Many real-world problems from operations research (OR) / management science (MS) are very complex in nature and quite hard to solve by conventional optimization techniques. One of them is Network Design which is used extensively in practice in an ever expanding spectrum of applications. Network optimization models such as shortest path, assignment, maxflow, transportation, transshipment, spanning tree, matching, traveling salesman, generalized assignment, vehicle routing, and multi-commodity flow constitute the most common class of practical network optimization problems. However, there is a large class of network optimization problems for which no reasonable fast algorithms have been developed. And many of these network optimization problems arise frequently in applications. Given such a hard network optimization problem, it is often possible to find an efficient algorithm whose solution is approximately optimal. Among such techniques, the genetic algorithm (GA) is one of the most powerful and broadly applicable stochastic search and optimization techniques based on principles from evolution theory. Simulating natural evolutionary processes of human beings results in stochastic optimization techniques called evolutionary algorithms (EAs) that can often outperform conventional optimization methods when applied to difficult real-world problems. EAs mostly involve metaheuristic optimization algorithms such as genetic algorithms (GA) [33, 37], evolutionary programming (EP), evolution strategies (ES), genetic programming [62,
Network Design Using Genetic Algorithm

63](GP) [38, 39]. Among them, genetic algorithms are perhaps the most widely known type of evolutionary algorithms used today.

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 approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space". Before applying the genetic algorithm approach, it is important to understand the representation of network. A network can be represented in different way like adjacency matrix or adjacency list. In this research work network is represented as an adjacency matrix.

5.1 Network Representation

The Network design problem can be considered as an undirected or directed graph, and represented with the help of adjacency matrix.

A Graph with node set $N = \{1, 2, \cdots, n\}$ is specified by an $(n \times n)$-matrix $A= (a_{ij})$, where $a_{ij} = 1$ if and only if $(i, j)$ is an arc of $G$, and $a_{ij} = 0$ otherwise. $A$ is called the adjacency matrix of $G$.

The adjacency matrix stores the distance in the form of number between two nodes and stores zero (0) in the case of diagonal of the matrix for same node to same node or non availability of the path. Figure 5.1 shows the various independent networks at different location where each location has one or more than one network. To connect these independent networks a graph can be considered which may be in the form of
directed or undirected, depending on the availability of the path. In figure 5.2(a), backbone network of figure 5.1 is shown. and figure 5.2(b) shows an undirected graph of backbone network of figure 5.2(a), and it adjacency matrix is shown in Table5.1. The diagonal of this Table5.1 shows only zero (0) means no path from same node to same node. Similarly other zeros represent non availability of the path. From figure 5.2, there is no path between node-1 to node-5, so in the adjacency matrix the distance from node-1 to node-5 is shown zero(0). Since this is an undirected graph, so from node-5 to node-1 will also be zero (0).

Figure 5.1 various independent networks
Network Design Using Genetic Algorithm

Figure 5.2(a) Backbone Network of fig 5.1

Figure 5.2(b) Undirected Graph of fig 5.2(a)
Table 5.1 Adjacency matrices of figure 5.1 graph

<table>
<thead>
<tr>
<th>node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In figure 5.3, a directed graph is shown of the same figure 5.1 and its adjacency matrix is shown in Table 5.2. The diagonal of this Table 5.2 shows only zero (0) means no path from same node to same node. Similarly other zeros represent non-availability of the path.

Figure 5.3 Directed Graph of fig 5.1
Table 5.2 Adjacency matrices of figure 5.2 graph

<table>
<thead>
<tr>
<th>node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If the path between all locations are available, then a complete graph will be used to represent the figure backbone network. Figure 5.4 shows the complete graph of figure 5.1.
5.2 Genetic Algorithm Approach

In genetic algorithm, the first question is how to represent the problem? The same problem can be represented in different way.

The basic genetic algorithm approach is started with the initialization of the population. Afterward population is evaluated and selected. Selected population are operated with the genetic operators and again evaluated. If the result is found, it is stopped otherwise the same process is repeated. The basic genetic algorithm is:

1. Initialisation of parent population
2. Evaluation
3. Selection
4. Crossover/recombination
5. Mutation
6. Evaluate child and Go to step 3 until termination criteria satisfies

5.2.1 Population Initialization

The first phase of the genetic algorithm is to initialize the population. The population means generation of chromosomes and it is also called parent population or parent chromosome. Generation of chromosome is dependent upon the problem presentation.

There are two parameters to be decided for initialization: the initial population size and the procedure to initialize the population. Initially, researchers thought that the population size needed to increase exponentially with the length of the chromosome string in order to generate good solutions. Recent studies have shown, however, that satisfactory results can be obtained with a much smaller population size. There are two ways to generate the initial population—random initialization and heuristic initialization. Random method, where for each gene, randomly generate an integer from a range of one to the number of nodes. The initial chromosomes need not represent a legal or feasible tree. In this thesis random method is used.
Chromosome Description

Table-5.3 shows ten (10) sets of randomly generated chromosomes. Each bit of the chromosome shows the connectivity with the corresponding position node. The logic behind association is that, the node [1] is connected with node 2; node [2] is connected with 5 and so on.

