NETWORK DESIGN USING GENETIC ALGORITHM

A Thesis submitted to

SAURASHTRA UNIVERSITY

RAJKOT, INDIA

For the award of the degree of

Doctor of Philosophy

in

Computer Science

In the Faculty of Science

Submitted by

ANAND KUMAR

(Registration No. 3893)

Under the Esteemed Guidance of

Dr. N. N. JANI

Ex. Prof. & Head, Department of Computer Science

SAURASHTRA UNIVERSITY

Director SKPIMCS (MCA)

Dean (Faculty of Computer Science & IT)

KADI SARVA VISHWAVIDYALAYA, GANDHINAGAR

April - 2010

Dedicated to my
beloved parents
and
my Vikramaditya

CERTIFICATE

This is to certify that the thesis entitled **"Network Design using Genetic Algorithm"** is a bonafide work done by Mr. Anand Kumar (Reg. No. 3893), in partial fulfillment of the requirement for the award of the degree ***"Doctor of Philosophy in Computer Science"*** at Department of Computer Science, Faculty of Science Saurashtra University under my guidance and supervision. This work is not submitted to any university for the award of any degree.

Guide

Dr. N. N. JANI
Ex. Prof. & Head, Computer Science Department
Saurashtra University
Director SKPIMCS – MCA
Dean (Computer Science)
Kadi Sarva Vishwavidyalaya, Gandhinagar

DECLARATION

This is to declare that the thesis entitled "**Network Design using Genetic Algorithm"** is a bonafide work done by me, in partial fulfillment of the requirement for the award of the degree ***"Doctor of Philosophy in Computer Science"*** at Department of Computer Science, Faculty of Science, Saurashtra University. This work is not submitted to any university for the award of any degree.

Anand Kumar
(Reg. No. 3893)
Ph.D. Scholar

Abstract

Network Design Problems are becoming increasingly critical & complex as telecommunication networks (and others) are expanded & upgraded in response to consumer's information needs. Network design 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. In this research work, a generalized network design problems (NDPs) is focused in the form of a large scale backbone network which belong to the family of NP-hard combinatorial optimization problems. The purpose of the backbone is to connect regional distribution networks and, in some instances, to provide connectivity to other peer networks. The primary objective of this research work is to develop a robust method based on genetic algorithm to solve NP-hard network design problem with minimum cost subject to a reliability constraint which meets the customer requirement. One fundamental problem in this area is the minimum spanning tree (MST) problem where all nodes in a graph have to be linked together in a circle-free structure in the cheapest possible way. The MST problem itself is easy to solve by polynomial-time algorithms like those of Prim or Kruskal, but adding additional constraints often make the corresponding optimization problem a hard one. One of these related problems is the degree-constrained MST problem, in which degree of each node is restricted with in a given range which is very important for the reliability and priority of the connecting node. Other possible constraints are path failure, node failure and connectivity which are requirement of the current network system. By adding this constraint, this network design problem becomes one of the hardest problems in NP-hard category. Due to the complexity of the problem these approaches are limited to relatively small instances with clearly less than 100 nodes when considering complete graphs. Therefore, in this research work methods have been developed to solve instances with up to 1000 and are applicable for more than

1000 nodes. However, there are also other problems that can be expressed as network design problems, such as traveling salesman problem (TSP), one has to find a round trip (Hamiltonian cycle) through a set of cities (nodes) of minimal length and Shortest Path problem. In this thesis these two problems are also considered and solved with genetic algorithm approach. Since network design is NP-hard problem and traditional heuristics have had only limited success in solving small to mid size problems. As a result, standard, traditional, optimization techniques are often not able to solve these problems of increased complexity with justifiable effort in an acceptable time period. The conventional search and optimization methods working on the commercial processors require hundreds of years to solve such a problem with limited number of components. However, evolutionary computation including Genetic Algorithms (GA) has shown promising performances to solve such problems Therefore, to overcome these problems, and to develop systems that solve these complex problems, researchers proposed using Genetic Algorithm. In this thesis it has been shown that, by this nature-inspired search method it is possible to overcome some limitations of traditional optimization methods, and to increase the number of solvable problem. In this study, Genetic Algorithm is considered as a one of the possible solutions for such kind of NP-hard problem where possible solutions are improved generation by generation and then there is more probability to find the exact solution. The main focus of this research is the consideration of up to1000 nodes and the proposed method can be applied for any possible size of the network. In this thesis various robust fitness functions have been developed. Twenty genetic operators are developed including new approaches required by the problem. Five hundred forty six different cases are considered for the fifteen different size of network. All the experimental results are described with the help of table and graph. All the developed functions are described with the help of figures and examples. Further new methods based on Genetic Algorithm have been developed for Shortest Path Problem and Traveling Salesman Problem.

All these functions and methods developed in this thesis are published in International Journals and in the proceedings of International Conference.

This research work shows that, genetic algorithm is an alternative solution for this NP hard problem where conventional deterministic methods are not able to provide the optimal solution.

Acknowledgement

First of all I am deeply grateful to my guide Prof. Dr N. N. Jani, who gave me the opportunity to do my Ph. D at the Saurashtra University and for his great supervision, support and encouragement during this work. I would also like to thank Prof. V. Leela, for her support and for fruitful discussions during my work to complete this thesis. I would like to express my special thanks to Mr. Vinod Kumar for his companionship in traveling the bumpy road towards the Ph.D. degree. I express my thanks to Ms. Aparna, for her kind support, indefatigable inspiration and continuous encouragement throughout this work. My gratitude is extended to all members of the MCA department for their help and kind support.

Last but not least I would like to express the warmest thank you to my parents, my brothers and my sister , for always backing me up and encouraging me to struggle on.

Anand Kumar

Contents

If I had a
Robust Network,
I could win this world

…Nepolian Bonapart

CHAPTER 1

Introduction

Network Design is very common problem which contributes key role in many real life applications which arose directly from everyday practice in engineering and management: determining shortest or most reliable paths in traffic or communication networks, maximal or compatible flows, or shortest tours; planning connections in traffic networks; coordinating projects; and solving supply and demand problems, electricity distribution, designing of digital circuit, designing of gas pipeline, layout planning roads and railway track, transportation and many more.

Because of vast real life application, network design has become the crucial problem in such real life applications. With the advancement of information technology our society is rapidly converting as an information society. The conversion of a society to information society means that extension of network. Each and every sector of our daily need is in the process of computerized network. Day by day it has to be extended. For developing country like India, where development is growing multidimensional, network is the primary issue. Each and every field of our daily need which is manual, has to be computerized and to materialize it, network is the key factor. If we are talking about the overload of population like the country India and China where the geographical extension is at the peak, again we need network to connect or extend these locations first physically for transportation, communication and management and then electronically in the form of computerized network,

network is required. By this discussion, the role of network is concluded in all our daily life applications and in the overall growth of the society. This is the main motivation behind this study Network Design. One fundamental problem in this area is the minimum spanning tree (MST) problem where all nodes in a graph have to be linked together in a circle-free structure in the cheapest possible way. However, there are also other problems that can be expressed as network design problems, such as various transportation and routing problems. For example shortest path problem, the famous traveling sales- man problem (TSP), one has to find a round trip (Hamiltonian cycle) through a set of cities (nodes) of minimal length[40,41]. A practical correspondent appears in the automated manufacturing of printed circuits when one wants to minimize the time required for drilling all holes by optimizing the path for moving the drill. Already this short list of problems should give a rough idea of the economical impact and therefore interest of solving such network design problems properly in general. Furthermore, network design is also important for complexity theory, an area in the common intersection of mathematics and theoretical computer science which deals with the analysis of algorithms. The term network design is involved in many contexts and there are several different aspects which deserve attention. In this study, they are regarded from a more theoretical point of view as graph theory problems, i.e. networks are modeled as graphs and optimization algorithms are applied on them. There are mainly four broad categories of network design- network topology design, network routing and flow control, network performance and network reliability. Since these all are separate category but all these categories are highly related.

In this research study, network design belongs to the category of network topology design. Further designing the topology of a large scale network can be divided into two problems, the backbone network design and the local network design. This research work is mainly focused on large scale backbone network design which is in the form of degree constraint minimum spanning tree with other constraints required by the network. The main objective behind the network design is to find the best way to connect the locations (nodes and arcs) to minimize the cost while meeting performance criterion such as transmission delay, throughput, fault tolerance and reliability. Exploring all the constraints for such a design problem, it becomes an NP-hard problem [1]. There are many methods such as Prim [3] or Kruskal [4] which solve minimum spanning tree problem in polynomial time but adding additional constraints often make the corresponding optimization problem a hard one and one of the hardest in NP category problems. There are other methods also like Breadth First Search , Depth First Search and Branch and Bound but these entire have also their limitations. These methods can only solve small networks because the number of arcs increases, the number of possible layouts grows faster than exponentially. There are other limitations also with these methods such as degree constraint for each node, fault tolerance and reliability in the case of failure of node and other constraints as per demand of the network.

Because of these complexities, these existing methods are not computationally feasible for deserving large scale network. As a result, standard, traditional, optimization techniques are often not able to solve these problems of increased complexity with justifiable effort in an acceptable time period. Therefore, to overcome these problems, and to develop systems that solve these complex problems, researchers proposed using genetic and evolutionary algorithms. Using these nature-inspired search methods it is possible to overcome some limitations of traditional optimization methods, and to increase the number of solvable problems. Given such a hard network optimization problem [2], 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. Genetic Algorithm (GA) is a probabilistic search heuristic that replicates the defining features of biological evolution: reproduction with variation, selection based on fitness, and repetition. GA maintains a population of data structures, called chromosomes that encode candidate solutions to its target problem. Attached to each chromosome is its fitness, a numerical value that indicates the quality of the solution the chromosome represents. The algorithm selects chromosomes to survive or reproduce so that those with better fitness are more likely to be selected. Crossover, also called recombination, combines genetic information from two parent chromosomes. Mutation randomly modifies one parent chromosome. When the EA has generated enough offspring, they replace their parents and the process continues. As these generations succeed each other, chromosomes that represent better solutions evolve. Therefore a heuristic search method based on genetic algorithm is developed to design such network, which has minimum cost and satisfy the required constraint demanded by the system.

This thesis is located in the area of combinatorial optimization, focusing on NP hard network design problems that occur in real world where multiple local area networks are interconnected by a backbone network. Depending on the demands of such a network, the underlying problem can either be formulated as the Generalized Minimum Spanning Tree problem or the even harder Degree Constrained Minimum Spanning Tree Problem. Given a connected, undirected graph G with n nodes, a spanning tree T is a subgraph of a G that connects all of G's nodes and contains no cycles. When every edge (i, j) is associated with a numerical costs c_{ij} , a minimum spanning tree (MST) is a spanning tree of the smallest possible total edge cost

$$C = \sum_{(i,j)\in T} c_{ij} \tag{1}$$

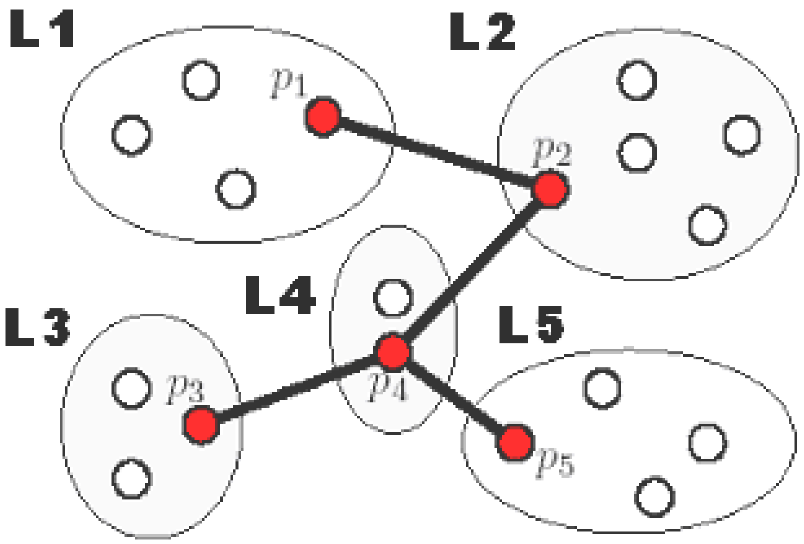

Figure 1. A Minimum Spanning Tree of Five Networks

1.1 Overview of Thesis

The further organization of this thesis is as follows: There are seven remaining chapters:

Chapter 2. Network Design

This chapter explains that what is network design, what are the different types of network with its mathematical formulation and what are the application where network design is the backbone of the system. It also describes that what are the different types of problem related with network design with its limitation.

Chapter 3. Methodologies

In this chapter various techniques to solve network design optimization problems are presented with their limitation and literature survey with relevance of the research work has been highlighted. This chapter is the motivation of this research work.

Chapter 4. Genetic Algorithm

This chapter explains the basic of genetic algorithm.

Chapter 5. Genetic Algorithm approach to Network Design

This chapter is the backbone of this research where redefined Genetic Algorithm approach is explained to design network. This chapter contains all the methods and algorithms developed for this study. This chapter starts with the explanation of genetic algorithm and it describes that how genetic algorithm can helps to solve this network design problem. It explains the improved genetic algorithm approach with the developed fitness function and various types of genetic operators developed in this thesis. All the functions developed in this chapter are published. Following papers are published for this chapter...

1. Anand Kumar, Dr. N.N. Jani , "An algorithm to detect cycle in an undirected graph" International Journal of Computational Intelligence Research ISSN 0973-1873, (Vol 6, No 2 (2010), pp 305-310)

2. Anand Kumar, Dr. N.N. Jani, "A Novel Genetic Algorithm Approach for Network Design with Robust Fitness Function" Proceeding of International Conference on Mathematics and Computer Science, 5-6 Feb 2010, Loyola College, Chennai, ISBN: 978-81-908234-2-5.

3. Anand Kumar and N.N. Jani, " Using A Genetic Algorithm approach to Design Backbone Core Communication Network" Proceeding of International Conference on Emerging Trends in Computing , 8-10 Jan 2009 , Kamaraj College of Engineering and technology, Virudhunagar, Tamilnadu.

Chapter 6. Experimental Design and Result

This chapter explains that how this experiment is carried out. The tools and data sets with the result have been presented here. This chapter is the proof of this research work. Various tables, graphs, diagrams and developed programs have been included in this chapter. Following papers are published for the experimental result.

4. Anand Kumar, Dr. N.N. Jani, “Network Design Problem Using Genetic Algorithm- An Empirical Study On Selection Operator” International Journal of Computer Science and Applications (IJCSA) ISSN: 0974-1003 (April/May 2010 Vol 3, No 2, pp 48-52)

5. Anand Kumar and Dr. N.N. Jani, “Genetic Algorithm for Network Design Problem- An Empirical Study of Crossover operator with Generation and Population Variation” International Journal of Information Technology and Knowledge Management, ISSN: 0973-4414, Vol III, Issue-I, June 2010.

Chapter 7. Genetic Algorithm approach to Solve Shortest Path and Traveling Salesman Problem

This chapter explains a new approach based on genetic algorithm to solve these NP-hard network design problems. To solve these problems various functions and operators are developed. These works are published also.

6. Anand Kumar, “A Nature based Evolutionary approach to solve Network Communication NP-Hard Traveling Salesman problem” International Journal of Computational Intelligence Research and Applications (IJCIRA) ISSN: 0973-6794, Volume 3 Number 1, January-June 2009, Page. No. 27-32.

7. Anand Kumar, Dr. N.N. Jani, “An Evolutionary Approach for Shortest Path Problem - Courier Delivery System” International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 6, Number 2 (2010), pp. 261–273.

8. Anand Kumar, Dr. N.N. Jani, “Genetic Algorithm Approach to Solve Hamiltonian Circuit Problem With Robust Fitness And Repair Function” Proceeding of IEEE International Advance Computing Conference 2009.Thapar University, Patiyala. ISBN NO: 978-981-08-2465-5

Chapter 8. Conclusions and Future Scope

This chapter is devoted for the discussion with conclusion and the extension of this research work.

CHAPTER 2

Network Design

Network design is one of the most important and most frequently encountered classes of optimization problems. In this chapter, it is explained that "what network design is?" The meaning of network design is a backbone network which is a connected network with all locations in the form of a tree. In this research work, a generalized network design problems (NDPs) is focused in the form of a large scale backbone network which belong to the family of NP-hard [6] combinatorial optimization problems[42,43]. The purpose of the backbone is to connect regional distribution networks and, in some instances, to provide connectivity to other peer networks. To connect the different locations, a minimum spanning tree (MST) [5] is required which is responsible for connecting all the locations with minimum distance without the formation of a circle. The MST problem itself is easy to solve by polynomial-time algorithms like those of Prim or Kruskal, but adding additional constraints often make the corresponding optimization problem a hard one. Additional constraint is important for the effective network design and one of the most important constraints is degree of each node. In the degree-constraint MST problem a bound on the degree, i.e., the number of incident edges, is imposed on every node in the tree to model that in a telecommunication network the used hardware (e.g., a router or switch) can only handle a limited amount of links However, there are also other problems that can be expressed as network design problems, such as various transportation and routing problems. For example, in the famous traveling salesman problem (TSP), one has to find a round trip (Hamiltonian cycle) through a set of cities (nodes) of minimal length, similarly Shortest path problems arise in a wide variety of practical problem such as transportation planning , salesperson routing, message routing in communication systems.

2.1 Graph Models

In the context of this chapter, the word network means a physical problem that can be modeled as a mathematical graph composed of nodes and links. The system is represented by a mathematical graph composed of nodes representing the computers and edges representing the communications links. The terms used to describe graphs are not unique; oftentimes, notations used in the mathematical theory of graphs and those common in the application fields are interchangeable. Thus a mathematics textbook may talk of vertices and arcs; an electrical engineering book, of nodes and branches; and a communications book, of sites and interconnections or links. In general, these terms are synonymous and used interchangeably. In some situations, communication can go only in one direction between a node pair; the link is represented by a directed edge (an arrowhead is added to the edge), and one or more directed edges in a graph result in a directed graph (digraph). If communication can occur in both directions between two nodes, the edge is non directed, and a graph without any directed nodes is an undirected graph. In this thesis undirected graph is considered. Following Figure-2 is the graph representation of network in Figure-1. It represents a complete graph where direct path exist to go from one node to any other node.

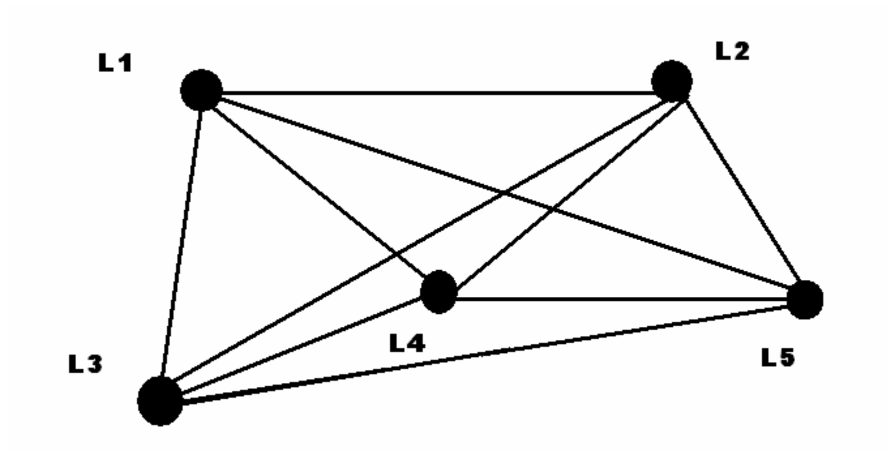

Figure 2:1 A Complete Graph of Five Nodes

2.2 Basic Network Models

Network design is used extensively in practice in an ever expanding spectrum of applications. Network optimization models such as shortest path, assignment, max-flow, 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. Following are the core models of network design.

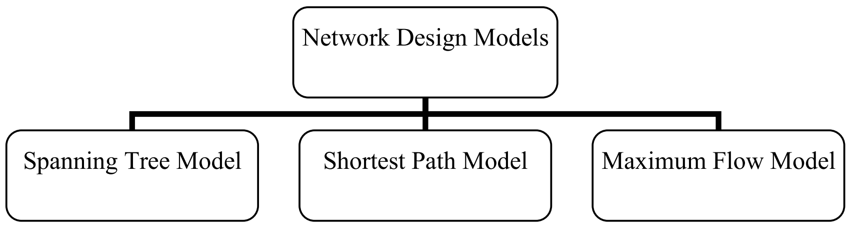

Figure 2: 2 Core Models of Network Design

These network models are used most extensively in applications and differentiated by their structural characteristics. This research work is based on spanning tree model. The descriptions of these models are as follows.

2.2.1 Spanning Tree Model

Spanning tree models play a central role within the field of network design. It generally arises in one of two ways, directly or indirectly. In some direct applications, It connects a set of points using the least cost or least length collection of arcs. Frequently, the points represent physical entities such as components of a computer chip, or users of a system who need to be connected to each other or to a central service such as a central processor in a computer system [2]. In indirect applications, either (1) wish to connect some set of points using a measure of performance that on the surface bears little resemblance to the minimum spanning tree objective (sum of arc costs), or (2) the problem itself bears little resemblance to an "optimal tree" problem – in these instances, it is often needed to be creative in modeling the problem so that it becomes a minimum spanning tree problem.

Applications

1. Backbone Network Design
2. Designing of digital circuit,
3. Designing of gas pipeline,
4. Layout planning roads and railway track and many more

2.2.2 Shortest Path Model

The shortest path model is the heart of network design optimization. It has several important reasons: (1) it arise frequently in practice since in a wide variety of applications, materials are sent (*e.g.*, a computer data packet, a telephone call, a vehicle) between two specified points in a network as quickly, as cheaply, or as reliably as possible; (2) as the simplest network models, they capture many of the most salient core ingredients of network design problems and so they provide both a benchmark and a point of departure for studying more complex network models; and (3) they arise frequently as sub problems when solving many combinatorial and network optimization problems [2]. Even though shortest path problems are relatively easy to solve, the design and analysis of most efficient algorithms for solving them requires considerable ingenuity. Consequently, the study of shortest path problems is a natural starting point for introducing many key ideas from network design problems, including the use of clever data structures and ideas such as data scaling to improve the worst case algorithmic performance.

Applications

1. Transportation planning: How to determine the route road that has prohibitive weight restriction so that the driver can reach the destination within the shortest possible time.
2. Salesperson routing: Suppose that a sales person want to go to Delhi from Patna and stop over in several cities to get some commission. How can he/she determine the route? (Traveling Salesman Problem)
3. Investment planning: How to determine the invest strategy to get an optimal investment plan.
4. Message routing in communication systems: The routing algorithm computes the shortest (least cost) path between the router and all the networks of the internet work It is one of the most important issues that has a significant impact on the network's performance.

2.2.3 Maximum Flow Model

The maximum flow model and the shortest path model are complementary. They are similar because they are both pervasive in practice and because they both arise as sub problems in algorithms for the minimum cost flow problem. The two problems differ because they capture different aspects. Shortest path problems model arc costs but not arc capacities; maximum flow problems model capacities but not costs. Taken together, the shortest path problem and the maximum flow problem combine all the basic ingredients of network design optimization. As such, they have become the cores of network optimization [2].

Applications

The maximum flow problems arise in a wide variety of situations and in several forms. For example, sometimes the maximum flow problem occurs as a sub problem in the solution of more difficult network problems, such as the minimum cost flow problems or the generalized flow problem. The problem also arises directly in problems as far reaching as machine scheduling, the assignment of computer modules to computer processors, the rounding of census data to retain the confidentiality of individual households, and tanker scheduling.

2.3 Network Design Problems

The basic problem in network design is connectivity of the node which is very important from the reliability point of view. Network Design problems can be classified on the basis of basic network models. The network design problems can be broadly classified as Spanning Tree problem and Shortest Path Problem.

2.3.1 Spanning Tree Problem

Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree and connects all the nodes together. A single graph can have many different spanning trees. It can also be assigned a weight to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A minimum spanning tree (MST) is a spanning tree with weight less than or equal to the weight of every other spanning tree

2.3.1.1 Minimum Spanning Tree problem.

A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

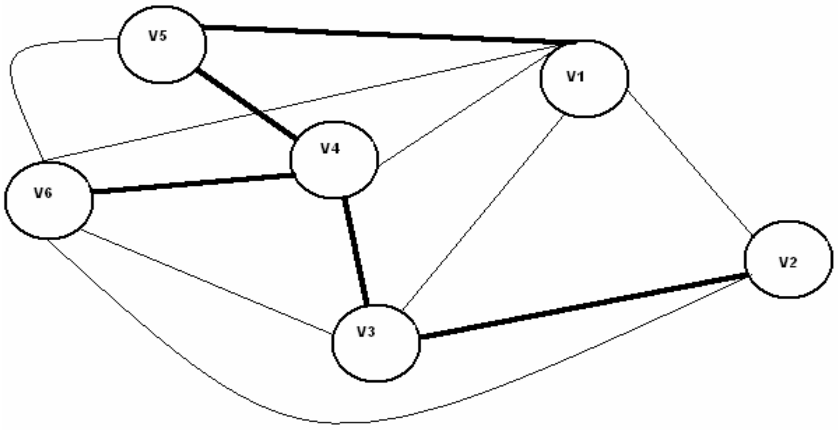

Figure 2: 3 Minimum Spanning Trees

Mathematical Formulation of Minimum Spanning Tree

The MST model attempts to find a minimum cost tree that connects all the nodes of the network. The links or edges have associated costs that could be based on their distance, capacity, and quality of line.

For an undirected connected graph $G = (V, E)$ a subgraph $T = (V, E')$ of G is a spanning tree of G if T is a tree.

V	: set of vertices
$\|V\|$	: number of vertex
E	: set of possible edges between pair of vertices
$(u, v) \in E$	: Each edge between pair of vertices u, v belongs to set of possible edges
$w(u, v)$	: Weight of each edge.

$$w(T) = \sum_{(u,v) \in T} w(u, v) \tag{2.1}$$

A spanning tree always consists of $|V| - 1$ edges and a complete graph G has $|V|^{|v|-2}$ spanning trees [8]. For example, for the four-node network of Figure.2.4 there are $4^{(4-2)} = 16$ spanning trees with $(4 - 1) = 3$ arcs

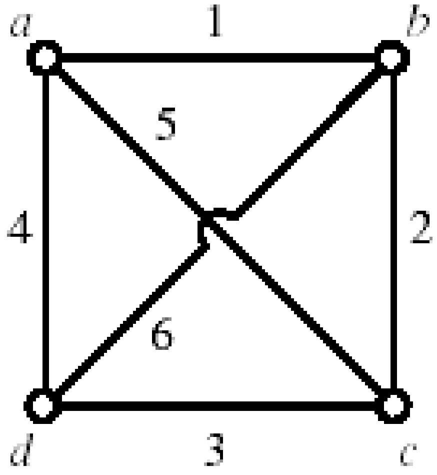

Figure. 2.4 A four-node graph representing a computer or communication network.

There will be possible $4^{(4-2)} = 16$ spanning trees with $(4 - 1) = 3$ arcs. In Figure 2.5 it is shown.

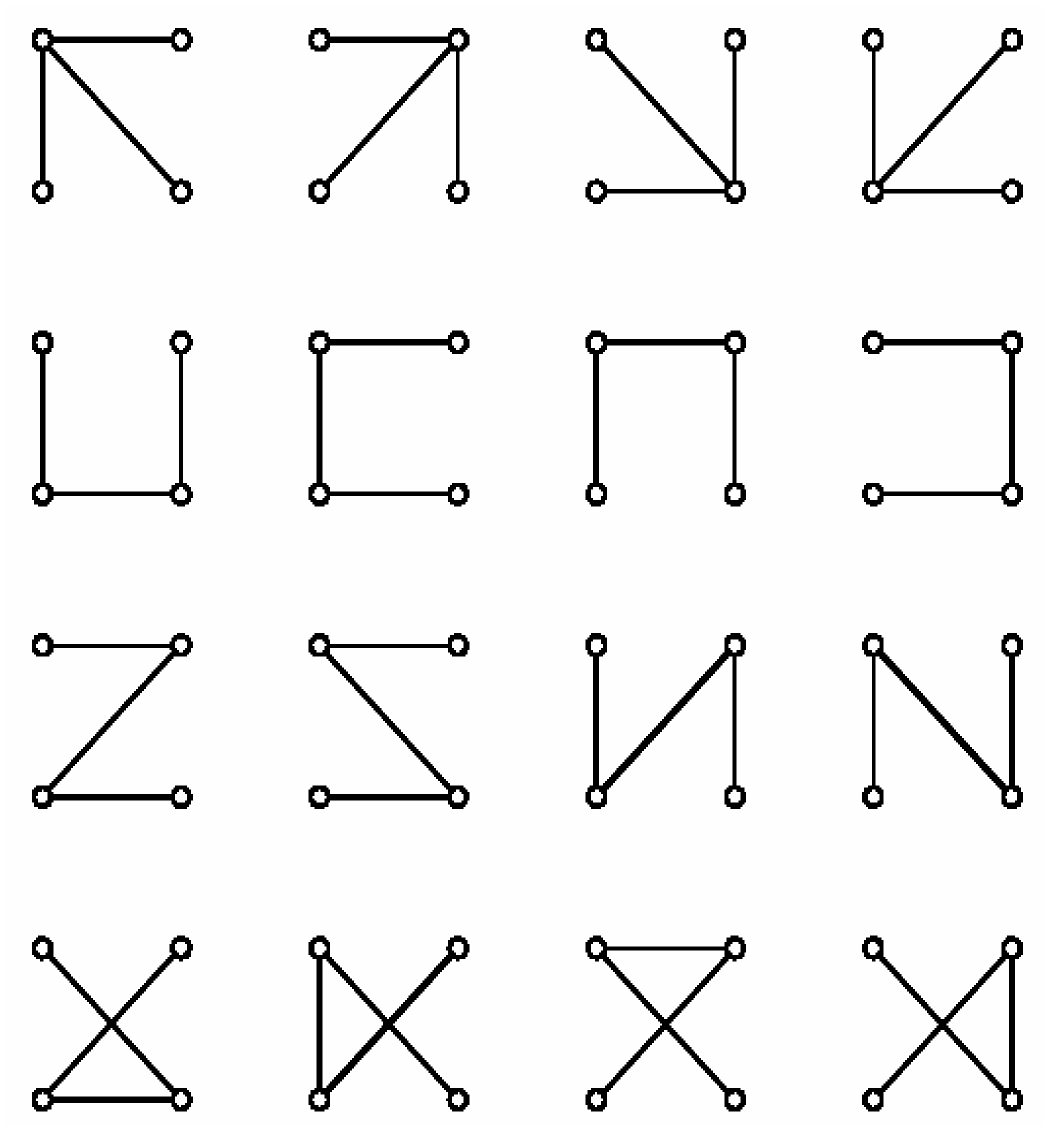

Figure 2.5 The 16 spanning trees for the network of Figure 2.4

2.3.1.2 Degree Constrained Minimum Spanning Tree

Degree-constrained minimum spanning tree (DCMST) is a special case of MST. The MST algorithm may occasionally generate a minimal spanning tree where all the links connect to one or two nodes. This solution, although optimal, may be highly vulnerable to failure due to over reliance on a few nodes. Furthermore, the technology to connect many links to a node may not be available or may be too expensive. Hence, it may be necessary to limit the number of links connecting to a node. Alternatively, from reliability perspective it is desirable to have more than one link connect to a node so that alternative routes can be selected in the case of a node or link failure [9], [10].. DCMSTP was specifically developed as a special case of MST with additional constraints to improve the reliability of the network and rerouting of traffic in the case of node failures. This is the extended version of minimum spanning tree problem where an extra constraint is added with each vertex. When this extra constraint is added this problem becomes NP-hard [7]. This constraint is usually motivated by the need to impose a limit on the number of ports in each node. In a shortest spanning tree resulting from the preceding construction, a vertex v_i can end up with any degree; that is

$$1 \leq d(v_i) \leq n-1$$

Where n is the total no of vertex and d is the degree denoted by $d(v_i)$ of a node $i(i=1,........n)$. The degree is the number of incident edges, and the degree of a graph is the maximum degree of its nodes. The degree constrained MST problem is to determine a spanning tree of the minimum total edge cost and degree no more than a given value k .

Then each node of a network must not be connected with k other nodes

$$d(v_i) \leq 3$$

So far, no efficient method of finding an arbitrarily degree constrained shortest spanning tree has been found. In this research work an attempt has been made to find the solution with more constraint which is required by the modern network.

2.3.2 Shortest Path Problem

Given a pair of nodes, the shortest path problem is to find a forward path that connects these nodes and has minimum cost. An analogy here is made between arcs and their costs, and roads in a transportation network and their lengths, respectively. Within this transportation context, the problem becomes one of finding the shortest route between two geographical points. Based on this analogy, the problem is referred to as the shortest path problem, and the arc costs and path costs are commonly referred to as the arc lengths and path lengths, respectively.

The shortest path problem is a classical and important combinatorial problem that arises in many contexts. This path is said to be shortest if it has minimum length over all forward paths with the same origin and destination nodes. The length of a shortest path is also called the shortest distance. The shortest path problem deals with finding shortest distances between selected pairs of nodes. The range of applications of the shortest path problem is very broad. The shortest path problem can be posed in a number of ways; for example, finding a shortest path from a single origin to a single destination, or finding a shortest path from each of several origins to each of several destinations.

Mathematical Formulation of Shortest Path Problem

Let $G=(N,A)$ $G=(N,A)$ be a directed network, which consists of a finite set of nodes $N = \{1,2,........n\}$ and a set of directed arcs $A=\{(i,j),(k,l),.....(s,t)\}$ connecting m pairs of nodes in N. Arc (i,j) is said to be incident with nodes i and j, and is directed from node i to node j. Each arc (i,j) has been assigned to a nonnegative value c_{ij}, the cost of (i,j). The Shortest Path Problem is to find the minimum cost z from a specified source node 1 to another specified sink node n, which can be formulated as follows

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{2.2}$$

$$s.t \sum_{j=1}^{n} x_{ij} - \sum_{k=1}^{n} x_{ki} = \begin{cases} 1, if\,(i=1) \\ 0, if\,(i=2,3....,n-1) \\ -1, if\,(i=n) \end{cases} \tag{2.3}$$

$$x_{ij} = 0 or 1 \forall i, j \tag{2.4}$$

Where x_{ij} : the link on an arc $(i, j) \in A$

2.3.3 Traveling Salesman Problem(TSP)

Historically, the TSP problem deals with finding the shortest tour in n-city situations where each city is visited exactly once. It is a prominent illustration of a class of problems in computational complexity theory which are classified as NP-hard. The problem is given a number of cities and the costs of traveling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the home city. In the Traveling Salesman Problem, the goal is to find the shortest distance between N different cities. The path that the salesman takes is called a tour. Testing every possibility for an N city tour would be N! math additions. A 30 city tour would have to measure the total distance of be 2.65 X 10^{32} different tours which will take unexpected time. Adding one more city would cause the time to increase by a factor of 31. Obviously, this is an impossible solution. Traveling Salesman Problem can be represented is in the form of Hamiltonian Circuit which has the smallest sum of the distances.

Mathematical Formulation of Traveling Salesman Problem

The problem, in essence, is an assignment model that excludes subtours. Specifically, in an *n* city situation, it is defined as

x_{ij} = 1, if city *j* is reached from city *i* otherwise 0.

Given that d_{ij} is the distance from city *i* to city *j* , the TSP model can be defined as

$$\min z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}, \qquad (2.5)$$

$$d_{ij} = \infty \ for\ all\ i = j$$

Subject to

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, 2.....n \qquad (2.6)$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, 2......, n \qquad (2.7)$$

$$x_{ij} = (0, 1) \qquad (2.8)$$

In following figure 2.6, there are nine cities, which have to be visited by a traveling salesman such that each city is to be visited exactly once. In figure 2.7, a Hamiltonian tour is shown which a traveling salesman tour is.

