

in cases where growing traffic loads or temporary traffic variations cause localized link congestions, routing optimization can be used to resolve or at least alleviate potential network performance problems. The idea is to adjust the paths, along which IP packets travel through the network, to current load situations and, thus, to better utilize available network resources, leading to improved Quality of Service (*QoS*) ⁽³⁾.

Genetic algorithm is an optimization and evolutionary algorithm that solves optimization problems. This project is aimed to solve the optimization problem in IP networks using genetic algorithm. The solution starts with finding alternative paths to alternate the overloaded path using genetic algorithm ⁽⁴⁾.

This project presents a genetic algorithmic approach to the shortest path (SP) routing problem. Variable-length chromosomes (strings) and their genes have been used for encoding the problem. The crossover operation exchanges partial chromosomes (partial routes) at appositionally independent crossing sites and the mutation operation maintains the genetic diversity of the population. The proposed algorithm can cure all the infeasible chromosomes with a simple repair function. Crossover and mutation together provide a search capability that results in improved quality of solution and enhanced rate of convergence. Computer simulations show that the proposed algorithm exhibits a much better quality of solution (route optimality) and a much higher rate of convergence than other algorithms ⁽⁵⁾.

2-GENETIC ALGORITHM

A genetic algorithm (or GA for short) is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a *fitness function* that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random. The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem. These promising candidates are kept and allowed to reproduce. Multiple copies are made of them, but the copies

are not perfect; random changes are introduced during the copying process. These digital offspring then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. Those candidate solutions which were worsened, or made no better, by the changes to their code are again deleted; but again, purely by chance, the random variations introduced into the population may have improved some individuals, making them into better, more complete or more efficient solutions to the problem at hand. Again these winning individuals are selected and copied over into the next generation with random changes, and the process repeats. The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered. As astonishing and counterintuitive as it may seem to some, genetic algorithms have proven to be an enormously powerful and successful problem-solving strategy, dramatically demonstrating the power of evolutionary principles. Genetic algorithms have been used in a wide variety of fields to evolve solutions to problems as difficult as or more difficult than those faced by human designers. Moreover, the solutions they come up with are often more efficient, more elegant, or more complex than anything comparable a human engineer would produce. In some cases, genetic algorithms have come up with solutions that baffle the programmers who wrote the algorithms in the first place ⁽⁶⁾.

3-METHODS OF REPRESENTATION TO PROBLEM

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to 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. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This technique was used, for example, in the work of Steffen Schulze-Kremer, who wrote a genetic algorithm to predict the three-dimensional structure of a protein based on the sequence of amino acids that go into it. Schulze-Kremer's GA used real-valued numbers to represent the so-called "torsion angles" between the peptide bonds that connect amino acids. (A protein is made up of a sequence of basic building blocks called amino acids, which are joined together like the links in a chain. Once all the amino acids are linked, the

protein folds up into a complex three-dimensional shape based on which amino acids attract each other and which ones repel each other. The shape of a protein determines its function.) Genetic algorithms for training neural network often use this method of encoding also.

A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution. One example of this technique is Hiroaki Kitano's "grammatical encoding" approach, where a GA was put to the task of evolving a simple set of rules called a context-free grammar that was in turn used to generate neural networks for a variety of problems.

The virtue of all three of these methods is that they make it easy to define operators that cause the random changes in the selected candidates: flip a 0 to a 1 or vice versa, add or subtract from the value of a number by a randomly chosen amount, or change one letter to another. Another strategy, developed principally by John Koza of Stanford University and called *genetic programming*, represents programs as branching data structures called trees. In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one subtree with another ⁽⁷⁾.

4-METHODS OF CHANGE TO CHROMOSOMES

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation. There are two basic strategies to accomplish this. The first and simplest is called mutation. Just as mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code. The second method is called crossover, and entails choosing two individuals to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. Common forms of crossover include single-point crossover, in which a point of exchange is set at a random location in the two individuals' genomes, and one individual contributes all its code from before that point and the other contributes all its code from after that point to produce an offspring, and uniform crossover, in which the value at any given location in the offspring's genome is either the value of one parent's

genome at that location or the value of the other parent's genome at that location, chosen with 50/50 probability ⁽⁸⁾.