Table 5.3 Randomly generated chromosomes

<table>
<thead>
<tr>
<th>Node →</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosom-1</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>Chromosom-2</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Chromosom-3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Chromosom-4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Chromosom-5</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Chromosom-6</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Chromosom-7</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>10</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Chromosom-8</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Chromosom-9</td>
<td>9</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Chromosom-10</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5.5 Illegal Spanning Tree based on chromosome-1.
Figure 5.6 Illegal Spanning Tree based on chromosome-2

Figure 5.7 Illegal Spanning Tree based on chromosome-3
Figure 5.8 Illegal Spanning Tree based on chromosome-4

Figure 5.9 Illegal Spanning Tree based on chromosome-5
Figure 5.10 Illegal Spanning Tree based on chromosome-6

Figure 5.11 Illegal Spanning Tree based on chromosome-7
Network Design Using Genetic Algorithm

Figure 5.12 Illegal Spanning Tree based on chromosome-8

Figure 5.13 Illegal Spanning Tree based on chromosome-9
After generating the chromosomes shown in Table 5.3, and drawing the tree based on these chromosomes (Fig 5.5 to Fig 5.14), it has been found that no derived solutions are spanning tree. These all chromosomes are applied with respect to complete graph shown in figure 5.4.

Following reasons have been found for being illegal tree-

- Figure 5.5 Illegal Spanning Tree based on chromosome-1, *because a cycle has been formed (5-8-9-10-7-6-5)*
- Figure 5.6 Illegal Spanning Tree based on chromosome-2, *because a cycle has been formed (1-2-5-3-1)*
- Figure 5.7 Illegal Spanning Tree based on chromosome-3, *because of isolated edge (8-10) and a cycle (5-4-7-6-5)*
- Figure 5.8 Illegal Spanning Tree based on chromosome-4, *because of multiple isolation (9-10), (5-4-8) and (3-1-2……from 1-6-7)*
- Figure 5.9 Illegal Spanning Tree based on chromosome-5, *because of self loop(2-2)*
Network Design Using Genetic Algorithm

- Figure 5.10 Illegal Spanning Tree based on chromosome-6, \textit{because of cycle (5-2-4-10-8-5) and degree violation if degree of node-5 is 3.}
- Figure 5.11 Illegal Spanning Tree based on chromosome-7, \textit{because of multiple cycles (1-2-5-3-1), (4-7-6-4) and (8-9-10-8).}
- Figure 5.12 Illegal Spanning Tree based on chromosome-8, \textit{because of self loop(3-3)}
- Figure 5.13 Illegal Spanning Tree based on chromosome-9, \textit{because of isolated edge(1-9), in this case there is no self loop and no cycle but isolation is found.}
- Figure 5.14 Illegal Spanning Tree based on chromosome-10, \textit{because of multiple self loop (1-1), (4-4) and an isolated edge (8-10-9)}

All these reasons are found for complete graph where direct path is available between any two nodes. Further directed graph (figure 5.3) is considered where all these reasons are available for illegal tree, but there is one more reason because of path constraint. Following chromosome shows the reason of being illegal.

\begin{table}[h]
\centering
\caption{Randomly generated chromosome}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|}
\hline
 Node $\rightarrow$ & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
 Chromosom-11 & & & & & & & & & & \\
\hline
\end{tabular}
\end{table}

Figure 5.15 Illegal Spanning Tree based on chromosome-11
• Figure 5.15 Illegal Spanning Tree based on chromosome-10 and directed simple graph (figure 5.3), because of path constraint (1-5) and (10-7), and (2-1)(9-7) for non directional.

5.2.2 Fitness Evaluation

Fitness evaluation is to check the solution value of the objective function subject to the problem constraints. In general, the objective function provides the mechanism evaluating each individual. However, its range of values varies from problem to problem. To maintain uniformity over various problem domains, one may use the fitness function to normalize the objective function to a range of 0 to 1. The normalized value of the objective function is the fitness of the individual, and the selection mechanism is used to evaluate the individuals of the population. When the search of GA proceeds, the population undergoes evolution with fitness, forming thus a new population. At that time, in each generation, relatively good solutions are reproduced and relatively bad solutions die in order that the offspring composed of the good solutions are reproduced. To distinguish between the solutions, an evaluation function (also called fitness function) plays an important role in the environment, and scaling mechanisms are also necessary to be applied in objective function for fitness functions. When evaluating the fitness function of some chromosome, decoding procedure is designed depending on the chromosome.

Evaluation is the most important phase of genetic algorithm where chromosomes are evaluated. If the required result is achieved then the process is terminated otherwise next generation is called. Evaluation is based on the fitness function and fitness function is the back bone of evaluation. To apply the fitness function it is important to know why chromosomes are unfit? In this previous section (5.2.1) it has been
observed the reason of illegal chromosomes. The following are the reasons of unfitness of chromosomes which leads to illegal spanning tree.

