



GENETIC ALGORITHM FOR SHORTEST PATH IN PACKET SWITCHING NETWORKS

¹ ANU CHAUDHARY, ² NEERAJ KUMAR PANDEY

¹Associate Professor, Dept. of IT, A.K.G. Engineering College, Ghaziabad (U.P)-India.

²Assistant Professor, Dept. of CS, College of Engineering Roorkee, (Uttarakhand)-India.

ABSTRACT

The problem of finding the shortest path between two nodes is a well known problem in network analysis. Optimal routing has been widely studied for interconnection networks this paper considers the problem of finding the shortest path. A Genetic algorithm based strategy is proposed and the algorithm has been developed to find the shortest (Optimal) Path Variable-length chromosomes (strings) and their genes (parameters) have been used for encoding the problem. The crossover operation exchanges partial chromosomes (partial routes) at positionally 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. Even though shortest path routing algorithms are already well established, there are researchers who are trying to find alternative methods to find shortest paths through a network. One such alternative is to use genetic algorithm (GA). GA is a multi-purpose search and optimization algorithm that is inspired by the theory of genetics and natural selection.

Keywords: *Genetic Algorithm (GA), Shortest Path, Packet Switching, Variable Length Chromosomes*

1. INTRODUCTION

For the information-oriented society in the early years of 21st century, communication by packet flow in large-scale computer networks becomes much more important in our daily life than ever before. The problem of finding the shortest path between two nodes is a well-known problem in network analysis. Shortest path algorithms have been a subject of extensive research, resulting in a number of algorithms for various conditions and constrain [1-3].

Adaptive routing algorithms [4-8], which can select the route of packets dynamically, have been widely studied to make the best use of bandwidth in interconnection networks of massively parallel computers and system area networks (SANs). Most of real SANs for PC clusters [9-10] have not employed adaptive routing. This is because adaptive routing introduces new problems in the networks. First, it does not guarantee in-order packet delivery in which some message passing libraries require. Second, switch complexity may be increased, because they compute alternative output channels and select one of them introducing selection logic. In the context of SANs, some works have also proposed simple methods to support adaptive routing in InfiniBand switches [11]. In a

packet switching network, communication between two hosts generally takes place in the following manner: the transmitting host delivers to a node a block of data, called a packet, which are addresses to the destination host. The objective of a routing strategy is essentially to minimize the mean delay of the packets in a network, subject to some reliability or capacity constraints [12-16]

There exist some extra mathematical programming methods which are based on certain properties of mean delay function to solve the problem of optimal routing [17, 18, 19].

The implementation of these methods necessitates complex and lengthy calculations. As a result, heuristic routing procedures have been used in order to determine, within reasonable computation time, the routes along which the packets must travel without causing network congestion [13, 15, 20, 21, 22].

In today's IP networks, routing protocols are responsible for building a path that carries a data packet to its destination [23]. Each router in the network has to send the packet to its next hop, independently from what other routers are doing at the time, on rules based only on its own knowledge



base. These routing tables are built based on topological and traffic information, send or capture

from information being received from other routers.

2. GENETIC ALGORITHM

Genetic algorithms have been developed by John Holland, the goals of their research have been twofold (i) to abstract and rigorously explain the adaptive processes of natural system, and (ii) to design artificial systems software that retains the important mechanisms of natural systems. This approach has led to important discoveries in both natural and artificial systems science. The central theme of research on genetic algorithms has been robustness, the balance between efficiency and efficacy necessary for survival in many different environments. Genetic algorithms have been implemented in scheduling problems and Finite State Machines [24].

Yinzhen et.al[25] presented a genetic algorithm for solving shortest path problems, which was based on the technology of dynamic coding of the priority of vertex and gene weight. The microevolution strategy was also considered fully in that paper. After putting forward the measure of the importance of a vertex in a network structure and its formula, the real coding priority of a vertex was generated in a non-uniform distribution function with the parameters of the measure of the vertex importance. Flexible fitness functions, elitist genes selection, the mountain climbing method for local optimum and other methods were adopted. Chang Wook Ahn, Ramakrishna, R.S.[26] also proposed a genetic algorithmic approach to the shortest path (SP) routing problem. Variable-length chromosomes (strings) and their genes (parameters) had been used for encoding the problem. The crossover operation exchanges partial chromosomes (partial routes) at positionally 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 provides a search capability that results in improved quality of solution and enhanced rate of convergence. They have also developed a population-sizing equation that facilitates a solution with desired quality. It was based on the gambler ruin model, the equation had been further enhanced and generalized. The equation relates the size of the population, quality of solution, cardinality of the alphabet, and other parameters of the proposed algorithm

A Genetic Algorithm starts with an initial population that evolves through generations. This evolution starts with an initial population randomly generated and the ability of an individual to span through different generations and to reproduce depends on its fitness.

The GA is based on two fundamental evolutionary concepts:

- (i) A Darwinian notation of fitness, which describes an individual's ability to survive, and
- (ii) Genetic operators, which determine the next generations genetic makeup based upon the current generation.