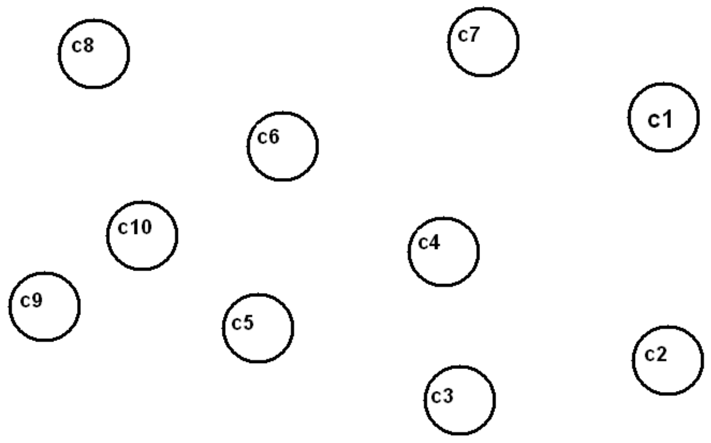

Figure 2.6 A number of cities visited by Traveling Salesman

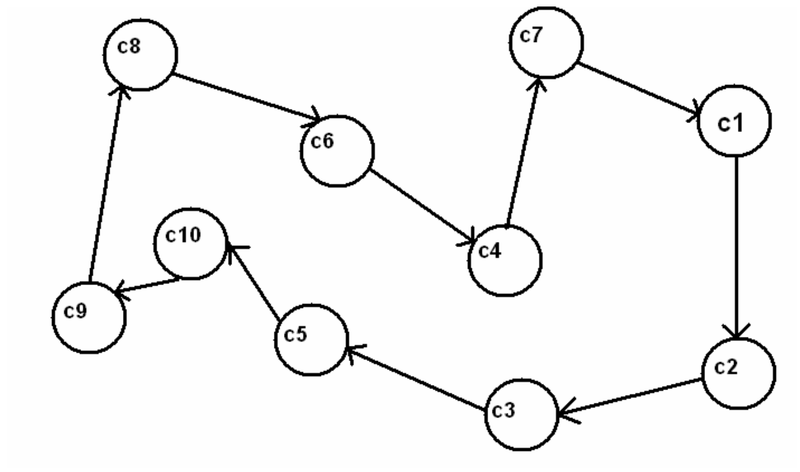

Figure 2.7 Path visited by Traveling Salesman

2.3.4 Considered Problems

In this research study, network design problem is mainly considered as backbone network design belongs to the category of network topology design which is in the form of degree constraint minimum spanning tree with other constraints required by the network. The problem is an NP-hard problem and there is no optimal solution still developed. Further other network design problem Traveling salesman and Shortest Path Problem is also considered with the same approach genetic algorithm. Since the first two problems have still not solved, a genetic algorithm approach is proposed in this study. For Shortest Path Problem, extra constraints have been applied which is the demand of current industry and then Genetic Algorithm approach is applied to solve the problem. Shortest Path Problem and Traveling Salesman Problem are discussed in section 2.6.2 and 2.6.3. There is no change in these two problems while backbone network design problem is redefined with other constraints.

2.3.5 Backbone Network Design Problem

A simple model for a backbone network is an undirected graph $G = (V, E)$ with node set V and edge set E. In this model, the nodes represent the connection points where LANs are hooked up to the backbone network via gateways. In addition to being connection points, the nodes are the processing units that carry out traffic management on the network by forwarding data packets to the nodes along their destinations (i.e., known as routers). The edges represent the high capacity multiplexed lines on which data packets are transmitted bi-directionally between the node pairs. In designing of backbone network connectivity is an important factor. For a reliable network, connectivity is an important constraint. If a node has degree one, there is more chance of isolation of this node in case of path failure. Further according to the importance of the node, degree can be extended which is in the form of more connectivity means more reliability. So degree constraint is an important constraint for backbone network design. In this backbone network design problem, degree of the node is kept between lower bound and upper bound. Other important constraint is existence of path between pair of nodes. If direct path exist from each node to each other node then it is the case of complete graph, but it is always not possible that path will be available from each node to each other node. So other important constraint is path constraint. Since real world network systems are becoming larger and more complex, the need of more sophisticated models arises. For example, with increasing number of local networks, it makes sense to connect them to a new global network. This involves choosing one computer from each local network to be used as an entrance gate for the global backbone. Obviously, the old model of MSTP is not sufficient anymore. This motivates the introduction of Backbone Network Design Problems. Since Degree Constrained Minimum Spanning Tree Problem itself is a NP-hard problem, including another constraint like path constraint makes it one of the hardest problem in category of NP-hard.

Mathematical Formulation of Backbone Network Design Problem

The mathematical formulation of the DCMST problem is presented below. The following notation is used in the research study.

Indices

i, j	: Index of nodes $i, j = 1, 2, ..., n$
V	: Set of nodes in the spanning tree.

Parameters

C_{ij}	: Cost to link nodes i to j
Ud_i	: Upper degree constraint on node i
Ld_i	: Lower degree constraint on node i
$\|N\|$	: Number of nodes in a subset N of nodes in V
$\|V\|$	: Number of the nodes in V

Decision Variables

X_{ij} : Equals one if the link between nodes i to j exists;

: zero, otherwise.

Minimize

$$\sum_{\substack{i,j<V \\ i<j}} C_{ij} X_{ij} \qquad (2.9)$$

Subject to

$$\sum_{\substack{j \in V \\ i \neq j}} X_{ij} \leq Ud_i \quad \forall i \in V \qquad (2.10)$$

$$\sum_{\substack{j \in V \\ i \neq j}} X_{ij} \geq Ld_i \quad \forall i \in V \qquad (2.11)$$

$$\sum_{\substack{i,j \in N \\ i<j}} X_{ij} \leq |N|-1 \quad \forall \mathrm{N} \subset V \qquad (2.12)$$

$$\sum_{\substack{i,j \in V \\ i<j}} X_{ij} = |V|-1 \qquad (2.13)$$

$$X_{ij} = 0 \ \ or \ \ 1 \ \ i,j \in V. \qquad (2.14)$$

The objective function (2.9) seeks to minimize the total connecting cost between nodes. The total cost could be distance cost, material cost, or customers' requirement cost. Constraint (2.10) and (2.11) specify the lower and upper bound constraints on the number of edges connecting to a node. Constraint (2.12) is an anti cycle constraint and constraint (2.13) indicates that the number of edges in a spanning tree is equal to the number of nodes minus one. Constraint (2.14) expresses the binary requirements of the decision variables. Constraint (2.12) increases exponentially with network node size, thereby making it impractical to solve large size problems.

CHAPTER 3

Methodologies

There are various techniques to solve optimization problems like these presented above. Roughly they can be classified into two main categories: Exact and, Heuristic. Further heuristic is classified as Metaheuristic Algorithms**.** Exact algorithms are guaranteed to always identify a provable optimal solution (if some exists), but often the runtime behavior does not scale satisfyingly with instance size. As a consequence, exact approaches often are only applied to small or moderately-sized instances while larger instances are solved by heuristics. Heuristics sacrifice the guarantee to reach the optimum for the sake of finding good solutions of acceptable quality within reasonable time. Somewhere in-between is the approximation algorithms: Mainly classified as heuristics they are able to give at least some provable bounds on the quality of the computed solution in relation to the optimum.

Examples for successful exact algorithms are Dynamic Programming (DP) [11], Branch&Bound, and especially the large family of (integer) linear programming (LP) based approaches, including in particular Linear Programming based Branch&Bound, Branch&Cut, Branch&Price, and Branch&Cut&Price [12, 13]. Concerning heuristics there exist constructive methods like Greedy Heuristics and techniques such as Local Search. Usually, these approaches are highly problem specific. More general solution strategies are the so-called metaheuristic [14, 15], which control and manage subordinate, often problem specific heuristics, using various strategies to escape local optima simple heuristics are frequently trapped in. Usually, metaheuristics are more reliable and robust in finding good solutions, making them an interesting choice to solve difficult optimization problems. Prominent representatives for Metaheuristics are Iterated Local Search [16] or Tabu Search (TS) [17]. Proven to sometimes be very effective are also algorithms inspired by nature and biology population-based approaches which are especially well suited for parallel processing like Evolutionary Algorithms (EA) [18, 19]. In this study, Genetic Algorithm approach is applied to solve the problem.

In this chapter first of all a brief description of Exact algorithms have been given then Heuristics have been described and at last Metaheuristic have been described.

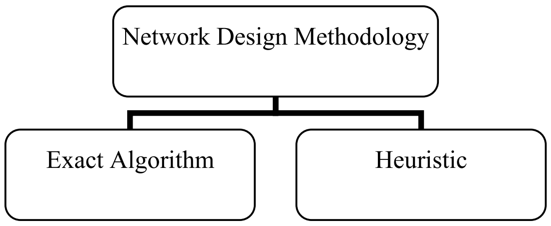

Figure 3.1 Core Methodology of Network design

3.1 Exact Algorithms

Many Combinatorial Optimization Problems (Cops) can be modeled as a (integer) linear program. While Linear Programs (LPs) can be solved efficiently in practice via the well known simplex algorithm and, from a theoretical point, even in polynomial time. Whenever possible, the first attempt should be to solve a given problem to proven optimality. Following are the main exact algorithms which are used to prove the solution optimally.

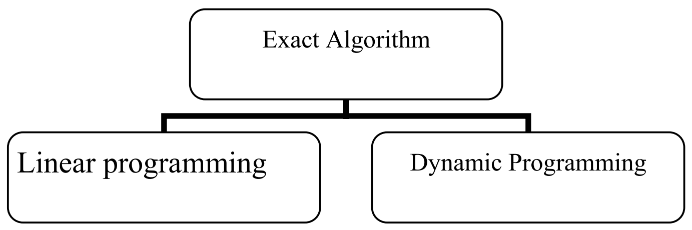

Figure 3.2 Exact Algorithms for Network design

3.1.1 Linear Programming

Linear programming (LP) is a mathematical method for determining a way to achieve the best outcome (such as maximum profit or lowest cost) in a given mathematical model for some list of requirements represented as linear equations. More formally, linear programming is a technique for the optimization of a linear objective function, subject to linear equality and linear inequality constraints. Given a polyhedron and a real-valued affine function defined on this polyhedron, a linear programming method will find a point on the polyhedron where this function has the smallest (or largest) value if such point exists, by searching through the polyhedron vertices. Linear programming is a considerable field of optimization for several reasons. Many practical problems in operations research can be expressed as linear programming problems. Certain special cases of linear programming, such as network flow problems and multicommodity flow problems are considered important enough to have generated much research on specialized algorithms for their solution. A number of algorithms for other types of optimization problems work by solving LP problems as sub-problems. Historically, ideas from linear programming have inspired many of the central concepts of optimization theory, such as duality, decomposition, and the importance of convexity and its generalizations. Likewise, linear programming is heavily used in microeconomics and company management, such as planning, production, transportation, technology and other issues. Although the modern management issues are ever-changing, most companies would like to maximize profits or minimize costs with limited resources. Therefore, many issues can boil down to linear programming problems.

Integer Linear Programming

If the unknown variables are all required to be integers, then the problem is called an integer programming (IP) or integer linear programming (ILP) problem. In contrast to linear programming, which can be solved efficiently in the worst case, integer programming problems are in many practical situations (those with bounded variables) NP-hard. 0-1 integer programming or binary integer programming (BIP) is the special case of integer programming where variables are required to be 0 or 1 (rather than arbitrary integers). This problem is also classified as NP-hard, and in fact the decision version was one of Karp's 21 NP-complete problems. If only some of the unknown variables are required to be integers, then the problem is called a mixed integer programming (MIP) problem. These are generally also NP-hard. There are however some important subclasses of IP and MIP problems that are efficiently solvable, most notably problems where the constraint matrix is totally unimodular and the right-hand sides of the constraints are integers. Advanced algorithms for solving integer linear programs include:

- cutting-plane method
- branch and bound
- branch and cut
- branch and price

- if the problem has some extra structure, it may be possible to apply delayed column generation.

3.1.1.1 Branch and bound

Branch and bound (BB) is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. It consists of a systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded *en masse*, by using upper and lower estimated bounds of the quantity being optimized.The method was first proposed by A. H. Land and A. G. Doig in 1960 for linear programming.Branch-and-bound may also be a base of various heuristics. For example, one may wish to stop branching when the gap between the upper and lower bounds becomes smaller than a certain threshold. This is used when the solution is "good enough for practical purposes" and can greatly reduce the computations required. This type of solution is particularly applicable when the cost function used is *noisy* or is the result of statistical estimates and so is not known precisely but rather only known to lie within a range of values with a specific probability. An example of its application here is in biology when performing cladistic analysis to evaluate evolutionary relationships between organisms, where the data sets are often impractically large without heuristics. For this reason, branch-and-bound techniques are often used in game tree search algorithms, most notably through the use of alpha-beta pruning.

3.1.2 Dynamic Programming

In mathematics and computer science, dynamic programming is a method of solving complex problems by breaking them down into simpler steps. It is applicable to problems that exhibit the properties of overlapping subproblems which are only slightly smaller and optimal substructure. When applicable, the method takes much less time than naive methods.

Top-down dynamic programming simply means storing the results of certain calculations, which are then re-used later because the same calculation is a sub-problem in a larger calculation. Bottom-up dynamic programming involves formulating a complex calculation as a recursive series of simpler calculations.

There are two key attributes that a problem must have in order for dynamic programming to be applicable: optimal substructure and overlapping subproblems which are only slightly smaller. When the overlapping problems are, say, half the size of the original problem the strategy is called "divide and conquer" rather than "dynamic programming". This is why merge sort, and quick sort, and finding all matches of a regular expression are not classified as dynamic programming problems.

Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Consequently, the first step towards devising a dynamic programming solution is to check whether the problem exhibits such optimal substructure. Such optimal substructures are usually described by means of recursion. For example, given a graph $G=(V,E)$, the shortest path p from a vertex u to a vertex v exhibits optimal substructure: take any intermediate vertex w on this shortest path p. If p is truly the shortest path, then the path p_1 from u to w and p_2 from w to v are indeed the shortest paths between the corresponding vertices. Hence, one can easily formulate the solution for finding shortest paths in a recursive manner, which is what the Bellman-Ford algorithm does.

3.2 Heuristics

When confronted with *NP*-hard combinatorial optimization problems, exact approaches often are only applicable to relatively small problem instances due to run time and sometimes also memory restrictions. Heuristics and especially *metaheuristics can* be seen as alternatives when large instances have to be solved in reasonable time, whereas these approaches are not able to guarantee to reach the optimum. Nevertheless, for real-world optimization problems they are often the only opportunity to get high-quality solutions with limited resources. The term metaheuristic has been introduced by Glover [20] and denotes a problem independent high-level solution strategy managing and controlling subordinate heuristics, which themselves are highly problem specific in general. This section starts with an introduction to some basic heuristics, and afterwards, Genetic Algorithm metaheuristics is explained

.

3.2.1 Kruskal's Algorithm

Examines edges in nondecreasing order of their lengths and include them in MST if the added edge does not form a cycle with the edges already chosen. The algorithm is attractive if the edges are already sorted in increasing order of their lengths. The procedure of Kruskal's algorithm is shown below in procedure and figure 3.3(b) is Kruskal based MST of graph shown in figure 3.3(a).

Procedure: Kruskal's Algorithm

```
Input: Graph $G = (V, E), weight\ w_{ij},\ \forall (i, j) \in V$
Output: Spanning Tree $T$
Begin
    $T \leftarrow \phi;$
    $A \leftarrow E;$          // $A$: Eligible edges

    while $|T| < |V| - 1$ do

        choose an edge $(u, v) \leftarrow$ argmin $\{w_{ij} \mid (i, j) \in A\};$

        $A \leftarrow A \setminus \{(u, v)\};$

        if $u$ and $v$ are yet not connected in $T$ then
            $T \leftarrow T \cup \{(u, v)\};$

        Output spanning tree $T$
End
```

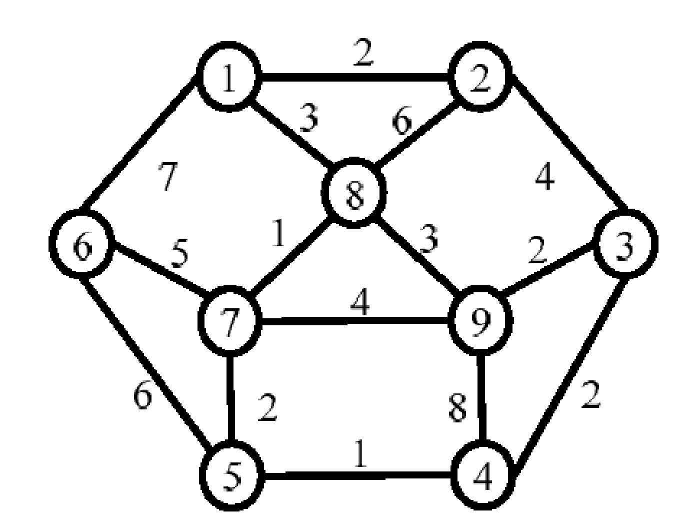

Figure 3.3 (a) A Graph $G = (V, E)$ and weight $w_{ij}, \forall (i, j) \in V$

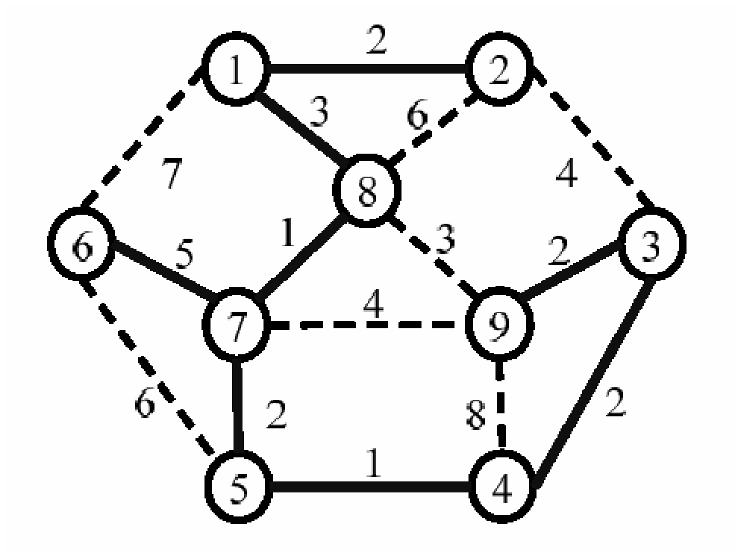

Figure 3.3 (b) MST

3.2.2 Prim's Algorithm

According to Prim, the spanning tree starts from an arbitrary root vertex and grows until the tree spans all the vertices in V. Prim's algorithm has the property that the edges in the set always form a single tree. The procedure of Prim's algorithm is shown in procedure and figure 3.4 is Prim based MST of graph shown in figure 3.3(a).

Procedure: Prim's Algorithm

```
Input: Graph $G = (V, E), weight\ w_{ij},\ \forall(i, j) \in V$
Output: Spanning Tree $T$
Begin
    $T \leftarrow \phi;$
    Choose a random starting node $s \in V$;
    $C \leftarrow C \cup \{s\}$; // $C$: set of connected nodes
    $A \leftarrow A \cup \{(s, v), \forall v \in V\}$; // $A$: eligible edges
```

```
        **While** $C \neq V$ **do**
            Choose an edge $(u,v) \leftarrow \textbf{argmin}\{w_{ij} \mid (i,j) \in A\}$;
            $A \leftarrow A \setminus \{(u,v)\}$;
            **if** $v \neq C$ **then**
                $T \leftarrow T \cup \{(u,v)\}$;
                $C \leftarrow C \cup \{v\}$;
                $A \leftarrow A \cup \{(v,w) \mid (v,w) \wedge w \notin C\}$;
        **Out put spanning tree** $T$
    **End**
```

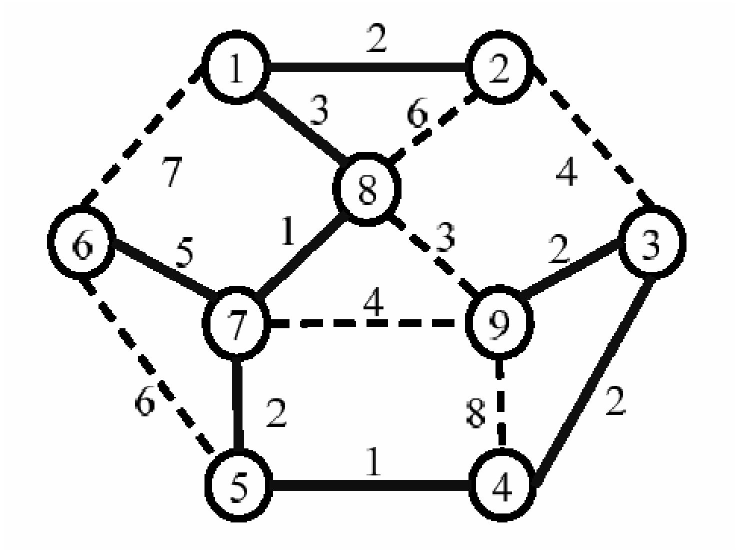

Figure 3.4 Prim based MST

3.2.3 Breadth First Search Algorithm(BFS)

Breadth-first traversal of a graph is a level-by-level traversal of an ordered tree. Start the traversal from an arbitrary vertex, visit all of its adjacent vertices; and then, visit all unvisited adjacent vertices of those visited vertices in last level. Continue this process, until all vertices have been visited. The procedure of BFS algorithm is shown in procedure and figure 3.5(b) is BFS based MST of graph shown in figure 3.5(a).

Procedure: Breadth First Search Algorithm

```
  BFS(G,s)
  for each vertex u in V
        visited[u] = false
     Report(s)
     visited[s] = true

    initialize an empty Q
    Enqueue(Q,s)
While Q is not empty
      do u = Dequeue(Q)
           for each v in Adj[u]
                 do if visited[v] = false
                      then Report(v)
                              visited[v] = true
                              Enqueue(Q,v)
```

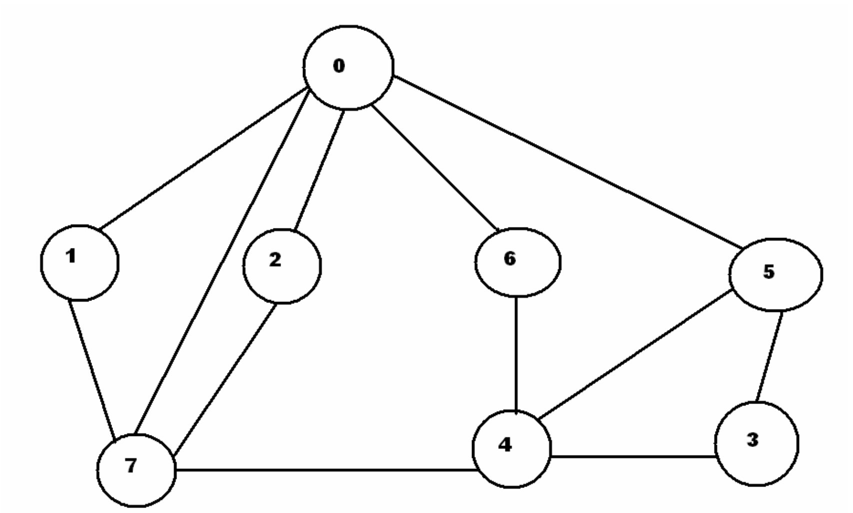

Figure 3.5(a) an Undirected Graph

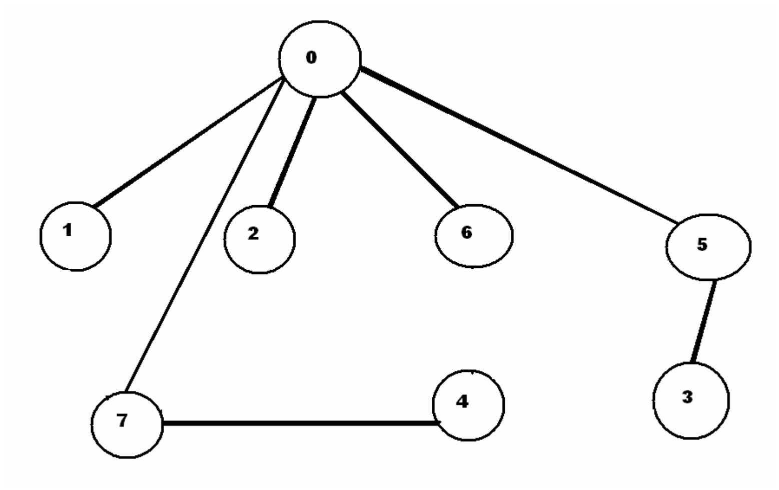

Figure 3.5(b) BFS based Spanning Tree

3.2.4 Depth First Search Algorithm(DFS)

It is based on depth of the graph. It starts from the given vertex, visit one of its adjacent vertices and leave others; then visit one of the adjacent vertices of the previous vertex; continue the process, visit the graph as deep as possible until: A visited vertex is reached or an end vertex is reached.

Procedure: Depth First Search Algorithm

```
 DepthFirst(Graph G)
     Vertex v;
     for (all v in G)
      visited[v] = FALSE;
     for (all v in G)
         if (!visited[v])
       Traverse(v);

Traverse(Vertex v)
      visited[v] = TRUE;
      Visit(v);
    for (all w adjacent to v)
```

```
                if (!visited[w])
                Traverse(w);
```

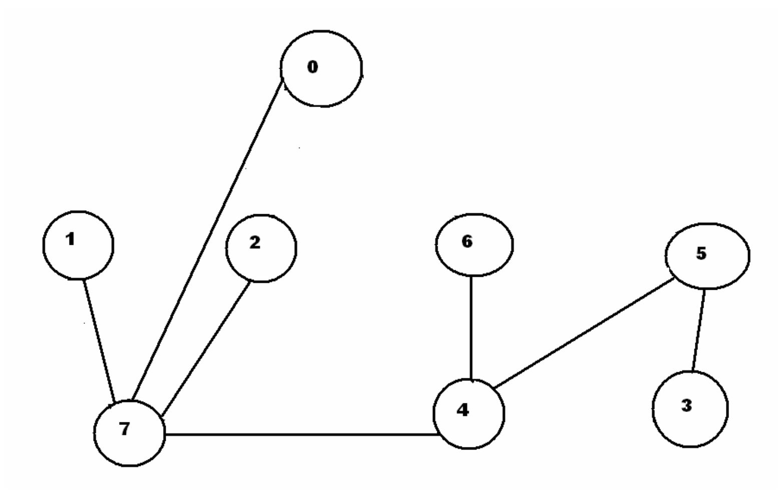

Figure 3.6 DFS based Spanning Tree

3.2.5 Dijkstra algorithm for Shortest Path

There are many algorithms for shortest path problem, but Dijkstra is the prominent among all of them. Here Dijkstra is explained only. Other shortest path algorithms are:

- Bellman-Ford Algorithm
- Floyd-Warshall Algorithm
- Incremental-shortest-path algorithms

Dijkstra algorithm provides a shortest route for weighted directed graph $G = (V,E)$ for the case in which all the edge weights are non negative. Therefore $w(u, v) >0$ for each edge $(u, v)\in E$. This algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined. That is, for all vertices

$v \in S$, $d[v] = \delta(s, v)$. The algorithm repeatedly selects the vertex $u \in V\text{-}S$ with the minimum shortest path , insert *u* into *S* and relax all edges leaving *u*. A priority queue *Q* is maintained that contains all the vertices in *V-S*, keyed by their *d* values. Graph *G* is assumed as adjacency list. The procedure of Dijkstra algorithm is shown in procedure and figure 3.7(b) is shortest path based on graph shown in figure 3.7(a).

Procedure: Dijkstra Algorithm

Input : Graph $G = (V, E), weight\ w_{ij},\ \forall(i, j) \in V$

Output: S*hortest Path*

```
Begin
Initialize- Single-Source (G, s)
$S \leftarrow \phi$
$Q \leftarrow V[G]$
 while $Q \neq \phi$
    do $u \leftarrow$ Extract-Min($Q$)
       $S \leftarrow S \cup \{u\}$
          for each vertex $v \in Adj[u]$
             do Relax$(u, v, w)$
 End
```

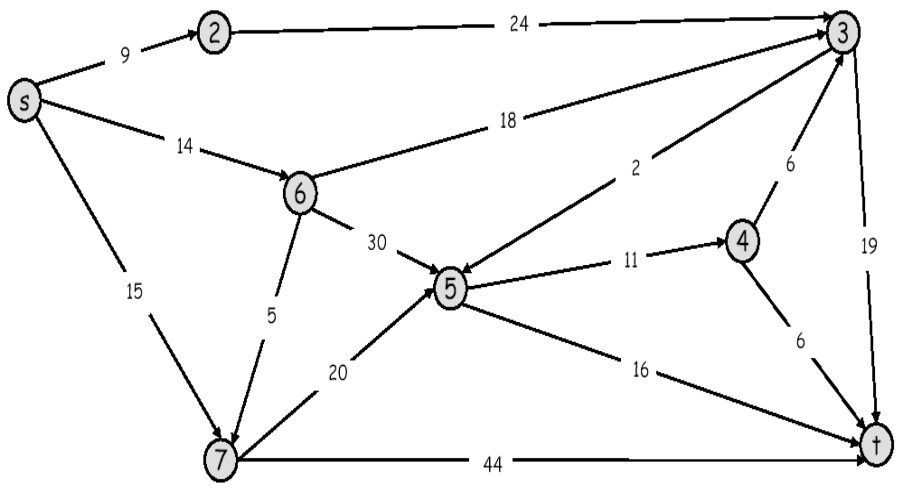

Figure 3.7(a) Directed Weighted Graph

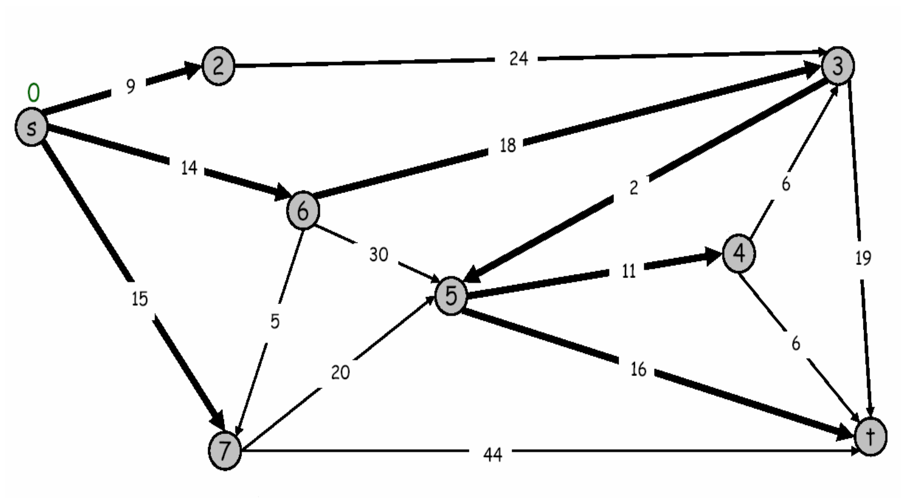

Figure 3.7(b) Shortest Path from s to t based on Dijkstra Algorithm

3.2.6 Metaheuristics

A metaheuristic is a heuristic method for solving a very general class of computational problems by combining user-given black-box procedures usually heuristics themselves in the hope of obtaining a more efficient or more robust procedure. The name combines the Greek prefix "meta" ("beyond", here in the sense of "higher level") and "heuristic" (from ευρισκειν, *heuriskein*, "to find").

Metaheuristics are generally applied to problems for which there is no satisfactory problem-specific algorithm or heuristic; or when it is not practical to implement such a method. Most commonly used metaheuristics are targeted to combinatorial optimization problems, but of course can handle any problem that can be recast in that form, such as solving boolean equations. For NP-hard optimization problems, it is often impossible to apply exact methods to large instances in order to obtain optimal solutions in acceptable time. In such cases, metaheuristics can be seen as alternatives, which are often able to provide excellent, but not necessarily optimal solutions in reasonable time. The term metaheuristic was first introduced by Glover and refers to a number of high-level strategies or concepts of how to solve optimization problems. It is somewhat difficult to specify the exact boundaries of this term. Voss gives the following definition:

A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method.

3.2.6.1 Genetic Algorithm

A Genetic Algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of Evolutionary Algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

Simple generational genetic algorithm pseudocode:

```
1. Choose the initial population of individuals
2. Evaluate the fitness of each individual in that population
3. Repeat on this generation until termination: (time limit, sufficient fitness
   achieved, etc.)
    1. Select the best-fit individuals for reproduction
    2. Breed new individuals through crossover and mutation operations to
       give birth to offspring
    3. Evaluate the individual fitness of new individuals
    4. Replace least-fit population with new individuals
```

Since this research is based on Genetic Algorithm only, it is explained in great detail in chapter 4.

3.2.6.2 Simulated Annealing

Simulated Annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global minimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter *T* (called the *temperature*), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when *T* is large, but increasingly "downhill" as *T* goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima which are the bane of greedier methods.

In the simulated annealing (SA) method, each point *s* of the search space is analogous to a state of some physical system, and the function *E*(*s*) to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from an arbitrary initial *state*, to a state with the minimum possible energy.

3.2.6.3 Local Search

In computer science, **local search** is a metaheuristic for solving computationally hard optimization problems. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the *search space*) until a solution deemed optimal is found or a time bound is elapsed. Some problems where local search has been applied are:

1. The vertex cover problem, in which a solution is a vertex cover of a graph, and the target is to find a solution with a minimal number of nodes;
2. The travelling salesman problem, in which a solution is a cycle containing all nodes of the graph and the target is to minimize the total length of the cycle;
3. The boolean satisfiability problem, in which a candidate solution is a truth assignment, and the target is to maximize the number of clauses satisfied by the assignment; in this case, the final solution is of use only if it satisfies *all* clauses.
4. The nurse scheduling problem where a solution is an assignment of nurses to shifts which satisfies all established constraints.
5. The k-medoid clustering problem and other related facility location problems for which local search offers the best known approximation ratios from a worst-case perespective.

Most problems can be formulated in terms of search space and target in several different manners. For example, for the travelling salesman problem a solution can be a cycle and the criterion to maximize is a combination of the number of nodes and the length of the cycle. But a solution can also be a path, and being a cycle is part of the target.

A local search algorithm starts from a candidate solution and then iteratively moves to a neighbor solution. This is only possible if a neighborhood relation is defined on the search space. As an example, the neighborhood of a vertex cover is another vertex cover only differing by one node. For boolean satisfiability, the neighbors of a truth assignment are usually the truth assignments only differing from it by the evaluation of a variable. The same problem may have multiple different neighborhoods defined on it; local optimization with neighborhoods that involve changing up to *k* components of the solution is often referred to as k-opt.

Typically, every candidate solution has more than one neighbor solution; the choice of which one to move to is taken using only information about the solutions in the neighborhood of the current one, hence the name *local* search. When the choice of the neighbor solution is done by taking the one locally maximizing the criterion, the metaheuristic takes the name hill climbing.

Termination of local search can be based on a time bound. Another common choice is to terminate when the best solution found by the algorithm has not been improved in a given number of steps. Local search algorithms are typically incomplete algorithms, as the search may stop even if the best solution found by the algorithm is not optimal. This can happen even if termination is due to the impossibility of improving the solution, as the optimal solution can lie far from the neighborhood of the solutions crossed by the algorithms.

Local search algorithms are widely applied to numerous hard computational problems, including problems from computer science (particularly artificial intelligence), mathematics, operations research, engineering, and bioinformatics. Examples of local search algorithm are WalkSAT and the 2-opt algorithm for the TSP. For specific problems it is possible to devise neighborhoods which are very large, possibly exponentially sized. If the best solution within the neighborhood can be found efficiently, such algorithms are referred to as very large-scale neighborhood search algorithms.

3.2.6.4 Best-First Search

It is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. Judea Pearl described best-first search as estimating the promise of node *n* by a "heuristic evaluation function *f*(*n*) which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain.

Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called greedy best-first search. Efficient selection[44, 74] of the current best candidate for extension is typically implemented using a priority queue. The A* search algorithm is an example of best-first search. Best-first algorithms are often used for path finding in combinatorial search.

3.2.6.5 Tabu Search

It is a metaheuristic algorithm that can be used for solving combinatorial optimization problems, such as the traveling salesman problem (TSP). Tabu search uses a local or neighbourhood search procedure to iteratively move from a solution *x* to a solution *x*' in the neighbourhood of *x*, until some stopping criterion has been satisfied. To explore regions of the search space that would be left unexplored by the local search procedure (see local optimality), tabu search modifies the neighbourhood structure of each solution as the search progresses. The solutions admitted to $N^*(x)$, the new neighbourhood, are determined through the use of memory structures. The search then progresses by iteratively moving from a solution x to a solution x' in $N^*(x)$. Perhaps the most important type of memory structure used to determine the solutions admitted to $N^*(x)$ is the tabu list. In its simplest form, a tabu list is a short-term memory which contains the solutions that have been visited in the recent past (less than n iterations ago, where n is the number of previous solutions to be stored (n is also called the tabu tenure)). Tabu search excludes solutions in the tabu list from $N^*(x)$. A variation of a tabu list prohibits solutions that have certain attributes (e.g., solutions to the traveling salesman problem (TSP) which include undesirable arcs) or prevent certain moves (e.g. an arc that was added to a TSP tour cannot be removed in the next n moves). Selected attributes in solutions recently visited are labeled "tabu-active." Solutions that contain tabu-active elements are “tabu”. This type of short-term memory is also called "recency-based" memory. Tabu lists containing attributes can be more effective for some domains, although they raise a new problem. When a single attribute is marked as tabu, this typically results in more than one solution being tabu. Some of these solutions that must now be avoided could be of excellent quality and might not have been visited. To mitigate this problem, "aspiration criteria" are introduced: these override a solution's tabu state, thereby including the otherwise-excluded solution in the allowed set. A commonly used aspiration criterion is to allow solutions which are better than the currently-known best solution.