5-IMPLEMENTATION OF PROPOSED METHOD

We started by creating a genetic algorithm class (called GA) that contains chromosome objects as instance variables and implements the basic methods for doing crossover, reproduction, mutations, and iterating through the simulated evolution.. The work of the algorithm will be organized through the main function and all the function will be called from the main. We created paths structure called (paths) which stores the nodes and the cost of each paths. The underlying structure is an array of integers within the range of 0 to the largest number of the nodes. Each object of this structure will be a path and the genes of the array will be the nodes that lying on that path and a random number that represents the cost to that array (path).

Step1: Generate the graph with cost for each path. Initialize the global variables and the population.

The first step is to generate a network (graph) with nodes and a cost assigned randomly to the link that connects two sequenced nodes in the path. The location of each network node is given. Nodes are perfectly reliable. Each cost is fixed and known. Each link is unidirectional. There are no redundant links in the network. Links are operational. No repair is considered. This function will do all this work using the [rand()] function the cost will be assigned randomly between two nodes. If the random function gives cost=zero to a link between two nodes so that indicates no link connects the two nodes.

Step2: Choose the source and destination node to generate all the paths between the desired nodes. Our task is to optimize a path in the routing table. Growing traffic loads or temporary traffic variations cause localized link congestions, routing optimization can be used to resolve or at least alleviate potential network performance problems. The idea is to adjust the paths, along which IP packets travel through the network, to current load situations and, thus, to better utilize available network resources as shown in Fig.(1).

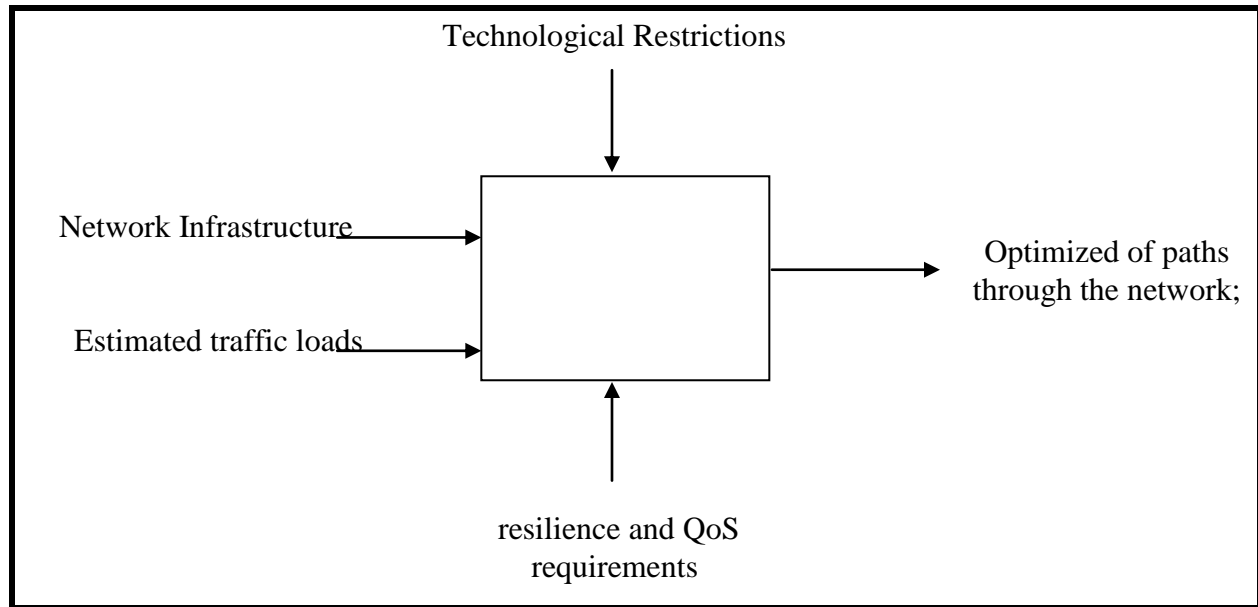


Fig.(1): Finding Path.