1. **Self loop**

2. **Cycle**

3. **Isolation**

4. **Degree Constraint**

5. **Path Constraint**

For all these reasons chromosomes are evaluated. To evaluate the chromosomes, for each of these reasons, fitness functions have been developed.

**Notation of Functions**

All these five functions accept the input in the form of a matrix (chromosomes) \( m \times n \), then calculate the fitness in the form of 0 and 1. Chromosome has been passed from the main function. All these functions calculate the fitness for each of the chromosome.

Last column of each chromosome has fitness value.

Chromosomes\((m \times n)\) : 

\( m = \text{No. of nodes} = \text{Row} \)

\( n = \text{No. of chromosomes} = \text{Col} \)

1. **Self Loop**

Self loop is formed when the position of node and the bit of chromosome, both are equal (figure 5.12).

For the connected graph
\[ G = (V, E) \]

Where \( V = \{v_1, v_2, \ldots , v_n\} \), sets of vertices
\( E = \{e_1, e_2, \ldots , e_{n-1}\} \), sets of edges where each edge \( e_k \) is associated with vertices \( (v_i, v_j) \)
\[
(v_i, v_j) \in e_k
\]

If \( i == j \) then it is called self loop for vertex \( v \).

---

**Procedure: Self Loop**

```plaintext
selfloop(chromosomes)

Begin

for i=1 to row do

    set 0 to fit;

    for j= 1 to (col – 1) do

        if(chromosomes(i, j) not equal to  j)
            Add 1 to fit;

    end

end

Accumulate  fit to chromosomes(i,s(2))
{chromosomes(i,s(2))= chromosomes(i,s(2))+ fit;}

end

return;

End
```

---

It returns 0 for each self loop and 1 for each non self loop occurrence. For 10 node network, 10 is the maximum fitness point for each chromosome for self loop.

2. Cycle

This is one of the most important works of this research work. A function is developed to detect the cycle in solution derived on the basis of randomly generated
Network Design Using Genetic Algorithm

chromosomes. When the solution is given by the chromosomes, it is completely unknown that whether it is tree or graph. It is also not known that, if it is graph then whether it is connected graph or unconnected graph. This function works in any of the condition.

---

**Procedure: Cycle**

Cycle(chromosomes)

**Begin**

Set k=0; t=(-1); b=1;e=5;

for i = 1 to N(number of node) do

new=0; s=i;

for j = 1 to(N + 1) do

if (new equal to 0)

set check = s;

else

check = chromosomes(s);

end

set l=1;

while(l<=k)do

if (p(l) equal to check)

if (new equal to 0)

break;

end

if (l great than equal to b)

if (k equal to (l+1))

t=-1;

break;
b=k+1;
come out from while loop;
end
if (k greater than(l+1))
e=0;
come out from while loop
end
end
if (l greater than b)
t=-1;
b=k+1;
come out from while loop
end
end
increment l by 1;
end
if (e == 0)
come out from j for loop;
end
if (l greater than k)
increment k by 1;
p(k) = check;
increment t by 1;
end
if (t equal to -1)
come out from j for loop;
end

if (new not equal to 0)
    s = chromosomes(s);
end

if (new equal to 0)
    new = 1;
end

end

if (e equal to 0)
    come out from loop
end

end

if (e equal to 0)
    disp('cycle');
else
    disp('no cycle');
end

End

This function checks the existence of cycle for each chromosome. In the case of existence of cycle it allocates 0 other wise 1. It returns 1 for non cycle and 0 for cycle for each chromosome.

**Cycle Description:**

To explain the working of this function, following example is considered:

<table>
<thead>
<tr>
<th>Table 5.5</th>
<th>Randomly generated chromosome</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node →</strong></td>
<td>1</td>
</tr>
<tr>
<td>Chromosom-12</td>
<td>2</td>
</tr>
</tbody>
</table>
A solution is drawn with the chromosome-12 from the Table 5.5

If it is started from node-1, and visit the node-2 given by chromosome12, at the same time a list is maintained to record the newly visited node. There must not be repeated entry of visited node in the list.

<table>
<thead>
<tr>
<th>Array location</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored nodes</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.17(a) List to store visited nodes
After storing the node-8 at location 7 in the list, the next node to be stored is node-9 because as per the chromosome12 (Table 5.5), but it already visited node and the location difference will be <=2, so a new search is started by finding the next not visited node with maintaining the ascending order, in this regard node-5 is the next node which has to be visited and this process continues until all the node are not visited. After the completion the list will be (Figure 5.17(b)) and the result is “CYCLE”.

<table>
<thead>
<tr>
<th>Array location</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored nodes</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Figure 5.17(b) List to store visited nodes

Now another case is considered with little variation where cycle is dissolved. Chromosome-13 (Table-5.6) and Figure 5.18.

