it has been concluded that there are three main reasons for illegal chromosome:

- Self loop
- Cycle and
- Isolated node or edge

Evaluation: By observing these problems, fitness functions have been developed (Kumar and Jani, 2010) to evaluate these chromosomes. On the basis of these fitness functions, fitness points are given to the chromosomes and on the basis of these fitness points chromosomes are selected as a child population for next generation. Following fitness functions have been developed to evaluate chromosomes:

- Self loop
- Isolated node or edge
- Cycle

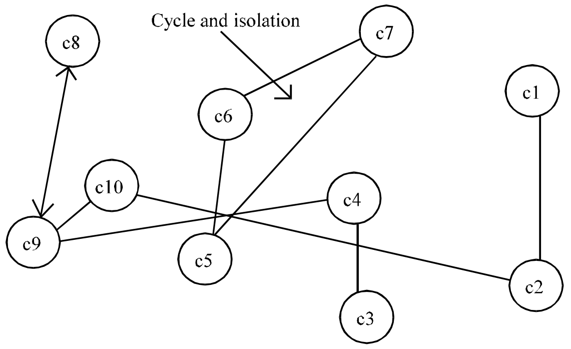

Fig. 1: Isolated circle

MUTATION

This is the main part of this study. Mutation is a background operator which produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. In GA, mutation serves the crucial role of either (a) replacing the genes lost from the population during the selection process so that they can be tried in a new context or (b) providing the genes that were not present in the initial population. The mutation probability (denoted by Pm) is defined as the percentage of the total number of genes in the population.

The mutation probability controls the probability with which new genes are introduced into the population for trial. If it is too low, many genes that would have been useful are never tried out while if it is too high, there will be much random perturbation, the offspring will start losing their resemblance to the parents and the algorithm (Knuth, 1997) will lose the ability to learn from the history of the search.

Up to now, several mutation operators have been proposed for real numbers encoding which can roughly be put into four classes as crossover can be classified. Random mutation operators such as uniform mutation, boundary mutation and plain mutation belong to the conventional mutation operators which simply replace a gene with a randomly selected real number with a specified range.

Dynamic mutation (non uniform mutation) is designed for fine-tuning capabilities aimed at achieving high precision which is classified as the arithmetical mutation operator. Directional mutation operator is a kind of direction-based mutation which uses the gradient expansion of objective function.

The direction can be given randomly as a free direction to avoid the chromosomes jamming into a corner. If the chromosome is near the boundary, the mutation direction given by some criteria might point toward the close boundary and then jamming could occur. Several mutation operators for integer encoding have been proposed. Inversion mutation selects two positions within a chromosome at random and then inverts the substring between these two positions.

- Insertion mutation selects a gene at random and inserts it in a random position
- Displacement mutation selects a substring of genes at random and inserts it in a random position Therefore, insertion can be viewed as a special case of displacement. Reciprocal exchange mutation selects two positions random and then swaps the genes on the positions

```
               ↓
Parent: 1 0 0 1 1 1 0 1 0 1
Child:  1 0 0 1 0 1 0 1 0 1
```

Following mutation operators have been developed and tested for different size of network. Following data structure have been used for all these developed functions.

Chromosomes: it is a matrix of size N X N to store N randomly generated chromosomes. After the fitness function, fitness point for each chromosome is stored in its last column. Subscript starts from 1.

S (1): It holds the no of row of matrix chromosome.

S (2): It holds the no of column of matrix chromosome.

Fitness: It is an array which holds the fitness value for each chromosome.

Mutation I: This mutation operator mutates only those chromosomes which does not have the maximum fitness. The logic applied behind this function is to simply find the chromosome and change its value with its position. If first chromosome is selected then its first place will be replaced by maximum number where maximum number is equal to number of node. Similarly if second unfit chromosome is selected then its second position will be replaced by maximum number-1 and so on.

```
Mutation1 (chromosome)
Begin
Set k = 1;
for i = 1 to row do
if (chromosome(i, col) not equal
  to maximum fitness)
      new-chromosome (i, i) = (col-i); end
end
for i = 1 to row
      for j = 1 to col-1 do mutated_chromosome (i, j) = new-chromosom
e(i, j); end
      end
End
```