3.2.6.6 Ant Colony Optimization

The Ant Colony Optimization algorithm (ACO), is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. This algorithm is a member of Ant Colony Algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Initially proposed by Marco Dorigo in 1992 , the first algorithm was aiming to search for an optimal path in a graph; based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

In the real world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food (see Ant communication).

Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over faster, and thus the pheromone density remains high as it is laid on the path as fast as it can evaporate. Pheromone evaporation has also the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained.

Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve. The original idea comes from observing the exploitation of food resources among ants, in which ants’ individually limited cognitive abilities have collectively been able to find the shortest path between a food source and the nest.

1. The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b)
2. Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive as the shortest route.
3. Ants take the shortest route, long portions of other ways lose their trail pheromones.

In a series of experiments on a colony of ants with a choice between two unequal length paths leading to a source of food, biologists have observed that ants tended to use the shortest route. A model explaining this behaviour is as follows:

1. *An ant (called "blitz") runs more or less at random around the colony;*
2. *If it discovers a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone;*
3. *These pheromones are attractive, nearby ants will be inclined to follow, more or less directly, the track;*
4. *Returning to the colony, these ants will strengthen the route;*
5. *If two routes are possible to reach the same food source, the shorter one will be, in the same time, traveled by more ants than the long route will*
6. *The short route will be increasingly enhanced, and therefore become more attractive;*
7. *The long route will eventually disappear, pheromones are volatile;*
8. *Eventually, all the ants have determined and therefore "chosen" the shortest route.*

Ants use the environment as a medium of communication. They exchange information indirectly by depositing pheromones, all detailing the status of their "work". The information exchanged has a local scope, only an ant located where the pheromones were left has a notion of them. This system is called "Stigmergy" and occurs in many social animal societies (it has been studied in the case of the construction of pillars in the nests of termites). The mechanism to solve a problem too complex to be addressed by single ants is a good example of a self-organized system. This system is based on positive feedback (the deposit of pheromone attracts other ants that will strengthen it themselves) and negative (dissipation of the route by evaporation prevents the system from thrashing). Theoretically, if the quantity of pheromone remained the same over time on all edges, no route would be chosen. However, because of feedback, a slight variation on an edge will be amplified and thus allow the choice of an edge. The algorithm will move from an unstable state in which no edge is stronger than another, to a stable state where the route is composed of strong edges.

3.2.6.7 Greedy Randomized Adaptive Search Procedure

The Greedy Randomized Adaptive Search Procedure (also known as GRASP) is a metaheuristic algorithm commonly applied to combinatorial optimization problems. GRASP typically consists of iterations made up from successive constructions of a *greedy randomized* solution and subsequent iterative improvements of it through a local search. The greedy randomized solutions are generated by adding elements to the problem's solution set from a list of elements ranked by a *greedy function* according to the quality of the solution they will achieve. To obtain variability in the candidate set of greedy solutions, well-ranked candidate elements are often placed in a *restricted candidate list* (also known as **RCL**), and chosen at random when building up the solution. This kind of greedy randomized construction method is also known as a semi-greedy heuristic, first described in Hart and Shogan (1987).

3.2.6.8 Artificial Bee Colony Algorithm

It is an optimization algorithm based on the intelligent foraging behaviour of honey bee swarm, proposed by Karaboga in 2005

In ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts. It is assumed that there is only one artificial employed bee for each food source. In other words, the number of employed bees in the colony is equal to the number of food sources around the hive. Employed bees go to their food source and come back to hive and dance on this area. The employed bee whose food source has been abandoned becomes a scout and starts to search for finding a new food source. Onlookers watch the dances of employed bees and choose food sources depending on dances. The main steps of the algorithm are given below:

- *Initial food sources are produced for all employed bees*
- *REPEAT*
    - *Each employed bee goes to a food source in her memory and determines a neighbour source, then evaluates its nectar amount and dances in the hive*
    - *Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source. After choosing a neighbour around that, she evaluates its nectar amount.*
    - *Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.*

    - *The best food source found so far is registered.*
- *UNTIL (requirements are met)*

In ABC which is a population based algorithm, the position of a food source represents a possible solution to the optimization problem and the nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of the employed bees is equal to the number of solutions in the population. At the first step, a randomly distributed initial population (food source positions) is generated. After initialization, the population is subjected to repeat the cycles of the search processes of the employed, onlooker, and scout bees, respectively. An employed bee produces a modification on the source position in her memory and discovers a new food source position. Provided that the nectar amount of the new one is higher than that of the previous source, the bee memorizes the new source position and forgets the old one. Otherwise she keeps the position of the one in her memory. After all employed bees complete the search process, they share the position information of the sources with the onlookers on the dance area. Each onlooker evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar amounts of sources. As in the case of the employed bee, she produces a modification on the source position in her memory and checks its nectar amount. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old one. The sources abandoned are determined and new sources are randomly produced to be replaced with the abandoned ones by artificial scouts.

3.2.6.9 Hill Climbing

In computer science, hill climbing is a mathematical optimization technique which belongs to the family of local search. It is relatively simple to implement, making it a popular first choice. Although more advanced algorithms may give better results, in some situations hill climbing works just as well.

Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained.

Hill climbing is used widely in artificial intelligence, for reaching a goal state from a starting node. Choice of next node and starting node can be varied to give a list of related algorithms.

3.2.6.10 Greedy Algorithm

A Greedy Algorithm is any algorithm that follows the problem solving metaheuristic of making the locally optimal choice at each stage with the hope of finding the global optimum. For example, applying the greedy strategy to the traveling salesman problem yields the following algorithm: "At each stage visit the unvisited city nearest to the current city".

In general, greedy algorithms have five pillars:

1. A candidate set, from which a solution is created
2. A selection function, which chooses the best candidate to be added to the solution
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
4. An objective function, which assigns a value to a solution, or a partial solution, and
5. A solution function, which will indicate when it has discovered a complete solution

Greedy algorithms produce good solutions on some mathematical problems, but not on others.

3.2.6.11 Memetic Algorithms

A common drawback of EAs is that there is no guarantee for the global best solution to be even local optimal. Though good diversification is present due to a large population, recombination and mutation mechanisms, EAs lack intensification in overall. Therefore, many successful EAs for complex combinatorial optimization problems additionally use hybridization to improve solution quality and/or running time. Pablo Moscato introduced the term Memetic Algorithm (MA) for local search and problem specific knowledge enhanced EAs. The term \meme" corresponds to a unit of imitation in cultural transmission. So while genetic algorithm is inspired by biological evolution, MAs attempts to mimic cultural evolution. In MAs, While the outer metaheuristic is an EA, individual solutions of the population are further improved e.g. via local search heuristics. If each intermediate solution is always turned into a local optimum, the EA would exclusively search the space of local optima (w.r.t. the neighborhood structure(s) of the local improvement procedure). So by adjusting how much effort is spent in the local improvement, it is possible to tune the balance between intensification and diversification.

3.3 Previous Work to Solve Network Design Problem

With the current demand of the industry, the reliability and service quality requirements of modern data communication networks and large investments in communication infrastructure have made it critical to design optimized networks that meet the performance parameters. Network means not only the simple connection of the locations rather an optimized reliable network which meets the requirement of the system. For designing the backbone network in the form of spanning tree, many algorithms are available but when additional constraints [47] are added, all these algorithms fail. The Network Design Problem is considered as Backbone Network Design Problem in this research work. A backbone network is not only spanning tree, it has some other constraints also like the connectivity of the node which is very important for the reliability of the network. In graph theory, this reliability is described as Degree Constrained Minimum Spanning Tree (DCMST). There is no any optimal method for this DCMST problem. Many of the network topology design problems start with the MST, which attempts to find a minimum cost tree that connects all the nodes of the network. The links or edges have associated costs that could be based on their distance, capacity, quality of line, etc. There might be other constraints imposed on the design such as the number of nodes in a sub tree, degree constraints on nodes, flow and capacity constraints on any edge or node, and type of services available on the edge or node. The MST problem is found in communication networks, circuit design, transportation, and logistics among others. The complexity of the MST problem increases as the number of nodes increases. Many heuristics have been developed to solve large problems, prominent among them being Kruskal [4] and Prim [3] algorithms heuristics which have been discussed in the section 3.2. One of the popular variations of the MST problem is the DCMST. The MST algorithm may occasionally generate a minimal spanning tree where all the links connect to one or two nodes. This solution, although optimal, may be highly vulnerable to failure due to over reliance on a few nodes. Furthermore, the technology to connect many links to a node may not be available or may be too expensive. Hence, it may be necessary to limit the number of links connecting to a node. Alternatively, from a reliability perspective it is desirable to have more than one link connect to a node so that alternative routes can be selected in the case of a node or link failure [4]. Hence, in practice, it may be added additional constraints[45, 46] that specify the upper and lower bound of the number of links connecting to a node. DCMST was specifically developed as a special case of MST with additional constraints[49, 50] to improve the reliability of the network and rerouting of traffic in the case of node failures. The problem of constrained trees has been studied for many years [25]. When additional constraints are added to MST problem it becomes NP-Hard Problem [26]. Both Prominent methods can't deal with extra constraint[48] and there are no exact methods to solve this problem. Similarly BFS and DFS have their own limitations. These two methods even can't deal with minimum spanning tree while they provide only simple spanning tree. Kruskal [4] algorithm works for the unconstrained MST [50, 51] by where in a first step the edges of the graph are sorted ascending according to their costs. Afterwards, the edges are considered in this order and an edge is accepted for the tree as long as it does not lead to a cycle. So this algorithm starts with a forest of independent trees (the single nodes) and iteratively connects them until this procedure results in a single spanning tree of minimum costs. This approach makes it much more difficult to impose additional constraints like a diameter restriction to the whole MST. However, on dense or even complete graphs Kruskal's algorithm has a higher run time complexity than Prim's MST algorithm since it is dominated by sorting all edges [21]. As a conclusion there is no method to deal with degree constrained minimum spanning tree which is the form of backbone network design. Adding extra constraints means one of the hardest problems in NP Hard category. To solve this NP-hard problem metaheuristic Genetic Algorithm is proposed here and all the experiments have been made here with genetic algorithm only.

Network Design Problem is categorized as NP-hard and exact methods and heuristics are not efficient to solve this problem, researchers[64, 65] moved towards metaheuristics. The earliest heuristic algorithm for DCMST was proposed by Obruca [27] as a solution to TSP. Narula and Ho [25] proposed three heuristic algorithms to solve the DCMST problem: primal, dual, and branch and bound. Savelsbergh and Volgenant [28] introduced an "edge exchange" algorithm that provided better performance. Although these methods solve experimental size problems, the computation time increases dramatically when the problem size gets larger. In recent years, researchers[66, 67] have attempted using meta-heuristics such as Tabu search and genetic (evolutionary) algorithms to solve these problems. Lixia Hanr and Yuping Wang [29] proposed a novel approach with the objective violation degree but it is limited for maximum 200 nodes. Berna Dengiz, Fulya Altiparmak and Alice E. Smith

[30] proposed a method which is very effective for the reliability of the network because they proposed that at least two different must exist between all pair of nodes. This approach is not effective for the cost point of view. They have considered only for three small size networks 5, 7 and 20. Rajeev Kumar and Nilanjan Banerjee [31] worked for multicriteria network design using evolutionary [52,61] algorithm, but only limited up to 36 nodes. In this research work network size is extended from 10 to 1000 which is attempted by few researchers with limited genetic operators. Other important consideration is various types of constraints with the variation of genetic operators.

CHAPTER 4

Genetic Algorithm

GAs are stochastic search algorithms based on the mechanism of natural selection and natural genetics. GA, differing from conventional search techniques, start with an initial set of random solutions called population satisfying boundary and/or system constraints to the problem. Each individual in the population is called a chromosome (or individual), representing a solution to the problem at hand. Chromosome is a string of symbols usually, but not necessarily, a binary bit string. The chromosomes evolve through successive iterations called generations. During each generation, the chromosomes are evaluated, using some measures of fitness. To create the next generation, new chromosomes, called offspring, are formed by either merging two chromosomes from current generation using a crossover operator or modifying a chromosome using a mutation operator. A new generation is formed by selection, according to the fitness values, some of the parents and offspring, and rejecting others so as to keep the population size constant. Fitter chromosomes have higher probabilities of being selected. After several generations, the algorithms converge to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem.

4.1 General Structure of a Genetic Algorithm

In general, a GA has five basic components:

1. *A genetic representation of potential solutions to the problem.*

2. A way to create a population (an initial set of potential solutions).

3. An evaluation function rating solutions in terms of their fitness.

4. Genetic operators that alter the genetic composition of offspring (Crossover, mutation, selection, etc.).

5. Parameter values that genetic algorithm uses (population size, probabilities of applying genetic operators, etc.).

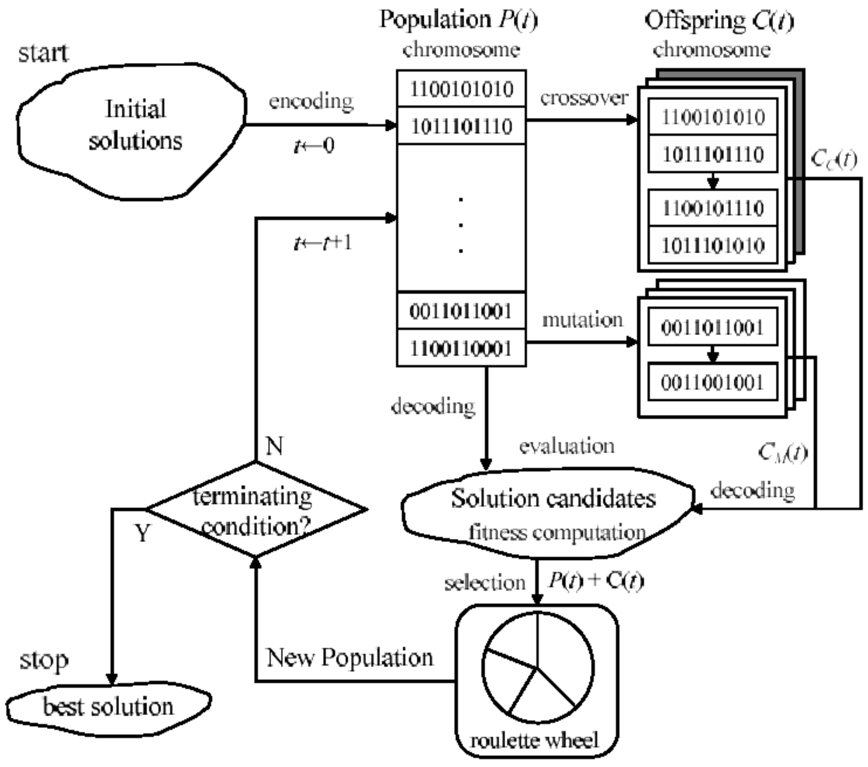

Figure 4.1 The general structure of genetic algorithms

Figure 4.1 shows a general structure of GA. Let P(t) and C(t) are parents and offspring in current generation t, respectively and the general implementation structure of GA is described as follows:

Procedure: Basic Genetic Algorithm

```
Input: problem data, GA parameters
Output: the best solution
Begin
    $t \leftarrow 0$;
     initialize $P(t)$ by encoding routine;
     evaluate $P(t)$ by decoding routine;
     while (not terminating condition) do
          create $C(t)$ from $P(t)$ by crossover
     routine;
          create $C(t)$ from $P(t)$ by mutation routine;
          evaluate $C(t)$ by decoding routine;
          select $P(t+1)$ from $P(t)$ and $C(t)$ by
     selection routine;
          $t \leftarrow t+1$;
     end
     output the best solution
  end
```

Figure 4.2 Pseudo Code of basic genetic algorithms

4.2 Exploitation and Exploration

Search is one of the more universal problem solving methods for such problems one cannot determine a prior sequence of steps leading to a solution. Search can be performed with either *blind strategies* or *heuristic strategies*. Blind search strategies do not use information about the problem domain. Heuristic search strategies use additional information to guide search move along with the best search directions. There are two important issues in search strategies: exploiting the best solution and exploring the search space. Michalewicz gave a comparison on hillclimbing search, random search and genetic search [32]. Hillclimbing is an example of a strategy which exploits the best solution for possible improvement, ignoring the exploration of the search space. Random search is an example of a strategy which explores the search space, ignoring the exploitation of the promising regions of the search space. GA is a class of general purpose search methods combining elements of directed and stochastic search which can produce a remarkable balance between exploration and exploitation of the search space. At the beginning of genetic search, there is a widely random and diverse population and crossover operator tends to perform wide-spread search for exploring all solution space. As the high fitness solutions develop, the crossover operator provides exploration in the neighborhood of each of them. In other words, what kinds of searches (exploitation or exploration) a crossover performs would be determined by the environment of genetic system (the diversity of population) but not by the operator itself. In addition, simple genetic operators are designed as general purpose search methods (the domain-independent search methods) they perform essentially a blind search and could not guarantee to yield an improved offspring.

4.3 Population-based Search

Generally, an algorithm for solving optimization problems is a sequence of computational steps which asymptotically converge to optimal solution. Most classical optimization methods generate a deterministic sequence of computation based on the gradient or higher order derivatives of objective function. The methods are applied to a single point in the search space. The point is then improved along the deepest descending direction gradually through iterations as shown in Fig. 4.2. This Point-to-point approach embraces the danger of failing in local optima. GA performs a multi-directional search by maintaining a population of potential solutions. The population-to-population approach is hopeful to make the search escape from local optima. Population undergoes a simulated evolution: at each generation the relatively good solutions are reproduced, while the relatively bad solutions die. GA uses probabilistic transition rules to select someone to be reproduced and someone to die so as to guide their search toward regions of the search space with likely improvement.

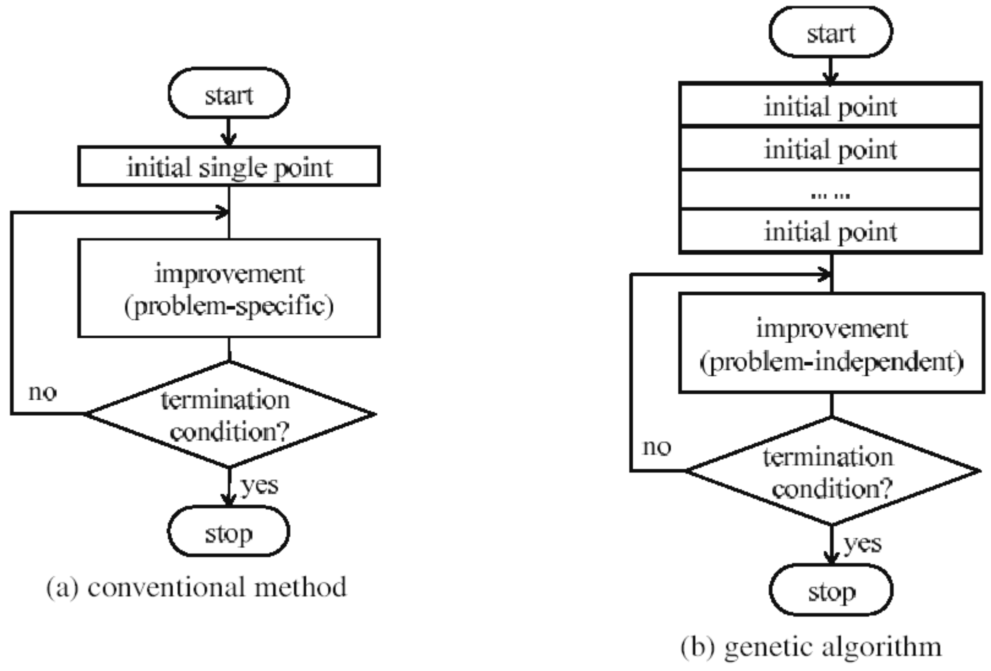

Figure 4.3 Comparison of conventional and genetic approaches

4.4 Major Advantages

GA has received considerable attention regarding their potential as a novel optimization technique. There are three major advantages when applying GA to optimization problems:

Adaptability: GA does not have much mathematical requirement regarding about the optimization problems. Due to the evolutionary nature, GA will search for solutions without regard to the specific inner workings of the problem. GA can handle any kind of objective functions and any kind of constraints, i.e., linear or nonlinear, defined on discrete, continuous or mixed search spaces.

2. Robustness: The use of evolution operators makes GA very effective in performing a global search (in probability), while most conventional heuristics usually perform a local search. It has been proved by many studies that GA is more efficient and more robust in locating optimal solution and reducing computational effort than other conventional heuristics.

3. Flexibility: GA provides us great flexibility to hybridize with domain-dependent heuristics to make an efficient implementation for a specific problem.

4.5 Implementation of Genetic Algorithm

In the implementation of GA, several components should be considered. First, a genetic representation of solutions should be decided (i.e., encoding); second, a fitness function for evaluating solutions should be given. (i.e., decoding); third, genetic operators such as crossover operator, mutation operator and selection methods should be designed; last, a necessary component for applying GA to the constrained optimization is how to handle constraints because genetic operators used to manipulate the chromosomes often yield infeasible offspring.

4.5.1 GA Vocabulary

Because GA is rooted in both natural genetics and computer science, the terminologies used in GA literatures are a mixture of the natural and the artificial. In a biological organism, the structure that encodes the prescription that specifies how the organism is to be constructed is called a chromosome. One or more chromosomes may be required to specify the complete organism. The complete set of chromosomes is called a genotype, and the resulting organism is called a phenotype. Each chromosome comprises a number of individual structures called genes. Each gene encodes a particular feature of the organism, and the location, or locus, of the gene within the chromosome structure, determines what particular characteristic the gene represents. At a particular locus, a gene may encode one of several different values of the particular characteristic it represents. The different values of a gene are called alleles. The correspondence of GA terms and optimization terms is summarized in Table 1.1.

Table 4.1 Explanation of GA terms

Genetic algorithms	**Explanation**
Chromosome (string, individual)	Solution (coding)
Genes (bits)	Part of solution
Locus	Position of gene
Alleles	Values of gene
Phenotype	Decoded solution
Genotype	Encoded solution

4.5.2 Encoding Issue

How to encode a solution of a given problem into a chromosome is a key issue for the GA. This issue has been investigated from many aspects, such as mapping characters from a genotype space to a phenotype space when individuals are decoded into solutions and the metamorphosis properties when individuals are manipulated by genetic operators

4.5.2.1 Classification of Encoding

In Holland's work, encoding is carried out using binary strings [33]. The binary encoding for function optimization problems is known to have severe drawbacks due to the existence of Hamming cliffs, which describes the phenomenon that a pair of encodings with a large Hamming distance belongs to points with minimal distances in the phenotype space. For example, the pair 01111111111 and 10000000000 belongs to neighboring points in the phenotype space (points of the minimal Euclidean distances) but have the maximum Hamming distance in the genotype space. To cross the Hamming cliff, all bits have to be changed at once. The probability that crossover and mutation will occur to cross it can be very small. In this sense, the binary code does not preserve locality of points in the phenotype space. For many real-world applications, it is nearly impossible to represent their solutions with the binary encoding. Various encoding methods have been created for particular problems in order to have an effective implementation of the GA. According to what kind of symbols is used as the alleles of a gene, the encoding methods can be classified as follows:

- Binary encoding
- Real number encoding
- Integer/literal permutation encoding
- A general data structure encoding

The real number encoding is best for function optimization problems. It has been widely confirmed that the real number encoding has higher performance than the binary or Gray encoding for function optimizations and constrained optimizations. Since the topological structure of the genotype space for the real number encoding method is identical to that of the phenotype space, it is easy for us to create some effective genetic operators by borrowing some useful techniques from conventional methods. The integer or literal permutation encoding is suitable for combinatorial optimization problems. Since the essence of combinatorial optimization problems is to search for a best permutation or combination of some items subject to some constraints, the literal permutation encoding may be the most reasonable way to deal with this kind of issue. For more complex real-world problems, an appropriate data structure is suggested as the allele of a gene in order to capture the nature of the problem. In such cases, a gene may be an array or a more complex data structure. According to the structure of encodings, the encoding methods also can be classified into the following two types:

- One-dimensional encoding
- Multi-dimensional encoding

In most practices, the one-dimensional encoding method is adopted. However, many real-world problems have solutions of multi-dimensional structures. It is natural to adopt a multi-dimensional encoding method to represent those solutions. According to what kinds of contents are encoded into the encodings, the encoding methods can also be divided as follows:

- Solution only
- Solution + parameters

In the GA practice, the first way is widely adopted to conceive a suitable encoding to a given problem. An individual consists of two parts: the first part is the solution to a given problem and the second part, called strategy parameters, contains variances and covariance of the normal distribution for mutation. The purpose for incorporating the strategy parameters into the representation of individuals is to facilitate the evolutionary self-adaptation of these parameters by applying evolutionary operators to them. Then the search will be performed in the space of solutions and the strategy parameters together. In this way a suitable adjustment and diversity of mutation parameters should be provided under arbitrary circumstances.

4.5.2.2 Infeasibility and Illegality

The GA works on two kinds of spaces alternatively: the encoding space and the solution space, or in the other words, the genotype space and the phenotype space. The genetic operators work on the genotype space while evaluation and selection work on the phenotype space. Natural selection is the link between chromosomes and the performance of the decoded solutions. The mapping from the genotype space to the phenotype space has a considerable influence on the performance of the GA. The most prominent problem associated with mapping is that some individuals correspond to infeasible solutions to a given problem. This problem may become very severe for constrained optimization problems and combinatorial optimization problems The *infeasibility* of chromosomes originates from the nature of the constrained optimization problem. Whatever methods are used, conventional ones or genetic algorithms, they must handle the constraints. For many optimization problems, the feasible region can be represented as a system of equalities or inequalities. For such cases, penalty methods can be used to handle infeasible chromosomes. In constrained optimization problems, the optimum typically occurs at the boundary between feasible and infeasible areas. The penalty approach will force the genetic search to approach the optimum from both sides of the feasible and infeasible regions. The *illegality* of chromosomes originates from the nature of encoding techniques. For many combinatorial optimization problems, problem-specific encodings are used and such encodings usually yield illegal offspring by a simple one-cut-point crossover operation. Because an illegal chromosome cannot be decoded to a solution, the penalty techniques are inapplicable to this situation. Repairing techniques are usually adopted to convert an illegal chromosome to a legal one. For example, the well-known PMX operator is essentially a kind of two-cut-point crossover for permutation representation together with a repairing procedure to resolve the illegitimacy caused by the simple two-cut-point crossover.

4.5.2.3 Properties of Encodings

Given a new encoding method, it is usually necessary to examine whether it can build an effective genetic search with the encoding. Several principles have been proposed to evaluate an encoding [34]:

Property 1 (*Space*): Chromosomes should not require extravagant amounts of memory.

Property 2 (*Time*): The time complexity of executing evaluation, recombination and mutation on chromosomes should not be a higher order.

Property 3 (*Feasibility*): A chromosome corresponds to a feasible solution.

Property 4 (*Legality*): Any permutation of a chromosome corresponds to a solution.

Property 5 (*Completeness*): Any solution has a corresponding chromosome.

Property 6 (*Uniqueness*): The mapping from chromosomes to solutions (decoding) may belong to one of the following three cases

Property 7 (*Heritability*): Offspring of simple crossover (*i.e.*, one-cut point crossover) should correspond to solutions which combine the basic feature of their parents.

Property 8 (*Locality*): A small change in chromosome should imply a small change in its corresponding solution.

4.5.2.4 Initialization

In general, there are two ways to generate the initial population, *i.e.,* the heuristic initialization and random initialization while satisfying the boundary and/or system constraints to the problem. Although the mean fitness of the heuristic initialization is relatively high so that it may help the GA to find solutions faster, in most large scale problems, for example, network design problems, the heuristic approach may just explore a small part of the solution space and it is difficult to find global optimal solutions because of the lack of diversity in the population. Usually it is to design an *encoding procedure* depending on the chromosome for generating the initial population.

4.5.3 Fitness Evaluation

A fitness function is a particular type of objective function that prescribes the optimality of a solution (that is, a chromosome) in a genetic algorithm so that that particular chromosome may be ranked against all the other chromosomes. Optimal chromosomes, or at least chromosomes which are *more* optimal, are allowed to breed and mix their datasets by any of several techniques, producing a new generation that will (hopefully) be even better. An ideal fitness function correlates closely with the algorithm's goal, and yet may be computed quickly. Speed of execution is very important, as a typical genetic algorithm[52, 55] must be iterated many, many times in order to produce a usable result for a non-trivial problem. This is one of the main drawbacks of GAs in real world applications and limits their applicability in some industries. It is apparent that amalgamation of approximate models may be one of the most promising approaches, especially in the following cases:

- Fitness computation time of a single solution is extremely high,
- Precise model for fitness computation is missing,
- The fitness function is uncertain or noisy.

Two main classes of fitness functions exist: one where the fitness function does not change, as in optimizing a fixed function or testing with a fixed set of test cases; and one where the fitness function is mutable, as in niche differentiation or co-evolving the set of test cases. Another way of looking at fitness functions is in terms of a fitness landscape, which shows the fitness for each possible chromosome. Definition of the fitness function is not straightforward in many cases and often is performed iteratively if the fittest solutions produced by GA are not what is desired. In some cases, it is very hard or impossible to come up even with a guess of what fitness function definition might be. Interactive genetic algorithms[53, 54] address this difficulty by outsourcing evaluation to external agents (normally humans).

4.5.4 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. Genetic diversity, the level of biodiversity, refers to the total number of genetic characteristics in the genetic makeup of a species. It is distinguished from genetic variability, which describes the tendency of genetic characteristics to vary. The academic field of population genetics includes several hypotheses and theories regarding genetic diversity. The neutral theory of evolution proposes that diversity is the result of the accumulation of neutral substitutions.

Diversifying selection is the hypothesis that two subpopulations of a species live in different environments that select for different alleles at a particular locus. This may occur, for instance, if a species has a large range relative to the mobility of individuals within it. Frequency-dependent selection is the hypothesis that as alleles become more common, they become more vulnerable. This is often invoked in host-pathogen interactions, where a high frequency of a defensive allele among the host means that it is more likely that a pathogen will spread if it is able to overcome that allele. When GA proceeds, both the search direction to optimal solution and the search speed should be considered as important factors, in order to keep a balance between exploration and exploitation in search space. In general, the exploitation of the accumulated information resulting from GA search is done by the selection mechanism, while the exploration to new regions of the search space is accounted for by genetic operators. The genetic operators mimic the process of heredity of genes to create new offspring at each generation. The operators are used to alter the genetic composition of individuals during representation. In essence, the operators perform a random search, and cannot guarantee to yield an improved offspring. There are three common genetic operators: crossover, mutation and selection.

4.5.4.1 Crossover

Crossover is the main genetic operator. It operates on two chromosomes at a time and generates offspring by combining both chromosomes' features. A simple way to achieve crossover would be to choose a random cut-point and generate the offspring by combining the segment of one parent to the left of the cut-point with the segment of the other parent to the right of the cut-point. This method works well with bit string representation. The performance of GA depends to a great extent, on the performance of the crossover operator used. The crossover probability (denoted by P_C) is defined as the probability of the number of offspring produced in each generation to the population size (usually denoted by popSize). This probability controls the expected number $P_C \times$ pop Size of chromosomes to undergo the crossover operation. A higher crossover probability allows exploration of more of the solution space, and reduces the chances of settling for a false optimum; but if this probability is too high, it results in the wastage of a lot of computation time in exploring unpromising regions of the solution space. Up to now, several crossover operators have been proposed for the real numbers encoding, which can roughly be put into four classes: conventional, arithmetical, direction-based, and stochastic. The conventional operators are made by extending the operators for binary representation into the real-coding case. The conventional crossover operators can be broadly divided by two kinds of crossover:

- Simple crossover: one-cut point, two-cut point, multi-cut point or uniform
- Random crossover: flat crossover, blend crossover

The arithmetical operators are constructed by borrowing the concept of linear combination of vectors from the area of convex set theory. Operated on the floating point genetic representation, the arithmetical crossover operators, such as convex, affine, linear, average, intermediate, extended intermediate crossover, are usually adopted. The direction-based operators are formed by introducing the approximate gradient direction into genetic operators. The direction-based crossover operator uses the value of objective function in determining the direction of genetic search. The stochastic operators give offspring by altering parents by random numbers with some distribution.

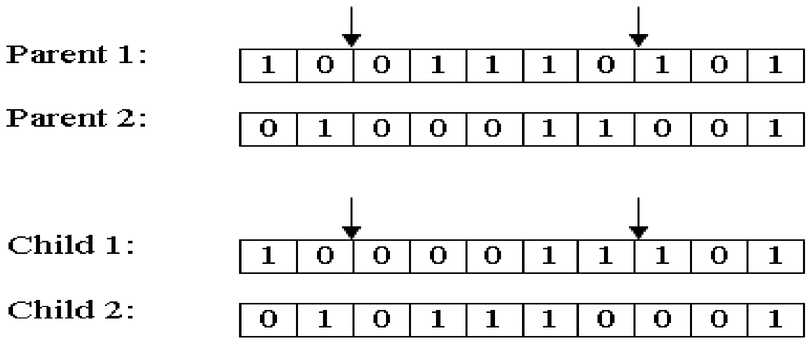

Figure 4.4 Crossover.

4.5.4.2 Mutation

Mutation is a background operator which produces spontaneous random changes in various chromosomes. A simple way to achieve mutation would be to alter one or more genes. In GA, mutation serves the crucial role of either (a) replacing the genes lost from the population during the selection process so that they can be tried in a new context or (b) providing the genes that were not present in the initial population. The mutation probability (denoted by P_m) is defined as the percentage of the total number of genes in the population. The mutation probability controls the probability with which new genes are introduced into the population for trial. If it is too low, many genes that would have been useful are never tried out, while if it is too high, there will be much random perturbation, the offspring will start losing their resemblance to the parents, and the algorithm [58]will lose the ability to learn from the history of the search. Up to now, several mutation operators have been proposed for real numbers encoding, which can roughly be put into four classes as crossover can be classified. Random mutation operators such as uniform mutation, boundary mutation, and plain mutation, belong to the conventional mutation operators, which simply replace a gene with a randomly selected real number with a specified range. Dynamic mutation (non uniform mutation) is designed for fine-tuning capabilities aimed at achieving high precision, which is classified as the arithmetical mutation operator. Directional mutation operator is a kind of direction-based mutation, which uses the gradient expansion of objective function. The direction can be given randomly as a free direction to avoid the chromosomes jamming into a corner. If the chromosome is near the boundary, the mutation direction given by some criteria might point toward the close boundary, and then jamming could occur. Several mutation operators for integer encoding have been proposed.

• Inversion mutation selects two positions within a chromosome at random and then inverts the substring between these two positions.

• Insertion mutation selects a gene at random and inserts it in a random position.

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

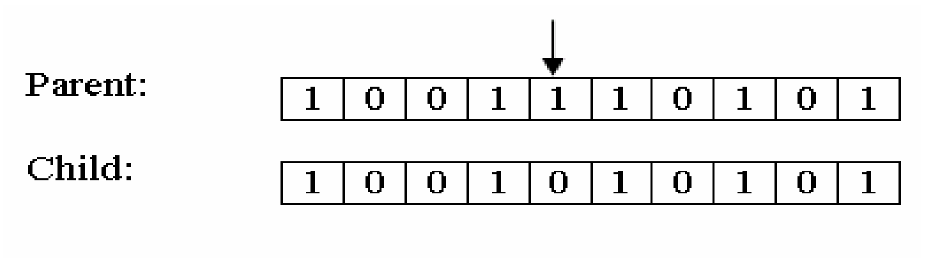

Figure 4.5 Mutation

4.5.4.3 Selection

Selection is the stage of a genetic algorithm in which individual genomes are chosen from a population for later breeding (recombination or crossover).

A generic selection procedure may be implemented as follows:

1. The fitness function is evaluated for each individual, providing fitness values, which are then normalized. Normalization means dividing the fitness value of

each individual by the sum of all fitness values, so that the sum of all resulting fitness values equals 1.

2. The population is sorted by descending fitness values.
3. Accumulated normalized fitness values are computed (the accumulated fitness value of an individual is the sum of its own fitness value plus the fitness values of all the previous individuals). The accumulated fitness of the last individual should of course be 1 (otherwise something went wrong in the normalization step!).
4. A random number *R* between 0 and 1 is chosen.
5. The selected individual is the first one whose accumulated normalized value is greater than *R*.

If this procedure is repeated until there are enough selected individuals, this selection method is called fitness proportionate selection or roulette-wheel selection. If instead of a single pointer spun multiple times equally spaced pointers on a wheel that spin once, is called stochastic universal sampling. Repeatedly selecting the best individual of a randomly chosen subset is tournament selection. Taking the best half, third or another proportion of the individuals is truncation selection.

There are other selection algorithms[59] that do not consider all individuals for selection, but only those with a fitness value that is higher than a given (arbitrary) constant. Other algorithms select from a restricted pool where only a certain percentage of the individuals are allowed, based on fitness value. Retaining the best individuals in a generation unchanged in the next generation, is called *elitism* or *elitist selection*. It is a successful (slight) variant of the general process of constructing a new population.