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosom-13</td>
<td>2</td>
<td>10</td>
<td>4</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 5.18 Legal Spanning Tree based on chromosome-13
Result: NO CYCLE

3. Isolation

- This function checks the isolated edge. (Figure 5.13) Illegal Spanning Tree based on chromosome-9, because of isolated edge (1-9), in this case there is no self loop and no cycle but isolation is found.

```
Procedure: Isolation

Isolate_Check(chromosomes)

Begin

for i=1 to row do

    set 0 to count ;

    for j=1 to (col -1)

        if(j equal to chromosomes(i, chromosomes(i,j)))

            increment count by 1;

        end

    end

end

if( count greater than 2)

    chromosomes(i, col) = chromosomes(i, col) + 0;

else

    chromosomes(i, col) = chromosomes(i, col) + 1;

end

End
```

It returns 1 for non isolation and 0 for isolation for each chromosome.
4. Degree Constraint

Degree-constrained spanning tree is a spanning tree where the maximum vertex degree is limited to a certain constant $k$.

For $n$-node undirected graph $G(V,E)$; positive integer $k \leq n$.

In this research work various network of various size have been studied and a relationship is observed between degree of spanning tree and sum of degree of each of the node.

For a spanning tree of $N$ node

$$d(N) = 2*N - 2 \quad (5.1)$$

This relationship has been derived on the basis of experimental data.

Proof: To prove this relationship four spanning tree (all the spanning tree of this study has been consdired) considered network of different size has been considered.
Figure 5.19(b) Legal Spanning Tree

Figure 5.19(c) Legal Spanning Tree
Network Design Using Genetic Algorithm

Figure 5.19(d) Legal Spanning Tree

---

**Procedure: Degree Constraint**

```plaintext
degree_constraint_check(chromosomes, degree)

Begin

N = size(degree);

for i=1 to row do
    set p=0 and total_degree=0;
    for j=1 to (col-1) do
        set d to 1
        for k=1 to (col-1) do
            if(chromosomes(i, k) equal to j)
                if(chromosomes(i, k) equal to k)
                    decrement d by -1;
                end
            end
        end
        if(chromosomes(i,chromosomes(i, k))not equal to k)
```
increment d by 1;
end
end
end

if((d greater than equal to degree(1,j)) && (d less than equal to degree(2,j)))
increment p by 1;
total_degree = total_degree + d;
else
out from inner loop;
end
end
end

if((p equal to (col-1)) AND total_degree equal to (2*N(2)-2))
chromosomes(i,col) = chromosomes(i,col) + 1;
end
end

End

This function checks the degree of each node within defined degree constraint range minimum and maximum and assigns 1 to those whose degree constraint is within range and equal to (2*N-2) otherwise 0

5. Path Constraint

This function is developed to check the existence of path between two nodes. In the case of complete graph, this function is of no use, but it is useful for the directed and simple incomplete graph. Here dist_matrix is the cost matrix of the graph or network.
Procedure: Path Constraint

path constraint(chromosomes, dist_matrix)

Begin

for i=1 to row do

    set t=0;

    for j=1 to col-1 do

        set k = chromosomes(i,j);

        if(dist_matrix(j, k) not to equal 0)

            increment t by 1;

        else

            if(j equal to k)

                increment t by 1;

            end

            if(j not equal to k)

                out of loop;

            end

        end

    end

end

if(t equal to (col-1))

    chromosomes(i,col) = chromosomes(i,col) + 1;

end

End

This function checks the path constraint for each chromosome according to
availability of path from dist_matrix. If path available for each gene of the chromosome then it assigns 1 otherwise it assigns 0

5.2.3 Selection

Selection provides the driving force in a GA. During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

In this research work seven selection functions have been designed.

1. Random Selection

This selection function simply selects the chromosome on the basis of randomly generated number. Randomly generated number decide the location of the chromosome to be selected. If the number is not in the range of the chromosome location, then it replaces with the fittest chromosome.

Procedure: Random Selection

random_selection(chromosomes)

Begin

set k=1;

for i=1 to row do

    r=randomly generate a number

    if((r equal to 0) OR (r greater than row))
Network Design Using Genetic Algorithm

for l=1 to row do
    if(chromosomes(l,col)) equal to  col+3
        Set r=l;
    end
end

if(r equal to  0)
    set r=1;
end

for j=1 to col do
    new_chromosomes(k, j) = chromosomes(r, j);
end

increment k by 1;
end

End

it stores the selected chromosomes in the new_chromosomes matrix.

2. **Roulette wheel Selection I**

It is based on the simple concept of roulette wheel.

s = sum of fitness of all the chromosomes in the generated population

r = random number generated from the range (0 to s)

following example shows the concept:
Table 5.7 Randomly generated chromosome with Fitness

<table>
<thead>
<tr>
<th>Chromosomes</th>
<th>Fitness</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>81</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>93</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>104</td>
</tr>
</tbody>
</table>

Table 5.7 shows the 10 randomly generated chromosomes with its corresponding fitness value. s is accumulated sum for roulette wheel procedure.