Mutation II: This mutation operator mutates only those chromosomes which does not have the maximum fitness value. Mutation is done to remove self loop. If the locus and allele both have the same vlaue than this value is replaced by (position+1). This function is also working as the repairing of chromosome.

```
Mutation II (chromosome)
Begin
set k = 1;
for i = 1 to row do
        if (chromosome (i, col) not equal
to maximum fitness)
              for j = 1 to col-1 do
                   if (chromosome (i,j) = = j)
                        if (j equal to col-1)
new-chromosome (i, j) = j-1;
                         else
new-chromosome (i,j) = j+1;
                                end
                         end
                  end
          end
                    end
for i = 1 to row do
     for j = 1 to col-1
          mutated-chromosome(i,j) =
new-chromosome(i,j);
     end
end
End
```

Random mutation: This mutationo peratormu tatesonly those chromosomes which does not have the maximum fitness value. Mutation is done by selecting a random position and replace its value with random number. It is considered that no self loop could form at the time of replacement.

```
Random_mutation (chromosome)
Begin
set k = 1;
for i = 1 to row do
       if (chromosome (i, col) not equal
to maximum fitness)
                  posi = randomly generated
number within limit;
                  val = randomly generated
number within limit;
                  if (posi equal to 0)
                        posi = 1;
                   end
                   if (val equal to 0)
                    val = 1;
               end
if ((posi equal to val) AND (posi = =col-1))
         chromosome (I, posi) = val-1;
                       else
                     chromosome (i,posi) = val;
                   end
             end
       end
for i = 1 to row do
        for j = 1 to col-1
              mutated_chromosome (i, j) =
new_chromosome(i,j);
        end
end
End
```

Swap mutation: This mutation operator swaps two random position of each of the chromosomes. If the randomly generated positions are 3 and 7.

```
                    ↓         ↓
Chromosome: 5 1 4 9 8 2 1 3 1 0 1
```

After mutation

Chromosome: 5 1 1 9 8 2 4 3 1 0 1

```
Swap_mutation (new_chromosome)
Begin
    Set k = 1;
          for i = 1 to row do

       p =  randomly generated number
within the limit;
      q = randomly generated number within the limit;
                   temp = new_chromosome (i, p);
                   new_chromosome (i, p) =
new_chromosome (i, q);
                 new_chromosome (i, q) = temp;
  end
End
```

Mutation inversion: This mutation operator inverts the genes between two random position for each of the chromosomes. For each chromosome there are different random position. If the randomly generated positions are 2 and 8.

```
                  ↓             ↓
Chromosome: 5 1 4 9 8 2 1 3 1 0 1
```

After mutation

Chromosome: 5 3 1 2 8 9 4 1 1 0 1

```
Mutation inversion(new_chromosome)
Begin
Set k = 1;
      for i = 1 to row do
       p =  randomly generated number
within the limit;
         q = randomly generated number
within the limit;
        sort p,q
        for x = p to q do
             temp = new_chromosome (i, x);
                new_chromosome (i, x) =
new_chromosome (i,q);
           new_chromosome (i, q) = temp;
           decrement q by - 1;
           if (x  = = q) || (x>q)
               break;
          end
      end
end
 End
```

Table 1: Minimum cost of network for various mutation operators

Network size	Random mutation	Mutation I	Mutation II	Swap mutation	Inversion mutation	Insertion mutation
10	226	281	247	241	266	234
20	624	682	646	567	652	680
40	1262	1293	1388	1281	1238	1306
60	2028	2026	2189	1977	2140	2203
80	2981	2944	3121	2999	2933	2813
100	3632	3555	3738	3464	3455	3368
200	7730	7390	7353	7561	7344	7407
300	11387	11305	11359	11559	11228	11300
400	15454	15401	15815	15066	15252	15337
500	19245	19296	19297	19256	19240	19295
600	23589	23315	22999	23440	23095	23246
700	27200	27421	28198	27626	26860	27234
800	32002	31335	31266	31251	30852	30833
900	35184	34643	35156	35383	35027	35294
1000	39172	39092	39315	39291	39100	39503