In this step we will choose the path that might be congested. Based on the network infrastructure all paths from the source and destination node will be taken and this is the first step of the algorithm. These alternative paths will be listed in a list with a cost for each path.

For example if the source node=0 and the destination node=4 then the list will be shown in Table(1).

Table(1):

X	Paths	No. of nodes (n)	path cost
0	0 – 1 – 2 – 4	4	C1
1	0 – 4	2	C2
2	0 – 3 – 2 – 4	4	C3

Here the path's cost will be the sum of all the costs assigned to each link in the path (the length of the path).

Crossover and mutation operations of the genetic algorithm will use this table to do there jobs.

The final routing solution does not only achieve good service quality, but that it is also robust enough to provide sufficient quality in cases where individual links or nodes of the network break down.

Step3: Calculate the fitness of all the paths generated in the previous step .There is a lot of routing algorithms which depends on cost to find the path between two nodes like Dijkstra. This algorithm calculates the shortest path between two nodes depending on the cost so the path between each source and destination is always the shortest one.

Figure (2) shows a simple example of routing optimization in a network with two traffic flows. In the network on the left-hand side, both traffic flows share several links, which is the shortest path between each source and destination, which possibly could lead to overload on the shortest path. On the right-hand side the routing is set in a way that the two flows have no link in common. This would typically result in better performance for both traffic flows. While the solution might be obvious in the given example, it becomes very difficult to determine in cases with tens of nodes, hundreds of links, and up to thousands of individual traffic flows. Efficient algorithms are necessary to solve the optimization problem.

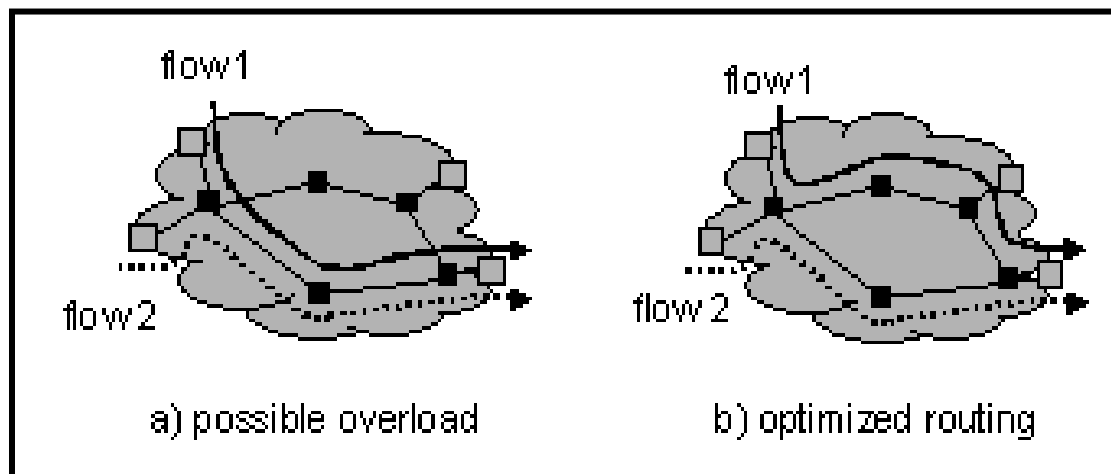


Fig.(2):Routing Optimization.

In our algorithm which depends on genetic algorithm we did not take the cost as a measurement to calculate the fitness of the path. We took other measurements to give a chance to find more paths, which may be solve the problem and enhance the network.

For example if the source node=0 and the destination node=4 then the list will be shown in Table(2).

Table(2):

X	Paths	No. of nodes (n)	path cost
0	0 – 1 – 2 – 4	4	C1
1	0 – 4	2	C2
2	0 – 3 – 2 – 4	4	C3

Where X is the number of possible paths between nodes 0 – 4 and n is the number of nodes in the path.