Conventionally, genetic operations are achieved through crossover and mutation operators. The crossover operator creates new individuals called offspring, by recombining the genetic material of two individuals, deemed the parents. Individuals with higher fitness scores are selected with greater probability to be parents and "pass on" their genes to the next generation. This is known as fitness proportional selection method.

Crossovers allow exploitation of successful subspace of the solution space. The mutation operator randomly alters one or more genes in an individual. Mutations add genetic diversity to the population. Through mutation, GAs can search previously unexplored sections of the solution-space. Mutations consequently assure that the entire search space is connected. Through crossover and mutations, GAs are able to simultaneously explore new subspace while exploiting successful ones.

3. SHORTEST PATH USING GENETIC ALGORITHM

Routing is one of the most important issues that have a significant impact on the network's performance [27], [28]. An ideal routing algorithm should strive to find an optimum path for packet transmission within a specified time so as to satisfy the Quality of Service (QoS) [28]-[30]. There are several search algorithms for the shortest path (SP) problem: the breadth-first search algorithm, the Dijkstra algorithm and the Bellman-Ford algorithm, to name a few [27]. Since these algorithms can solve SP problems in polynomial time, they will be effective in fixed infrastructure wireless or wired networks. But, they exhibit unacceptably high computational complexity for real-time communications involving rapidly changing net-

work topologies [29], [30]. In most of the current packet-switching networks, some form of SP computation is employed by routing algorithms in the network layer [28], [30]. Specifically, the network links are weighted, the weights reflecting the link transmission capacity, the congestion of networks and the estimated transmission status such as the queuing delay of head-of-line (HOL) packet or the link failure. The SP problem can be formulated as one of finding a minimal cost path that contains the designated source and destination nodes. In other words, the SP routing problem involves a classical combinatorial optimization problem arising in many designs and planning contexts [28]-[33]. Since neural networks (NNs) [28]-[30] and genetic algorithms (GAs) (and other evolutionary algorithms) [31] [34] promise solutions to such complicated problems, they have been used successfully in various practical applications. On the other hand, NNs and GAs may also not be promising candidates for supporting real-time applications in packet switching networks because they involve a large number of iterations in general. However, hardware implementations of NNs or GAs are extremely fast. Furthermore, they are not very sensitive to network size [30], [35]. The quality of the solution (i.e., computed path) returned by NNs is constrained by their inherent characteristics. GAs is flexible in this regard. Various paths of a network are shown in fig-1:

The quality (of the solution) can be adjusted as a function of population. In addition, NN hardware is limited in size: it cannot accommodate networks of arbitrary size because of its physical limitation. GA hardware, on the other hand, scales well to networks that may not even fit within the memory. It is realized by employing parallel GA over several nodes. Therefore, GAs (especially hardware implementations) is clearly quite promising in this regard.

4. PROPOSED APPROACH

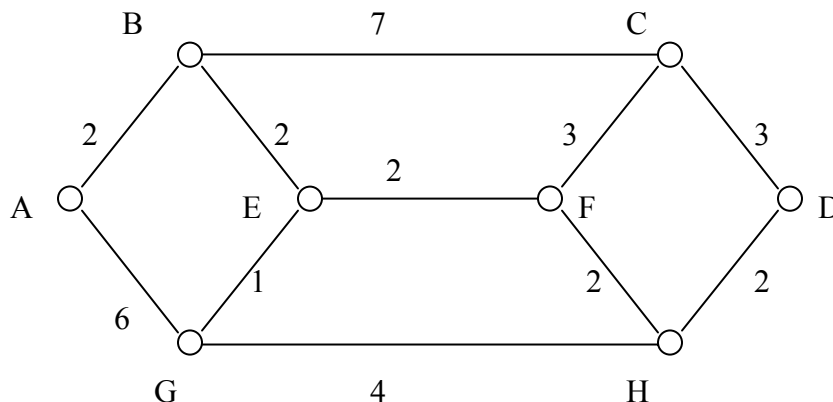
4.1 CODING OF SOLUTIONS: The coding of an individuals is composed by m strings {e₁, e₂, e₃,.....e_m}, each {e_i} represents distance between two nodes. Our fitness function is used to minimize the distance from source node to destination node with continuity.

Objective function = min (∑e_i, source → destination), with continuity

Where, (i = 1, 2, 3,.....m)

CHROMOSOME REPRESENTATION:

A network can be thought of interconnection of nodes where distance between two nodes is represented by {e_i}



(Fig-1)

- Let, e₁ = AB = 2
- e₂ = BC = 7
- e₅ = AG = 6
- e₆ = EF = 2
- e₉ = DH = 2
- e₁₀ = FH = 2

- e₃ = CD = 3
- e₄ = GH = 4
- e₇ = BE = 2
- e₈ = CF = 3
- e₁₁ = GE = 1



One of the combinations of edges can be:
 $\{e_5(6), e_2(7), e_4(4), e_6(2)\}$

Each edge is represented by four bit string therefore the above combination of edges can be represented by following strings:

0110 0111 0100 0010

And,

$\{e_1(2), e_{11}(1), e_{10}(2), e_3(3)\}$ these combinations can be represented by following Strings:

0010 0001 0010 0011

4.2 INITIAL POPULATION: The initial population is randomly generated, based on distance between nodes of the networks. The distance between each node is coded into 4 bit strings and total string length is 20 bit long.