The value of r must be in the range of s such that.

\[
11 \leq r \leq 104
\]

if r is 29.8998

\[\Rightarrow r < s(3), \text{ so 3rd chromosome will be selected and so on until } n \text{ chromosomes are not selected. Where } n \text{ is total no. of chromosomes.}\]
# Procedure: Roulette wheel Selection I

Roulette_wheel_selectionI(chromosomes)

**Begin**

\[ r = \text{randomly generated number} \]

set \( s(row,1) = 0; \)

set \( \text{temp} = 0; \)

for \( i = 1 \) to row do

\[ \text{temp} = \text{temp} + \text{chromosomes}(i,\text{col}); \]

\[ s(i) = \text{temp}; \]

end

set \( k = 1; \)

for \( i = 1 \) to row do

\[ r = \text{random generated number} \times \text{temp}; \]

for \( j = 1 \) to row do

if \( r < s(j) \)

for \( t = 1 \) to \( \text{col} \) do

\[ \text{new}_\text{chromosomes}(k,t) = \text{chromosomes}(j,t); \]

end

increment \( k \) by 1;

out of loop;

end

end

end

**End**
3. **Roulette wheel Selection II**

It is also based on the simple concept of roulette wheel but with change.

\[ s = \text{sum of fitness probability} \]

\[ \text{fitness probability} = \frac{\text{fitness}}{\text{avg}} \]

\[ \text{avg} = \frac{\text{sum of fitness}}{\text{no of chromosome}} \]

\[ r = \text{random number generated from the range (0 to s)} \]

Following example shows the concept:

\[ \text{avg} = \frac{109}{10} = 10.900 \]

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Fitness</th>
<th>Fitness probability</th>
<th>( s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>1.1009</td>
<td>1.1009</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>.9174</td>
<td>2.0183</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>.9174</td>
<td>2.9358</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>1.0092</td>
<td>3.9450</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>1.1927</td>
<td>5.1376</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>1.0092</td>
<td>6.1468</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>1.1927</td>
<td>7.3394</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0.8257</td>
<td>8.1651</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>0.9174</td>
<td>9.0826</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>0.9174</td>
<td>10.000</td>
</tr>
</tbody>
</table>
If r is 3.47 then

⇒ r < s(4), so 4th chromosome will be selected and so on until n chromosomes are not selected. Where n is total no. of chromosomes.

---

**Procedure: Roulette wheel Selection II**

Roulette_wheel_selectionII (chromosomes)

**Begin**

r= randomly generated number

set s(row,1)=0;

set temp=0;

for i=1 to row do

    temp = temp+chromosomes(i,col);

    s(i)= temp;

end

avg = temp/row;

for i=1 to row do

    temp=(chromosomes(i, col)/avg);

end

set temp=0;

for i=1 to row do

    temp=temp+(chromosomes(i,col)/avg);

    s(i)=temp;

end

temp = s(i);

set k = 1;
for i = 1 to row do
    r = random * temp;
    for j = 1 to row
        if(r less than s(j))
            for t = 1 to col
                new_chromosomes(k, t) = chromosomes(j, t);
            end
        end
        increment k by 1;
        out of loop ;
    end
end
End

4. Sort Selection

This function sorts the chromosome, then sorted chromosome is selected.

Procedure: Sort_Selection

Sort_selection(chromosomes)

Begin
    Set k=1;
    Set t=2;
    while(k less than equal to row) do
        for i=1 to row do


if(chromosomes(i, col) equal to (col+t))

for j = 1 to col do

   new_chromosomes(k, j)= chromosomes(i, j);

end

increment k by 1;

end

end

decrement t by 1

end

End

5. Fittest Selection

This function selects only the fittest chromosome up to a fixed fitness level

Procedure: Fittest_Selection

Fittest_selection(chromosomes)

Begin

Set k=1;

Set t=2;

while(k less than equal to row)

   for i=1 to row do

      if(chromosomes((i, col) equal to (col+t))

         for j=1 to col do

            new_chromosomes(k, j)=chromosomes(i, j);

         end

      end

   end

   decrement t by 1

end

End
6. Selection Sort SelectionI

This selection function is based on selection sort. It generates two random numbers for two random positions. These two position chromosomes are selected, compared and the greatest one is selected. It repeats n times where n are no of chromosome.

Procedure: Selection_Sort_SelectionI

Selection_Sort_Selection(chromosomes)

Begin

Set k=1;

Set t=2;

    for i=1 to row do
        p  = randomly generated number;
        q  = randomly generated number;
        if(p equal to 0)
            set p=1;
        end
        if(q equal to 0)
            set q=1;
        end
        if(chromosomes(p,col) greater than chromosomes(q,col))
            for j=1 to col do
                new_chromosomes(k,j)= chromosomes(p,j);
            end
            increment k by 1;
        else
            for j= 1 to col do
                new_chromosomes(k,j)=chromosomes(q,j);
            end
            increment k by 1;
        end
    end

End
7. Selection Sort SelectionII

This selection function is based on selection sort. It generates two random numbers for two random positions. These two position chromosomes are selected, compared and the smallest one is selected. It repeats n times where n are no of chromosome

Procedure: Selection_Sort_SelectionII

Selection_Sort_SelectionII(chromosomes)