The first step in the fitness function is to find F(x) for the path, where .

$$F(X) = \frac{X}{\sum X} + \frac{n}{\sum n} \dots\dots\dots 1$$

Where **X** is the number of the path in the previous list of possible paths, $\sum n$ is the number of nodes in the hall network, $\sum X$ is the sum of all the paths (e.g 0+1+2+3+4+...+Max).as shown in Table(3).

Table(3):

X	Paths	No. of nodes (n)	F(x)	path cost
0	0 – 1 – 2 – 4	4	F(1)	C1
1	0 – 4	2	F(2)	C2
2	0 – 3 – 2 – 4	4	F(3)	C3

After finding F(x) for each possible path then the fitness for each one is calculated like this:

$$Fitness = \frac{F(X)}{Max.F(X)} \dots\dots\dots 2$$

Where Max. F(x) is the maximum F(x) in the list so one of the paths which has the maximum F(x) will be given Fitness = 1

For example assume the maximum

$$F(X) = 0.68 \text{ then } \frac{0.68}{0.68} = 1$$

So table(4) is final table .

Table(4):

X	Paths	No. of nodes (n)	F(x)	Fitness	path cost
0	0 – 1 – 2 – 4	4	F(1)	F1	C1
1	0 – 4	2	F(2)	F2	C2
2	0 – 3 – 2 – 4	4	F(3)	F3	C3

The operations of the genetic algorithm will use this table to perform there operations.

Step4: Do reproduction, Crossover and Mutation to produce the best solution.

The operations of the genetic algorithm give all the genes in a chromosome (good or week) a chance to appear in the offspring. They change the order of the chromosomes and exchange the genes between the chromosomes. These process my produce a good or bad offspring depends on the chromosomes and the order of the genes inside the chromosomes. These operations are reproduction, crossover and mutation.

5-1. REPRODUCTION (SELECTION)

The reproduction (selection) operator is intended to improve the average quality of the population by giving the high-quality chromosomes a better chance to get copied into the next generation. The selection thereby focuses the exploration on promising regions in the solution space. Selection pressure characterizes the selection schemes. It is defined as the ratio of the probability of selection of the best chromosome in the population to that of an average chromosome. Hence, a high selection pressure results in the population's reaching equilibrium very quickly, but it inevitably sacrifices genetic diversity
Selection picks the chromosomes based on their fitness and on a random number so from Table (5).

Table (5):

X	Paths	No. of nodes (n)	F(x)	Fitness	path cost
0	0 – 1 – 2 – 4	4	F(1)	F1	C1
1	0 – 4	2	F(2)	F2	C2
2	0 – 3 – 2 – 4	4	F(3)	F3	C3

$R_n = \text{Rand}()$ $\rightarrow R_n$ is a random number

iF $R_n < \text{Fitness}$;

Select the path as a solution ;

Else

Skip the path ;

Usually a list of paths will be provided in this step as the much redundant paths.

5-2.CROSSOVER

Crossover examines the current solutions in order to find better ones. Physically, crossover in the SP routing problem plays the role of exchanging each partial route of two chosen chromosomes in such a manner that the offspring produced by the crossover represents only one route. This dictates selection of one-point crossover as a good candidate scheme for the proposed GA. One partial route connects the source node to an intermediate node, and the other partial route connects the intermediate node to the destination node. The crossover between two dominant parents chosen by the selection gives higher probability of producing offspring having dominant traits. But the mechanism of the crossover is not the same as that of the conventional one-point crossover. We choose two chromosomes for crossover should have at least one node(gene) except the source and destination nodes, but there is no requirement that they be located at the same location. That is, the crossover does not dependent on the position of nodes in routing paths.

- 1) The probability of the crossover (PC) is equal to 0.6 (60%). In this operation the chromosomes will be the number of the path in the list of the alternative paths for the given path in the problem as shown in table(6).