During the past two decades, many selection methods have been proposed, examined, and compared. Common selection methods are as follows:

- Roulette wheel selection
- Tournament selection
- Truncation selection
- Elitist selection
- Ranking and scaling
- Sharing

Roulette wheel selection, proposed by Holland, is the best known selection type. The basic idea is to determine selection probability or survival probability for each chromosome proportional to the fitness value. Then a model roulette wheel can be made displaying these probabilities. The selection process is based on spinning the wheel the number of times equal to population size, each selecting a single chromosome for the new procedure.

Tournament selection runs a tournament" among a few individuals chosen at random from the population and selects the winner (the one with the best fitness). Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Truncation selection is also a deterministic procedure that ranks all individuals according to their fitness and selects the best as parents. *Elitist selection* is generally used as supplementary to the proportional selection process.

Ranking and Scaling *mechanisms* are proposed to mitigate these problems. The scaling method maps raw objective function values to positive real values, and the survival probability for each chromosome is determined according to these values. Fitness scaling has a twofold intention: (1) to maintain a reasonable differential between relative fitness ratings of chromosomes, and (2) to prevent too-rapid takeover by some super-chromosomes to meet the requirement to limit competition early but to stimulate it later.

Sharing selection is used to maintain the diversity of population for multi-model function optimization. A sharing function optimization is used to maintain the diversity of population. A sharing function is a way of determining the degradation of an individual's fitness due to a neighbor at some distance. With the degradation, the reproduction probability of individuals in a crowd peak is restrained while other individuals are encouraged to give offspring.

4.5.5 Handling Constraints

A necessary component for applying GA to constrained optimization is how to handle constraints because genetic operators used to manipulate the chromosomes often yield infeasible offspring. There are several techniques proposed to handle constraints with GA . The existing techniques can be roughly classified as follows:

• Rejecting strategy

• Repairing strategy

• Modifying genetic operators strategy

• Penalizing strategy

Each of these strategies have advantages and disadvantages.

4.5.5.1 Rejecting strategy

Rejecting strategy discards all infeasible chromosomes created throughout an evolutionary process. This is a popular option in many GA. The method may work reasonably well when the feasible search space is convex and it constitutes a reasonable part of the whole search space. However, such an approach has serious limitations. For example, for many constrained optimization problems where the initial population consists of infeasible chromosomes only, it might be essential to improve them. Moreover, quite often the system can reach the optimum easier if it is possible to "cross" an infeasible region (especially in non-convex feasible search spaces).

4.5.5.2 Repairing Strategy

Repairing a chromosome means to take an infeasible chromosome and generate a feasible one through some repairing procedure. For many combinatorial optimization problems, it is relatively easy to create a repairing procedure. Repairing strategy depends on the existence of a deterministic repair procedure to converting an infeasible offspring into a feasible one. The weakness of the method is in its problem dependence. For each particular problem, a specific repair algorithm should be designed. Also, for some problems, the process of repairing infeasible chromosomes might be as complex as solving the original problem. The repaired chromosome can be used either for evaluation only, or it can replace the original one in the population.

4.5.5.3 Modifying Genetic Operators Strategy

One reasonable approach for dealing with the issue of feasibility is to invent problem-specific representation and specialized genetic operators to maintain the feasibility of chromosomes. Michalewicz [35]. pointed out that often such systems are much more reliable than any other genetic algorithms based on the penalty approach. This is a quite popular trend: many practitioners use problem-specific representation and specialized operators in building very successful genetic algorithms in many areas [31]. However, the genetic search of this approach is confined within a feasible region.

4.5.5.4 Penalizing Strategy

These strategies above have the advantage that they never generate infeasible solutions but have the disadvantage that they consider no points outside the feasible regions. For highly constrained problem, infeasible solution may take a relatively big portion in population. In such a case, feasible solutions may be difficult to be found if it just confine genetic search within feasible regions.

4.6 Hybrid Genetic Algorithms

GA has proved to be a versatile and effective approach for solving optimization problems. Nevertheless, there are many situations in which the simple GA does not perform particularly well, and various methods of hybridization have been proposed. One of most common forms of hybrid genetic algorithm (HGA) is to incorporate local optimization as an add-on extra to the canonical GA loop of recombination and selection. With the hybrid approach, local optimization is applied to each newly generated offspring to move it to a local optimum before injecting it into the population. GA is used to perform global exploration among a population while heuristic methods are used to perform local exploitation around chromosomes. Because of the complementary properties of GA and conventional heuristics, the hybrid approach often outperforms either method operating alone. Another common form is to incorporate GA parameters adaptation. The behaviors of GA are characterized by the balance between exploitation and exploration in the search space. The balance is strongly affected by the strategy parameters such as population size, maximum generation, crossover probability, and mutation probability. How to choose a value to each of the parameters and how to find the values efficiently are very important and promising areas of research on the GA.

CHAPTER 5

Genetic Algorithm approach to Network Design

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 strategys (ES), genetic programming [62, 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[60]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 ai j = 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).

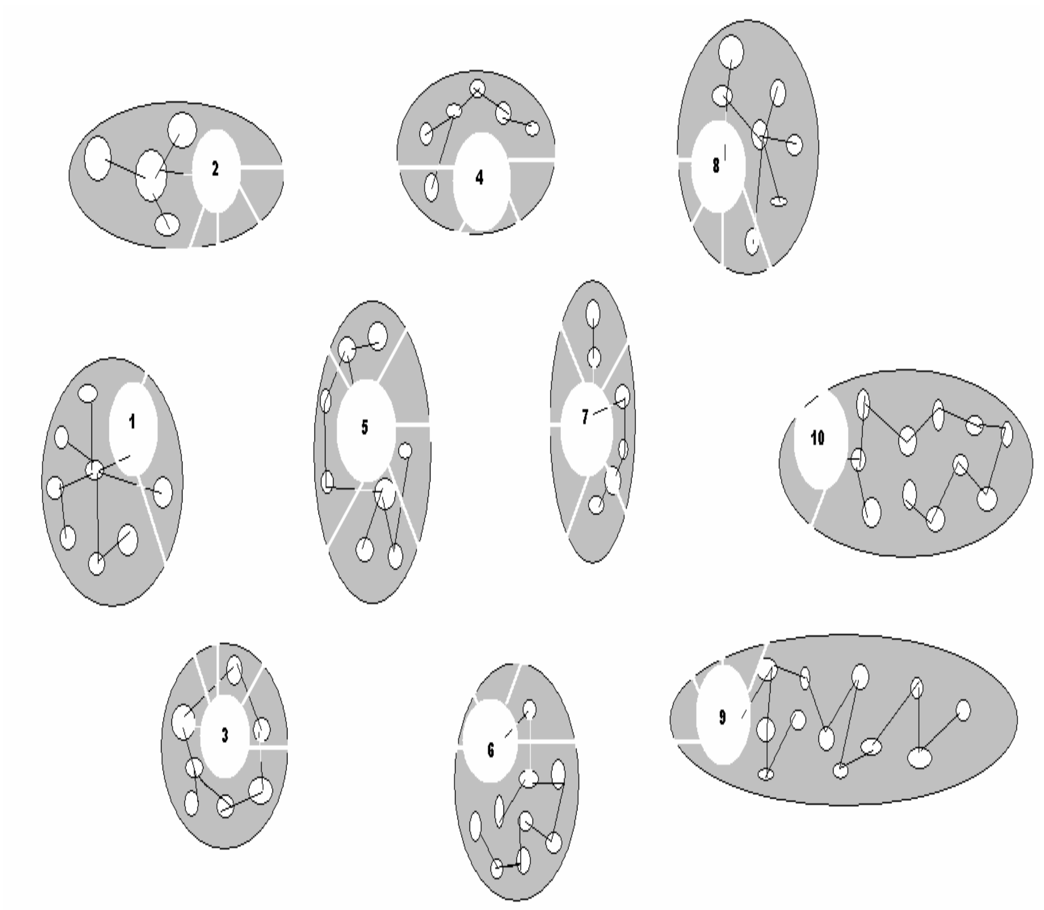

Figure 5.1 various independent networks

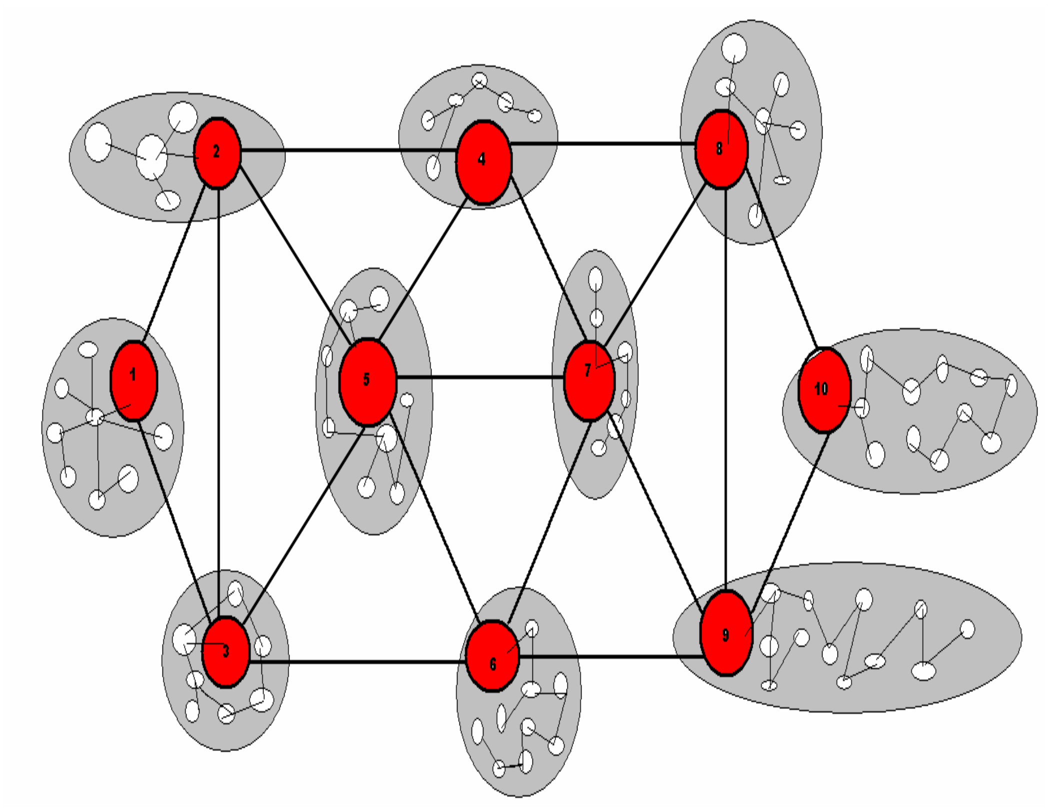

Figure 5.2(a) Backbone Network of fig 5.1

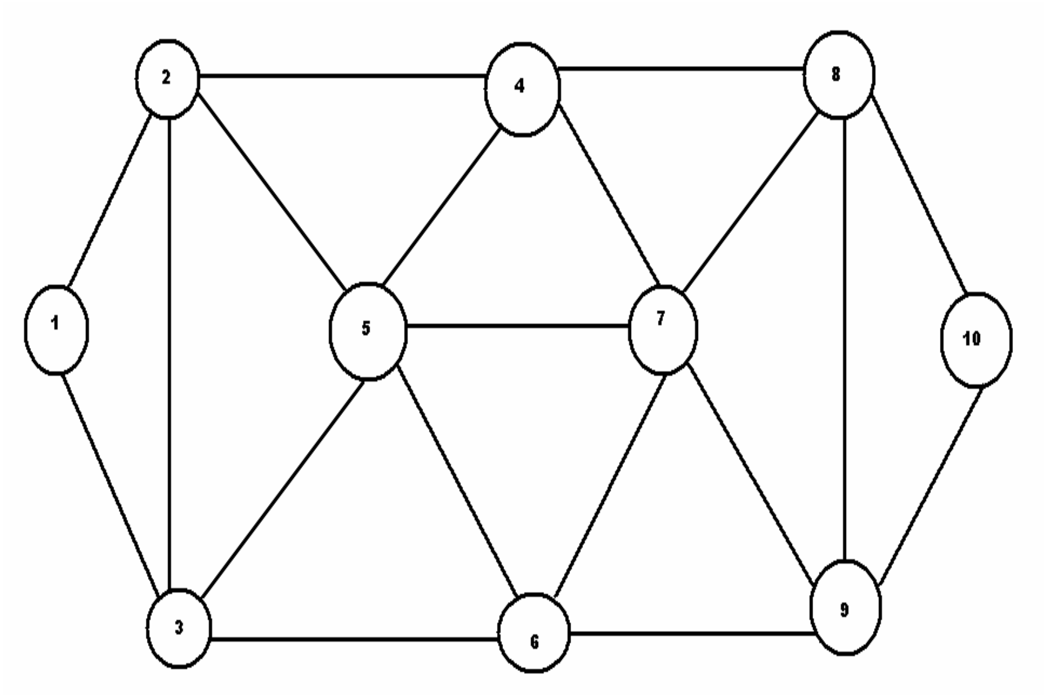

Figure 5.2(b) Undirected Graph of fig 5.2(a)

Table 5.1 Adjacency matrices of figure 5.1 graph

node	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	0	0	0	0	0	0
2	1	0	1	1	1	0	0	0	0	0
3	1	1	0	0	1	1	0	0	0	0
4	0	1	0	0	1	0	1	1	0	0
5	0	1	1	1	0	1	1	0	0	0
6	0	0	1	0	1	0	1	0	0	0
7	0	0	0	1	1	1	0	1	1	0
8	0	0	0	1	0	0	1	0	1	1
9	0	0	0	0	0	0	1	1	0	1
10	0	0	0	0	0	0	0	1	1	0

In figure 5.3, a directed graph is shown of the same figure 5.1 and it adjacency matrix is shown in Table5.2. The diagonal of this Table5.2 shows only zero (0) means no path from same node to same node. Similarly other zeros represent non availability of the path.

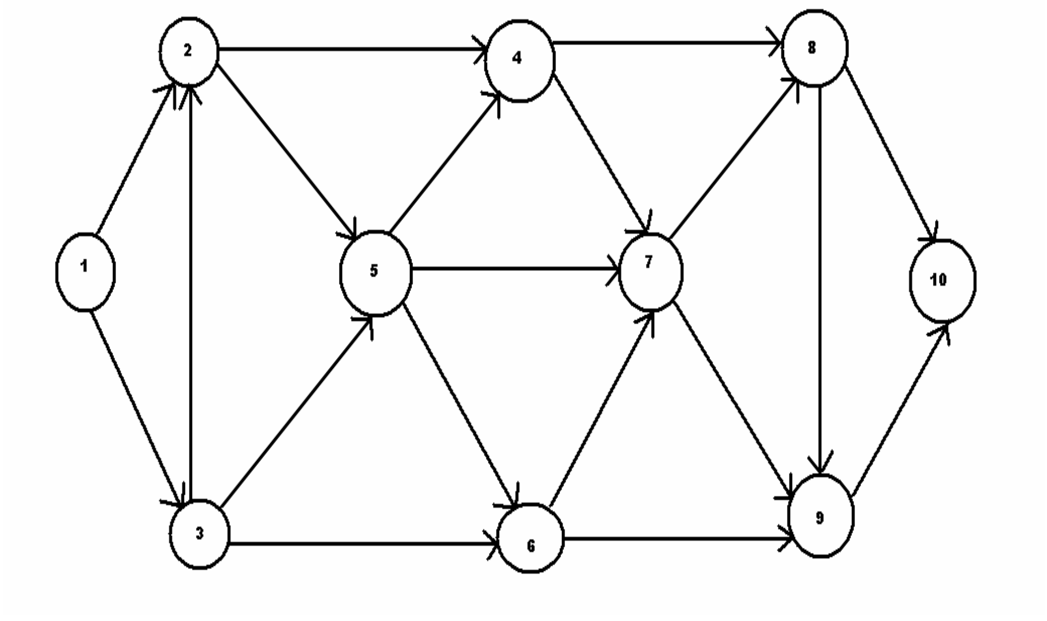

Figure 5.3 Directed Graph of fig 5.1

Table 5.2 Adjacency matrices of figure 5.2 graph

node	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	0	0	0	0	0	0
2	0	0	0	1	1	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0
4	0	0	0	0	0	0	1	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	0	0	1	0	1	0
7	0	0	0	0	0	0	0	1	1	0
8	0	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

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

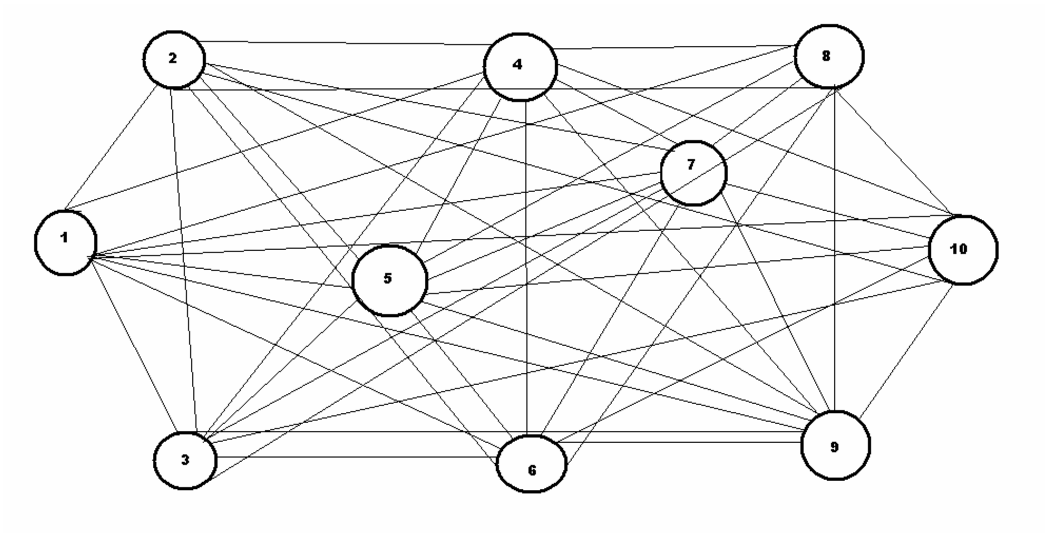

Figure 5.4 Complete Graph of fig 5.1

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

Node →	1	2	3	4	5	6	7	8	9	10
Chromosom-1	2	5	6	5	8	5	6	9	10	7
Chromosom-2	2	5	1	5	3	8	4	9	7	8
Chromosom-3	5	5	6	7	4	5	6	10	7	8
Chromosom-4	3	1	1	5	4	1	6	4	10	9
Chromosom-5	2	2	5	5	1	5	9	7	10	6
Chromosom-6	5	4	5	10	2	5	6	5	8	8
Chromosom-7	2	5	1	6	3	7	4	10	8	9
Chromosom-8	2	5	3	7	4	3	8	10	6	9
Chromosom-9	9	3	7	10	6	4	3	7	1	8
Chromosom-10	1	3	1	4	1	5	2	10	10	8

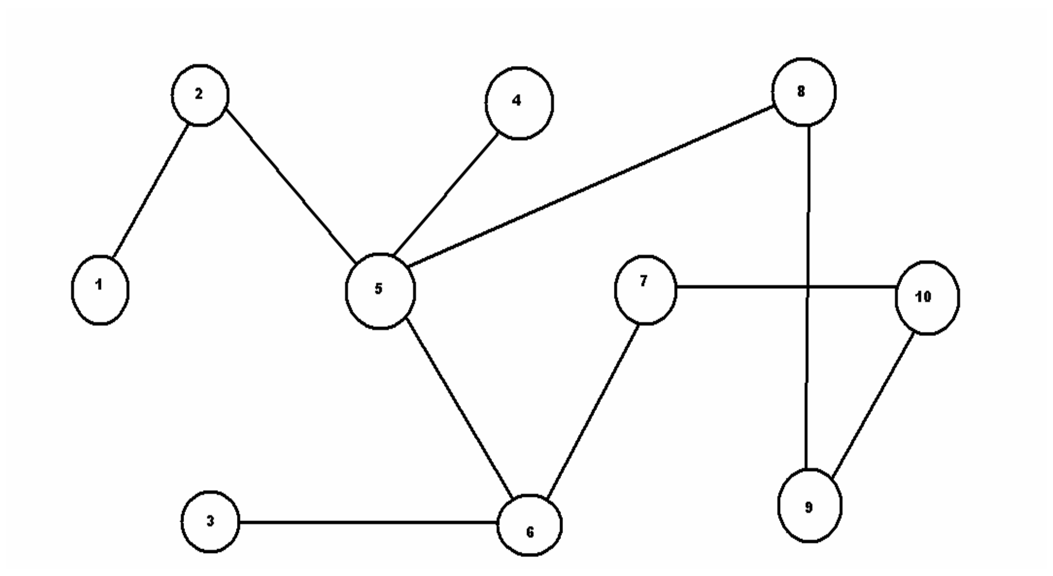

Figure 5.5 Illegal Spanning Tree based on chromosome-1.

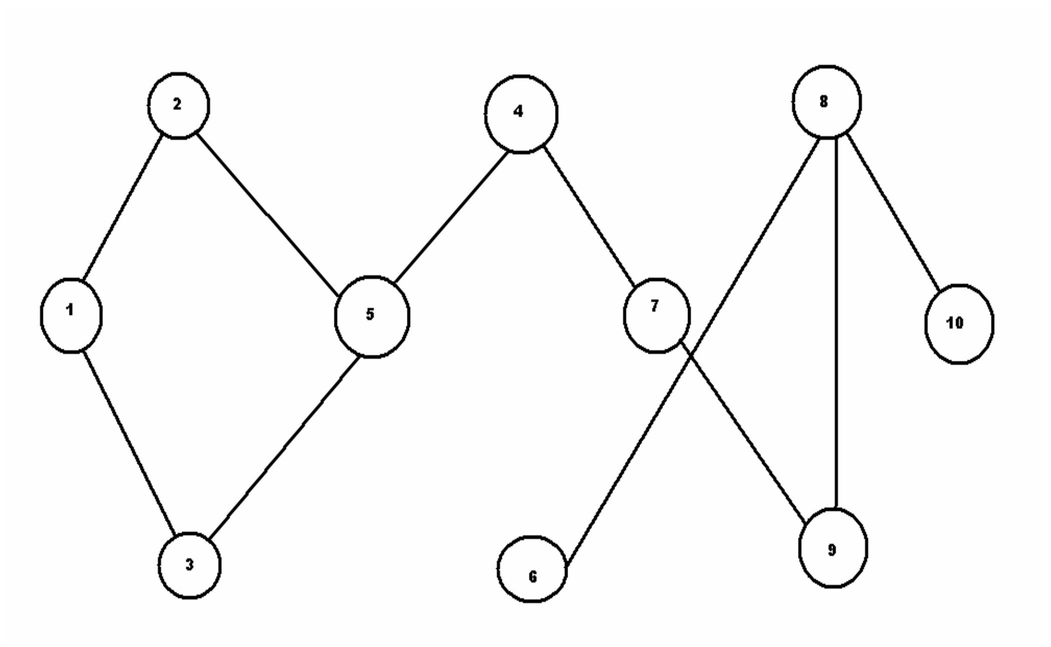

Figure 5.6 Illegal Spanning Tree based on chromosome-2

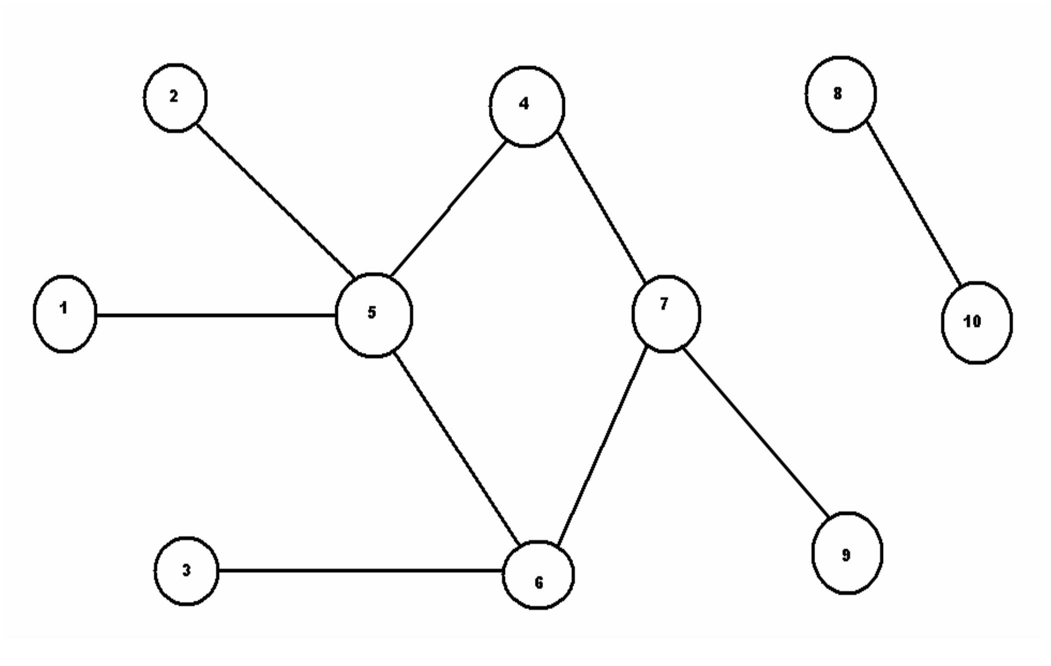

Figure 5.7 Illegal Spanning Tree based on chromosome-3

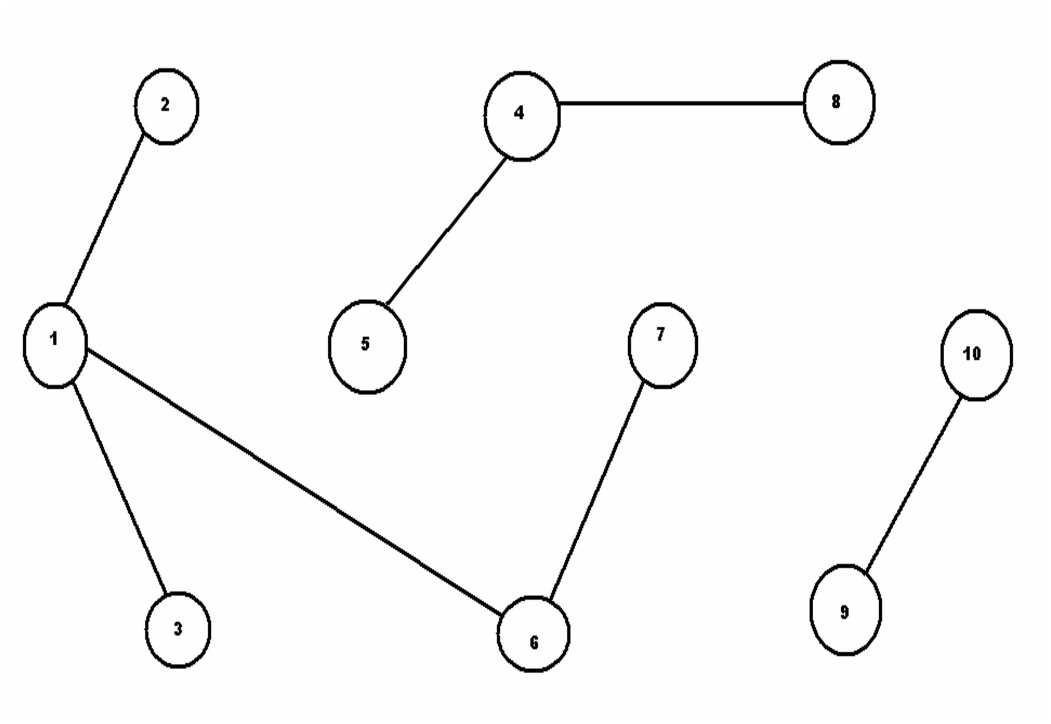

Figure 5.8 Illegal Spanning Tree based on chromosome-4

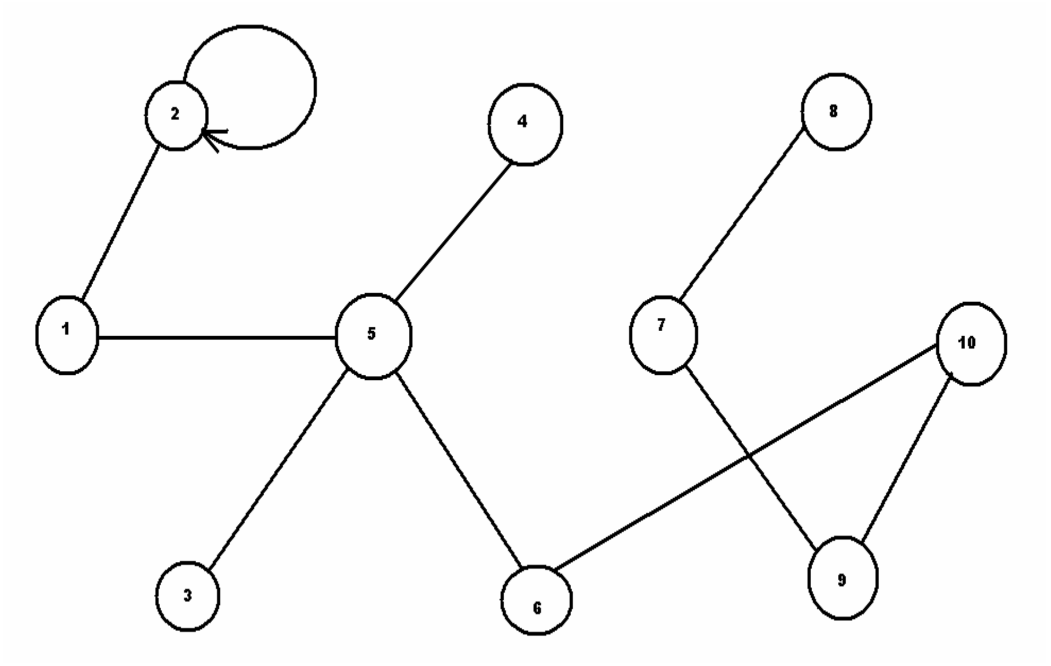

Figure 5.9 Illegal Spanning Tree based on chromosome-5

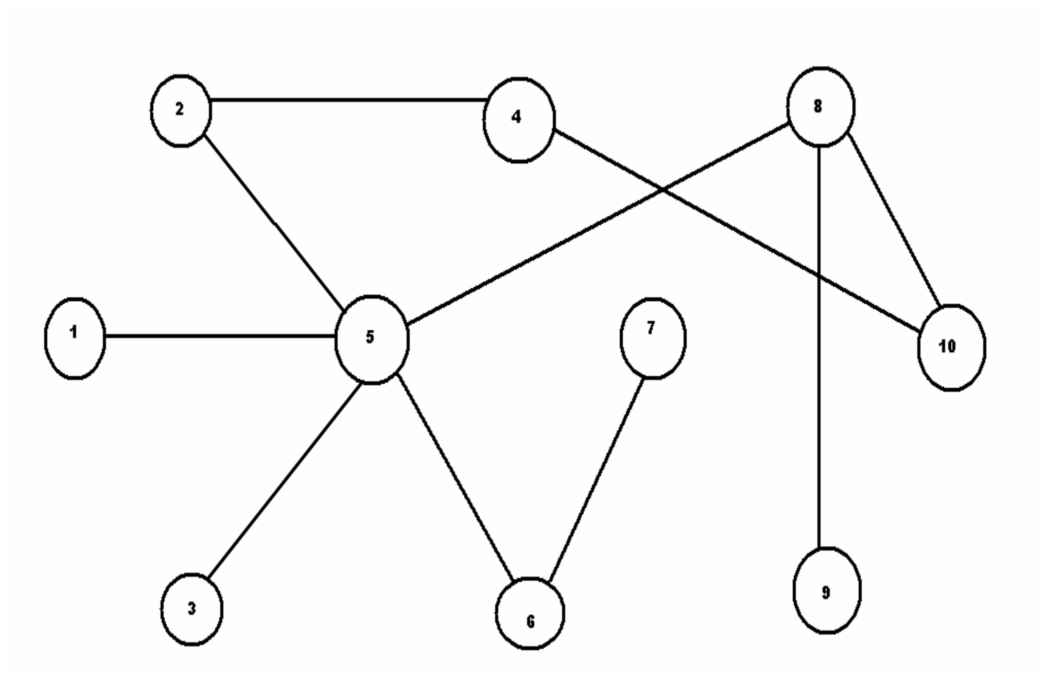

Figure 5.10 Illegal Spanning Tree based on chromosome-6

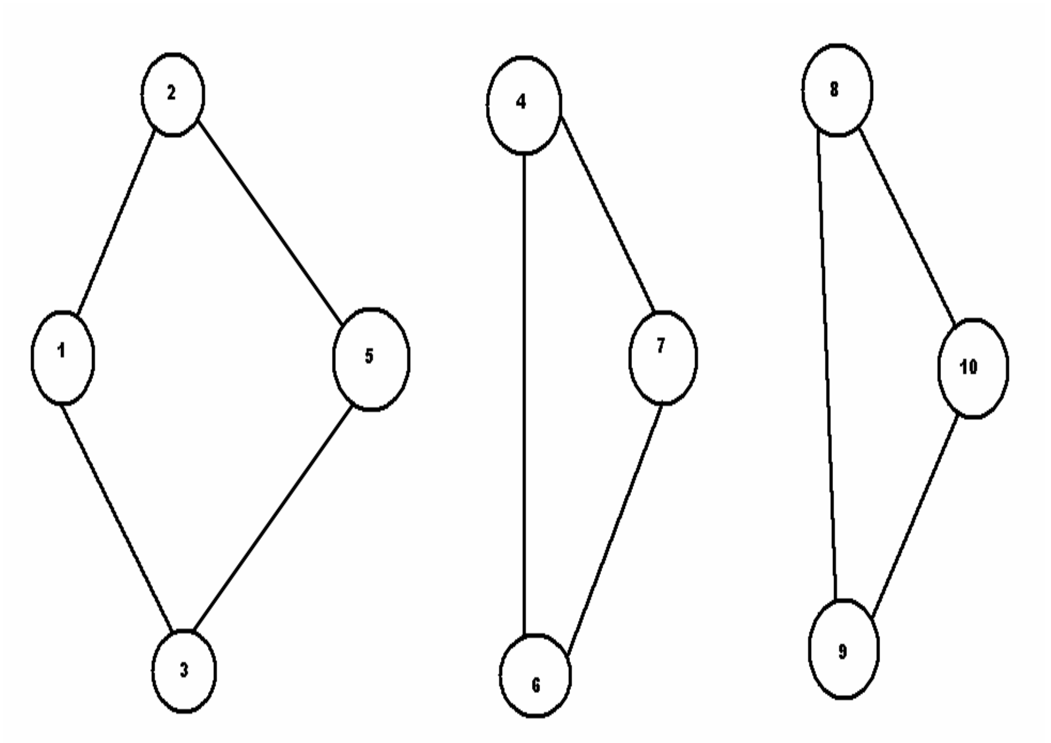

Figure 5.11 Illegal Spanning Tree based on chromosome-7

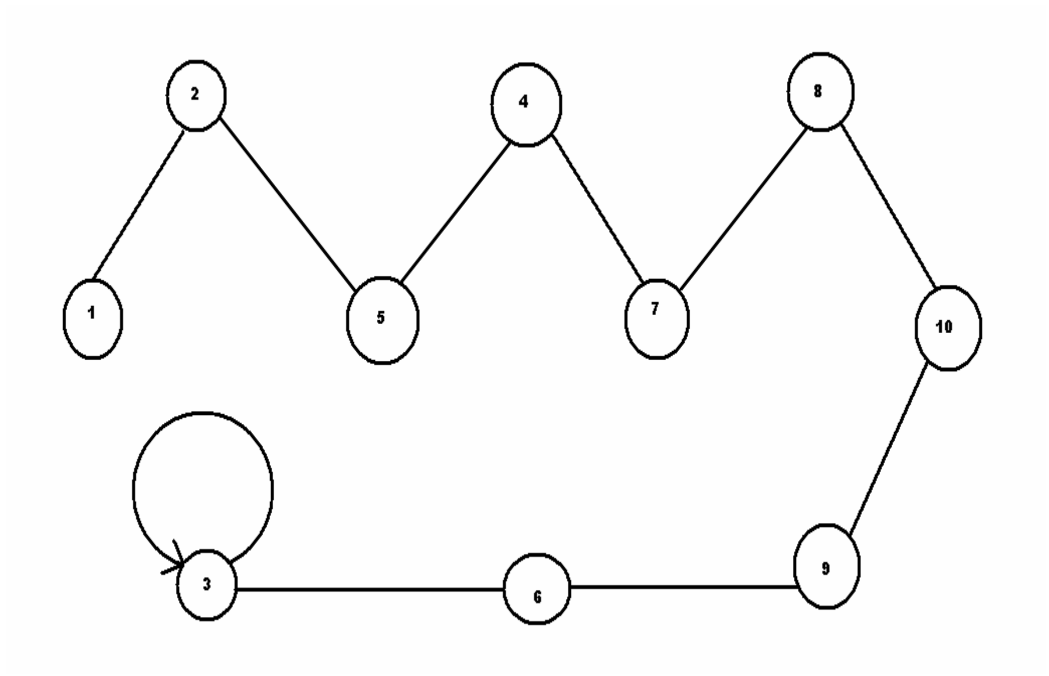

Figure 5.12 Illegal Spanning Tree based on chromosome-8

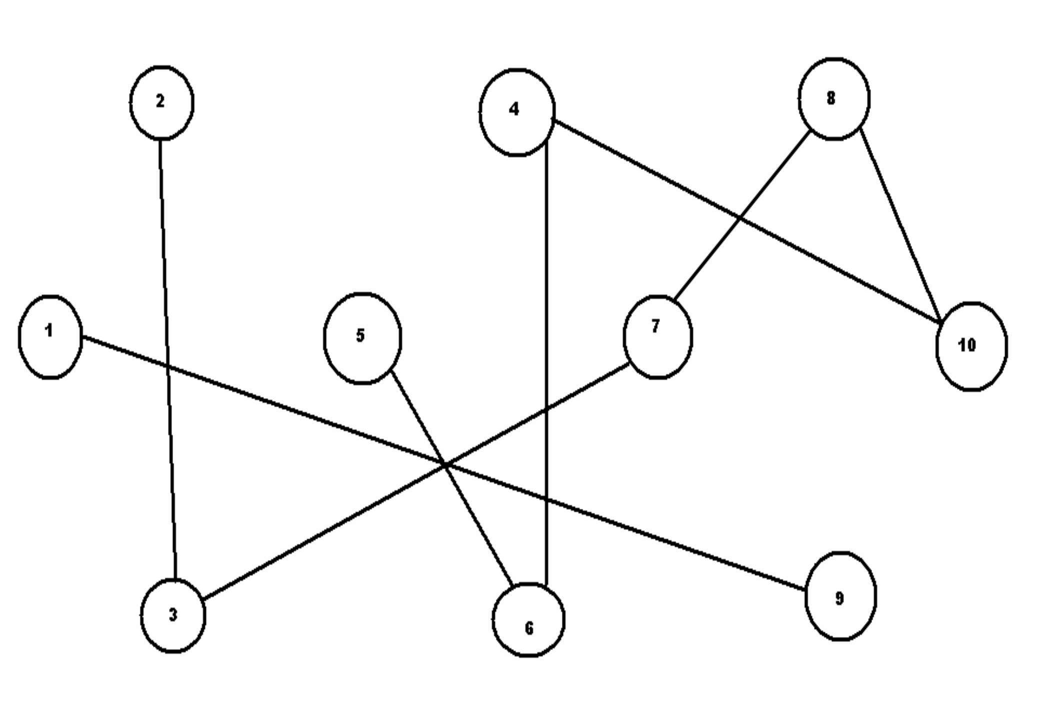

Figure 5.13 Illegal Spanning Tree based on chromosome-9

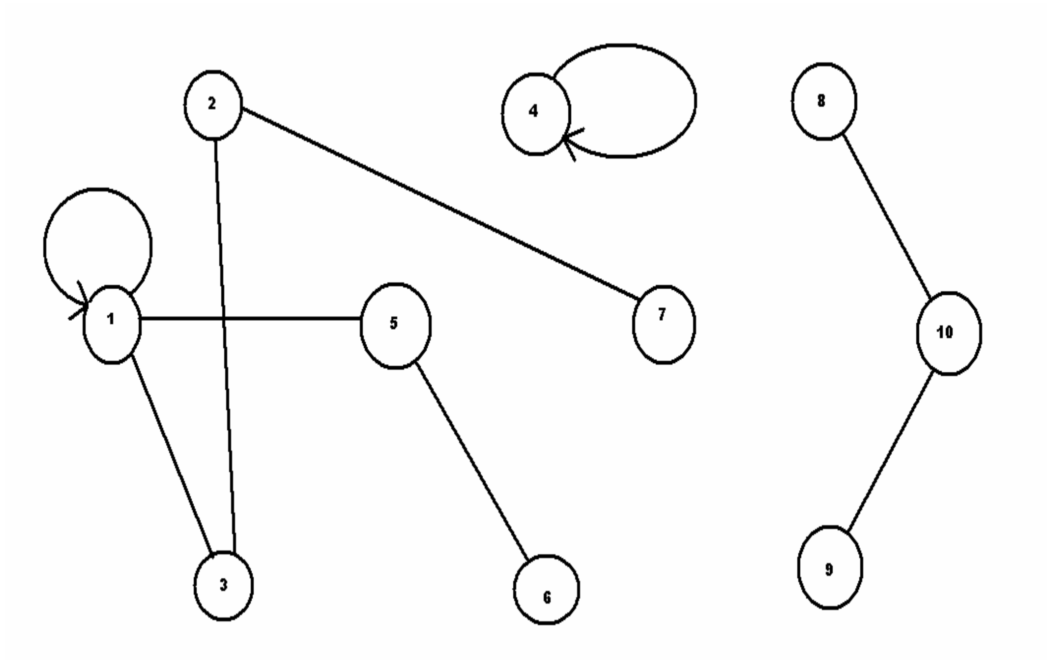

Figure 5.14 Illegal Spanning Tree based on chromosome-10

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)***

- Figure 5.10 Illegal Spanning Tree based on chromosome-6, ***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, ***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, ***because of self loop(3-3)***
- 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***.
- Figure 5.14 Illegal Spanning Tree based on chromosome-10, ***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.