Begin
Set k=1;
Set t=2;
for i=1 to row do
    p  = randomly generated number;
    q  = randomly generated number;
    if(p equal to 0)
        set p=1;
    end
    if(q equal to 0)
        set q=1;
    end
    if(chromosomes(p,col) smaller than chromosomes(q,col))
        for j=1 to col do
            new_chromosomes(k,j)= chromosomes(p,j);
        end
        increment k by 1;
else
    for j = 1 to col do
        new_chromosomes(k,j) = chromosomes(q,j);
    end
    increment k by 1;
end

End

5.2.4 Genetic Operators

A genetic operator is an operator used in genetic algorithms to maintain genetic diversity. Genetic variation is a necessity for the process of evolution. Genetic operators used in genetic algorithms are analogous to those which occur in the natural world: survival of the fittest, or selection; reproduction (crossover, also called recombination); and mutation

5.2.4.1 Crossover

In genetic algorithms, crossover is a genetic operator used to vary the programming[68,69] of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. In this research work six different crossover function is developed:
5.2.4.1.1 Variable Point Crossover

It is a single point crossover where point is changed with all pair of cromosome.

Following example explains the logic behind this crossover operator.

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>1</th>
<th>4</th>
<th>6</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Chromosome3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Chromosome4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 5.20(a)

↓

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>1</th>
<th>4</th>
<th>6</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5.20(b)

In figure 5.20(b) it has been shown that crossover will take place on first place (first point) after the crossover it will become (figure 5.20(c)).

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5.20(c)

Since it is variable point crossover, the next two crossover3 and crossover4 will be exchange their bits on second place (second point)

↓-----↓

<table>
<thead>
<tr>
<th>Chromosome3</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 5.20(d)

<table>
<thead>
<tr>
<th>Chromosome3</th>
<th>6</th>
<th>7</th>
<th>2</th>
<th>1</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome4</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 5.20(e)
**Procedure: Variable_point_crossover**

Variable_point_crossover(chromosomes)

**Begin**

Set \( t = 1 \) and \( i = 1 \);

while( \( i \) less than row) do

for \( j = 1 \) to \( t \) do

    temp = chromosomes(\( i, j \));

    shift chromosomes(\( i, j \)) = chromosomes((\( i+1 \), j));

    chromosomes((\( i+1 \), j)) = temp;

end

increment \( i \) by 2;

increment \( t \) by 1;

if(\( t \) greater than \( > \) col)

    set \( t = 1 \);

end

end

**End**

---

**5.2.4.1.2 Fixed Two Point Crossover**

It is a two point crossover where both the point begin and end, is fixed for all pair of chromosomes. These two points are randomly generated and fixed for all the chromosomes. The following example explains the logic behind this crossover operator. Same figure 5.20(a) is considered here.
After fixed two point crossover—

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>1</th>
<th>4</th>
<th>6</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Chromosome3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Chromosome4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Procedure: Fixed_two_point_crossover

```python
Fixed_two_point_crossover(chromosomes)
Begin
Set t=1 and i=1;
p= randomly generated number within the limit;
q= randomly generated number within the limit;
if(p equal to 0)
    set p=1;
end
if(q equal to 0)
    set q=1;
end
if(p greater than q)
    p1=q;
p2=p;
```
Network Design Using Genetic Algorithm

else
    p1=p;
    p2=q;
end

while( I less than row) do
    for j=p1 to p2 do
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end
    increment i by 2;
    increment t by 1;
    if(t greater than col)
        set t=1;
    end
end
End

5.2.4.1.3 Variable Two Point Crossover

It is a two point crossover where both the point begin and end, is different for each pair of chromosome. These two points are randomly generated for each pair chromosomes. Following example explains the logic behind this crossover operator. Same figure 5.20(a) is splitted here as 5.20(h) and 5.20(i).
After Variable two point crossover—

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Chromosome3</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Chromosome4</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 5.20(j)

---

**Procedure: Variable_two_point_crossover**

Variable_two_point_crossover(chromosomes)

Begin

Set \( i = 1; \)

while( \( i \) less than row)

\[ p = \text{randomly generated number within the limit;} \]

if(p equal to 0)

set \( p = 1; \)

end
q= randomly generated number within the limit;
if(q equal to 0)
    set q=1;
end
if(p greater than q)
    p1=q;
    p2=p;
else
    p1=p;
    p2=q;
end
for j=p1 to p2 do
    temp = chromosomes(i,j);
    shift chromosomes(i,j) = chromosomes((i+1),j);
    chromosomes((i+1),j)=temp;
end
increment i by 2;
end

5.2.4.1.4 Uniform Crossover

It is a multi point crossover where multiple random points are generated and bits are exchanged between these points only. These points are fixed for each pair of chromosomes. Following example explains the logic behind this crossover operator.
Randomly generated points are considered 2, 5, 8 and 9. So exchange of bits will occur between 2-5 and 8-9, remaining bits will be unchanged.