4.3 CROSSOVER: We applied two points crossover on initial population

Suppose four randomly generated individuals are:

$e_2(7) e_4(4) e_3(3) e_5(6) e_8(3)$ sum of edges $\rightarrow 23$

$e_1(2) e_{11}(1) e_5(6) e_2(7) e_6(2)$ sum of edges $\rightarrow 18$

$e_4(4) e_7(2) e_3(3) e_{11}(1) e_6(2)$ sum of edges $\rightarrow 12$

$e_1(2) e_3(3) e_6(2) e_2(7) e_7(2)$ sum of edges $\rightarrow 16$

Before Crossover,

0111 010 0	0011 011 0 0011	sum of edges $\rightarrow 23$
0010 000 1	0110 011 1 0010	sum of edges $\rightarrow 18$
0100 001 0	0011 000 1 0010	sum of edges $\rightarrow 12$
0010 001 1	0010 011 1 0010	sum of edges $\rightarrow 16$

(Applying two point Crossover)

After Crossover,

(7) (5) (6) (7) (3)	0111 0101 0110 0111 0011	sum of edges $\rightarrow 28$
(2) (0) (3) (6) (2)	0010 0000 0011 0110 0010	sum of edges $\rightarrow 13$
(4) (3) (2) (7) (2)	0100 0011 0010 0111 0010	sum of edges $\rightarrow 18$
(2) (2) (3) (1) (2)	0010 0010 0011 0001 0010	sum of edges $\rightarrow 27$

4.4 MUTATION: Mutation of a string is implemented through a very simple protocol. We will replace first four bits with source and last four bits with destination.

Since in our network source node is A and destination node is D, therefore we replace first four bits by 0010 and last four bits by 0010

(2) (5) (6) (7) (2)	0010 0101 0110 0111 0010	sum of edges $\rightarrow 22$
(2) (0) (3) (6) (2)	0010 0000 0011 0110 0010	sum of edges $\rightarrow 13$
(2) (3) (2) (7) (2)		

0010 0011 0010 0111 0010 sum of edges $\rightarrow 16$

(2) (2) (3) (1) (2)

0010 0010 0011 0001 0010 sum of edges $\rightarrow 10$

4.5 SELECTION:

In the above problem our fitness function is $= \min(\sum e_i)$, with continuity

After mutation, we have minimum path length from source node to destination node is: 10 (min path length from A \rightarrow D) (0010 0010 0011 0001 0010)

Path \rightarrow AB BE FC EG FH

We Observed that it is not a continuous path, therefore we have to select minimum path with continuity.

After iterations we get, minimum path length 10 with continuity

10 (min path length after selection) (0010 0010 0010 0010)

Path \rightarrow AB BE EF FH HD

This is the most optimal path.

5. ALGORITHM FOR THE CODING:

Algorithm 1:- Algorithm for identifying path from source to destination

```

path(source, destination, *source node)
{
    Identify source and destination in given graph
    push(source);
    while(top!=NULL)
    {
        temp node = pop(&top);
        check weather the node is visited or not
        if not visited set flag=1
        {
            while(temp node->info!=destination && temp arc!=NULL)//
            {
                push(temp node);
                p[i][j]=temp arc;
                temp node=temp arc->adj;
                temp arc=t node->edg;
                j=j+1;
                if(tempnode->info==destination)
                {
                    p[i][j]=NULL;
                    i=i+1;
                }
            }
            End if
        }
        End while
    }
    End while
}
End }
```



Algorithm 2:- Algorithm for create node from source to destination

```

randompath(*source_node) //In the following
algorithms the '*' symbol use to represent pointer
notion in c language
//Create a Structure for create a node
{
store all edges in an array
intilize len=0
repeat step 3,4,5 while(len<length_of_array)
use mode function index=(3*random_num)%4 to
select an edge from array
assign the selected edge to two diminsion array
len=len+1
End
}
    
```

```

{
Then *source=temp
else
tc=*source
{
while(tc->next!=NULL)
{
tc=tc->next
End While
}
tc->next=temp
}
End Else
}
End If
}
}
    
```

Algorithm 3:- Algorithm for Push function

```

Push(node,*data,**top) //Create a Structure for
store the edges between the nodes
{
Declare a temp pointer variable of stack
Structure type
Initialize size of Stack in temp
Assign the data in add part of temp node
Assign the top reference in link part of temp node
//Create a Structure for push data from the Stack
}
    
```

Algorithm 4:- Algorithm for store values in binary encoded form through crossover process

```

PushBin(val,stackbin **top,**temp //Create a
Structure for creating a stack
//Create a Structure for creating a stack which
stores the encoded decimal number in binary form
{
Assign size of stackbin of structure type in
temp
Insert val in data part of temp node
Assign Refrence of *top in Link part of
temp node
*top = temp
}
    
```