Table 5.4 Randomly generated chromosome

Node →	1	2	3	4	5	6	7	8	9	10
Chromosom-11										

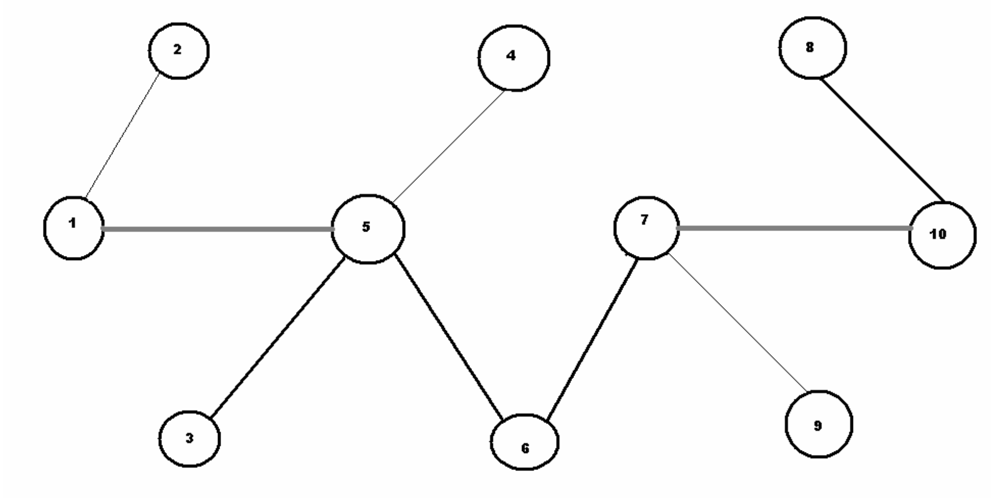

Figure 5.15 Illegal Spanning Tree based on chromosome-11

- 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 new a 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 x 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 x n) :

m = No. of nodes = Row

n = No. of chromosomes = 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 $V = \{v_1, v_2 \ldots\ldots v_n\}$, sets of vertices

$E = \{e1, e2 \ldots\ldots en\text{-}1\}$, 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

```
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 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;
                             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 thank)
                   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 5.5 Randomly generated chromosome

Node →	1	2	3	4	5	6	7	8	9	10
Chromosom-12	2	10	4	9	6	7	5	9	8	3

A solution is drawn with the chromosome-12 from the Table 5.5

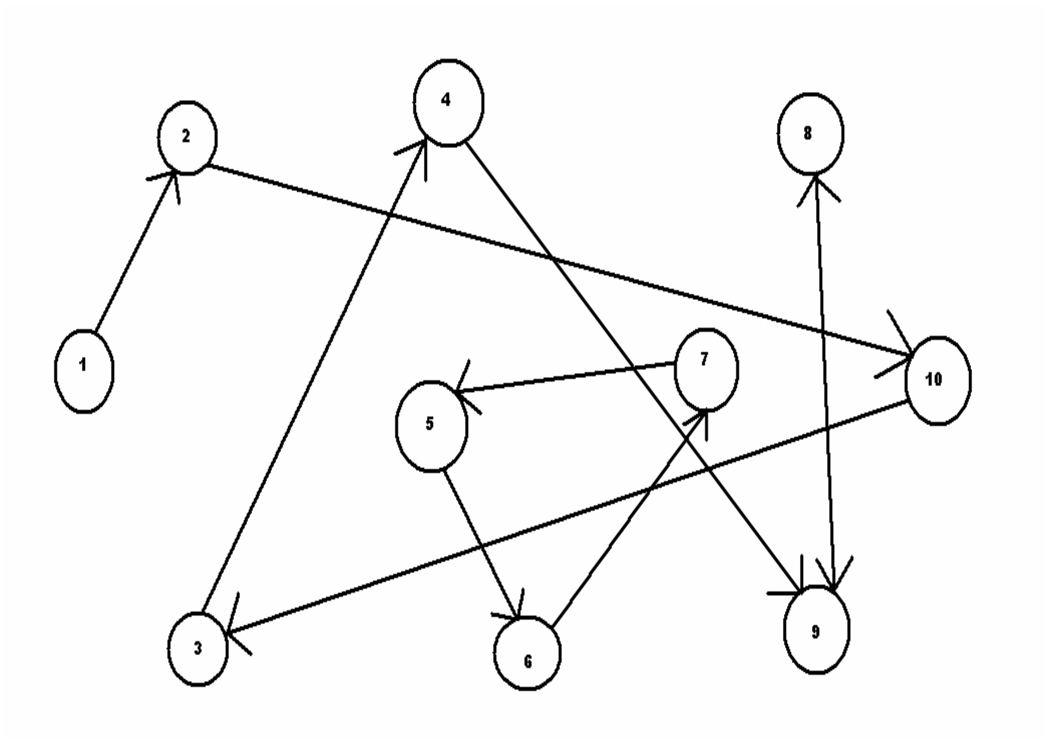

Figure 5.16 Illegal Spanning Tree (Cycle) based on chromosome-12

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.

Array location	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Stored nodes	1	2	10	3	4	9	8			

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".

Array location	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Stored nodes	1	2	10	3	4	9	8	5	6	7

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 5.6 Randomly generated chromosome

Node →	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Chromosom-13	**2**	**10**	**4**	**9**	**6**	**7**	**5**	**9**	**8**	**3**

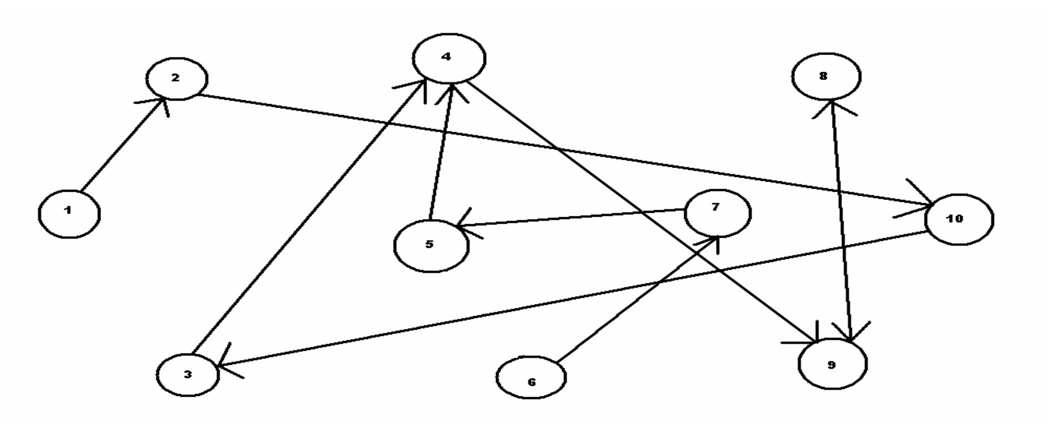

Figure 5.18 Legal Spanning Tree based on chromosome-13

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
    if ( count greater than 2)
       chromosomes(i, col) = chromosomes(i, col) + 0;
    else
        chromosomes(i,col) = chromosomes(i,col) + 1;
    end
 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

$$\mathbf{d(N) = 2*N-2} \tag{5.1}$$

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.

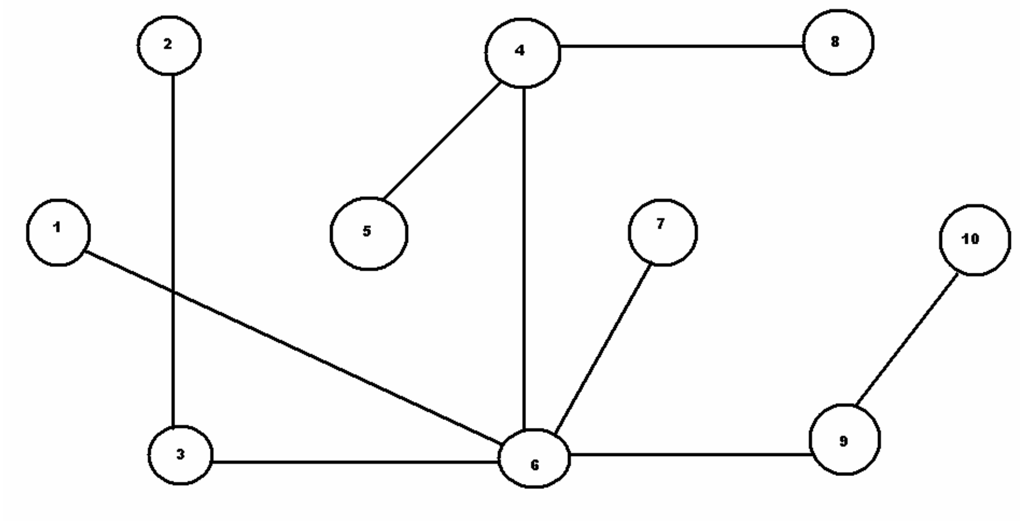

Figure 5.19(a) Legal Spanning Tree

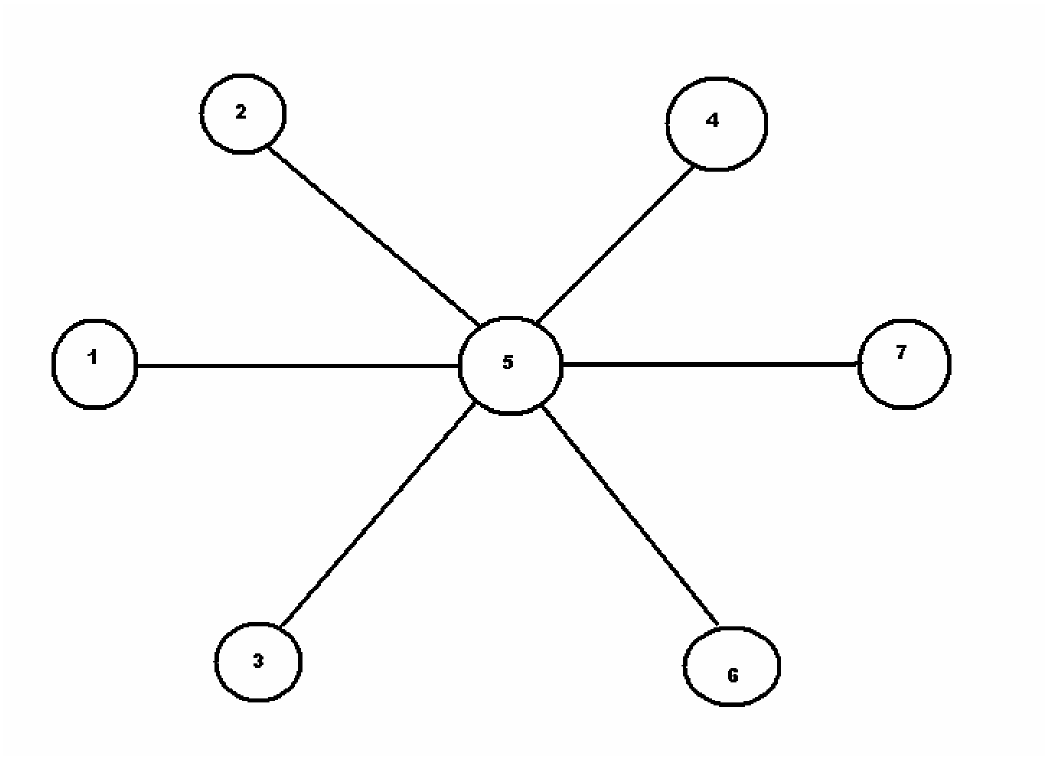

Figure 5.19(b) Legal Spanning Tree

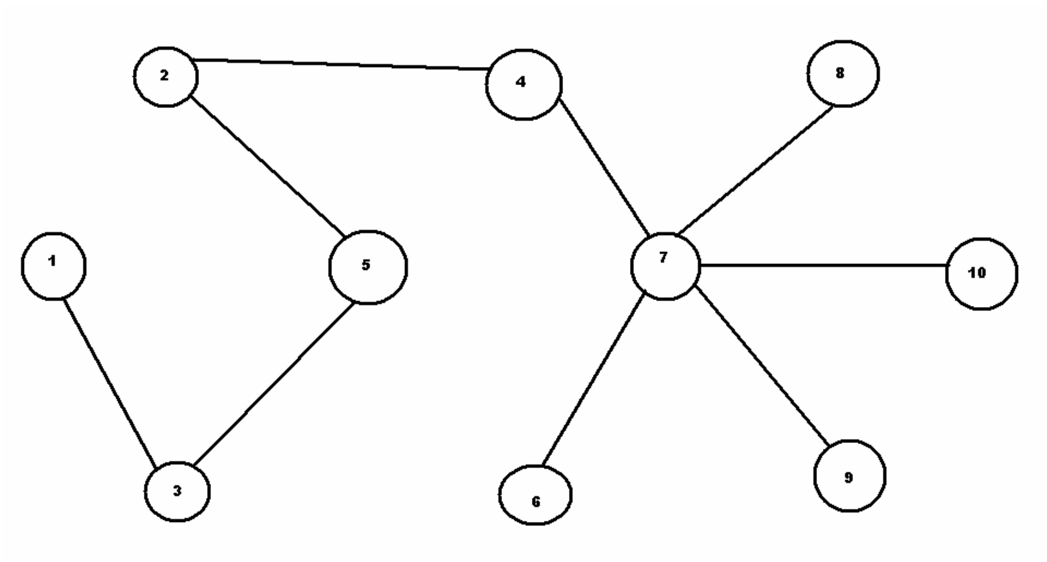

Figure 5.19(c) Legal Spanning Tree

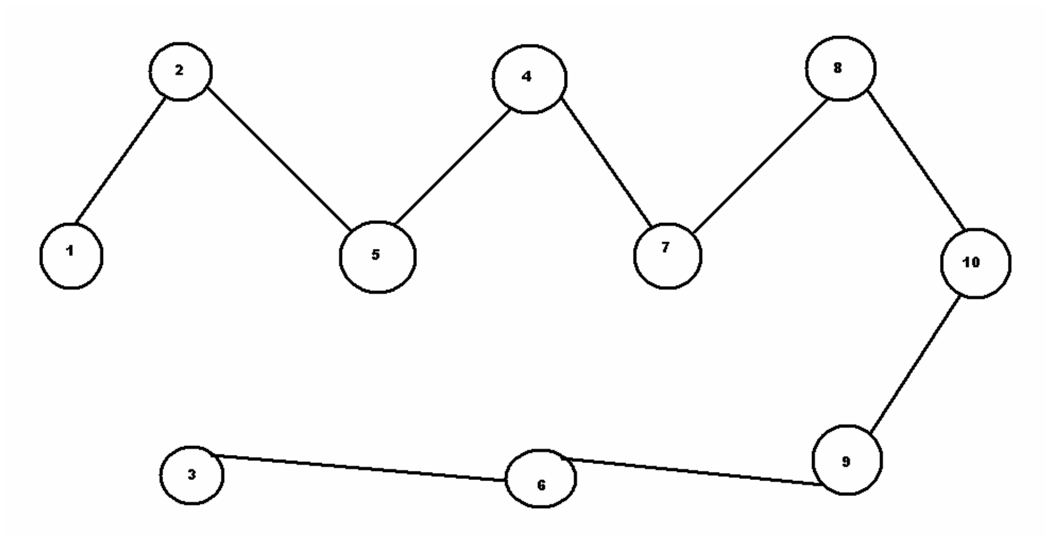

Figure 5.19(d) Legal Spanning Tree

Procedure: Degree Constraint

```
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
                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
     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_matrx is the cost matrix of the graph or network.

Procedure: Path Constraint

```
 path constraint(chromosomes, dist_matrx)
Begin
for i=1 to row do
    set t=0;
    for j=1 to col-1 do
        set k = chromosomes(i,j);
        if(dist_matrx(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
    if(t equal to (col-1))
        chromosomes(i,col) = chromosomes(i,col) + 1;
    end
end
End
```

This function checks the path constraint for each chromosome according to availability of path from dist_matrx. 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))
```

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

Chromosomes	**Fitness**	**s**
1	11	11
2	10	21
3	11	32
4	8	40
5	10	50
6	10	60
7	10	70
8	11	81
9	12	93
10	11	104

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.

$$\mathbf{11 \leq r \leq 104}$$

if r is 29.8998

$\Rightarrow$ $r < s(3)$, so 3rd chromosome will be selected and so on until *n* chromosomes are not selected. Where *n* is total no. of chromosomes.

Procedure: Roulette wheel Selection I

```
Roulette_wheel_selectionI(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
 set k=1;
 for i=1 to row do
     r= random generated number * temp;
         for j= 1 to row do
         if(r less than s(j))
             for t=1 to col do
                 new_chromosomes(k,t)=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 = sum of fitness probability

fitness probability = fitness / avg

avg = sum of fitness / no of chromosome

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

Following example shows the concept:

avg = 109 / 10 = 10.900

Table 5.8 Randomly generated chromosome with Fitness

Chromosome	Fitness	Fitness probability	s
1	12	1.1009	1.1009
2	10	.9174	2.0183
3	10	.9174	2.9358
4	11	1.0092	3.9450
5	13	1.1927	5.1376
6	11	1.0092	6.1468
7	13	1.1927	7.3394
8	9	0.8257	8.1651
9	10	0.9174	9.0826
10	10	0.9174	10.000

If r is 3.47 then

$\Rightarrow$ 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
             increment k by 1;
             out of loop ;
         end
     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
          increment k by 1;
       end
     end
   if(k greater than row)
      out of loop;
   end
   decrement t by 1;
   if(t less than (-1))
       set t=2;
   end
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
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.

Cromosome1	1	4	6	9	8
Cromosome2	2	3	5	7	8
Cromosome3	4	3	2	1	8
Cromosome4	6	7	7	5	4

Figure 5.20(a)

↓

Cromosome1	**1**	4	6	9	8
Cromosome2	**2**	3	5	7	8

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)).

Cromosome1	**2**	4	6	9	8
Cromosome2	**1**	3	5	7	8

Figure 5.20(c)

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

↓------↓

Cromosome3	**4**	**3**	2	1	8
Cromosome4	**6**	**7**	7	5	4

Figure 5.20(d)

Cromosome3	**6**	**7**	2	1	8
Cromosome4	**4**	**3**	7	5	4

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 cromosome. these two points are randomly generated and fixed for all the chromosomes Following example explains the logic behind this crossover operator. Same figure 5.20(a) is considered here

↓------------------↓

Cromosome1	1	4	6	9	8
Cromosome2	2	3	5	7	8
Cromosome3	4	3	2	1	8
Cromosome4	6	7	7	5	4

Figure 5.20(f)

After fixed two point crossover—

Cromosome1	1	4	**5**	**7**	**8**
Cromosome2	2	3	**6**	**9**	**8**
Cromosome3	4	3	**7**	**5**	**4**
Cromosome4	6	7	**2**	**1**	**8**

Figure 5.20(g)

Procedure: Fixed_two_point_crossover

```
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;
```

```
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 cromosome. 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).

↓----------------↓

Cromosome1	1	4	6	9	8
Cromosome2	2	3	5	7	8

Figure 5.20(h)

↓-------------------------↓

Cromosome3	4	3	2	1	8
Cromosome4	6	7	7	5	4

Figure 5.20(i)

After Variable two point crossover—

Cromosome1	**2**	**3**	**5**	9	8
Cromosome2	**1**	**4**	**6**	7	8
Cromosome3	4	**7**	**7**	**5**	**4**
Cromosome4	6	**3**	**2**	**1**	**8**

Figure 5.20(j)

Procedure: Variable_two_point_crossover

```
Variable_two_point_crossover(chromosomes)

Begin

Set i=1;

while( i less than row)


    p= 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
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 cromosome. 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.

↓-----------↓ ↓---↓

Cromosome1	8	5	10	1	9	5	9	9	8	9
Cromosome2	4	4	3	4	9	8	9	3	4	7

Figure 5.20(k)

After uniform crossover-

Cromosome1	8	**4**	**3**	**4**	**9**	5	9	**3**	**4**	9
Cromosome2	4	**5**	**10**	**1**	**9**	8	9	**9**	**8**	7

Figure 5.20(l)

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}
```

```
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
    incremetnt i by 2
end
End
```

5.2.4.1.5 Hybrid CrossoverI

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 cromosome.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.

↓-----------↓ ↓---↓

Cromosome1	8	4	3	4	9	5	9	3	4	9
Cromosome2	4	5	10	1	9	8	9	9	8	7
Chromosome3	6	3	5	7	1	1	2	9	8	6

Figure 5.20(m)

After uniform crossover-

Cromosome1	8	**5**	**10**	**1**	**9**	5	9	**9**	**8**	9
Cromosome2	4	**3**	**5**	**7**	**1**	8	9	**9**	**8**	7
Chromosome3	6	**4**	**3**	**4**	**9**	1	2	**3**	**4**	6

Figure 5.20(n)

Procedure: Hybrid_crossoverI

```
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
```

```
    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
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 cromosome. 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.

↓------------↓ ↓---↓

Cromosome1	8	5	10	1	9	5	9	9	8	9
Cromosome2	4	3	5	7	1	8	9	9	8	7
Chromosome3	6	4	3	4	9	1	2	3	4	6
Cromosome4	5	1	4	9	6	2	1	3	10	1
Cromosome5	4	2	5	8	7	3	4	5	10	3
Chromosom6	3	3	6	7	8	2	1	9	1	9

↑------------------↑ ↑----------------↑

Figure 5.20(o)

After Crossover-

Cromosome1	8	**3**	**5**	**7**	**1**	5	9	**9**	**8**	9
Cromosome2	4	**4**	**3**	**4**	**9**	8	9	**3**	**4**	7
Chromosome3	6	**5**	**10**	**1**	**9**	1	2	**9**	**8**	6
Cromosome4	4	**2**	**5**	**8**	**6**	2	**4**	**5**	**10**	**3**
Cromosome5	3	**3**	**6**	**7**	**7**	3	**1**	**9**	**1**	**9**
Chromosom6	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= 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}
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

```
Mutation1(chromosome)
Begin
Set k=1;
for i=1 to row do
     if(chromosome(i, col) not equal to maximum fitness)
         new_chromosome(i,i) = (col-i);
     end
end
for i=1 to row
    for j=1 to col-1 do
        mutated_chromosome(i,j) = new_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
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
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.

			↓				↓			
Chromosom	**5**	**1**	**4**	**9**	**8**	**2**	**1**	**3**	**10**	**1**

Figure 5.20(q)

After mutation-

Chromosom	**5**	**1**	**1**	**9**	**8**	**2**	**4**	**3**	**10**	**1**

Figure 5.20(r)

Procedure: Swap_mutation

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

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

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.

		↓						↓		
Chromosom	**5**	**1**	**4**	**9**	**8**	**2**	**1**	**3**	**10**	**1**

Figure 5.20(s)

After mutation-

Chromosom	**5**	**3**	**1**	**2**	**8**	**9**	**4**	**1**	**10**	**1**

Figure 5.20(t)

Procedure: Mutation_Inversion

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

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

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.

		↓						↓		
Chromosom	**5**	**1**	**4**	**9**	**8**	**2**	**1**	**3**	**10**	**1**

Figure 5.20(u)

After mutation-

↓ →

Chromosom	**5**	**1**	**3**	**4**	**9**	**8**	**2**	**1**	**10**	**1**

Figure 5.20(v)

Procedure: Mutation_Insertion

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

CHAPTER 6

Experimental Design and Results

In order to examine the proposed genetic algorithm, various different size of networks are considered. The network is considered as a weighted graph which is further represented with the adjacency matrix. The adjacency matrix contains the distance between node. The network size varies from 10 to 1000 nodes. For the connectivity of the node, a table is maintained for each graph which contains the details of degree of each node of that graph.

All the experimenatal data is separately maintained because it is bulky and not possible to include in thesis. In the experimenatal data, execution of all the 546 cases are saved with step by step execution and the final result.

Genetic Algorithm is a step by step process where the process starts from population generation and further evaluation, selection and genetic operation. The general working of genetic algorithm is discussed in section 4.1 of chapter 4. In this proposed genetic algorithm, the actual steps of genetic algorithm are kept as it is but evaluation process is repeated because of the uncertain nature of this genetic algorithm. At the same time the best result is preserved for the next generation also. The result is replaced with the better result whenever it is derived from the next generation otherwise previous better result is maintained. Keeping all this in consideration, the proposed genetic algorithm is designed as following-

Procedure: Proposed Genetic Algorithm

1. **Initialization of parent population.**
2. **Evaluation based on the following fitness functions**
    a) self loop()
    b) cycle()
    c) path constraint()
    d) degree constraint()
    e) isolation()
    f) store the complete fit chromosome
3. **Selection of the chromosome for the next generation.**

    Selection based on following function.

    1. random selection
    2. roulette wheel- I selection
    3. roulette wheel- II selection
    4. sort selection
    5. fittest selection
    6. selection sort selection-I
    7. selection sort selection-II
4. **Crossover/recombination based on following function**
    a) variable one point crossover
    b) Fixed two point crossover
    c) variable two point crossover
    d) uniform crossover
    e) hybrid crossover-1
    f) hybrid crossover-2