Mutation insertion: This mutation operator inserts one gene with another gene by displacing other genes. Two random positions are generated to denote two gene then one random place gene is inserted with the another random place gene. Other inbetween genes are shifted. For each chromosome there are different random position. If the randomly generated positions are 2 and 8.

```
                    ↓             ↓
Chromosome: 5 1 4 9 8 2 1 3 1 0 1
After mutation
                      ↓ →
Chromosome: 5 1 3 4 9 8 2 1 1 0 1
```

```
Mutation_insertion(new_chromosome)
Begin
   Set k = 1;
        for i = 1 to row do
         p = randomly generated number
within the limit;
         q = randomly generated number
within the limit;
        sort p, q
        temp = new_chromosome (i, q);
        if (p not equal to q)
               x = q-1;
 While (x greater than equal to p+1)
              new_chromosome (i,x+1) =
new_chromosome (i,x);
              decrement x by -1;
         end
new_chromosome (i,p+1) = temp;
       end
   end
End
```

Crossover/recombination: Chromosomes have been done on a single point.

EXPERIMENTAL RESULT

Experiment based on Mutation Operator for small tolarge size network. The experiment is done in MATLAB R2008a version 7.6.0.324. Following parameters have been considered:

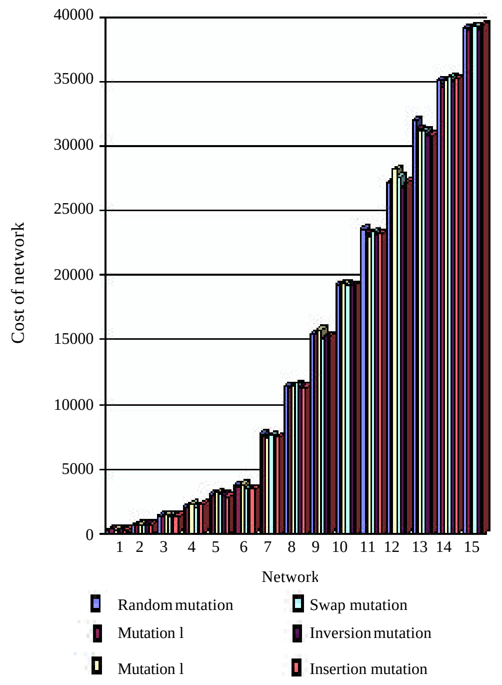

Fig. 2: Network cost chart based on mutation function

Population size: 100
No of generations: 100
Selection: Roulette wheel selection
Crossover: Uniform

From the above experimental result of Table 1 and the chart shown in Fig. 2, it is clear that, mutation operator is one of the important factor of genetic algorithm.

CONCLUSION

From these six mutation function it is observed that insertion, inversion and swap mutation operator gives the better result. This is the improved approach of evolutionary computing which gives the very positive result. The importance of mutation operator. The effectiveness of the methodology however can be increased by applying the various genetic operators with variations of network size as the densely connected locations.

ACKNOWLEDGEMENT

Researchers thanks to the reviewers for their cogent and insightful comments.

REFERENCES

Basu, S.K., 2005. Design Methods and Analysis of Algorithms. Prentice-Hall of India Pvt. Ltd., India, ISBN-10: 8120326377.

Deo, N., 2000. Graph Theory with Applications to Engineering and Computer Science. Prentice-Hall, Englewood Cliffs, New Jersey.

Knuth, D., 1997. The Art of Computer Programming: Sorting and Searching. 3rd Edn., Addison-Wesley, USA.

Kumar, A. and N.N. Jani, 2010. Genetic algorithm for network design problem: An empirical study of crossover operator with generation and population variation. Int. J. Inform. Technol. Knowledge Manage., 2: 605-611.

Melanie, M., 1998. An Introduction to Genetic Algorithm. 1st Rev. Edn., MIT Press, Cambridge, MA., ISBN-10: 0262631857.

Taha, H.A., 2007. Operation Research: An Introduction. 8th Edn., Pearson Education, India, ISBN-13: 9788131711040.

Vose, M.D., 1999. The Simple Genetic Algorithm, Foundation and Theory. A Bradford Book. MIT Press, Cambridge, Massachusetts. London, England.