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>8</th>
<th>5</th>
<th>10</th>
<th>1</th>
<th>9</th>
<th>5</th>
<th>9</th>
<th>9</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
</tbody>
</table>

After uniform crossover-

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>8</th>
<th>4</th>
<th>3</th>
<th>4</th>
<th>9</th>
<th>5</th>
<th>9</th>
<th>3</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Procedure: Uniform_crossover**

Uniform_crossover(chromosomes)

Begin

t=1; i=1;
p= randomly generated number within the limit;
q= randomly generated number within the limit;
r= randomly generated number within the limit;
s= randomly generated number within the limit;

sorted_pos =sort(p,q,r,s); { all these four numbers are sorted}
Network Design Using Genetic Algorithm

while( i less than row) do
    for j = p to q do
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end
    for j = r to s do
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end
    increment i by 2
end

End

_____________________________________________________________________

5.2.4.1.5 Hybrid Crossover I

It is a multi point, multi parent crossover where multiple random points are generated and bits are exchanged between these points only. These points are fixed for each pair of chromosome. Exchanged are made in multiple parents at a time. Since multiparents have been used so it is called hybrid crossover. Following example explains the logic behind this crossover operator.

Consider the randomly generated points are 2,5,8 and 9. so exchange of bits will occur between 2-5 and 8-9, remaining bits will be unchanged.
Anand Kumar (Registration No: 3893)

---

**Figure 5.20(m)**

After uniform crossover-

<table>
<thead>
<tr>
<th>Chromosome1</th>
<th>8</th>
<th>4</th>
<th>3</th>
<th>4</th>
<th>9</th>
<th>5</th>
<th>9</th>
<th>3</th>
<th>4</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chromosome2</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Chromosome3</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

**Figure 5.20(n)**

---

**Procedure: Hybrid_crossover1**

```plaintext
Hybrid_crossover(chromosomes)

Begin

t=1;i=1;
p= randomly generated number within the limit;
q= randomly generated number within the limit;
r= randomly generated number within the limit;
s= randomly generated number within the limit;

sorted_pos =sort(p,q,r,s); { all these four numbers are sorted}

while( i less than row) do
```

134
Network Design Using Genetic Algorithm

for j = p to q do
    temp = chromosomes(i,j);
    chromosomes(i,j) = chromosomes((i+1),j);
    chromosomes(i+1,j) = chromosomes((i+2),j);
    chromosomes((i+2),j)=temp;
end

for j = r to s do
    temp = chromosomes(i,j);
    chromosomes(i,j) = chromosomes((i+1),j);
    chromosomes(i+1,j) = chromosomes((i+2),j);
    chromosomes((i+2),j)=temp;
end

increment i by 3; { it's a 3 parent crossover }

End

5.2.4.1.6 Hybrid CrossoverII

It is a multi point, multi parent and variable crossover where multiple random points are generated and bits are exchanged between these points only. These points are randomly generated for each parent combination of chromosome. Exchange are made in multiple parents at a time. since multiparents have been used so it is called hybrid crossover. Following example explains the logic behind this crossover operator.

There are combination of three parent. For the first three parents randomly generated points are 2,5,8 and 9. so exchange of bits will occur between 2-5 and 8-9, remaining
bits will be unchanged. For the next three parents randomly generated numbers are 1,4,7,10. so exchange will be made between 1-4 and 7-10, remaining bits will be exchanged.

| Chromosome1 | 8 | 5 | 10 | 1 | 9 | 5 | 9 | 9 | 8 | 9 |
| Chromosome2 | 4 | 3 | 5 | 7 | 1 | 8 | 9 | 9 | 8 | 7 |
| Chromosome3 | 6 | 4 | 3 | 4 | 9 | 1 | 2 | 3 | 4 | 6 |
| Chromosome4 | 5 | 1 | 4 | 9 | 6 | 2 | 1 | 3 | 10 | 1 |
| Chromosome5 | 4 | 2 | 5 | 8 | 7 | 3 | 4 | 5 | 10 | 3 |
| Chromosome6 | 3 | 3 | 6 | 7 | 8 | 2 | 1 | 9 | 1 | 9 |

| Chromosome1 | 8 | 3 | 5 | 7 | 1 | 5 | 9 | 9 | 8 | 9 |
| Chromosome2 | 4 | 4 | 3 | 4 | 9 | 8 | 9 | 3 | 4 | 7 |
| Chromosome3 | 6 | 5 | 10 | 1 | 9 | 1 | 2 | 9 | 8 | 6 |
| Chromosome4 | 4 | 2 | 5 | 8 | 6 | 2 | 4 | 5 | 10 | 3 |
| Chromosome5 | 3 | 3 | 6 | 7 | 7 | 3 | 1 | 9 | 1 | 9 |
| Chromosome6 | 5 | 1 | 4 | 9 | 8 | 2 | 1 | 3 | 10 | 1 |

Figure 5.20(o)

After Crossover-

| Chromosome1 | 8 | 3 | 5 | 7 | 1 | 5 | 9 | 9 | 8 | 9 |
| Chromosome2 | 4 | 4 | 3 | 4 | 9 | 8 | 9 | 3 | 4 | 7 |
| Chromosome3 | 6 | 5 | 10 | 1 | 9 | 1 | 2 | 9 | 8 | 6 |
| Chromosome4 | 4 | 2 | 5 | 8 | 6 | 2 | 4 | 5 | 10 | 3 |
| Chromosome5 | 3 | 3 | 6 | 7 | 7 | 3 | 1 | 9 | 1 | 9 |
| Chromosome6 | 5 | 1 | 4 | 9 | 8 | 2 | 1 | 3 | 10 | 1 |