```
5. Evaluation (same as step-2)
6. Mutation based on following function
      1. mutation-1
      2. mutation-2
      3. random mutation
      4. swap mutation
      5. inversion mutation
      6. hybrid (insertion) mutation
7. Evaluate child and Go to step 3 until termination
   criteria satisfies
```

6.1 Experimental Design of the Backbone Network

The experimental design of this research work is based on the procedure "proposed genetic algorithm". The experiment is carried out in step by step manner as the steps are mentioned in the above procedure. To show the experimental design a network is considered of 10 nodes which is further represented by a complete graph. For the experiment, this complete graph is considered as an adjacency matrix which consists of the distance between each pair of node in the complete graph. First of all network is represented then procedure "proposed genetic algorithm" is followed step by step.

6.1.1 Backbone Network Representation

The Figure 6.1 shows the different locations to be connected, all these locations have direct path to reach to other locations. All these locations are connected with each other and represented as a complete graph in Figure 6.1.

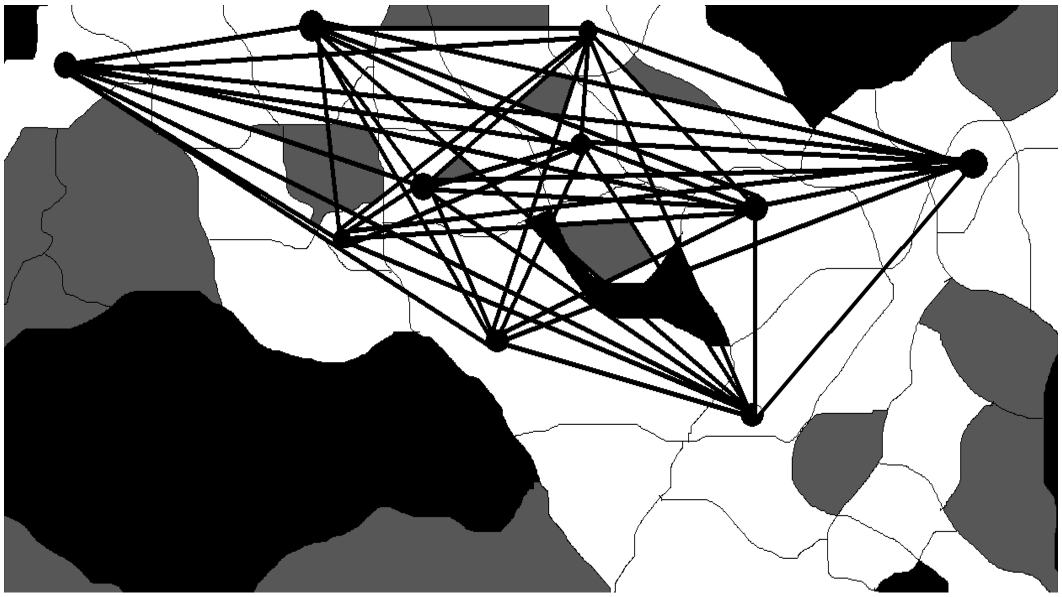

Figure 6.1 Ten different locations to be connected

A complete weighted graph is considered here which represents a backbone network to connect different locations.

Table 6.1 Adjacency matrix of complete graph of Figure 6.1

Node	1	2	3	4	5	6	7	8	9	10
1	0	42	29	43	25	62	46	41	52	33
2	42	0	57	28	25	39	62	36	6	51
3	29	57	0	57	14	41	58	39	38	56
4	43	28	57	0	37	9	30	39	27	55
5	25	25	14	37	0	35	67	52	24	71
6	62	39	41	9	35	0	50	15	34	48
7	46	62	58	30	67	50	0	68	40	69
8	41	36	39	39	52	15	68	0	55	28
9	52	6	38	27	24	34	40	55	0	65
10	33	51	56	55	71	48	69	28	65	0

6.1.2 Initialization of parent population

This is the first step of genetic algorithm after the network presentation. In this research work, parent population is generated randomly. Population means chromosomes. For the simplicity, 10 sets of chromosomes have been generated here, since this is the first population so it is called parent population. The length of each chromosome is equal to the number of node present in graph to be connected as a backbone network. Table 6.2 shows the 10 sets of randomly generated chromosomes.

Table 6.2 Randomly generated chromosomes

Chromosome1	8	3	6	2	2	6	3	10	1	9
Chromosome2	2	8	4	7	4	1	2	1	2	7
Chromosome3	1	8	10	4	1	9	5	9	2	10
Chromosome4	8	9	2	8	6	8	2	1	1	8
Chromosome5	7	5	7	7	2	7	8	7	1	3
Chromosom6	7	6	6	6	1	8	10	8	6	7
Chromosome7	6	10	4	2	7	4	1	5	9	2
Chromosome8	4	4	1	10	3	6	5	9	4	3
Chromosome9	4	1	1	3	7	6	1	9	4	7
Chromosome10	8	9	4	9	4	5	8	6	10	5

These randomly generated chromosomes represent 10 different networks. And now these chromosomes will be evaluated by fitness functions.

6.1.3 Evaluation based on fitness functions

To evaluate these chromosomes, five different fitness functions have been developed as it has been discussed in the section 5.2.2, the reason of being illegal chromosome.

6.1.3.1 Cycle

This function assigns 0 for cycle and 1 for NO-cycle to each chromosome. Table 6.3 presents the status of each of the chromosomes. The last column of this table shows the fitness of each of the chromosome. Each 0 represent the presence of the cycle and 1 cycle free network.

Table 6.3 Fitness of chromosomes after cycle check

Cromosome1	8	3	6	2	2	6	3	10	1	9	**0**
Cromosome2	2	8	4	7	4	1	2	1	2	7	**0**
Chromosome3	1	8	10	4	1	9	5	9	2	10	**0**
Cromosome4	8	9	2	8	6	8	2	1	1	8	**1**
Cromosome5	7	5	7	7	2	7	8	7	1	3	**1**
Chromosom6	7	6	6	6	1	8	10	8	6	7	**1**
Chromosome7	6	10	4	2	7	4	1	5	9	2	**1**
Cromosome8	4	4	1	10	3	6	5	9	4	3	**0**
Cromosome9	4	1	1	3	7	6	1	9	4	7	**0**
Chromosom10	8	9	4	9	4	5	8	6	10	5	**0**

6.1.3.2 Path Constraint

This function is not applicable here because it is a complete graph and paths are available from each node to each node. This function will be applicable for incomplete or partial complete graph where paths are not available between certain nodes. this function is very useful in the case of restricted path where specifically paths have mentioned. This function plays a very important role for the shortest path problem which is discussed in next chapter. So in this case each of the chromosome will be assigned fitness 1 and it will be added with the existing fitness. Table 6.4 presents the fitness status of each of the chromosomes.

Table 6.4 Fitness of chromosomes after path constraint check

Cromosome1	8	3	6	2	2	6	3	10	1	9	**1**
Cromosome2	2	8	4	7	4	1	2	1	2	7	**1**
Chromosome3	1	8	10	4	1	9	5	9	2	10	**1**
Cromosome4	8	9	2	8	6	8	2	1	1	8	**2**
Cromosome5	7	5	7	7	2	7	8	7	1	3	**2**
Chromosom6	7	6	6	6	1	8	10	8	6	7	**2**
Chromosome7	6	10	4	2	7	4	1	5	9	2	**2**
Cromosome8	4	4	1	10	3	6	5	9	4	3	**1**
Cromosome9	4	1	1	3	7	6	1	9	4	7	**1**
Chromosom10	8	9	4	9	4	5	8	6	10	5	**1**

6.1.3.3 Self loop

This function assigns fitness 1 to each gene of the chromosome, it means if there is no self loop in a chromosome, the total fitness for the chromosome will be 10. Table 6.5 presents the fitness status of each of the chromosomes.

Table 6.5 Fitness of chromosomes after self loop check

Cromosome1	8	3	6	2	2	6	3	10	1	9	**10**
Cromosome2	2	8	4	7	4	1	2	1	2	7	**11**
Chromosome3	1	8	10	4	1	9	5	9	2	10	**8**
Cromosome4	8	9	2	8	6	8	2	1	1	8	**12**
Cromosome5	7	5	7	7	2	7	8	7	1	3	**12**
Chromosom6	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome7	6	10	4	2	7	4	1	5	9	2	**11**
Cromosome8	4	4	1	10	3	6	5	9	4	3	**10**
Cromosome9	4	1	1	3	7	6	1	9	4	7	**10**
Chromosom10	8	9	4	9	4	5	8	6	10	5	**11**

Table 6.5 shows that only chromosome 4 and chromosome 5 has full fitness after self loop check.

6.1.3.4 Isolation

This function assigns 0 for isolation and 1 for NO-isolation to each chromosome. Table 6.6 presents the status of each of the chromosomes.

Table 6.6 Fitness of chromosomes after isolation check

Cromosome1	8	3	6	2	2	6	3	10	1	9	**11**
Cromosome2	2	8	4	7	4	1	2	1	2	7	**12**
Chromosome3	1	8	10	4	1	9	5	9	2	10	**8**
Cromosome4	8	9	2	8	6	8	2	1	1	8	**13**
Cromosome5	7	5	7	7	2	7	8	7	1	3	**12**
Chromosom6	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome7	6	10	4	2	7	4	1	5	9	2	**11**
Cromosome8	4	4	1	10	3	6	5	9	4	3	**11**
Cromosome9	4	1	1	3	7	6	1	9	4	7	**11**
Chromosom10	8	9	4	9	4	5	8	6	10	5	**12**

6.1.3.5 Degree constraint

This function is one of the important functions which converts this minimum spanning tree to degree constrained minimum spanning tree. For each node, degree is fixed which shows the connectivity of the node with other node. The degree varies from minimum to maximum. For the fixed degree, minimum and maximum are equal.

Table 6.7 Degree of each node of the network

Degree	**Nodes**									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Minimum	1	1	1	1	1	2	1	1	1	1
maximum	4	4	4	4	4	2	4	4	4	4

Table 6.8 presents the status of each of the chromosomes

Table 6.8 Fitness of chromosomes after degree constraint check

Cromosome1	8	3	6	2	2	6	3	10	1	9	**11**
Cromosome2	2	8	4	7	4	1	2	1	2	7	**12**
Chromosome3	1	8	10	4	1	9	5	9	2	10	**8**
Cromosome4	8	9	2	8	6	8	2	1	1	8	**14**
Cromosome5	7	5	7	7	2	7	8	7	1	3	**12**
Chromosom6	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome7	6	10	4	2	7	4	1	5	9	2	**11**
Cromosome8	4	4	1	10	3	6	5	9	4	3	**11**
Cromosome9	4	1	1	3	7	6	1	9	4	7	**11**
Chromosom10	8	9	4	9	4	5	8	6	10	5	**12**

Table 6.8 shows that only chromosome 4 has full fitness after degree constraint check.

6.1.3.6 Storage of completely fit chromosome

First of all distance is calculated for each of the chromosome and then on the basis of fitness table 6.9, completely fit chromosome is stored. If more than one chromosome are present, then the minimum distance completely fit chromosome is stored.

Table 6.9 Fitness T able for 10 Node network

Fitness function	**Fitness value**
Cycle	1
Self loop	10
Path constraint	1
Degree constraint	1
Isolation	1
Total Fitness	**14**

Distance of each chromosome is calculated on the basis of adjacency matrix Table 6.1

Table 6.10 Distance of chromosomes

Chromosome1	395
Chromosome2	442
Chromosome3	279
Chromosome4	335
Chromosome5	385
Chromosome6	278
Chromosome7	372
Chromosome8	374
Chromosome9	435
Chromosome10	422

On the basis of Table 6.8 it is clear that only one chromosome, chromosome-4 is completely fit, so this chromosome will be stored as a fittest chromosome. Table6.11 shows the fittest chromosome with the last as a distance of the chromosome.

Table 6.11 Fittest chromosome

FittestCromosome	8	9	2	8	6	8	2	1	1	8	**335**

6.1.4 Selection of the chromosome for the next generation.

Seven selection functions have been developed in this research work but only one can be used at a time. Here Roulette wheel selection method is considered. Table 6.12 shows the new set of chromosomes **(child population)** after the selection based on roulette wheel method for next generation**.**

Table 6.12 selected child population (chromosomes)

Chromosome1	6	10	4	2	7	4	1	5	9	2
Chromosome2	4	4	1	10	3	6	5	9	4	3
Chromosome3	7	6	6	6	1	8	10	8	6	7
Chromosome4	7	6	6	6	1	8	10	8	6	7
Chromosome5	7	6	6	6	1	8	10	8	6	7
Chromosom6	7	6	6	6	1	8	10	8	6	7
Chromosome7	8	3	6	2	2	6	3	10	1	9
Chromosome8	4	4	1	10	3	6	5	9	4	3
Chromosome9	8	9	4	9	4	5	8	6	10	5
Chromosome10	8	9	2	8	6	8	2	1	1	8

6.1.5 Genetic Operator Applications

There are mainly two types of genetic operator, discussed in section 5.2.4.

6.1.5.1 Crossover/Recombination

Six crossover functions have been developed here, but any one can be used at a time. Here hybrid crossover method is considered**.** . Table 6.13 shows the status of chromosome bits after hybrid crossover.

Table 6.13 Hybrid crossoverd chromosomes

Chromosome1	6	10	4	10	3	4	1	5	4	3
Chromosome2	4	4	1	6	1	6	5	9	6	7
Chromosome3	7	6	6	2	7	8	10	8	9	2
Chromosome4	7	6	6	6	1	8	10	8	6	7
Chromosome5	7	6	6	6	1	8	10	8	6	7
Chromosom6	7	6	6	6	1	8	10	8	6	7
Chromosome7	8	3	6	10	3	6	3	10	4	3
Chromosome8	4	4	1	9	4	6	5	9	10	5
Chromosome9	8	9	4	2	2	5	8	6	1	9
Chromosome10	8	9	2	8	6	8	2	1	1	8

According to the proposed genetic algorithm, these hybrid crossovered chromosomes will be evaluated. For evaluation the same 5 functions will be applied. Table 6.14 shows the total evaluation of each of the chromosome.

Table 6.14 Hybrid crossoverd chromosomes ater evaluation

Chromosome1	6	10	4	10	3	4	1	5	4	3	**12**
Chromosome2	4	4	1	6	1	6	5	9	6	7	**13**
Chromosome3	7	6	6	2	7	8	10	8	9	2	**11**
Chromosome4	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome5	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome6	7	6	6	6	1	8	10	8	6	7	**11**
Chromosome7	8	3	6	10	3	6	3	10	4	3	**12**
Chromosome8	4	4	1	9	4	6	5	9	10	5	**11**
Chromosome9	8	9	4	2	2	5	8	6	1	9	**12**
Chromosome10	8	9	2	8	6	8	2	1	1	8	**14**

As it is seen that there is only one completely fit chromosome, chromosome-10 which has the fitness 14 and the previously stored chromosome has the fitness14 also so this new chromosome will replace the previous one. Since both the previous and new chromosome have the same fitness value and same distance, so replacement does not make any difference but here it is replaced because of the maintenance of new set of value.

Table 6.15 shows the current fittest chromosome.

Table 6.15 Fittest chromosome

FittestCromosome	8	9	2	8	6	8	2	1	1	8	**335**

6.1.5.2 Mutation

Six mutation functions have been developed here, but any one can be used at a time. Here inversion mutation is considered.

Table 6.16 shows the status of chromosome bits after inversion mutation.

Table 6.16 Inversion mutated chromosomes

Chromosome1	6	10	3	10	4	4	1	5	4	3
Chromosome2	4	6	1	4	1	6	5	9	6	7
Chromosome3	7	6	6	2	7	8	10	8	9	2
Chromosome4	10	8	1	6	6	6	7	8	6	7
Chromosome5	7	6	6	6	1	8	10	8	6	7
Chromosom6	8	10	8	1	6	6	6	7	6	7
Chromosome7	8	3	6	10	3	6	3	10	4	3
Chromosome8	4	4	9	5	6	4	9	1	10	5
Chromosome9	8	9	8	5	2	2	4	6	1	9
Chromosome10	8	9	2	8	2	8	6	1	1	8

According to the proposed genetic algorithm, these inversion mutated chromosomes will be evaluated. For evaluation the same 5 functions will be applied. Table 6.17 shows the total evaluation of each of the chromosome.

Table 6.17 Inversion mutated chromosomes after evaluation

Chromosome1	6	10	3	10	4	4	1	5	4	3	**13**
Chromosome2	4	6	1	4	1	6	5	9	6	7	**11**
Chromosome3	7	6	6	2	7	8	10	8	9	2	**11**
Chromosome4	10	8	1	6	6	6	7	8	6	7	**9**
Chromosome5	7	6	6	6	1	8	10	8	6	7	**12**
Chromosome6	8	10	8	1	6	6	6	7	6	7	**12**
Chromosome7	8	3	6	10	3	6	3	10	4	3	**12**
Chromosome8	4	4	9	5	6	4	9	1	10	5	**12**
Chromosome9	8	9	8	5	2	2	4	6	1	9	**12**
Chromosome10	8	9	2	8	2	8	6	1	1	8	**14**

As it is seen that there is only one completely fit chromosome, chromosome-10 which has the fitness 14, with less distance 313 and the previously stored chromosome has the fitness14, with distance 335, This is one of the objectives to apply the evaluation after each application of genetic operator. It has been proposed in this study and here it is proved also.

Table 6.18 shows the current fittest chromosome.

Table 6.18 Fittest chromosome

FittestCromosome	8	9	2	8	2	8	6	1	1	8	**313**

6.1.6 Termination

There may be many termination criteria, here for simplicity four generations have been considered. After first generation, a chromosome of distance 313 is found. All the remaining three generation details are given here. For simplicity, status of chromosome is presented here after the evaluation.

After Second Generation

Evaluation chromosomes after crossover

Table 6.19 Crossovered chromosomes after evaluation

Chromosome1	8	6	1	4	1	2	4	6	6	7	**11**
Chromosome2	4	9	8	5	2	4	9	1	1	9	**12**
Chromosome3	4	4	9	5	6	6	5	9	10	5	**12**
Chromosome4	6	9	2	8	2	4	1	5	1	8	**12**
Chromosome5	4	10	3	10	4	6	5	9	4	3	**11**
Chromosome6	8	6	1	4	1	8	6	1	6	7	**11**
Chromosome7	8	10	3	10	4	6	6	7	4	3	**11**
Chromosome8	4	10	8	1	6	6	5	9	6	7	**11**
Chromosome9	6	6	1	4	1	4	1	5	6	7	**12**
Chromosome10	8	9	2	8	2	8	6	1	1	8	**14**

Chromosome-10 which has the fitness 14 and the previously stored chromosome has the fitness14 also so this new chromosome will replace the previous one. Since both the previous and new chromosome have the same fitness value and same distance, so replacement does not make any difference but here it is replaced replace because of the maintenance of new set of value.

Evaluation after mutation

Table 6.20 Mutated chromosomes after evaluation

Chromosome1	8	4	1	6	1	2	4	6	6	7	**12**
Chromosome2	4	9	5	8	2	4	9	1	1	9	**12**
Chromosome3	4	4	9	5	6	5	10	9	5	6	**14**
Chromosome4	6	1	4	2	8	2	9	5	1	8	**13**
Chromosome5	10	3	10	4	4	6	5	9	4	3	**10**
Chromosome6	8	6	1	4	8	1	6	1	6	7	**11**
Chromosome7	8	10	3	10	4	6	6	7	3	4	**10**
Chromosome8	4	10	8	1	6	9	5	6	6	7	**12**
Chromosome9	6	4	1	6	1	4	1	5	6	7	**13**
Chromosome10	8	9	2	8	2	8	6	1	1	8	**14**

Here chromosome-3 (distance 377) and chromosome-10 (distance 313) both are completely fit, but the minimum distance calculated for chromosome-10 is 313. Further previously stored chromosome has the distance 313. So replacement does not make any difference but here it is replaced because of the maintenance of new set of value.

Table 6.21 Fittest chromosome

FittestCromosome	8	9	2	8	2	8	6	1	1	8	**313**

After Third Generation

Evaluation after crossover

Table 6.22 Crossovered chromosomes after evaluation

Chromosome1	8	10	3	2	8	8	9	5	1	8	**11**
Chromosome2	6	1	4	8	2	6	6	1	1	8	**13**
Chromosome3	8	9	2	10	4	2	6	7	3	4	**13**
Chromosome4	4	9	5	1	6	6	5	6	6	7	**11**
Chromosome5	4	10	8	10	4	4	6	7	3	4	**14**
Chromosome6	8	10	3	8	2	9	9	1	1	9	**11**
Chromosome7	6	4	1	2	8	9	9	5	1	8	**11**
Chromosome8	6	1	4	1	6	4	5	6	6	7	**12**
Chromosome9	4	10	8	6	1	2	1	5	6	7	**12**
Chromosome10	6	4	1	6	1	4	1	5	6	7	**13**

Here chromosome-5 (distance 390) is the completely fit, but its distance is greater than previously stored chromosome (distance 313), so there will be no replacement of the chromosome and the fittest chromosome will be as it is Table 6.21

Evaluation after mutation

Table 6.23 Mutated chromosomes after evaluation

Chromosome1	8	10	3	9	8	8	2	5	1	8	**11**
Chromosome2	6	1	4	1	6	6	2	8	1	8	**11**
Chromosome3	8	9	2	10	4	2	6	7	3	4	**13**
Chromosome4	4	9	5	1	6	6	7	6	6	5	**10**
Chromosome5	4	10	8	3	7	6	4	4	10	4	**11**
Chromosome6	8	10	3	8	2	9	9	9	1	1	**11**
Chromosome7	2	1	4	6	8	9	9	5	1	8	**12**
Chromosome8	6	1	5	4	6	1	4	6	6	7	**11**
Chromosome9	4	10	5	1	2	1	6	8	6	7	**11**
Chromosome10	6	4	1	6	1	1	4	5	6	7	**13**

As it is seen from Table 6.23, there is no fit chromosome, so there will be no replacement with the previously stored chromosome.

After Fourth Generation

Evaluation after crossover

Table 6.24 Crossovered chromosomes after evaluation

Chromosome1	8	9	5	1	8	6	7	6	6	8	**11**
Chromosome2	4	10	8	3	6	6	4	4	10	5	**11**
Chromosome3	4	10	3	9	7	8	2	5	1	4	**11**
Chromosome4	8	1	5	4	4	1	4	6	6	4	**11**
Chromosome5	6	10	5	1	6	1	6	8	6	7	**11**
Chromosome6	4	9	2	10	2	2	6	7	3	7	**12**
Chromosome7	8	1	5	4	4	1	4	6	6	4	**11**
Chromosome8	6	1	4	1	6	6	2	8	1	7	**11**
Chromosome9	6	9	2	10	6	2	6	7	3	8	**12**
Chromosome10	4	10	5	1	2	1	6	8	6	7	**11**

Again as from Table 6.24, there is no fit chromosome, so there will be no replacement with the previously stored chromosome.

Evaluation after mutation

Table 6.25 Mutated chromosomes after evaluation

Chromosome1	8	9	5	1	6	8	7	6	6	8	**11**
Chromosome2	4	10	8	10	4	4	6	6	3	5	**12**
Chromosome3	4	10	3	9	4	1	5	2	8	7	**11**
Chromosome4	4	6	6	4	1	4	4	5	1	8	**12**
Chromosome5	6	6	1	6	1	5	10	8	6	7	**10**
Chromosome6	4	9	2	10	2	6	2	7	3	7	**11**
Chromosome7	8	1	5	4	1	4	4	6	6	4	**12**
Chromosome8	1	8	2	6	6	1	4	1	6	7	**12**
Chromosome9	8	3	7	6	2	6	10	2	9	6	**11**
Chromosome10	4	10	6	1	2	1	5	8	6	7	**10**

Again from Table 6.25, there is no fit chromosome, so there will be no replacement with the previously stored chromosome.

So after four generation the fittest chromosome has the minimum distance 313 to connect 10 different locations of the Figure 6.1.

6.2 Experimental Results

There are several factors which affects the result of network design problem using genetic algorithm. These factors can be broadly classified in to following categories:

1. Genetic Algorithm Operators and Methods
2. Types of network
3. Constraints imposed by the requirement of the network

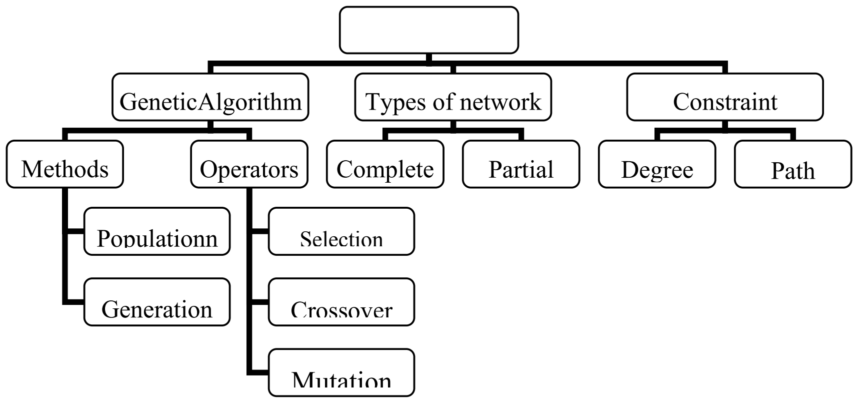

Figure 6.2 (a) Factors affects the performance of Network Design

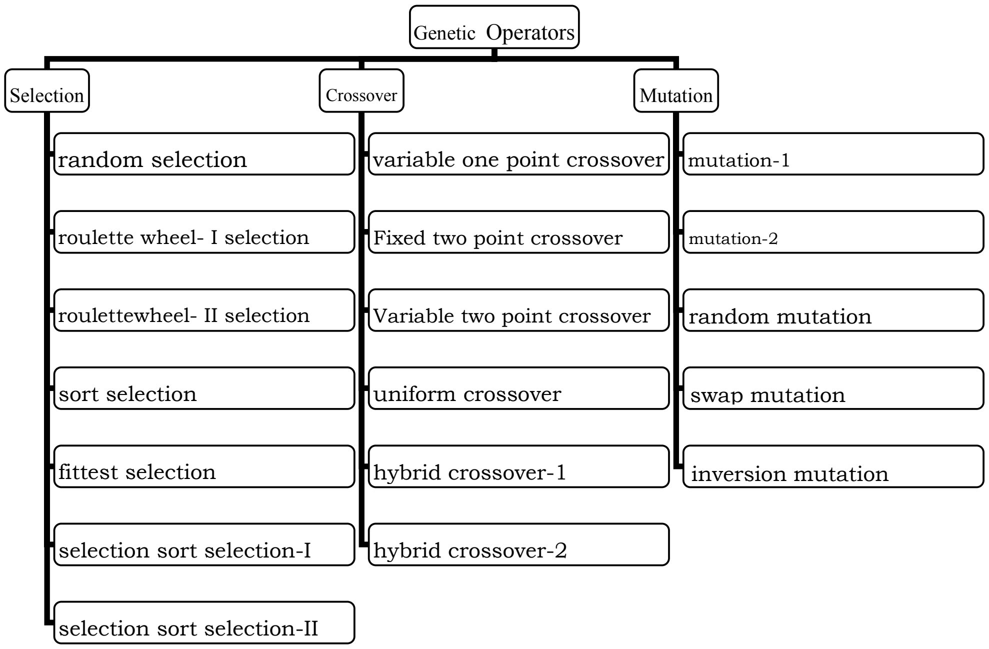

Figure 6.2 (b) GA Factors that affects the performance of Network Design

Networks

In this research work, 15 networks have been considered of the size 10,20,30,40,50,60,70,80,90,100,200,300,400,500,600,700,800,900 and 1000. All these networks have been represented with the help adjacency matrix. Each network has degree constraint table. Each degree constraint table represents the degree of each node of that table.

In Figure 6.1(a) it is shown that there are there are three main factors which affect the performance of network design problem. Further these factors are classified at specific level shown in Figure 6.1(b).

In this experiment 15 networks have been considered with **546** different cases. These parameters are basically from genetic operators. The considered parameters are Selection, Crossover, Mutation, Population size and Number of generation.

First of all experiment is made for a small size network upto100 nodes, dividing into three groups of network size 10, 20, 60 with the variation of population and generation for the different crossover function.

6.2.1 Experiment based on crossover operator with generation and population variation in genetic algorithm for small network design problem

Three different network of size 10, 60 and 100 have been used. The experiment is done in MATLAB R2008a version 7.6.0.324. [56, 57]The entire crossover is experimented with various sizes of network and population –generation combination. In this case following parameters have been considered:

Population size	: 10 to 100
No of Generations	: 10 to 100
Selection	: Roulette Wheel
Mutation	: Mutation I

Following tables and figures display the result:

TABLE -6.26

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-

NETWORK SIZE -10

Population	generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
10	100	307	297	315
20	100	257	302	277
30	100	286	260	252
40	100	286	308	276
50	100	269	274	233
60	100	251	221	263
70	100	247	286	251
80	100	271	240	278
90	100	262	254	245
100	100	231	268	207
200	100	237	245	254
300	100	242	258	252
400	100	248	229	226
500	100	249	225	195
1000	100	210	197	237

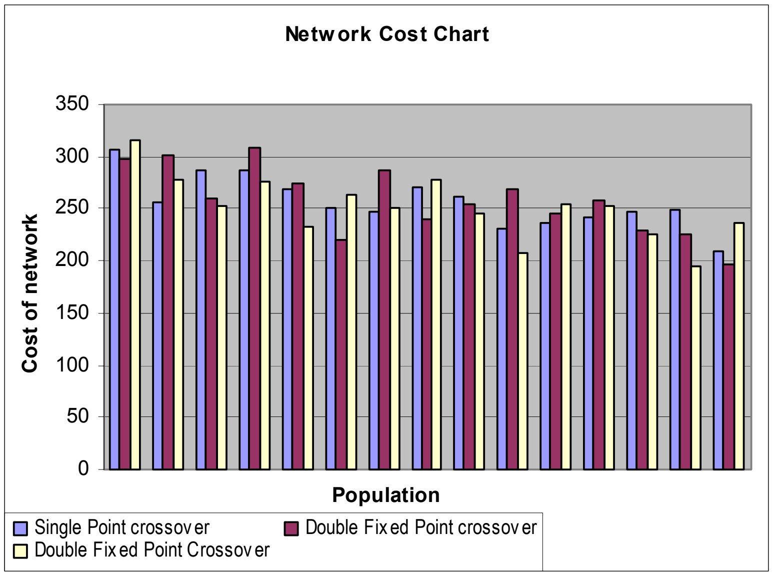

Figure 6.3 (a)

TABLE -6.27

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-NETWORK SIZE -10

Population	generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
100	10	317	298	291
100	20	248	265	263
100	30	249	280	261
100	40	256	251	284
100	50	255	245	271
100	60	222	260	267
100	70	220	223	258
100	80	265	229	225
100	90	242	274	228
100	100	241	226	255
100	200	237	246	262
100	300	234	266	209
100	400	250	253	241
100	500	248	235	270
100	600	232	255	233
100	700	208	248	224
100	800	240	251	238
100	900	265	213	240
100	1000	254	272	256

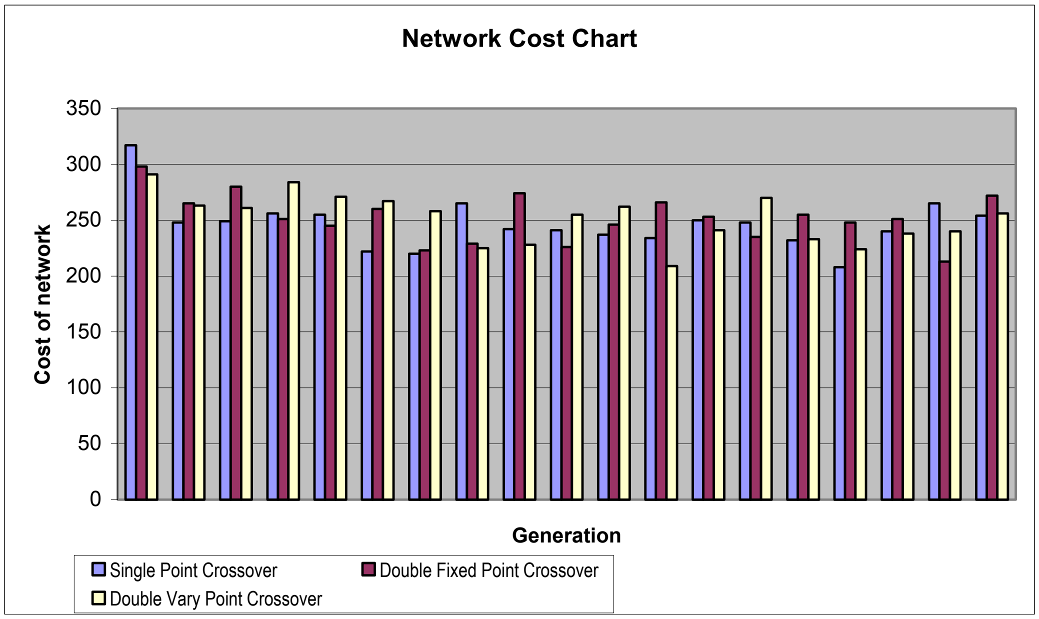

Figure 6.3 (b)

TABLE -6.28

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-NETWORK SIZE -60

Population	generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
20	100	0	2233	0
40	100	2457	2425	0
60	100	0	2285	2211
80	100	2246	2253	2166
100	100	2169	2166	2273
200	100	2161	2091	2142
300	100	2014	2084	2010
400	100	2053	2130	1933
500	100	2047	2066	1954
600	100	2031	2177	2065
700	100	2041	2047	2040
800	100	2013	1905	1983
900	100	2004	1983	2105
1000	100	1962	2032	2041

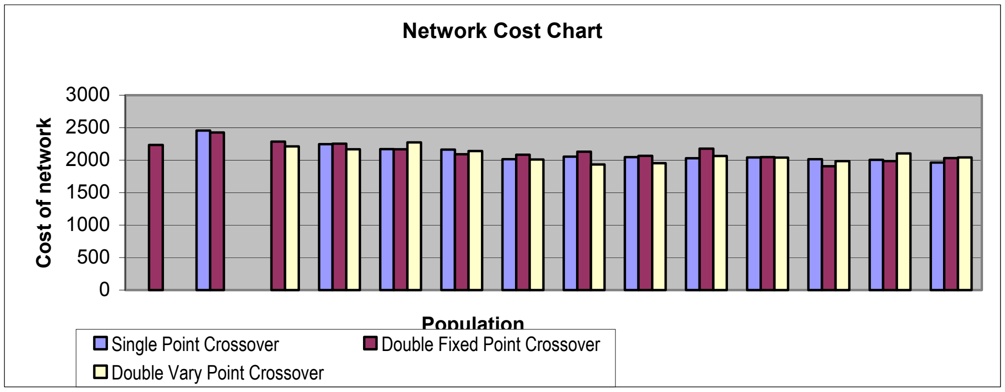

Figure 6.4 (a)

TABLE -6.29

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-NETWORK SIZE -60

Population	generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
100	10	0	0	2161
100	20	2133	2364	2386
100	30	2179	2212	2190
100	40	2189	2178	2170
100	50	2203	2201	2284
100	60	2061	2074	2187
100	70	2276	2216	2209
100	80	2248	2256	2151
100	90	2084	2013	2059
100	100	2254	2145	2092
100	200	2226	2218	2184
100	300	2081	2111	2234
100	400	2212	2152	2381
100	500	2076	2083	2245
100	600	2228	2151	2121
100	700	2140	2106	2139
100	800	2123	2180	2107
100	900	2134	2154	2144
100	1000	0	2238	2064

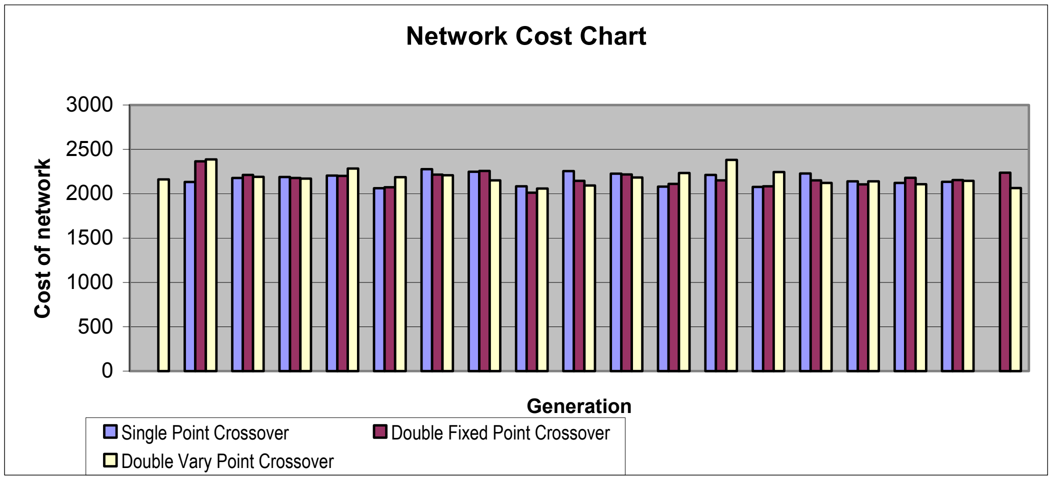

Figure 6.4 (b)

TABLE -6.30

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-NETWORK SIZE -100

Population	generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
100	10	0	0	0
100	20	0	0	0
100	30	0	0	0
100	40	0	3789	3256
100	50	0	0	0
100	60	0	0	0
100	70	0	0	0
100	80	0	0	0
100	90	3883	0	0
100	100	0	0	0
100	200	0	0	0
100	300	0	0	0
100	400	0	0	3456
100	500	0	0	0
100	600	0	0	0
100	700	0	0	0
100	800	4356	0	4123
100	900	0	0	0
100	1000	0	3245	0

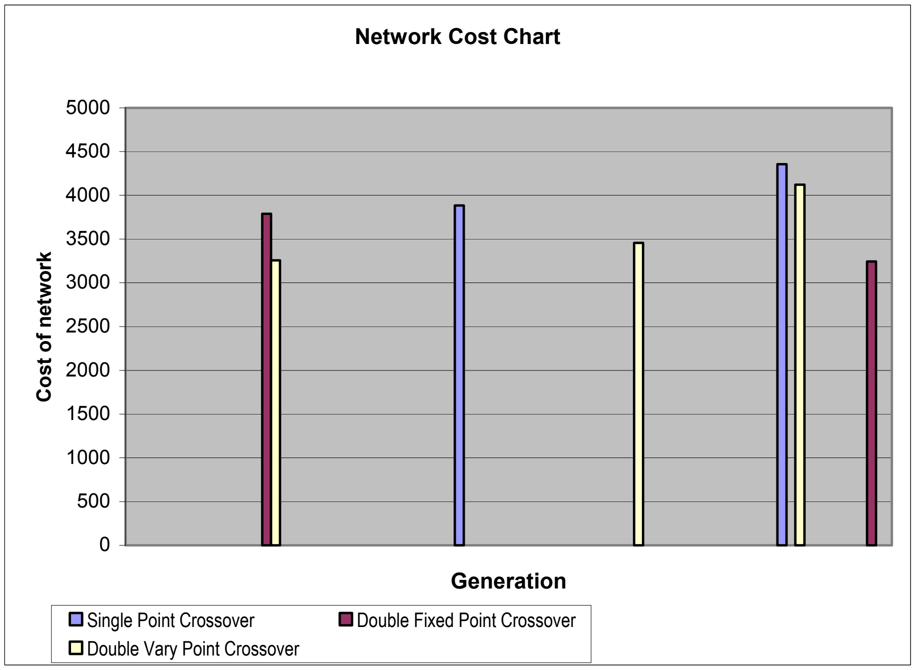

Figure 6.5 (a)

TABLE -6.31

MINIMUM COST OF NETWORK FOR VARIOUS CROSSOVER OPERATORS-NETWORK SIZE -100

Population	Generation	Single Point crossover	Double_ fixed point crossover	Double vary-point crossover
50	100	0	0	0
100	100	0	0	3768
200	100	0	0	3478
300	100	3877	3855	3659
400	100	3666	3543	3624
500	100	3767	3874	3693
600	100	4013	3669	3547
700	100	3886	3335	3699
800	100	3692	3998	3678
900	100	3673	3321	3378
1000	100	3819	3330	3221

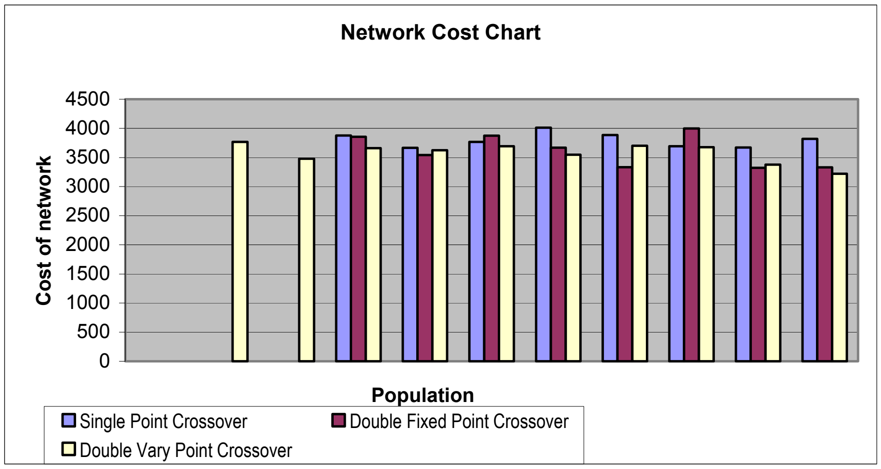

Figure 6.5 (b)

6.2.2 Experiment based on Selection Operator for small to large size network

In this case following parameters have been considered:

Population size	: 100
No of Generations	: 100
Crossover Methods	: Fixed two point crossover
Mutation	: Random mutation

Table 6.32 Experimental Result based on different Selection function

Network Size	Random Selection	Roulette wheel selection I	Roulette wheel selection II	Sort Selection	Fittest Selection	Selection Sort-selectionI	Selection Sort-selectionII
10	301	219	232	253	239	303	296
20	703	587	654	601	633	657	788
40	1577	1333	1381	1351	1235	1330	0
60	2353	2039	2231	2046	2186	2236	0
80	3014	3054	2844	2964	2841	2830	0
100	3810	3533	3648	3500	3502	3521	3795
200	8033	7358	7462	7580	7351	8171	0
300	11928	11175	11172	11571	10978	12013	11627
400	15838	15015	15485	15234	14923	0	0
500	20402	19503	19501	19247	19158	0	0
600	0	23690	23486	23532	23191	24014	0
700	27929	27145	27377	26874	26712	0	0
800	0	31338	30633	30462	31187	32049	0
900	35423	35386	35519	35314	35044	0	35933
1000	40217	39660	39330	38477	35889	0	0

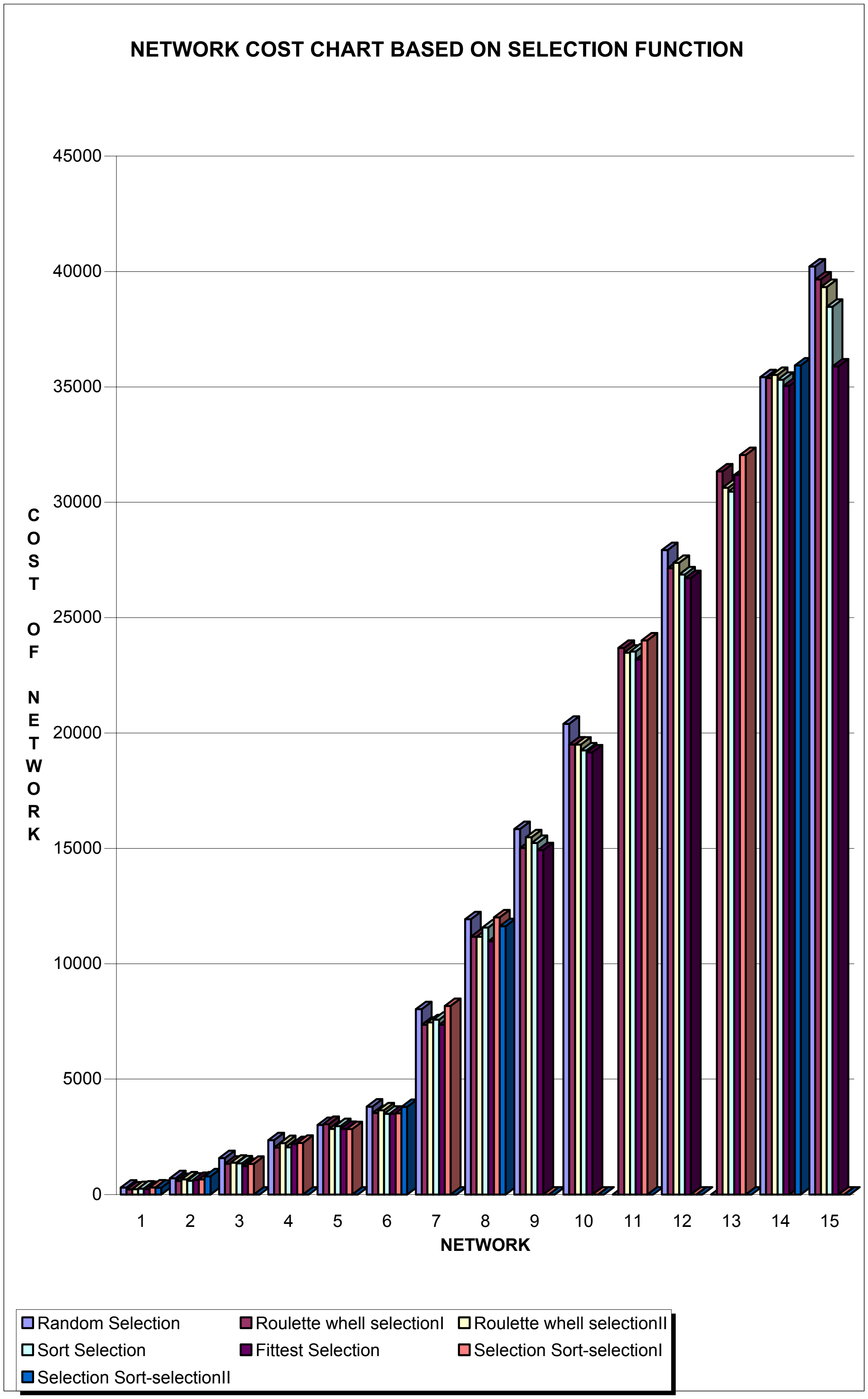
NETWORK COST CHART BASED ON SELECTION FUNCTION
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
COST OF NETWORK
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
NETWORK
Random Selection
Roulette whell selectionI
Roulette whell selectionII
Sort Selection
Fittest Selection
Selection Sort-selectionI
Selection Sort-selectionII

Figure 6.6

6.2.3 Experiment based on Crossover Operator for small to large size network

In this case following parameters have been considered:

Population size	: 100
No of Generations	: 100
Selection	: Roulette Wheel Selection
Mutation	: Random mutation

Table 6.33 Experimental Result based on different Crossover function

Network Size	**Variable two point crossover**	**Fixed two point crossover**	**One point crossover**	**Uniform Crossover**	**Hybrid Crossover**	**Hybrid CrossoverII**
10	248	227	259	248	288	268
20	607	638	664	626	644	639
40	1358	1376	1333	1305	1324	1353
60	2061	2201	2070	2098	2145	2196
80	2874	3030	3131	2987	2828	3111
100	3615	3480	3291	3501	3524	3571
200	7569	7548	7657	7586	7629	7375
300	11609	11184	11532	11512	11222	11252
400	15363	15056	15350	15612	15342	15617
500	19275	19461	19044	18992	19418	19116
600	23778	23269	23602	22855	23918	23492
700	27317	26711	0	27245	27404	27340
800	31687	0	31624	30899	32083	31417
900	35107	0	0	35959	34856	34694
1000	40673	39228	39723	39945	40127	39896

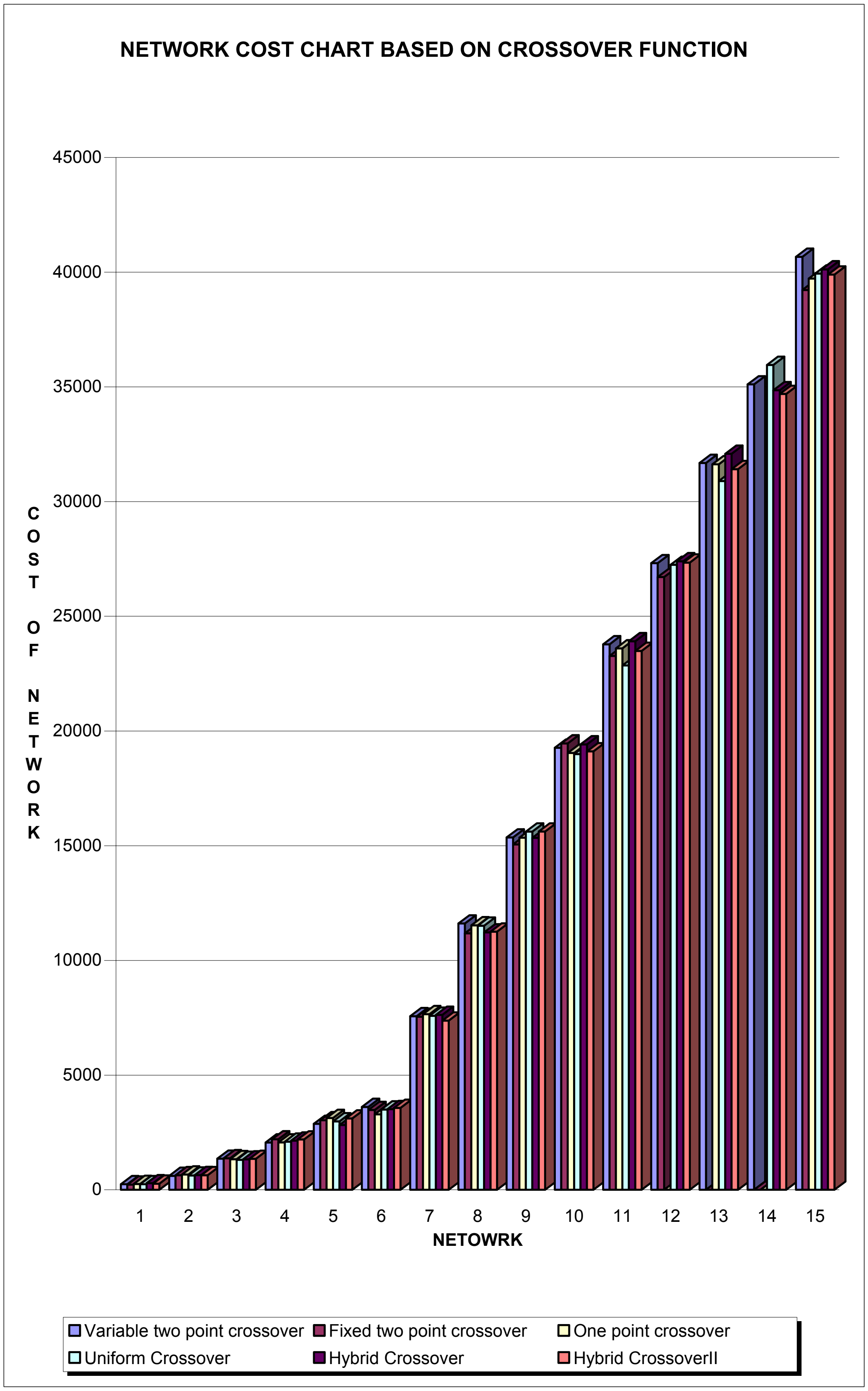
NETWORK COST CHART BASED ON CROSSOVER FUNCTION
COST OF NETWORK
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
NETOWRK
Variable two point crossover
Fixed two point crossover
One point crossover
Uniform Crossover
Hybrid Crossover
Hybrid CrossoverII

Figure 6.7

6.2.4 Experiment based on Mutation Operator for small to large size network

In this case following parameters have been considered:

Population size	: 100
No of Generations	: 100
Selection	: Roulette Wheel Selection
Crossover	: Uniform

Table 6.34 Experimental Result based on different Mutation function

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

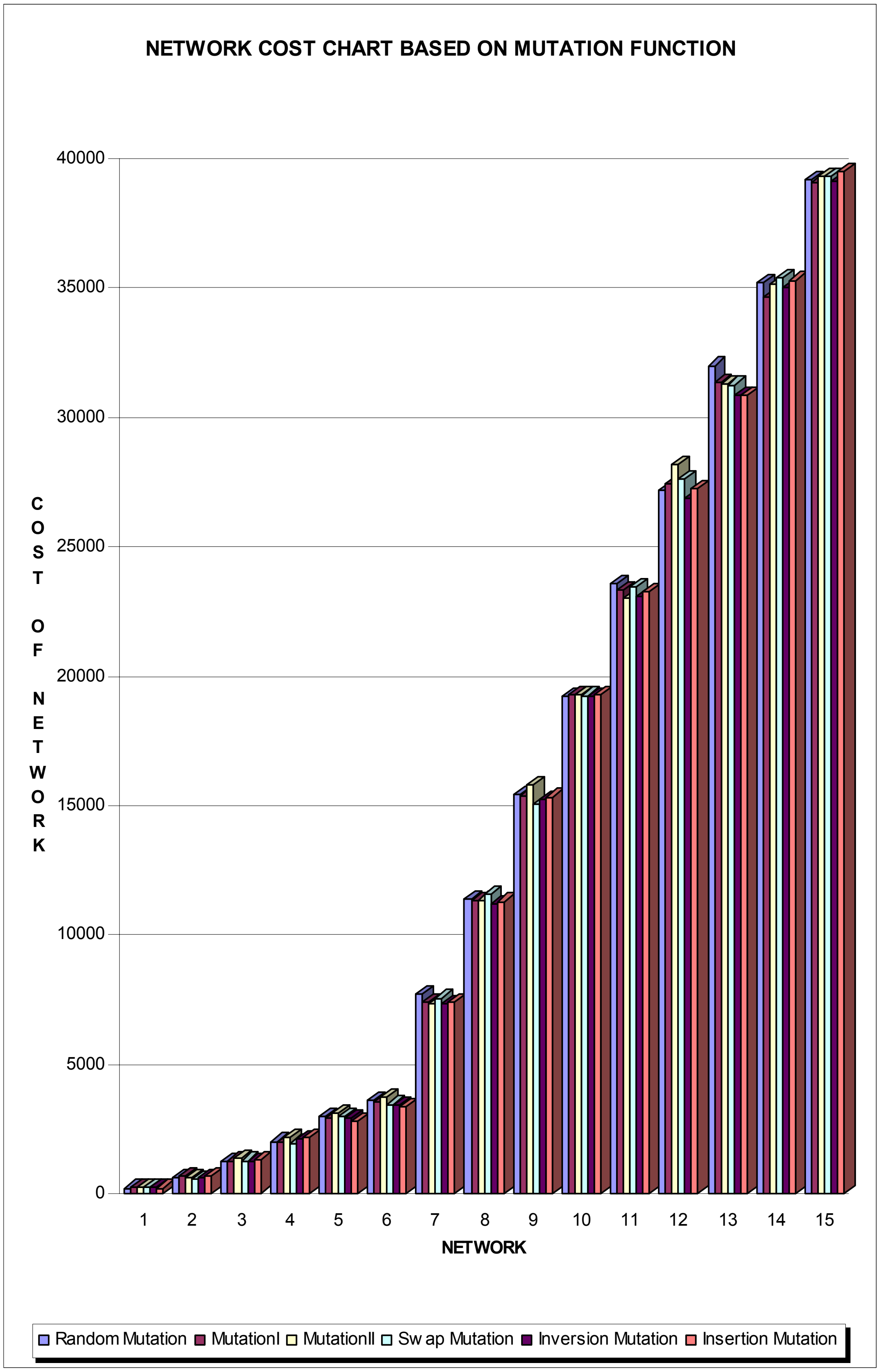
NETWORK COST CHART BASED ON MUTATION FUNCTION
40000
35000
30000
25000
20000
15000
10000
5000
0
COST OF NETWORK
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
NETWORK
Random Mutation
MutationI
MutationII
Sw ap Mutation
Inversion Mutation
Insertion Mutation

Figure 6.8

6.3 Experimental Code developed in MATLAB, Version7.6.0.324(R2008a)

Main Program