Algorithm 5:- Algorithm for creating a node

```

CreateNode(**source,*temp,*tc,c,data)
{
Assign a block of size from the memory
heap in to temp
temp->next=NULL
temp->edg=NULL
temp->info=data
{
If(*source==NULL)
    
```

Algorithm 6:- Algorithm for Connect Nodes which are present in the whole graph

```

ConnectNodes(*source,*temp,*back,*track,*tc,*ptr)
{
temp=source
back=temp
while(temp->info!=p)
{
temp=temp->next
back=temp
End While
}
temp=source
track=temp
while(temp->info!=c)
{
temp=temp->next
track=temp
End While
}
Print : Enter the Weight of edge
Assign a block of size from memory heap
to the tc
tc->w=wit
tc->adj=track
tc->nextptr=NULL
tc->flag=0
If(back->edg==NULL)
{
Then back->edg=tc
End If
}
Else
{
ptr=back->edg
while(ptr->nextptr!=NULL)
    
```



```

ptr=ptr->nextptr;
End While
}
ptr->nextptr=tc;
End Else
}

```

Algorithm7:- Convert Decimal values into Binary Values for performing Encoding

```

//In this function we need a 2d array int data[][]
dimensions are depend on our choice and a 1d array
int decimal data[] to store final ecoded binary
values
Convert
Decimal(data[i][j],decimaldata[k],CurrentPosition,
multi)
{
    Initialize i=0,j=0
    Start While(data[i][j] != -1)
    {
        Initialize decimalData[i]=0
        Start While(data[i][j] != -1)
        {
            If(CurrentPosition == 0) Then
            {
                Assign CurrentPosition = 4
                End If
            }
            multi =Call power( CurrentPosition )
            decimalData[i] = decimalData[i] +
            (data[i][j] * multi)
            Increment j=j+1
            Decrement CurrentPosition =
            CurrentPosition - 1
            End While
        }
        Assign j = 0
        Increment i=i+1
        End While
    }
    decimalData[i] = -1
    Initialize i=0 and j=0
    Start While(data[i][j] != -1)
    {
        Start While(data[i][j] != -1)
        {
            Print data[i][j]
            Increment j = j + 1
            Increment len = len + 1
            Print decimalData[i]
            Assign j= 0 , len = 0
            Increment i = i + 1
            End While
        }
        End While
    }
}

```

Algorithm 8:-Algorithm for Crossover operation

```

Crossover (data [ ][ ])
{
    For i = 0 to 3
    {
        For j= 0 to 15
        {
            If (j==4)
            {
                Data [i][j] =! data[i][j]
            }
            If (j==11)
            {
                Data [i][j]=!data[i][j]
            }
        }
    }
}

```

Algorithm 9:-Algorithm for Performing Mutation

```

Mutation(data[i][j],position, counter, limit) //For
performing Mutation we need a 2d array
{
    Initialize position = 0, i = 1, j = 0 and
    counter = 1
    Start while( i < limit)
    {
        pos=pos+4
        Increment i=i+1
        End While
    }
    Assign i=0
    Start while( data[i][0] != -1 )
    {
        Assign counter=1
        Assign j=0
        Start while( j < 4 )
        {
            If( counter == 3 ) Then
            data[i][j] = 1
            End If
            Else data[i][j]=0
            End Else
            Increment counter = counter + 1
            Increment j=j+1
        }
    }
}

```



```

Assign j=pos
Assign counter=1
Start while( j < ( pos + 4 ) )
{
    If( counter == 3 )
    {
        Assign data[i][j]=1
        End If
    }
    Else data[i][j]=0
    {
        End Else
    }
    Increment counter = counter + 1
    Increment j = j + 1
    End While
}
Increment i = i + 1
Printf "After Mutation " Call displaybin(data)
//Call to displaybin() function with passing an
argument data.
    Call convertDecimal( data ) //Call
convertDecimal function with passing an argument
data
}
}

Start while( position >= 0 )
{
    data[ 1 ][ k + difference ] = popbin(&top)
    Decrement position=position - 1
    Increment k = k + 1
    End While
    position = k - length
    If( length < 4 ) Then
    {
        Start while( difference > 0 )
        Assign data[1][position] = 0
        Decrement difference =difference - 1
        Increment pos = pos + 1
        Increment k = k + 1
        End while
    }
    End If
}
End For
}
Assign data[1][k] = -1
Increment l = l + 1
Assign j = 0
End For
}
Assign data[1][0] = -1
Call crossover(data) //Call to
Crossover function with passing an argument data.
}
}
    
```

Algorithm 10:- Algorithm for Convert binary data into decimal data.