Table(6):

X	Paths	No. of nodes (n)	F(x)	Fitness	path cost
0	0 – 1 – 2 – 4	4	F(1)	F1	C1
1	0 – 4	2	F(2)	F2	C2
2	0 – 3 – 2 – 4	4	F(3)	F3	C3

0 → 0000

0	0	0	0
---	---	---	---

1 → 0001

0	0	0	1
---	---	---	---

2 → 0010

0	0	1	0
---	---	---	---

- 2) Crossover in the SP routing problem plays the role of exchanging each of the two chosen chromosomes. Our algorithm divide the list of the alternative paths into pairs of chromosomes table(7).

Table (7):

X	Paths	No. of nodes (n)	F(x)	<u>Fitness</u>	path cost
0	0-1-2-4	4	F(0)	F0	C0
1	0-4	2	F(1)	F1	C1
2	0-1-4	3	F(2)	F2	C2
3	0-3-2-4	4	F(3)	F3	C3

For each pare a random number (Rn) is generated, if that random number is smaller than PC then crossover will operates on that pairs.

If the first pair (1, 2) $\rightarrow \rightarrow$ (0001, 0010). A random point in the chromosome will be taken to split the chromosome.

(1)	(2)	(3)
$\begin{array}{r} 0001 \\ 0010 \end{array}$	$\begin{array}{r} 0000 \\ 0011 \end{array}$	$\begin{array}{r} 0000 \\ 0011 \\ \hline 0011 \end{array}$

The product is 3. It means the path 3. Then apply this algorithm to all the pairs in the list and save the result in another which contains the number of the path and number of redundancy as shown in table(8).

Table(8):

X	redundancy
0	R1
1	R2
2	R3

Where R is the number of redundancy of the path.

In this way we will provide more probabilities and results, where the more redundant one is best.

Usually one path will be chosen in this step as the much redundant one .

5-3.MUTATION

The chromosomes undergo mutation by an actual change or flipping of one of the genes, thereby keeping away from local optima. Physically, it generates an alternative route from the mutation node to the destination node in this GA.

1) The probability of the mutation (PM) is 0.7 (70%). Each path in the list of alternative paths will be a chromosome.

0 → 0000

0	0	0	0
---	---	---	---

1 → 0001

0	0	0	1
---	---	---	---

2) For each chromosome a random number (Rn) is generated. If this Rn is smaller than PM then mutation will operates on that chromosome.

A random gene in the chromosome will be taken to flip it (0 → 1 and 1 → 0).



0000

→ →

0001

The product is 1. It means the path 1. Then apply this algorithm to all the chromosomes in the list and save the result in another which contains the number of the path and number of redundancy as shown in table(9).

Table(9):

X	redundancy
0	R1
1	R2
2	R3

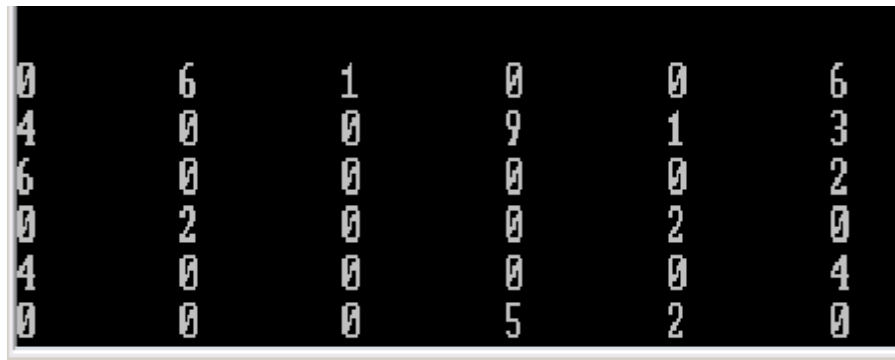
Where R is the number of redundancy of the path.

In this way we will provide more probabilities and results, where the more redundant one is best. Usually one path will be chosen in this step as the much redundant one.