```
disp('*************************************************')
;
disp('BACKBONE NETWORK DESIGN PROGRAM USING GENETIC
ALGORITHM ');
disp('  WRITTEN BY MR. ANAND KUMAR Ph.D Scholar
Registration N0. 3893)');
disp('DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF SCIENCE,
SAURASHTRA UNIVERSITY,RAJKOT INDIA');
disp('    Not to be used without permission of Anand
Kumar, Reproduction is not permitted.');

disp('*************************************************'
);
clear all;

% INITIALISATION OF PARENT POPULATION-(GENERATE
CHROMOSOME)

no_node = input ('ENTER THE NUMBER OF NODE   :');
no_chromos = input ('ENTER THE NUMBER OF CHROMOSOME (even
number only)  :');
chromosomes = round(rand(no_chromos,no_node)* no_node);
for i=1:no_chromos
   for j=1:no_node
        if(chromosomes(i,j) == 0)
            chromosomes(i,j)=1;
        end
    end
end
```

```
%disp('randomly generated chromosomes are   ');
%disp(chromosomes);

%DISTANCE MATRIX%
%distance matrix generation is previously generate and
%stored such that all the diagonals are zero.
%here fixed .mat file is used for the matrix size
%10,20,26,40......100,150,200......1000.

size_matrix = input('ENTER THE SIZE OF MATRIX
10,20,26,40,60,80,100,150,200,250......1000     :','s');

load (size_matrix);
dist_matrx = d;
%disp('DISTANCE AMONG NODES');
%disp(dist_matrx);
%pause;

%EVALUATE CHROMOSOME
%THIS FUNCTION CALCULATES FITNESS FOR cycles
      chromosomes=cycle_calculatetry(chromosomes);
%disp('CHROMOSOMES AFTER CYCLE FITNESS');
%disp(chromosomes);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%THIS FUNCTION CALCULATES FITNESS FOR path constraint

chromosomes=path_constraint(chromosomes,dist_matrx);
%disp('CHROMOSOMES AFTER PATH CONSTRAINT FITNESS');
%disp(chromosomes);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

       %THIS FUNCTION CALCULATES FITNESS FOR SELF LOOP
       chromosomes=selfloop_calculate6(chromosomes);
       %disp('CHROMOSOMES AFTER SELF LOOP FITNESS');
       %disp(chromosomes);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%THIS FUNCTION CALCULATES FITNESS FOR ISOLATED EDGE OR
ISOLATED TREE
chromosomes=isolate_calculate6(chromosomes);
%disp('CHROMOSOMES AFTER ISOLATION FITNESS');
%disp(chromosomes);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%THIS FUNCTION CALCULATES FITNESS FOR DEGREE CONSTRAINT
FOR
%EACH NODE
for i=1:10
degree(i) = input('degree?');
end
degree_matrix = input('ENTER THE OF DEGREE constraint of
MATRIX EXAMPLE ddegree10,
ddegree20,ddegree26.........sequence to size    ;','s');
load (degree_matrix);
disp('DEGREE FOR EACH NODE');
disp(degree);
pause;
chromosomes=degree_constraint_calculate11(chromosomes,deg
ree);
disp('CHROMOSOMES AFTER DEGREE CONSTRAINT FITNESS');
disp(chromosomes);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%calculation of distance for each randomly generated
chromosomes%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
total_distance =
distance_calculate6(chromosomes,dist_matrx);
disp('DISTANCE FOR EACH CHROMOSOME');
%disp(total_distance);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIND THE FITTEST CHROMOSOME WITH MINIMUM DISANCE%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fittest_chromosome =
fittest_calculate_pc(chromosomes,total_distance);
disp('FITTEST CHROMOSOME WITH DISTANCE');
disp(fittest_chromosome);
PAUSE = input('PRESS 1 TO CONTINUE....');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%NEW GENERATION FOR CHILD POPULATION%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%% selection%%%%%%%%%%%%%%%%%%%%
no_of_generation = input('ENTER THE NUMBER OF REQUIRED
GENERATION');
for LOOP = 1 :  no_of_generation

disp('#########################GENERATION#############');
disp(LOOP);
pause;

%new_chromosome = simple_selection9(chromosomes);
%new_chromosome = selection11(chromosomes);
%new_chromosome = selection12(chromosomes);
%new_chromosome = selection13(chromosomes);
new_chromosome = selection14(chromosomes);
%new_chromosome = selection15(chromosomes);
%new_chromosome = selection16(chromosomes);
%disp('NEW CHROMOSOME AFTER SELECTION');
%disp(new_chromosome);

%%%%%%%%%%%%%%%%% crossover%%%%%%%%%%%%%%%%%%%%%%%%

%new_chromosome = hybrid_crossoverII(new_chromosome);
%new_chromosome = hybrid_crossover(new_chromosome);
%new_chromosome = uniform_crossover(new_chromosome);
```

```
%new_chromosome = one_point_crossover6(new_chromosome);
new_chromosome = two_point_crossover12(new_chromosome);
%new_chromosome = two_point_crossover13(new_chromosome);
%disp('NEW CHROMOSOME AFTER   CROSSOVER');
%disp(new_chromosome);

%%%%%%%%%%%%%%%evaluation%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

new_chromosome=cycle_calculatetry(new_chromosome);
new_chromosome=path_constraint(new_chromosome,dist_matrx)
;
new_chromosome=selfloop_calculate6(new_chromosome);
new_chromosome=isolate_calculate6(new_chromosome);
new_chromosome=degree_constraint_calculate11(new_chromoso
me,degree);
%disp('EVALUATED CHROMOSOME AFTER  CROSSOVER');
%disp(new_chromosome);

%%%%%%%%%%%%%%%finding fittest chromosome%%%%%%%%%%%%%%
%%% first it calculates distance for each chromosome%%%%%
%%%%then if found minimum than previous generation it
%%%%replaces the previous one%%%%%%%%%%%%%%%%%%%%%%%%%%%


total_distance =
distance_calculate6(new_chromosome,dist_matrx);
%disp('DISTANCE FOR EACH CHROMOSOME');
%disp(total_distance);
%disp('OLD FITTEST CHROMOSOME');
%disp(fittest_chromosome);
fittest_chromosome =
main_fittest_calculate_pc(new_chromosome,total_distance,f
ittest_chromosome);
%disp('NEW FITTEST CHROMOSOME WITH DISTANCE ');
%disp(fittest_chromosome);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%Mutation%%%%%%%%%%%%%%%%%%%

%mutated_chromosome = mutation_insertion(new_chromosome);
%mutated_chromosome = mutation_inversion(new_chromosome);
%mutated_chromosome = mutation_swap(new_chromosome);
%mutated_chromosome = mutation_simple9(new_chromosome);
%mutated_chromosome = mutation_13(new_chromosome);
mutated_chromosome = mutation_14(new_chromosome);
%disp('NEW CHROMOSOME AFTER MUTATION ');
%disp(mutated_chromosome);

%%%%%%%%%%%%%%%%evaluation%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mutated_chromosome=cycle_calculatetry(mutated_chromosome);
mutated_chromosome=path_constraint(mutated_chromosome,dist_mat
rx);
mutated_chromosome=selfloop_calculate6(mutated_chromosome);
mutated_chromosome=isolate_calculate6(mutated_chromosome);
mutated_chromosome=degree_constraint_calculate11(mutated_chrom
osome,degree);
 %disp('EVALUATED CHROMOSOME AFTER MUTATION');
%disp(mutated_chromosome);

%%%%%%%%%%%%%%%%finding fittest chromosome%%%%%%%%%%%%%
%%%first it calculates distance for each chromosome%%%%%
%%%then if found minimum than previous generation it
%%%replaces the previous one%%%%%%%%%%%%%%%%%%%%%%%%%


total_distance =
distance_calculate6(mutated_chromosome,dist_matrx);
%disp('DISTANCE FOR EACH CHROMOSOME');
%disp(total_distance);
%disp('OLD FITTEST CHROMOSOME');
%disp(fittest_chromosome);
```

```
fittest_chromosome =
main_fittest_calculate_pc(mutated_chromosome,total_distan
ce,fittest_chromosome);
%disp('NEW FITTEST CHROMOSOME WITH DISTANCE ');
%disp(fittest_chromosome);
chromosomes=mutated_chromosome;
%PAUSE = input('PRESS 1 TO CONTINUE....')
end

disp('FITTEST CHROMOSOME WITH DISTANCE ');
disp(fittest_chromosome);

disp('***********************END********************');
```

SELECTION OPERATORS

```
%******************RANDOM SELECTION*********************
% this function simply select the chromosome on the basis
%of their fitness function.
% if r is out of the range of the no. of chromosome then
it
%finds  the  position  of  the  fittest  chromosome  and
replaces
%with that chromosome.

function [new_chromosomes] =
simple_selection9(chromosomes)
a=size(chromosomes);
row = a(1);
col = a(2)-1;
k=1;
for i=1:row
    r=row*round(rand());
    if((r == 0) || (r > row))

        for l=1:row
            if(chromosomes(l,a(2)) == col+3)
```

```
                r=1;
            end

         end
    end
    if(r == 0)
        r=1;
    end
    for j=1:col
        new_chromosomes(k,j) = chromosomes(r,j);
    end
    k=k+1;
end
disp(new_chromosomes);

return;
end
```

```
%******ROULETTE WHEEL SELECTION I***********************
% BASED ON SIMPLE ROULETTE WHELL ALGORITHM%%

function [new_chromosomes] = selection11(chromosomes)
a=size(chromosomes);

%disp(chromosomes);

row = a(1);
col = a(2)-1;
r=row*round(rand());
s(row,1)=0;
temp=0;
for i=1:row
    temp=temp+chromosomes(i,a(2));
    s(i)=temp;
end
disp('DISP');
disp(s);
```

```
 k=1;
 for i=1:row
     r=rand*temp;
     disp('r');
     disp(r);
     for j=1:row
         if(r<s(j))
             for t=1:col
                 new_chromosomes(k,t)=chromosomes(j,t);
             end
             k=k+1;
             break;
         end
     end

 end
%disp('***NEW SELECTED CHROMOSOMES******************');
%disp(new_chromosomes);
%disp('*********************************************');

return;
end
```

____

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%ROULETTE_WHEEL_SELECTIONII %%%%%%%%%%%

function [new_chromosomes] = selection12(chromosomes)
a=size(chromosomes);
disp('********inside selection11 Function************');
disp(chromosomes);
disp('*********************************************');

row = a(1);
col = a(2)-1;
r=row*round(rand());
s(row,1)=0;
temp=0;
```

```
for i=1:row
    temp=temp+chromosomes(i,a(2));
end
avg=temp/row;
disp('avg');
disp(avg);
disp('**temp**');
for i=1:row
    temp=(chromosomes(i,a(2))/avg);
    disp(temp);
end
temp=0;
for i=1:row
    temp=temp+(chromosomes(i,a(2))/avg);
    s(i)=temp;
end
temp=s(i);
disp('s');
disp(s);
 k=1;
 for i=1:row
     r=rand*temp;
     disp('r');
     disp(r);
     for j=1:row
         if(r<s(j))
             for t=1:col
                 new_chromosomes(k,t)=chromosomes(j,t);
             end
             k=k+1;
             break;
         end
     end
end
disp('***NEW SELECTED CHROMOSOMES********************');
disp(new_chromosomes);
disp('**********************************************');
return;
```

```
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%% SELECTION BASED ON SORTING %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [new_chromosomes] = selection13(chromosomes)
a=size(chromosomes);

%disp(chromosomes);
row = a(1);
col = a(2)-1;
k=1;
 t=3;
while(k<=row)
   for i=1:row

       if(chromosomes(i,a(2)) == (a(2)+t))
          for j=1:col
           new_chromosomes(k,j)=chromosomes(i,j);
          end
          k=k+1;
       end
   end
   t=t-1;

end
disp('***************NEW SELECTEDD CHROMOSOMES********');
%disp(new_chromosomes);
%disp('*********************************************');
pause;
return;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%SELECTION METHOD  to select only fittest chromosome of A
%FIXED LEVEL OF FITNESS VALUE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [new_chromosomes] = selection14(chromosomes)
a=size(chromosomes);

disp(chromosomes);
%disp('********************************************');

row = a(1);
col = a(2)-1;
k=1;
 t=3;
 n=1;
while(k<=row)
   for i=1:row

       if(chromosomes(i,a(2)) == (a(2)+t))
          for j=1:col
           new_chromosomes(n,j)=chromosomes(i,j);
          end
          n=n+1;
       end
   end
   if(n>row)
      break;
   end
  k=k+1;
   t=t-1;
   if(t< (-8))
       t=3;
   end
end
disp('****NEW SELECTED CHROMOSOMES*******************');
disp(new_chromosomes);
disp('********************************************');
```

```
pause;
return;
end
```

____

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%SELECTION METHOD SELECTION SORT based on two random
%position and select the best one this procedure is
%repeated times

function [new_chromosomes] = selection15(chromosomes)
a=size(chromosomes);

disp(chromosomes);
disp('**********************************************');
pause;
row = a(1);
col = a(2)-1;
k=1;
 t=2;
   for i=1:row

            p  = round(rand()* (a(2)-1));
            q  =  round(rand()* (a(2)-1));
             if(p == 0)
                 p=1;
             end

             if(q == 0)
                 q=1;
             end
             disp('P');
             disp(p);
             disp('Q');
             disp(q);
       if(p > row)
           p = row;
       end
       if(q > row)
           q = row;
       end

       if(chromosomes(p,a(2)) >= chromosomes(q,a(2)))
          for j=1:col
           new_chromosomes(k,j)=chromosomes(p,j);
          end
          k=k+1;
       else
           for j=1:col
           new_chromosomes(k,j)=chromosomes(q,j);
          end
          k=k+1;
       end
   end
disp('*NEW SELECTED CHROMOSOMES***********************');
disp(new_chromosomes);
disp('************************************************');
return;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%SELECTION METHOD SELECTION SORT based on two random
%position and select the smallest one
%This procedure is repeated n times

function [new_chromosomes] = selection16(chromosomes)
a=size(chromosomes);

disp(chromosomes);
disp('************************************************');
row = a(1);
col = a(2)-1;
k=1;
 t=2;
```

```
   for i=1:row

            p  = round(rand()* (a(2)-1));
            q =  round(rand()* (a(2)-1));
             if(p == 0)
                 p=1;
             end

             if(q == 0)
                 q=1;
             end
       if(p > row)
           p = row;
       end
       if(q > row)
           q = row;
       end
       if(chromosomes(p,a(2)) < chromosomes(q,a(2)))
          for j=1:col
           new_chromosomes(k,j)=chromosomes(p,j);
          end
          k=k+1;
       else
           for j=1:col
           new_chromosomes(k,j)=chromosomes(q,j);
          end
          k=k+1;
       end

   end

disp('****NEW SELECTED CHROMOSOMES*******************');
disp(new_chromosomes);
disp('************************************************');
return;
end
```

CROSSOVER OPERATORS

```
%****************HYBRID CROSSOVER-II*****************%


function [chromosomes] = hybrid_crossoverII(chromosomes)
disp(chromosomes);

a=size(chromosomes);
row = a(1);
col = a(2);
t=1;i=1;
p=ceil(col.*rand());
q=ceil(col.*rand());
r=ceil(col.*rand());
s=ceil(col.*rand());

pos = [p,q,r,s];
disp (pos);
sorted_pos =sort(pos);
disp('SORTED POSITION');
disp(sorted_pos);
p=sorted_pos(1);
q=sorted_pos(2);
r=sorted_pos(3);
s=sorted_pos(4);
disp(p);
disp(q);
disp(r);
disp(s);
pause;
disp('begin point');
disp(p);
if(p==0)
        p=1;
end
```

```
if(q==0)
       q=1;
end
disp('end point');
disp(q);
if(p>q)
   p1=q;
   p2=p;
else
   p1=p;
   p2=q;
end
while( i<row)
    for j=p:q
        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:s
        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
    i=i+3;


p=ceil(col.*rand());
q=ceil(col.*rand());
r=ceil(col.*rand());
s=ceil(col.*rand());

pos = [p,q,r,s];
disp (pos);
sorted_pos =sort(pos);
```

```
disp('SORTED POSITION');
disp(sorted_pos);
p=sorted_pos(1);
q=sorted_pos(2);
r=sorted_pos(3);
s=sorted_pos(4);
disp(p);
disp(q);
disp(r);
disp(s);
pause;
end

%disp('%begin%%%%%%% After uniform crossover function ');
%disp(chromosomes);
%disp('%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%');

return;
end
```

```
%***************HYBRID CROSSOVER*************************

function [chromosomes] = hybrid_crossover(chromosomes)
disp(chromosomes);
a=size(chromosomes);
row = a(1);
col = a(2);
t=1;i=1;
p=ceil(col.*rand());
q=ceil(col.*rand());
r=ceil(col.*rand());
s=ceil(col.*rand());
pos = [p,q,r,s];
disp (pos);
sorted_pos =sort(pos);
disp('SORTED POSITION');
disp(sorted_pos);
```

```
p=sorted_pos(1);
q=sorted_pos(2);
r=sorted_pos(3);
s=sorted_pos(4);
while( i<row)
    for j=p:q
        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:s
        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
    i=i+3;
end
disp(chromosomes);
disp('%end%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%');
 return;
end
```

```
% ***********UNIFORM CROSSOVER**************************

function [chromosomes] = uniform_crossover(chromosomes)

a=size(chromosomes);
row = a(1);
col = a(2);
t=1;i=1;
p=ceil(col.*rand());
q=ceil(col.*rand());
r=ceil(col.*rand());
s=ceil(col.*rand());
```

```
pos = [p,q,r,s];
disp (pos);
sorted_pos =sort(pos);
disp('SORTED POSITION');
disp(sorted_pos);
p=sorted_pos(1);
q=sorted_pos(2);
r=sorted_pos(3);
s=sorted_pos(4);
while( i<row)
    for j=p:q
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end

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

end
return;
end
```

```
%***VARIABLE ONE POINT CROSSOVER***********************

function [chromosomes] =
one_point_crossover6(chromosomes)
a=size(chromosomes);
row = a(1);
col = a(2);
t=1;i=1;
while( i<row)
```

```
  for j=1:t
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end
    i=i+2;
    t=t+1;
    if(t>col)
        t=1;
    end
end
return;
end
```

```
%****FIXED TWO POINT CROSSOVER***************************

function [chromosomes] =
two_point_crossover12(chromosomes)
a=size(chromosomes);
row = a(1);
col = a(2);
t=1;i=1;
p=round(rand()*col);
q=round(rand()*col);
disp('begin point');
disp(p);
if(p==0)
        p=1;
end

if(q==0)
        q=1;
 end
disp('end point');
disp(q);
if(p>q)
    p1=q;
    p2=p;
else
    p1=p;
    p2=q;
end
while( i<row)
    for j=p1:p2
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end
    i=i+2;
    t=t+1;
    if(t>col)
        t=1;
    end
end

return;
end
```

```
%*****VARIABLE TWO POINT CROSSOVER**********************

function [chromosomes] =
two_point_crossover13(chromosomes)

a=size(chromosomes);
row = a(1);
col = a(2);
i=1;
 while( i<row)
    p=round(rand()*col);
    if(p==0)
        p=1;
    end
q=round(rand()*col);
 if(q==0)
```

```
        q=1;
 end
 if(p>q)
    p1=q;
    p2=p;
else
    p1=p;
    p2=q;
end
disp('begin point');
disp(p1);
disp('end point');
disp(p2);
    for j=p1:p2
        temp = chromosomes(i,j);
        chromosomes(i,j) = chromosomes((i+1),j);
        chromosomes((i+1),j)=temp;
    end

    i=i+2;

end

return;
end
```

MUTATION OPERATORS

```
%*********MUTATION INSERTION***************************

function [mutated_chromosome] =
mutation_insertion(new_chromosome)
a=size(new_chromosome);

row = a(1);
col = a(2)-1;
```

```
k=1;
for i=1:row

    p=ceil(col.*rand());
    q=ceil(col.*rand());
    pos = [p,q];
    sorted_pos =sort(pos);
    p=sorted_pos(1);
    q=sorted_pos(2);
 disp('random position for swap');
 disp(p);
 disp(q);

        temp =  new_chromosome(i,q);
        if(p ~= q)
            x = q-1;
          while (x >= p+1)

              new_chromosome(i,x+1) =
new_chromosome(i,x);
              x = x-1;

          end
           new_chromosome(i,p+1) = temp;
        end
end

for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end

return;
end
```

```
%*************** MUTATION INVERSION********************

function [mutated_chromosome] =
mutation_inversion(new_chromosome)
a=size(new_chromosome);
row = a(1);
col = a(2)-1;

k=1;
for i=1:row

    p=ceil(col.*rand());
    q=ceil(col.*rand());
    pos = [p,q];
    sorted_pos =sort(pos);
    p=sorted_pos(1);
    q=sorted_pos(2);
 disp('random position for swap');
 disp(p);
 disp(q);

        for x = p : q
         temp = new_chromosome(i,x);
         new_chromosome(i,x) = new_chromosome(i,q);
         new_chromosome(i,q) = temp;
         q = q - 1;
         if (x == q) || (x > q)
             break;
         end
        end
end

for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end
```

```
return;
end
```

```
% ***************MUTATION SWAP************************

function [mutated_chromosome] =
mutation_swap(new_chromosome)
a=size(new_chromosome);
row = a(1);
col = a(2)-1;

k=1;
for i=1:row

    p=ceil(col.*rand());
    q=ceil(col.*rand());
    pos = [p,q];
    sorted_pos =sort(pos);
    p=sorted_pos(1);
    q=sorted_pos(2);
 disp('random position for swap');
 disp(p);
 disp(q);
         temp = new_chromosome(i,p);
         new_chromosome(i,p) = new_chromosome(i,q);
         new_chromosome(i,q) = temp;

end
 for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end
return;
end
```

```
%MUTATIONI
function [mutated_chromosome] =
mutation_simple9(new_chromosome)
a=size(new_chromosome);
row = a(1);

k=1;
for i=1:row
     if(new_chromosome(i,a(2)) ~= (a(2)+3))
         new_chromosome(i,i) = (a(2)-i);
     end
end

for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end

return;
end
```

```
% MUTATION II

function [mutated_chromosome] =
mutation_13(new_chromosome)

a=size(new_chromosome);
row = a(1);
k=1;
for i=1:row
     if(new_chromosome(i,a(2)) ~= (a(2)+2))
         for j=1:a(2)-1
             if(new_chromosome(i,j) == j)
                 if(j == a(2)-1)
                      new_chromosome(i,j) = j-1;
                 else
                   new_chromosome(i,j) = j+1;
```

```
                end

            end
        end

    end
end

for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end
return;
end
```

____

```
%%%%%%%%%%%%%RANDOM_MUTATION%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [mutated_chromosome] =
mutation_14(new_chromosome)
a=size(new_chromosome);
row = a(1);
k=1;
for i=1:row
     if(new_chromosome(i,a(2)) ~= (a(2)+3))

             posi = round(rand()* (a(2)-1));
             val =  round(rand()* (a(2)-1));
             if(posi == 0)
                 posi=1;
             end

             if(val == 0)
                 val=1;
             end

             if((posi == val) && (posi == a(2)-1))
                  new_chromosome(i,posi) = val-1;
             else
                  new_chromosome(i,posi) = val;
             end
    end
end

for i=1:row
    for j=1:a(2)-1
        mutated_chromosome(i,j) = new_chromosome(i,j);
    end
end
return;
end
```

____

FITNESS FUNCTIONS

```
%******************CYCLE CHECK FUNCTION******************
% This function checks the existence of cycle for each
%chromosome. In the case of existence of cycle it
allocate
%0 other wise 1.

function [chromosomes] = cycle_calculatetry(chromosomes)
a=size(chromosomes);

disp('INSIDE cycle');
disp(chromosomes);

  for m = 1 : a(1)
    k=0; t=(-1); b=1;e=5;
    for i = 1:a(2)
        new=0; s=i;
        for j=1:(a(2) + 1)
           if (new == 0)
               check = s;
           else
```