We need a self referential structure edge of pointer type and a 2d array data[i][j]

```

ConvertBinary(struct arc
p[i][j],weight,length,difference,position,i,,j,k,l,binar
y,data[i][j])
{
    Assign struct stackbin *top = Null
    //Assign Null in to top of Dynamic Stack
    {
        Initialize l=0 and j=0
        Start For( i=0 ; p[i][j] != Null ; i++)
        Initialize k=0
        Start For( j=0 ; p[i][j] != Null ; j++)
        {
            Assign length = 0
            weight = p[i][j] -> w
            Start while( weight > 0 )
            {
                binary = weight % 2
                Call pushbin( binary , &top )
                weight = weight / 2
                Increment length=length+1
                End While
            }
        }
        difference = 4 - length
        position = length - 1
    }
}
    
```

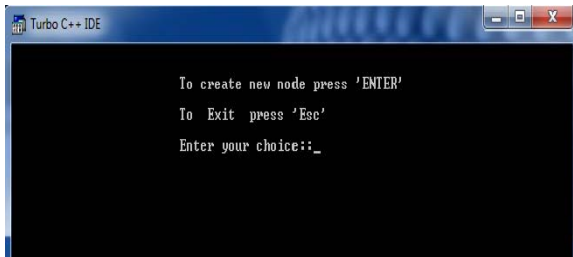
Algorithm 11:- Algorithm for Main Function

We need a pointer variable source of node structure type and Source and Destination in Graph

```

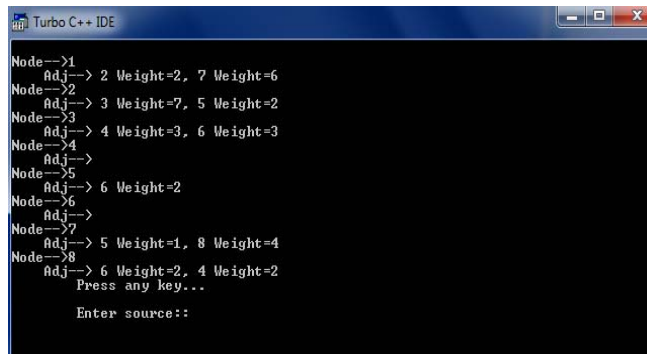
Main(source,destination,struct node *source)
{
    Initialize source = Null
    Call createNode(&source)
    If(source != Null) Then
    {
        Call connectNodes(source)
        Input source and destination
        Call randomPath(source)
        Call path(source,destination,Source)
    }
}
    
```

6. RESULT:



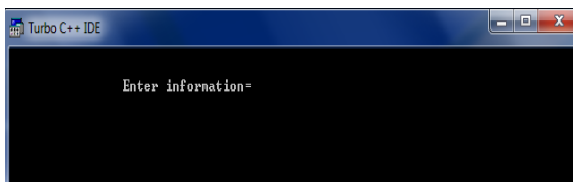
```
To create new node press 'ENTER'
To Exit press 'Esc'
Enter your choice:~
```

Figure 6.1: Result for identifying path from source to destination



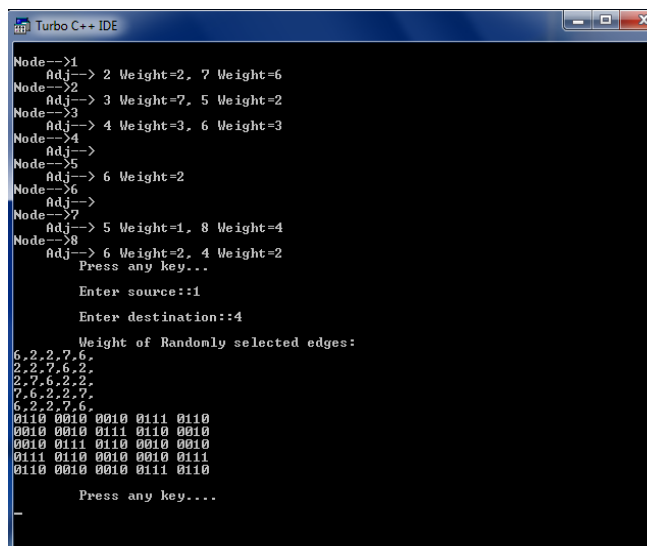
```
Node-->1
Adj--> 2 Weight=2, 7 Weight=6
Node-->2
Adj--> 3 Weight=7, 5 Weight=2
Node-->3
Adj--> 4 Weight=3, 6 Weight=3
Node-->4
Adj-->
Node-->5
Adj--> 6 Weight=2
Node-->6
Adj-->
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...
Enter source:~
```

Figure 6.5: Result for identifying source and destination node.