6) RESULTS

After doing the three operations the results will be taken as alternative paths but there is one best solution which is the more redundant one in the three operations. The program will not choose a path. We leave it to the user to choose the more redundant one so we suggest selecting it. Here an example to illustrate our program and how it works.

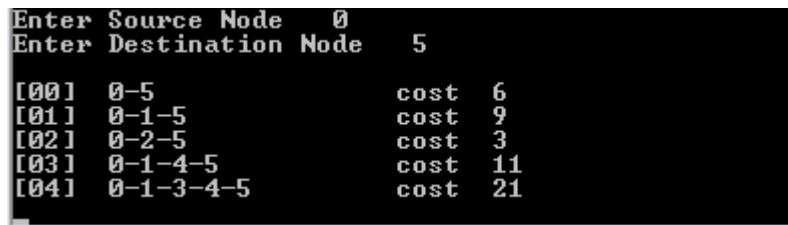
1. The first choice is to build a random graph.
2. The next step is to print the graph as we seen in fig(3) so the 2-dimentional array shows the paths between two nodes and the cost for each link connect them .



0	6	1	0	0	6
4	0	0	9	1	3
6	0	0	0	0	2
0	2	0	0	2	0
4	0	0	0	0	4
0	0	0	5	2	0

Fig.(3): Build Graph.

3. Then we choose the path that we want by entering the source node and the destination node, so the program will give us the alternative paths and the cost for each path as shown in fig (4).



```

Enter Source Node  0
Enter Destination Node  5
[00]  0-5          cost  6
[01]  0-1-5        cost  9
[02]  0-2-5        cost  3
[03]  0-1-4-5      cost 11
[04]  0-1-3-4-5    cost 21
  
```

Fig.(4):Choose Bath.

4. The fitness function is divided into two steps. The first one is to find $F(x)$ as shown in fig(5).

```

The F(x) Values Are:
0.203333
0.346667
0.226667
0.47
0.753333
    
```

Fig.(5):Find $F(X)$.

5. Then we find the fitness value for each $F_x()$ and the reproduction table for the paths as shown in fig(6).

```

The Fitness Values Are:

<0,0.269912>    <1,0.460177>    <2,0.300885>    <3,0.623894>
<4,1>
    ---> Average Fitness is 0.663717

Reproductions
4          3          4          4          3
Repeat For Each Reproduction
<0,0>    <1,0>    <2,0>    <3,2>    <4,3>
Mostly Repeated Paths Are:
[4]  0-1-3-4-5
    
```

Fig.(6):Find Fitness Value.

6. After that we have an important step that is used to find the crossover for 2 paths as shown in Fig.(7) .

```

3. Generate Paths
4. Step 1: Find F(x) values
5. Step 2: Find Fitness Values And Reproductions
6. Step 3: Crossover
7. Mutation
8. Exit

---> Enter Your Choice 6

Repeat For Each Reproduction After Crossover:
<0,0> <1,0> <2,0> <3,0> <4,4>
Mostly Repeated Paths Are:
[4] 0-1-3-4-5

```

Fig.(7):Find crossover.

7. Then the second important step is the mutation as shown in Fig.(8)

```

2. Print The Graph
3. Generate Paths
4. Step 1: Find F(x) values
5. Step 2: Find Fitness Values And Reproductions
6. Step 3: Crossover
7. Mutation
8. Exit

---> Enter Your Choice 7

Repeat For Each Reproduction After Mutation:
<0,0> <1,0> <2,0> <3,0> <4,0>
Mostly Repeated Paths Are:
...

```

Fig.(8): Mutation.

From this example we can see the results from the three operations.

Reproduction → Most repeated path is (4).

Crossover → Most repeated path is (4).

Mutation → No repeated paths.

So we can see that path 4 is the most repeated one. So we can choose it as the best alternative path. By this operation we can optimize any path in the network..