```
     check = chromosomes(m,s);
end
   l=1;
   while(l <=k )
       if (p(l) == check)
          if (new == 0)
              break;
          end
          if (l>=b)
              if (k == (l+1))
                  t=-1;
                  b=k+1;
                  break;
              end
              if (k >(l+1))
                  e=0;
                  break;
              end
          end
          if (l<b)
              t=-1;
              b=k+1;
              break;
          end
       end
    l = l + 1;
   end
    if (e == 0)
        break;
    end
    if (l>k)
        k=k+1;
        p(k) = check;
        t = t + 1;
    end
    if (t == -1)
        break;
    end
```

```
                if (new ~= 0)
                    s = chromosomes(m,s);
                end
                if (new == 0)
                    new = 1;
                end
        end
         if ( e == 0)
             break;
         end
    end
if( e == 0)
    chromosomes(m,(a(2)+1))= 0;
else
    chromosomes(m,(a(2)+1)) =1;
end

  end
disp('INSIDE cycle');
disp(chromosomes);
return;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%THIS FUNCTION CHECKS THE PATH CONSTRAINT FOR EACH
%CHROMOSOME ACCORDING TO AVAILABILTIY OF PATH FROM
%DIST_MATRX IF PATH AVAILABLE FOR EACH GENE OF THE
%CHROMOSOME THEN IT ASSIGNS 1 OTHERWISE IT ASSIGNS 0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [chromosomes] =
path_constraint(chromosomes,dist_matrx)
a=size(chromosomes);

disp(chromosomes);
row=a(1);
col=a(2);
```

```
for i=1:row
    t=0;
    for j=1:col-1
        k=chromosomes(i,j);
        if(dist_matrx(j,k) ~= 0)
            t=t+1;
        else
            if(j==k)
                t = t+1;
            end
            if(j~=k)
               break;
            end
        end
    end
 disp('t');
 disp(t);
    if(t== (col-1))
        chromosomes(i,col) = chromosomes(i,col) + 1;
    end
end
return;
end
```

```
% THIS FUNCTION CALCULATES FITNESS FOR SELF LOOP SUCH
THAT IF COLUMN
% POSITION NOT EQUALS TO ALLELE OF CHROMOSOME THAN IT
ASSIGN 1 AND THEN
% CALCULATE TOTAL NO OF 1 FOR EACH CHROMOSOME
function [chromosomes] = selfloop_calculate6(chromosomes)
s=size(chromosomes);

for i=1:s(1)
    fit = 0;
    for j=1:(s(2)-1)
        if(chromosomes(i,j) ~= j)
            fit = fit + 1;
        end
    end
    chromosomes(i,s(2))= chromosomes(i,s(2))+ fit;
end
return;
end
```

____

```
% THIS FUNCTION CALCULATES FITNESS FOR ISOLATED EDGE OR
%ISOLATED PART OF
% IT  ASSIGNS 1 IF NO ISOLATION OR OTHERWISE 0
% one problem for 10 node network

function [chromosomes] = isolate_calculate6(chromosomes)
s=size(chromosomes);

for i=1:s(1)
    count = 0;
    for j=1:(s(2) -1)
        if(j == chromosomes(i,chromosomes(i,j)))
            count = count + 1;
        end
    end
    if ( count > 2)
       chromosomes(i,s(2)) = chromosomes(i,s(2)) + 0;
    else
        chromosomes(i,s(2)) = chromosomes(i,s(2)) + 1;
    end
 end
return;

end
```

____

```
%THIS FUNCTION DEGREE CONSTRIANT FOR EACH NODE
% DEGREE CONSTRAINT RANGE, DEGREE IS BETWEEN MINIMUM AND
%MAXIMMUM IT ASSIGNS 1 TO THOSE WHO'S DEGREE CONSTRAINT
IS
```

```
%EQUAL TO (2*N-2) SATISFIES OTHERWISE 0
function [chromosomes] =
degree_constraint_calculate11(chromosomes,degree)
s=size(chromosomes);
N=size(degree);
  for i=1:s(1)
    p=0;total_degree=0;
    for j=1:(s(2)-1)

        d=1;
        for k=1:(s(2)-1)
            if(chromosomes(i,k) == j)

                if(chromosomes(i,k) == k)
                    d=d-1;
                end
                if(chromosomes(i,chromosomes(i,k))~= k)
                    d=d+1;
                end
            end
        end
        if((d >=degree(1,j)) && (d<=degree(2,j)))
            p=p+1;
            total_degree = total_degree + d;
        else
            break;
        end

    end
     if((p==(s(2)-1)) && (total_degree == (2*N(2)-2 )))
         chromosomes(i,s(2)) = chromosomes(i,s(2)) + 1;
     end

  end
return;
end
```

```
%THIS FUNCTION CALCULATES TOTAL DISTANCE FOR
EACHCHROMOSOME
function [total_distance] =
distance_calculate6(chromosomes,dist_matrx)
s=size(chromosomes);
 for i=1:s(1)
    sum=0;
    for j=1:(s(2)-1)

        k=chromosomes(i,j);
        if(j ~= chromosomes(i,k))
        sum = sum + dist_matrx(j, (chromosomes(i,j)));
        end

        if(j == chromosomes(i,k))
            if(j > chromosomes(i,j))
             sum = sum + dist_matrx(j,
(chromosomes(i,j)));
            end
        end
      end
    total_distance(i) = sum;
 end
return;
end
```

```
% THIS FUNCTION finds the fittest chromosome with least
%distance
function [fittest_chromosome] =
fittest_calculate_pc(chromosomes,total_distance)
s=size(chromosomes);
N = s(2)-1;
t=0;
posi=0;
for i=1:s(1)

    if(chromosomes(i,s(2)) == (N+4))
```

```
        if (t==0)
            posi = i;
            distance1 = total_distance(i);
            t=1;
        end
        if(t==1)
            if(total_distance(i)<distance1)
               posi = i;
               distance1 = total_distance(i);
            end
        end
    end
end

if (posi ~= 0)
    for j=1:N
         fittest_chromosome(j) = chromosomes(posi,j);
    end

fittest_chromosome(N+1) = distance1;
end

if(posi == 0)
    fittest_chromosome = 0;
end
return;

end
```

____

```
% THIS FUNCTION CALCULATES TOTAL DISTANCE FOR EACH
%CHROMOSOME
function [fittest_chromosome] =
main_fittest_calculate_pc(chromosomes,total_distance,fitt
est_chromosome)
s=size(chromosomes);
N = s(2)-1;
t=0;
```

```
posi=0;
distance = 0;
for i=1:s(1)

    if(chromosomes(i,s(2)) == (N+4))
       if (t==0)
           posi = i;
           distance = total_distance(i);
           t=1;
       end
       if(t==1)
           if(total_distance(i)<distance)
              posi = i;
              distance = total_distance(i);
           end
       end
    end
end
if(distance ~= 0)
    if(fittest_chromosome == 0)
        fittest_chromosome = chromosomes(posi,:);
        fittest_chromosome(N+1) = distance;
        pause;
    end
   if(distance<fittest_chromosome(N+1))
   for j=1:N
        fittest_chromosome(j) = chromosomes(posi,j);
   end
fittest_chromosome(N+1) = distance;
disp('found');
pause;
   end
end
return;
end
```

CHAPTER 7

Genetic Algorithm approach to Solve Shortest Path and Travelling Salesman Problem

Shortest Path, Traveling Salesman and Hamiltonian Cycle are the other network design problem. These problems are very common to back bone network design problem. In all these three problems, the main difference is the degree of the node which is strictly two. Further, these three problems are very similar with each other. In the case of Shortest Path and Traveling salesman problem, a Hamiltonian Cycle is checked in the possible solution. Due to this similarity, these three problems are also considered in this research work. Shortest Path is considered in the terms of decision making.

This research work considers the problem for selecting a shortest route to deliver couriers to their destination address. The shortest route is defined as a route starts from the courier office to visit a number of destinations and at last returns to its source address. It has been explored the use of genetic algorithm where possible solutions are improved generation by generation and then there is more probability to find the exact solution. Fitness function is the backbone of the concept of genetic algorithm which directly affects the performance; since this is NP problem and traditional heuristics have had only limited success in solving small to mid size problems.

7.1 Shortest Route Problem Presentation

Given a connected, undirected graph *G* with *n* nodes, a least cost Hamiltonian circuit H is a sub graph of a *G* that connects all of *G's* nodes and contains one cycle. In this graph every edge (*We*, *j*) is associated with a numerical costs (distance) *cij*. A shortest route Hamiltonian circuit is the graph of the smallest possible total distance traveled

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

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

The Shortest route Courier delivery problem is represented with the help of Fig 1. Where each small circles represents a location and the magnified circles are those location where the couriers are to deliver. The locations are 13, 20, 34, 49, 57, 63, 73, 84, 92 and 10. The distance and type of route between two locations has been shown in Table-1 and Table-2 respectively.

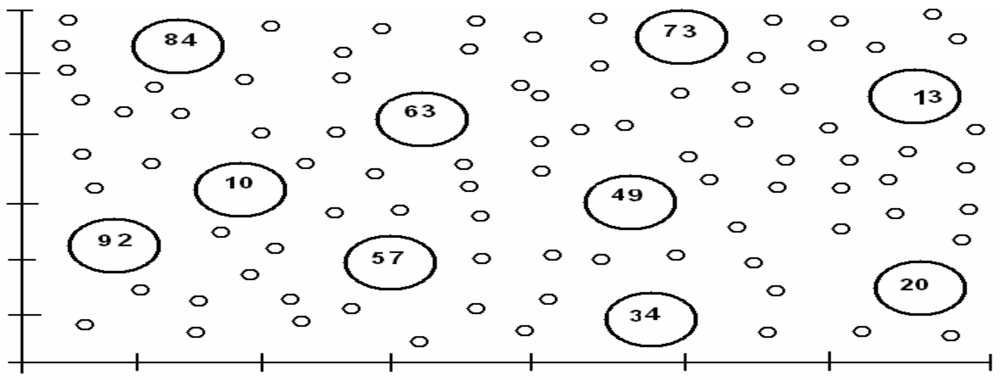

Figure 7.1. Locations to deliver courier

These locations are represented as a node of an undirected graph and it is represented in the form of an adjacency matrix in Table –7.1.This table contains the distance between two locations.

TABLE -7.1

ADJACENCY MATRIX OF THE GRAPH

	13	20	34	49	57	63	73	84	92	10
13	*	5	0	8	17	12	6	21	30	25
20	5	*	4	7	15	0	0	23	~~31~~	24
34	0	4	*	0	7	0	12	15	~~17~~	0
49	8	7	0	*	6	4	9	13	15	0
57	17	15	7	6	*	0	13	0	7	4
63	12	0	0	4	0	*	5	5	7	4
73	6	0	12	9	13	5	*	8	~~12~~	10
84	21	23	15	13	0	5	8	*	7	6
92	30	~~31~~	~~17~~	15	7	7	~~12~~	7	*	3
10	25	24	0	0	4	4	~~10~~	6	3	*

In this table, non zero numbers represent the distance between two locations. Zero (0) represents no path between two locations and strikethrough numbers represent the path constraint between two locations due to sudden change in route or due to emergency or heavy traffic load. Table-7.2 is used here to show the type of route between two locations.

TABLE -7.2

TYPES OF ROUTE

	13	20	34	49	57	63	73	84	92	10
13	*	D	0	S	D	S	D	D	S	H
20	D	*	H	S	S	0	0	S	~~H~~	D
34	0	H	*	0	S	0	S	D	~~D~~	0
49	S	S	0	*	H	D	H	S	D	0
57	D	S	S	H	*	0	S	0	S	D
63	S	0	0	D	0	*	S	S	S	D
73	D	0	S	H	S	S	*	H	~~H~~	S
84	D	S	D	S	0	S	H	*	D	S
92	S	~~H~~	~~D~~	D	S	S	~~H~~	D	*	D
10	H	D	0	0	D	D	~~S~~	S	D	*

This table contains three types of route: Heavy, Smooth and Difficult. These three types represent three speed ranges which are used to calculate the time between two locations. Table -7.3 represents the behaviour of three types of route-

TABLE -7.3

BEHAVIOUR OF EACH TYPES OF ROUTE

Type	**Description**	**Speed Range (KM/H)**	**Average Speed (KM/H)**
H	Heavy Traffic	10-30	20
D	Difficult	30-50	35
S	Smooth	50-70	60

7.1.1 Initialisation of parent population

Parent solutions are generated randomly with the help of a function. The function has the constraint that an allele of each chromosome must not be repeated in that chromosome. It is called parent population. Each chromosome is the combination of ten numbers (allele). Each chromosome represents a Courier delivery tour (Hamiltonian cycle) [3] where an each allele represents itself as a location and a path between location and its fixed position. All these Locations are numbered in a sequence. 1, 2, 3…..10.where 1 represent location 13, 2 represents 20 and so on.

TABLE -7.4

LOCATION CONNECTION

Location	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
chromosome	**2**	**3**	**5**	**7**	**10**	**1**	**6**	**4**	**8**	**9**

7.1.2 Evaluation

Evaluation is based on fitness function and total minimum distance travelled in each tour. All these tours are evaluated with fitness functions. The tour represented by each chromosome, may be illegal due to four reasons-

1) Self Loop
2) Violation of degree constraint or missing node
3) Hamiltonian Cycle
4) Isolated edge or path.

7.1.3 Fitness function

There are four reasons for the Illegality of the tour; therefore four Fitness functions have been developed here to check the fitness. 1 mark is assigned to pass each fitness function, while 0 marks are assigned in the case of failure. Chromosome is implemented in the form of array of size [10], where array index shows the fixed position and its value is an allele of generated chromosome. The representation of chromosome is as following

Chromo [1] = 2;Chromo [2] = 3;Chromo [3] = 5;
Chromo [4] = 7;Chromo [5] = 10;Chromo [6] = 1;
Chromo [7] = 6;Chromo [8] = 4;Chromo [9] = 8;
Chromo [10] = 9;

7.1.3.1 Self Loop

For the undirected connected graph

$$G = (V, E)$$

Where $V = \{v_1, v_2 \ldots\ldots v_n\}$
$E = \{e_1, e2 \ldots\ldots e_{n-1}\}$, each edge e_k is associated with vertices (v_i, v_j)

$(v_i, v_j) \in e_k$

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

```
Function self_loop()
Begin
 Set WE = 1 and N = 10 (where N is total no of location)
      for WE = 1 to N by 1 do
                   If chromo[WE] == WE
                   Print:  " self loop", Terminate fr
           endif
      endfor
 End.
```

7.1.3.2 Degree Constraint (missing node or repeated node)

Since each location has to be visited once, the location will be connected with two other cities. In-degree and out-degree for each location will be 1. If an allele of a chromosome is not repeated then it ensures that there each location is connected with two other locations.

$d(v_i) == 2$; where d denotes the degree of vertex We.

```
Function degree_constraint()
Begin
 Set WE = 1 and N = 10 (where N is total no of location)
      for WE = 1 to N by 1 do

            Set C = 0

               for J = 1 to N by 1 do
                  If chromo[WE] == WE
                    Increment C by 1
                    terminate the inner loop
                  endif
               endfor
```

```
                   if (C = 0)
                        print: "missing node"
                        terminate the outer loop
                   endif
       endfor
   End.
```

7.1.3.3 Isolated edge

If the pair of locus (array index) and allele (value) is same with other locus and allele in the same chromosome, then the edge will be isolated.

For any generated chromosome, pair of its locus and allele is defined as

$$\text{Chromo}(\ We \leftarrow v)$$

Where We is locus and v is the allele at this locus and its value vary from

$$1 >= We <= N \quad \text{and } 1 >= v <= N$$

where N is the total no of node.

Chromo(We ← v) = Chromo(j ← z)

If (We = z) and (v = j) then edge e_{iv} or e_{jz} is isolated.

```
Function isolated_edge()
Begin
  Set WE = 1 and N = 10 (where N is total no of location)
      for WE = 1 to N by 1 do
           Set  v = chromo [WE]

              If chromo[v] == WE
                        Print :  " isolated edge"
                       Terminate from the loop
               endif
        endfor
 End.
```

7.1.3.4 Hamiltonian Cycle

For each chromosome Chromo[N] there must be a Hamiltonian cycle. , Two vectors **Chromo** and **A** of size N are considered and initialized with value null.

For a chromosome Chromo [N]

Function Hamiltonian_cycle()

```
Begin
Set   j =1, p = 1 , t = 1 and N = 10
 (where N is total no of location)
    for WE = 1 to N-1 by 1 do
                If (chromo[j] == 1)
                   Terminate the loop
             Endif
                       Set j = chromo[j]
             If ( p > 1)
                   For l = 1 to p-1 by 1 do
                          If (a[l] == j)
                                Set t = 0
                                 Terminate the loop
                           Endif
                    Endfor
             Endif

           If( t == 0)
             Terminate the loop
          endif
          Set A[p] = j
          Increment p by 1
   Endfor


   If ( We < 10)
       Print : NO Hamiltonian Cycle
  Else
       Print : Hamiltonian Cycle exist
End
```

7.1.4 Result of fitness function.

After applying the fitness function it is found that all these tours are legal and have some cost which is in the form of total distance traveled. For passing each fitness function, 1 point will be given and in the case of failure 0. Following fitness point and distance earned by each chromosome (TABLE -7.5)

TABLE -7.5

FITNESS OF PARENT POPULATION

Chromosome	Fitness	Distance
a	4	69
b	4	87
c	4	70
d	4	62
e	4	157

Selection

In genetic algorithm fit solution are likely to survive and bad solution are likely to die off. So some of the best fit chromosomes are selected from parent population according to some selection criteria (e.g. Roulette wheel selection). Simply maximum point and minimum distance criteria is considered here. Selected chromosomes are **a, b, c, and d.**

Crossover/Recombination

Selected solutions are used for crossover. One point cross over is considered.

Mutation

It is the process to change the value of an allele of solution with some small probability value e.g. 1% Motivation is to explore new point in the solution space. A new concept is approached to mutate all those allele which are repeated a chromosome and it will be mutated (replaced) with the missing value in low to high order of the missing value.

Missing values (a1, a2, a3……..an)

Where a1<a2<a3…………<an

Repeated allele (x1, y1, z1……x1…..y1……..N)

Replace x1 with a1 and y1 with a2, where x1<y1.

Since there are no repetition of an allele in chromosome x and y, no any allele will be replaced while chromosome p and q will be mutated with their missing values.

For chromosome p, missing values are 4 and 10 and repeated alleles are 7 and 9 which will be replaced with 4 and 10 respectively. Similarly chromosome q will be mutated.

Evaluation of child population

After applying the fitness function, it is found the following fitness value for each of the child population

TABLE -7.6

FITNESS OF CHILD POPULATION

Chromosome	Fitness	Distance
x	4	54
y	4	102
p	2	--
q	0	--

TABLE -7.7

POSSIBLE PATH

Path No.	**Distance(km)**	**Time(hour)**	**Type(km)**		
			H	**D**	**S**
1.	69	1.8	13	19	**37**
2.	87	2.85	**35**	20	32
3.	70	1.7	04	**34**	32
4.	62	1.6	12	15	**35**
5.	157	3.55	00	**79**	78
6.	54	1.67	13	**29**	12
7.	102	3.78	35	10	**57**

On the basis of Table-6, if the selection criteria of the path is minimum time and driver's comfort, Path No. 2 is the best recommended option. If this Path No 2 is selected, its detail is shown in Figure-7.2 Table-7.8.

TABLE -7.8

SELECTED PATH DESCRIPTION

Distance Type	**Distance (km)**	**Average speed(km/h)**	**Time (hour)**	**Total time**
Difficult	15	35	0.42	1.60 hr
Heavy Traffic	12	20	0.6	
Smooth	35	60	0.58	

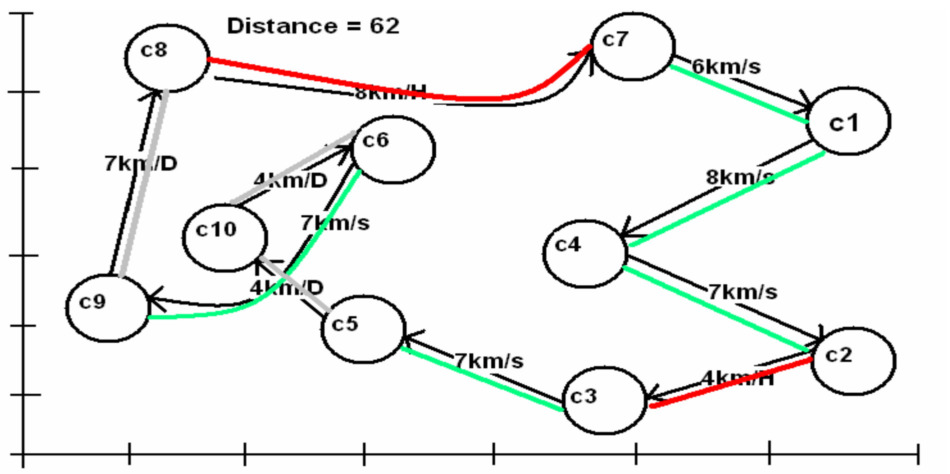

Figure 7.2. Selected Path No 2.

So selected path is number 2 with the total distance covered is 62 Km.

CHAPTER 8

Conclusion and Future Scope

Network Design Problem is an NP-hard combinatorial optimization problem. This thesis proposes Genetic Algorithm approach to solve this network design problem. It has been shown that, traditional methods are not capable to design the network required by the time. Traditional heuristics have the limitations with the possible constraints. The proposed genetic algorithm approach can provide the good results with the required constraints. For the network design problems, the aim of this thesis was to develop tools that find feasible high quality solutions of practical relevance within reasonable cost. To solve this problem all the possible genetic operators are developed. The size of network is considered from 10 to 1000 nodes. Researchers have tried to solve this problem but only up to mid size of network usually 200-300 nodes. In this research work network, up to 1000 node is considered and solution is derived which shows the robustness of this proposed genetic algorithm method. Various required constraints are imposed on the network which is the requirement of the current network. Degree constraint is one of major constraint and so far, no efficient method of finding an arbitrary degree constraint network has been developed. This thesis proposes a robust network design method which can derive the good solution for bigger size of network with the possible degree constraint. For the degree constraint an empirical relationship is derived on the basis of experimental data. In this research work various fitness functions have been developed. One of the fitness function is cycle check which checks the existence of the cycle in any undirected graph of any size. Various selection functions (7), crossover operators (6) and mutation operators (6) are developed and experimented with various size of network. In this research work total 546 different cases are considered for 15 different size of network from 10 to 1000 size of nodes. Further traveling salesman problem and shortest path problem are considered which are the special case of degree constrained spanning tree problem. For the shortest path problem various functions have been developed and experimented, and it has been shown that how does it help in decision making. For the traveling sales man problem, Hamiltonian cycle function is developed.

These all are the network design problems which belong to the NP-hard category. One of the objective of this research work is to show that genetic algorithm is an alternative solution for this NP hard problem where conventional deterministic methods are not able to provide the optimal solution. The proposed method is a robust method which finds the solution for almost any size of network (1000 node) for any possible network constraint. Any new constraint required by the network can be easily added with out changing the other functions. The proposed method also provides multiple parallel solutions which helps in decision making. Last but not least, the proposed network design method based on genetic algorithm has potential to achieve better results for any size of network. Altogether, these are some interesting research challenges for the near future.

The research work can be extended for different hybrid selection, crossover and mutation operators. The same problem can be considered for the reliable network where each node must have at least to connection. The same research work can be applied for directed graphs also. The proposed approach can be applied for various advanced network models like logistic network, task scheduling models, container terminal network model, vehicle navigation routing models etc. The same approach can also be used for allocation of frequencies in cells of cellular network.

Bibliography

[1]. M. Gerla and L. Klensock, on the topological design of distributed computer networks IEEE trans. Communication 25(1): 48-60, 1977.

[2]. Ahuja, R. K., Magnanti, T. L. & Orlin, J. B. (1993). Network Flows, New Jersey: Prentice Hall.

[3]. R. C. Prim. Shortest connection networks and some generalizations. Bell System Technical Journal, 36:1389.1401, 1957.

[4]. J. B. Kruskal. On the shortest spanning sub tree of a graph and the travelling salesman problem. Proc. of the American Mathematics Society, 7(1):48.50, 1956.

[5]. Narsingh Deo, 2000. Graph Theory with Applications to Engineering and Computer science: (PHI)

[6]. Ellis Horwitz, Sartaj Sahni and Sanguthevar Rajasekaran, Computer algorithms, University Press,2007

[7]. Cunha, A. and A. Lucena, “Algorithms for the degree-constrained minimum spanning tree problem,” Electronic Notes in Discrete Mathematics, volume 19, pages 403-409, and 2005.

[8]. A. Cayley. A theorem on trees. Quarterly Journal of Mathematics, vol. 23, pp. 376–378, 1889.

[9]. F. Harary and J. P. Hayes, “Node fault tolerance in graphs,” Networks, vol. 27, no. 1, pp. 19–23, 1996.

[10]. H. K. Ku and J. P. Hayes, “Optimally edge fault-tolerant trees,” Networks, vol. 27, no. 3, pp. 203–214, 1996.

[11]. R. E. Bellman. Dynamic Programming. Dover Publications Inc., 1957/2003.

[12]. G. L. Nemhauser and L. A. Wolsey. Integer and Combinatorial Optimization. Wiley-Interscience, 1988.

[13]. C. H. Papadimitriou and K. Steiglitz. Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, 1982.

[14]. F. Glover and G. Kochenberger. Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science. Kluwer Academic Publishers, Norwell, MA, 2003.

[15]. H. Hoos and T. St¨utzle. Stochastic Local Search – Foundations and Applications. Morgan Kaufmann, San Francisco, CA, 2004.

[16]. H. R. Louren¸co, O. Martin, and T. St¨utzle. Iterated local search. In Handbook of Metaheuristics [53], pages 321.353.

[17]. F. Glover and M. Laguna. Tabu Search. Kluwer Academic Publishers, Boston, MA, 1997.

[18]. Melanie M. (1998). An Introduction to genetic Algorithm (PHI) ISBN 81-203-1385-5

[19]. Michael D. Vose. 1999. The simple genetic algorithm : (PHI) ISBN 61-203-2459-5

[20]. F. Glover. Future paths for integer programming and links to arti.cial intelligence. Decision Sciences, 8:156.166, 1977.

[21]. B. M. E. Moret and H. D. Shapiro. An empirical analysis of algorithms for constructing a minimum spanning tree. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 400.411. Springer, 1991.

[22]. M. Ruthmair and G. R. Raidl. A Kruskal-based heuristic for the rooted delay-constrained minimum spanning tree problem. In A. Quesada-Arencibia et al.,editors, Twelfth International Conference on Computer Aided Systems Theory(EUROCAST 2009), Gran Canaria, Spain, to appear 2009. Springer LNCS.

[23]. F. Harary and J. P. Hayes, "Node fault tolerance in graphs," Networks, vol. 27, no. 1, pp. 19–23, 1996.

[24]. H. K. Ku and J. P. Hayes, "Optimally edge fault-tolerant trees," Networks, vol. 27, no. 3, pp. 203–214, 1996.

[25]. S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning tree," Comput. Oper. Res., vol. 7, no. 4, pp. 239–249, 1980.

[26]. D. S. Johnson, "The NP-completeness column: An ongoing guide," J. Algorithms, vol. 6, no. 1, pp. 145–159, 1985.

[27]. A. K. Obruca, "Spanning tree manipulation and the travelling-salesman problem," Comput. J., vol. 10, no. 4, pp. 374–377, 1968.

[28]. M. Savelsbergh and T. Volgenant, "Edge exchanges in the degree-constrained spanning tree problem," Comput. Oper. Res., vol. 12, no. 4, pp. 341–348, 1985.

[29]. Lixia Hanr† and Yuping Wang, A Novel Genetic Algorithm for Degree-Constrained Minimum Spanning Tree Problem, IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.7A, July 2006

[30]. Berna Dengiz and Fulya Altiparmak, Department of Industrial Engineering Gazi University, Ankara, turkey 06570 alice e. Smith1, "A Genetic Algorithm approach to optimal topological design of all terminal networks"

[31]. Rajeev Kumar and Nilanjan Banerjee, Multicriteria Network Design Using Evolutionary Algorithm, GECCO 2003, LNCS 2724, pp. 2179–2190, 2003.Springer-Verlag Berlin Heidelberg 2003

[32]. Michalewicz, Z. (1994). Genetic Algorithm + Data Structures = Evolution Programs. New York: Springer-Verlag.

[33]. Holland, J. (1992). Adaptation in Natural and Artificial System, Ann Arbor: University o Michigan Press; 1975, MA: MIT Press.

[34]. Raidl, G. R. & Julstrom, B. A. (2003). Edge Sets: An Effective Evolutionary Coding of Spanning Trees, IEEE Transactions on Evolutionary Computation, 7(3), 225–239.

[35]. Michalewicz, Z. (1995). A survey of constraint handling techniques in evolutionary computation methods, in McDonnell et al. eds. Evolutionary Programming IV, MA: MIT Press.

[36]. Michalewicz, Z., Dasgupta, D., Riche, R. G. L. & Schoenauer, M. (1996). Evolutionary algorithms for constrained engineering problems, Computers and Industrial Engineering, 30(4), 851–870.

[37]. Goldberg, D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Reading, MA: Addison-Wesley.

[38]. Koza, J. R. (1992). Genetic Programming, Cambridge: MIT Press.

[39]. Koza, J. R. (1994). Genetic Programming II, Cambridge: MIT Press

[40]. S.K. Basu, 2005. Design Methods and Analysis of algorithms (PHI) ISBN : 81-203-2637-7

[41]. Behrouz A Forouzan. 2006. Data Communications and Networking. The McGraw-Hill Company. ISBN-13: 978-0-07-06341-5

[42]. Hamdy A. Taha. 2007. Operation Research An Introduction

[43]. Kalyanmoy Deb, 1995. Optimization for engineering design (PHI)

[44]. Genetic Algorithms Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand by John H. Holland

[45]. Bryant a. Julstrom, codings and operators in two genetic algorithms for the leaf-constrained minimum spanning tree problem, int. J. Appl. Math. Comput. Sci., 2004, vol. 14, no. 3, 385–396

[46]. G¨unther R. Raidl and Bryant A. Julstrom, A Weighted Coding in a Genetic Algorithm for the Degree-Constrained Minimum Spanning Tree Problem, SAC ’2000 Como, Italy

[47]. A.T. Haghighat1, K. Faez2, M. Dehghan3, A. Mowlaei2, and Y. Ghahremani2, A Genetic Algorithm for Steiner Tree Optimization with Multiple Constraint Using Prüfer Number, EurAsia-ICT 2002, LNCS 2510, pp. 272–280, 2002. © Springer-Verlag Berlin Heidelberg 2002.

[48]. Tanenbaum, A.S. (1981) Computer Networks, Prentice- Hall, Englewood Cliffs, New Jersey

[49]. N. Deo and A. Abdalla. Computing a diameter-constrained minimum spanning tree in parallel. In G. Bongiovanni, G. Gambosi, and R. Petreschi, editors, Algorithms and Complexity, number 1767 in LNCS, pages 17.31, Berlin, 2000. Springer-Verlag.

[50]. M. Gruber and G. Raidl. A new 0.1 ILP approach for the bounded diameter minimum spanning tree problem. In L. Gouveia and C. Mour.ao, editors, Proc. of the Int. Network Optimization Conference, volume 1, pages 178.185, Lisbon, Portugal, 2005.

[51]. G. R. Raidl and B. A. Julstrom. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. In G. Lam-ont et al., editors, Proc. of the ACM Symposium on Applied Computing, pages747.752. ACM Press, 2003.

[52]. Kenneth A. De Jong, Evolutionary Computation, A Unified Approach, PHI, 2006.

[53]. A. Kershenbaum, “When genetic algorithms work best,” INFORMS J. Comput, vol. 9, no. 3, pp. 254–255, 1997.

[54]. L. Davis, Ed., Genetic Algorithm and Simulated Annealing. San Mateo, CA: Morgan Kaufmann, 1987.

[55]. L. Davis, “Adapting operator probabilities in genetic algorithms,” in Proceedings of the Third International Conference on Genetic Algorithms,

[56]. K. A. DeJong, “Analysis of the behavior of a class of genetic adaptive systems,” Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1975.

[57]. Brian R. Hunt, Ronald L. Lipsman, Jonathan M., A Guide to Matlab, Cambridge University Press.

[58]. Donald Knuth. The Art of Computer Programming, Volume 1: Fundamental Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 1.2.11: Asymptotic Representations, pp.107–123.

[59]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0262032937. Section 3.1: Asymptotic notation, pp.41–50.

[60]. Michael Sipser (1997). Introduction to the Theory of Computation, PWS Publishing. ISBN 0-534-94728-X. Pages 226–228 of section 7.1: Measuring complexity.

[61]. Srinivas, N. and Kalyanmoy Deb. "Multiobjective optimization using nondominated sorting in genetic algorithms." *Evolutionary Computation*, vol.2, no.3, p.221-248 (Fall 1994).

[62]. Koza, John, Forest Bennett, David Andre and Martin Keane. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, 1999.

[63]. Koza, John, Martin Keane, Matthew Streeter, William Mydlowec, Jessen Yu and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence.* Kluwer Academic Publishers, 2003.

[64]. www.galeb.etf.bg.ac.yu/~vm Option:Tutorials

[65]. www.hao.ucar.edu/public/research/si/pikaia/tutorial.html

[66]. www.epcc.ed.ac.uk/overview/publications/training_material/tech_watch

[67]. http://samizdat.mines.edu/ga_tutorial

[68]. http://www.genetic-programming.com/gpanimatedtutorial.html

[69]. http://www.geneticprogramming.com/Tutorial/tutorial.html

[70]. http://online.engr.uiuc.edu/shortcourses/innovation/index.html

[71]. http://www.aic.nrl.navy.mil/galist/

[72]. http://www.geatbx.com/index.html.

[73]. http://www.wired.com/wired/archive/10.03/everywhere.html?pg=2.

[74]. http://www.natural-selection.com/NSIPublicationsOnline.htm.

[75]. http://www.trnmag.com/Stories/062800/Genetically_Enhanced_Engine_062800.

[76]. http://spaceflightnow.com/news/n0110/18orbits/.

[77]. http://www.space.com/news/darwin_satellites_011016.html.

[78]. http://www.salon.com/tech/feature/1999/08/10/genetic_programming/.

Publications

1. Anand Kumar, Dr. N.N. Jani , "An algorithm to detect cycle in an undirected graph" International Journal of Computational Intelligence Research ISSN 0973-1873, (Vol 6, No 2 (2010), pp 305-310)

2. Anand Kumar, Dr. N.N. Jani, "Network Design Problem Using Genetic Algorithm- An Empirical Study On Selection Operator" International Journal of Computer Science and Applications (IJCSA) ISSN: 0974-1003, April/May 2010, Vol 3, No 2, pp 48-52.

3. Anand Kumar, Dr. N.N. Jani, "An Evolutionary Approach for Shortest Path Problem - Courier Delivery System" International Journal of Computational Intelligence Research ISSN 0973-1873 Volume 6, Number 2 (2010), pp. 261–273.

4. Anand Kumar, Dr. N.N. Jani, and "An Evolutionary Approach to Allocate Frequency in Cellular Telephone System" International Journal on Futuristic Computer Applications (IJFCA) 25-Feb, 2010 Bangalore, ISSN: 0975 - 8887. The manuscript is published in International Journal of Computer Applications. URI: http://www.ijcaonline.org/archives/number7/157-280

5. Anand Kumar, "A Nature based Evolutionary approach to solve Network Communication NP-Hard Traveling Salesman problem" International Journal of Computational Intelligence Research and Applications (IJCIRA) ISSN: 0973-6794, Volume 3 Number 1, January-June 2009, Page. No. 27-32.

6. Anand Kumar, Dr. N.N. Jani, "Genetic Algorithm Approach to Solve Hamiltonian Circuit Problem With Robust Fitness And Repair Function" Proceeding of **IEEE** International Advance Computing Conference 2009.Thapar University, Patiyala. ISBN NO: 978-981-08-2465-5

7. Anand Kumar, Dr. N.N. Jani, "A Novel Genetic Algorithm Approach for Network Design with Robust Fitness Function" Proceeding of International Conference on Mathematics and Computer Science, 5-6 Feb 2010, Loyola College, Chennai, ISBN: 978-81-908234-2-5.

8. Anand Kumar and N.N. Jani, " Using A Genetic Algorithm approach to Design Backbone Core Communication Network" Proceeding of International Conference on Emerging Trends in Computing , 8-10 Jan 2009 , Kamaraj College of Engineering and technology, Virudhunagar, Tamilnadu.

9. Anand Kumar and Dr. N.N. Jani, "Genetic Algorithm for Network Design Problem- An Empirical Study of Crossover operator with Generation and Population Variation" International Journal of Information Technology and Knowledge Management, ISSN: 0973-4414, Vol-III, Issue-I, June 2010

****