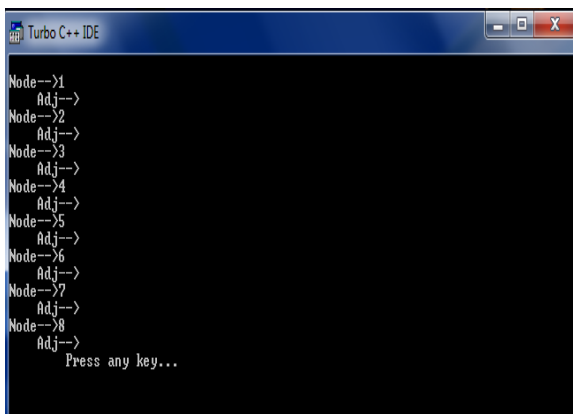
```
Enter information=
```

Figure 6.2 : Result for creating a node



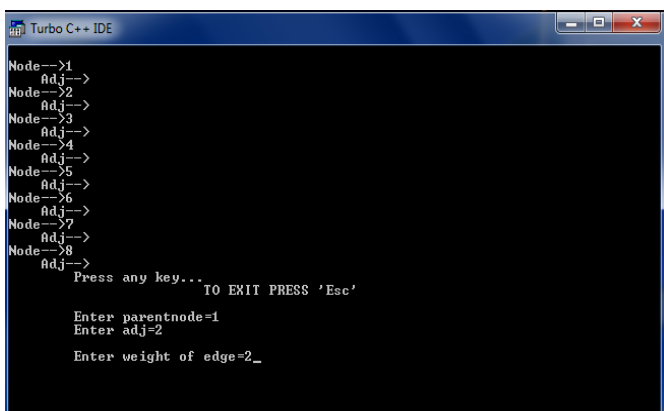
```
Node-->1
Adj--> 2 Weight=2, 7 Weight=6
Node-->2
Adj--> 3 Weight=7, 5 Weight=2
Node-->3
Adj--> 4 Weight=3, 6 Weight=3
Node-->4
Adj-->
Node-->5
Adj--> 6 Weight=2
Node-->6
Adj-->
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...
Enter source:~1
Enter destination:~4
Weight of Randomly selected edges:
6,2,2,7,6,
2,2,7,5,2,
2,7,6,2,2,
7,6,2,2,7,
6,2,2,7,6,
0110 0010 0010 0111 0110
0010 0010 0111 0110 0010
0010 0111 0110 0010 0010
0111 0110 0010 0010 0111
0110 0010 0010 0111 0110
Press any key....
```

Figure 6.6: Result for Convert Decimal values into Binary Values for performing Encoding



```
Node-->1
Adj-->
Node-->2
Adj-->
Node-->3
Adj-->
Node-->4
Adj-->
Node-->5
Adj-->
Node-->6
Adj-->
Node-->7
Adj-->
Node-->8
Adj-->
Press any key...
```

Figure 6.3: Result for Connect Nodes which are present in the whole graph



```
Node-->1
Adj-->
Node-->2
Adj-->
Node-->3
Adj-->
Node-->4
Adj-->
Node-->5
Adj-->
Node-->6
Adj-->
Node-->7
Adj-->
Node-->8
Adj-->
Press any key...
TO EXIT PRESS 'Esc'
Enter parentnode=1
Enter adj=2
Enter weight of edge=2~
```

Figure 6.4: Result for Identify the parent node, adjacent node & weight of the edge.


```
Turbo C++ IDE
Node-->1
Adj--> 2 Weight=2, 7 Weight=6
Node-->2
Adj--> 3 Weight=7, 5 Weight=2
Node-->3
Adj--> 4 Weight=3, 6 Weight=3
Node-->4
Adj-->
Node-->5
Adj--> 6 Weight=2
Node-->6
Adj-->
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...

Enter source::1

Enter destination::4

Weight of Randomly selected edges:
6,2,2,7,6,
2,2,7,6,2,
2,7,6,2,2,
7,6,2,2,7,
6,2,2,7,6,
0110 0010 0010 0111 0110
0010 0010 0111 0110 0010
0010 0111 0110 0010 0010
0111 0110 0010 0010 0111
0110 0010 0010 0111 0110

Press any key...

After CROSSOVER:
0110 0010 0111 0111 0110
0010 0010 0010 0110 0010
0010 0110 0010 0010 0010
0111 0111 0110 0010 0111
0110 0010 0010 0111 0110

Press any key...
```

Figure 6.7: Result for Crossover operation

```
Turbo C++ IDE
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...

Enter source::1

Enter destination::4

Weight of Randomly selected edges:
6,2,2,7,6,
2,2,7,6,2,
2,7,6,2,2,
7,6,2,2,7,
6,2,2,7,6,
0110 0010 0010 0111 0110
0010 0010 0111 0110 0010
0010 0111 0110 0010 0010
0111 0110 0010 0010 0111
0110 0010 0010 0111 0110

Press any key...

After CROSSOVER:
0110 0010 0111 0111 0110
0010 0010 0010 0110 0010
0010 0110 0010 0010 0010
0111 0111 0110 0010 0111
0110 0010 0010 0111 0110

Press any key...

After Mutation:
0010 0010 0111 0111 0010
0010 0010 0010 0110 0010
0010 0110 0010 0010 0010
0010 0111 0110 0010 0010
0010 0111 0110 0010 0010
0110 0010 0010 0111 0110

Press any key...

0010 0010 0111 0111 0010      46
0010 0010 0010 0110 0010      28
0010 0110 0010 0010 0010      34
0010 0111 0110 0010 0010      36
0010 0010 0010 0111 0010      30
1 >2 >3 >4
1->3->4
1->8->6->
```