Figure 5.20(p)

Procedure: Hybrid_crossoverII

Hybrid_crossoverII(chromosomes)

Begin

\[ t=1; i=1; \]

\[ p= \text{randomly generated number within the limit}; \]

\[ q= \text{randomly generated number within the limit}; \]

\[ r= \text{randomly generated number within the limit}; \]

\[ s= \text{randomly generated number within the limit}; \]
Network Design Using Genetic Algorithm

sorted_pos = sort(pos); { all these four numbers are sorted}

while( i less than row) do
    for j = p to q do
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes(i+1,j) = chromosomes((i+2),j);
        chromosomes((i+2),j)=temp;
    end
    for j = r to s do
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes(i+1,j) = chromosomes((i+2),j);
        chromosomes((i+2),j)=temp;
    end
    increment I by 3;{ for 3 parent combination}
    p= randomly generated number within the limit;
    q= randomly generated number within the limit;
    r= randomly generated number within the limit;
    s= randomly generated number within the limit;
    sorted_pos = sort(pos); {all these four numbers are sorted}
end

End

_____________________________________________________________________
5.2.4.2 Mutation

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.

In this research work six different mutation function is developed:

5.2.4.2.1 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.

**Procedure: MutationI**

MutationI(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_chromosome(i,j);

    end

end

**End**

**5.2.4.2.2 MutationII**

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.
Procedure: Mutation-II

mutationII(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

for i=1 to row do

    for j=1 to col-1

        mutated_chromosome(i,j) = new_chromosome(i,j);

    end

end

End
5.2.4.2.3 Random Mutation

This mutation operator mutates only 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.

**Procedure: Random_mutation**

```
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
End
```
for i=1 to row do
    for j=1 to col-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end

End

5.2.4.2.4 Swap Mutation

This mutation operator swaps two random position of each of the chromosomes.

If the randomly generated positions are 3 and 7.

\[
\begin{array}{cccccccccccc}
5 & 1 & 4 & 9 & 8 & 2 & 1 & 3 & 10 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
5 & 1 & 1 & 9 & 8 & 2 & 4 & 3 & 10 & 1 \\
\end{array}
\]

Figure 5.20(q)

After mutation-

\[
\begin{array}{cccccccccccc}
5 & 1 & 1 & 9 & 8 & 2 & 4 & 3 & 10 & 1 \\
\end{array}
\]

Figure 5.20(r)

Procedure: Swap_mutation

Swap_mutation(new_chromosome)

Begin

Set k=1;

for i=1 to row do

end
Network Design Using Genetic Algorithm

\[ p = \text{randomly generated number within the limit}; \]
\[ q = \text{randomly generated number within the limit}; \]
\[ \text{temp} = \text{new_chromosome}(i,p); \]
\[ \text{new_chromosome}(i,p) = \text{new_chromosome}(i,q); \]
\[ \text{new_chromosome}(i,q) = \text{temp}; \]

End

_____________________________________________________________________

5.2.4.2.5 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.

\[
\begin{array}{cccccccc}
\text{Chromosom} & 5 & 1 & 4 & 9 & 8 & 2 & 1 & 3 & 10 & 1 \\
\end{array}
\]

Figure 5.20(s)

After mutation-

\[
\begin{array}{cccccccc}
\text{Chromosom} & 5 & 3 & 1 & 2 & 8 & 9 & 4 & 1 & 10 & 1 \\
\end{array}
\]

Figure 5.20(t)

Procedure: Mutation_Inversion

Mutation inversion(new_chromosome)

Begin

Set \( k=1; \)

for \( i=1 \) to \( \text{row} \) 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

5.2.4.2.6 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.

\[ \begin{array}{cccccccc}
5 & 1 & 4 & 9 & 8 & 2 & 1 & 3 & 10 & 1 \\
\end{array} \]

Figure 5.20(u)
Network Design Using Genetic Algorithm

After mutation-

\[
\begin{array}{cccccccccc}
\text{Chromosome} & 5 & 1 & 3 & 4 & 9 & 8 & 2 & 1 & 10 & 1 \\
\end{array}
\]

Figure 5.20(v)

---

Procedure: Mutation_Insertion

mutation_insertion(new_chromosome)

Begin

Set k=1;

for i=1 to row do

\[ p = \text{randomly generated number within the limit}; \]
\[ q = \text{randomly generated number within the limit}; \]

sort p, q

\[ \text{temp} = \text{new_chromosome}(i, q); \]

if\( p \neq q \)

\[ x = q - 1; \]

While\( x \geq p + 1 \)

\[ \text{new_chromosome}(i, x+1) = \text{new_chromosome}(i, x); \]

decrement x by -1;

end

\[ \text{new_chromosome}(i, p+1) = \text{temp}; \]

end

End