7-DISCUSSION

When we begin to build a random path by choosing the source node(1) and destination nodes(5) using our algorithm and Dijkstra's Algorithm), so the program will give us the alternative paths and the cost for each path in each methods, after we build a path between the nodes, The two algorithms uses the common two genetic operators.

crossover and *mutation*. Crossover recombines two 'parent' paths to produce two 'children' new paths in the next generation. Two points crossover is used. Both parent paths are divided randomly into three parts respectively and recombined. The middle part of the first path between crossover bit positions and the middle part of the second path are exchanged to produce the new children.

The mutation process is also applied to flip randomly a bit position in the chromosomes , after apply these process to our path we find the final path .

```
0 5 1 0 0 5
4 2 5 1 3 4
0 2 2 0 0 1
1 3 5 2 0 1
1 1 0 2 4 5
4 5 1 0 2 1
```

In our algorithm the repeated path is (4), in Dijkstra's Algorithm we find the path (2).That means the shortest path from node 1 to node 5 has to pass via nodes 0,1, 3, 5, with minimum cost 39. The second path in our work 0,1,3,4,5 with cost 21 .After these result we find the path with the minimum cost is not best path. The program is tested with other source and destination points. The obtained results affirmed the potential of the proposed algorithm where the convergence was guaranteed to obtain the optimal path in each case .

8-CONCLUSIONS

It is not important that the lowest cost is the best. From here we found after experiments that GA gives best results in some cases compared with the old studies on this field like Dijkstra algorithm. So I suggest using the two algorithms together (GA and Dijkstra). First use the Dijkstra but if the path takes long time to deliver the packets because of overloading immediately use the genetic algorithm.

This Paper presented a genetic algorithm for solving the shortest path routing problem. The reproduction, crossover and the mutation operations work on variable-length chromosomes. The crossover is simple and independent of the location of crossing site. Consequently, the algorithm can search the solution space in a very effective manner. The mutation introduces, in part, a new alternative route.

REFERENCES

1. H.L. Christensen, R.L. Wainwright and D.A. Schoenefeld, "A Hybrid Algorithm for The Point to Multipoint Routing Problem", Proceedings of the 1997 ACM Symposium on Applied Computing, ACM Press, 1997, pp 263-268.
2. Liming Zhu, Roger L. Wainwright, and Dale A. Schoenefeld, "A Genetic Algorithm for the point to multipoint Routing Problem with Varying Number of Requests" Mathematical and Computer Science Department The University of Tulsa, 1998.
3. T. Al-Qahtani, M. Abedin, S. Ahson, "Dynamic Routing in Homogenous ATM Networks using Genetic Algorithms", Proceeding of the 1998 IEEE International Conference on Evolutionary Computing (ICEC'98), part of WCCI, Anchorage, Alaska, May 4-9, 1998.
4. David E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.
5. Michael J. Alexander and Gabriel Robins, "New Performance-Driven FPGA Routing Algorithms," Proceedings of ACM/SIGDA Design Automation Conference, June 1995.
6. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D, Morgan Kaufmann, "Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications, (1998).
7. Melanie Mitchell. An introduction to genetic algorithms. MIT Press, 1998.
8. James Cunha Werner, Terence C. Fogarty "Map Algorithm in Routing Using Genetic Algorithm", South Bank University, London 12-July-2002 .

تحديد المسارات باستخدام الخوارزميات الجينية في الشبكات الكبيرة

يسرى احمد فاضل

مدرس مساعد

كلية الهندسة-جامعة ديالى

الخلاصة

ان الأداء والإتقان للانترنت يعتمد بدرجة كبيرة على سياقات اختيار المسارات . اليوم Internet protocol يقوم بعملية حساب المسارات بالاعتماد على ال Network topology بدون الاهتمام للزخم في المسارات . هذا البحث يبحث بتحقيق أفضل المسارات بالاعتماد على الخوارزمية الجينية . سنقوم بدراسة وتحليل مشكلة اختيار أفضل المسارات في الشبكات الكبيرة وسنقدم شرح للخوارزمية الجينية من اجل الحصول على جدول المسارات واختيار افضل المسارات الممكنة.