Figure 6.9: Result for Convert binary data into decimal data

```
Turbo C++ IDE
Node-->4
Adj--> 4 Weight=3, 6 Weight=3
Node-->4
Adj-->
Node-->5
Adj--> 6 Weight=2
Node-->6
Adj-->
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...

Enter source::1

Enter destination::4

Weight of Randomly selected edges:
6,2,2,7,6,
2,2,7,6,2,
2,7,6,2,2,
7,6,2,2,7,
6,2,2,7,6,
0110 0010 0010 0111 0110
0010 0010 0111 0110 0010
0010 0111 0110 0010 0010
0111 0110 0010 0010 0111
0110 0010 0010 0111 0110

Press any key...

After CROSSOVER:
0110 0010 0111 0111 0110
0010 0010 0010 0110 0010
0010 0110 0010 0010 0010
0111 0111 0110 0010 0111
0110 0010 0010 0111 0110

Press any key...

After Mutation:
0010 0010 0111 0111 0010
0010 0010 0010 0110 0010
0010 0110 0010 0010 0010
0010 0111 0110 0010 0010
0010 0111 0110 0010 0010

Press any key...
```

Figure 6.8: Result for Mutation operations

```
Turbo C++ IDE
Node-->1
Adj--> 2 Weight=2, 7 Weight=6
Node-->2
Adj--> 3 Weight=7, 5 Weight=2
Node-->3
Adj--> 4 Weight=3, 6 Weight=3
Node-->4
Adj-->
Node-->5
Adj--> 6 Weight=2
Node-->6
Adj-->
Node-->7
Adj--> 5 Weight=1, 8 Weight=4
Node-->8
Adj--> 6 Weight=2, 4 Weight=2
Press any key...

Selected path:
1->3->5
Total Weight=6
```

Figure 6.10: Result for (Main function) shortest path between source to destination



6. CONCLUSION:

Genetic Algorithm provides a useful problem solving technique. The proposed approach shows, how GA can be used to solve a very general version of shortest path problem. A GA encoding along with the genetic operators is defined. The performance of the algorithm is better than previous work. For the practical implementation of the proposed work Coding of the algorithm is also included. This technique can be very useful to evaluate the shortest path in various networks. This research work presented a genetic algorithm for solving the SP routing problem. The 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. In essence, it maintains the diversity of population thereby avoiding local traps. A treatment for infeasible solutions (chromosomes) has also been investigated without unduly compromising on computational requirements. The proposed algorithm can search the solution space effectively and speedily compared with other algorithms.

REFERENCES:

- [1]. E.W. Dijkstra, "A note on two papers in connection with graphs", *Numeriske Mathematics* 1 pp.269-271, 1959.
- [2]. D.Eppstein, "Finding the k shortest paths", *SIAM journal on Computing* 28(2) pp.653-674, 1998.
- [3]. R.W Floyd, "Algorithm 97: Shortest paths", *Communications of the ACM* 5 pp.345- 357, 1962.
- [4]. A.A. Chien, J.H. Kim, "Planar-adaptive routing:low-cost adaptive networks for multiprocessors", *J. ACM* 42(1) pp. 91-123 1995.
- [5]. W.J.Dally,H Aoki, "Deadlock-free adaptive routing in multicomputer networks using Virtual channels", *IEEE Trans. Parallel Distribu. Systems* 4(4) pp.466-475, 1993.
- [6]. J. Duato, "A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks", *IEEE Trans.Parallel Distrib. Systems* 6(10) pp.1055-1067,2005.
- [7]. M. Koibuchi , A . Funahashi , A . Jouraku, H . Amano, "L-turn routing:An adaptive routing in irregular networks", *Proc. International Conference on Parallel Processing,* pp.374-383 Sep 2001.
- [8]. F. Silla, J. Duato, "High -performance routing in networks of workstations with irregular Topology," *IEEE Trans. Parallel Distrib. Systems* 11(7) pp. 699-719, 2000.
- [9]. N. J. Boden ,et.al, "Myrinet:a gigabit-per-second local area network", *IEEE Micro.* 15(1) pp.29-35, 1995.
- [10]. T. Kudoh, S. Nishimura, J. Yamamoto, H. Nishi,O. Tatebe,H.Amano, "RHINET:A Network for high performance parallel computing using locally distributed computing", *Proc IWIA* pp. 69-73, Nov 1999.
- [11]. J.C.Martinez,J.Flich,A.Robles,P.Lopez,J. Duato, "Supporting adaptive routing in IBA switches", *Systems Architect* 49 pp. 441-449, 2004.
- [12]. Baransel C,Dobosiewicz W, Gburzynski p, "Routing in multihop packet switching networks:Gb/s challenges", *IEEE Network* 9(3) pp.38-61, 1995.
- [13]. Beaubrun R, Pierre S, "Routing algorithm for distributed communication networks", *Proc 22nd IEEE Conference on Computer Networks,LCN* 97 pp. 99-105, Nov 1997.
- [14]. Kershenbaum A, Kermani P,Grover GA, "MENTOR:an algorithm for mesh Network topological optimization and routing", *IEEE Trans Comm* pp.503-513, 1991.
- [15]. Khasnabish B, "A new method for evaluating packet routing policies in supra-high-Speed metropolitan (or wide) area networks", *Comp Networks ISDN Syst* pp.195- 216, 1993.
- [16]. Suk-Gwon C, "Fair integration of routing and flow control in communication networks", *IEEE Trans Commun* 40(4) pp. 821-34, 1992.
- [17]. Courtois PJ,Semal P, "flow assignment algorithm based on the flow deviation method", *Proc of the ICC* pp.77-83, 1980.
- [18]. Gavish B, "Topological design of computer networks-the overall design problem", *Eur J Oper Res* 58 pp. 149-72, 1992.
- [19]. Neumann I, "System A. For priority routing and capacity assignment in packet switched networks",*Ann Oper Res* 36 pp. 225-46, 1992.
- [20]. Lee S,Chang S, "Neural network for routing of communication networks with Unreliable components", *IEEE Trans Neural Networks* 4(5) pp. 854-63, 1993.
- [21]. Mehmet M,Kamoun F, "Neural networks for shortest path computation and routing In

- computer networks”, *IEEE Trans Neural networks* 4(6) pp.941-54, 1993.
- [22]. Moopenn A, Thakoor AP, Duong T, “A neural network for Euclidean distance minimization”, *Proc IEEE Int Conf Neural Networks* 2 349-56, 1988.
- [23]. *Internetworking Technology Handbook: Internet Protocols (IP)*, Cisco Sytems, Inc., 2002.
- [24]. Chyzy Mariusz, Kosinski Witold, “Evolutionary Algorithm for State Assignment of Finite State Machines”, *Proc IEEE of the Euromicro Symposium on Digital System Design (DSD’02)* pp. 7695-99, 2002.
- [25]. “Evolutionary Computation”, *IEEE Transactions on Communication Volume 6, Issue 6, Dec 2002* pp. 566 – 579 Digital Object Identifier 10.1109/TEVC.2002.804323.
- [26]. Yinzhun Li, Ruichun He, Yaohuang Guo, “Faster Genetic Algorithm for Network Path”, *The Sixth International Symposium on Operations Research and Its Applications* pp. 382-389, 2000.
- [27]. W. Stallings, *High-Speed Networks: TCP/IP and ATM Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1998.
- [28]. M. K. Ali and F. Kamoun, “Neural networks for shortest path computation and routing in computer networks”, *IEEE Trans. Neural Networks*, vol. 4, pp. 941-954, Nov. 1993.
- [29]. D. C. Park and S. E. Choi, “A neural network based multi-destination routing algorithm for communication network” in *Proc. Joint Conf. Neural Networks*, pp. 1673-1678, 1998.
- [30]. C. W. Ahn, R. S. Ramakrishna, C. G. Rang, and I. C. Choi, “Shortest path routing algorithm using hopfield neural network”, *Electron. Lett.*, vol. 37, no. 19, pp. 1176-1178, Sept. 2001.
- [31]. M. Munemoto, Y. Takai, and Y. Sato, “A migration scheme for the genetic adaptive routing algorithm”, in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, pp. 2774-2779, 1998.
- [32]. J. Inagaki, M. Haseyama, and H. Kitajima, “A genetic algorithm for determining multiple routes and its applications”, in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 137-140, 1999.
- [33]. Y. Leung, G. Li, and Z. B. Xu, “A genetic algorithm for the multiple destination routing problems”, *IEEE Trans. Evol. Comput.*, vol. 2, pp. 150-161, Nov. 1998.
- [34]. G. Syswerda, “Uniform crossover in genetic algorithms”, in *Proc. 3rd Int. Conf. Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann, pp. 2-9, 1989.
- [35]. G. Tufte and P. C. Haddow, “Prototyping a GA pipeline for complete hardware evolution”, in *Proc. 1st NASA/DoD Workshop on Evolvable Hardware*, pp. 76-84, 1999.
- [36]. X. Hue, “Genetic algorithms for optimization: Background and applications”, Edinburgh Parallel Computing Centre, Univ. Edinburgh, Edinburgh, Scotland, Ver 1.0, Feb. 1997

AUTHOR PROFILES:



Dr. Anu Chaudhary received MCA degree from Madras University in 2001 and Ph.D degree in Computer Science from G.K. University, Hardwar (U.K)-India, in 2010. He is currently working as an Associate Professor at AKG Engineering College Ghaziabad (U.P)-India. His interests are in Computer Communication and Performance evaluation of networks.



Mr. N.K. Pandey received the B.Tech. degree in Computer Science and engineering from the Dr. Bhim Rao Ambedkar University, Agra (U.P)- India, in 2003. He received M.Tech. degree in Computer Science engineering from Shobhit University Meerut (U.P)-India in 2010. Currently, he is Working as Assist. Professor in College Of Engineering Roorkee in department of Computer Science & Engineering . His interests are in Theory of Automata & Formal Language, Operating System.