Scaled Evolutionary Development of Topological Network Designs through Specialised Genetic Operators

Daniel Bratton

Supervisor: Andrew Philippides

University of Sussex

Evolutionary and Adaptive Systems MSc, 2003

Acknowledgements

Thanks go to Inman Harvey, the EASy course director, and Andy Philippides, my dissertation supervisor, for their help and wisdom in guiding me in my research and interests throughout this project. Thanks also go to Ezequiel Di Paolo, who provided he impetus for my interest in this topic and encouragement in my early research in the subject. Personal thanks are most heartily deserved by my family as well as the Rose family for all of their support, by my colleagues on the course, and by Sean "Blown Tire" Whalen, for last-minute proofreading and general encouragement. Finally, my gratitude goes to Mariamne Rose for all of her support, understanding, and affection.

Abstract

The network design problem can be defined as the search for optimal topological configurations of links connecting a set of fixed nodes. Most of the previous work on the subject has been restricted to small networks of fewer than 20 nodes due to processing and memory requirements as well as the huge increase in complexity seen as network size is increased. This project explores scaling a genetic algorithm to adapt to solving the problem for much larger networks of up to 200 nodes while still obtaining optimal or near-optimal solutions for all node configurations. A variety of techniques are used, including specialised genetic operators and heuristics, to improve both the efficiency and performance of the GA in developing solutions to the increasingly complex problems.

Table of Contents

1. Introduction

1.1 A Background of Evolutionary Computing

Evolutionary computing (EC) is a field that has enjoyed a surge in growth and research in recent years. Its concepts are rooted in biology and computer science, which are combined to produce a discipline that encompasses all methods for solving problems that are inspired by the traditional evolutionary method seen in nature. These processes are designed and modelled after biological evolution to produce the same developmental effects on a population over a period of time.

The ideas behind EC were first conceived and modelled by Holland (1975) in his seminal book *Adaptation in Natural and Artificial Systems*. The book was the first introduction to genetic algorithms (GAs), which are the basis on which most of EC is built. Holland outlined the basic concepts of evolution and their application to computer models. A GA is designed to solve a specific problem, without any prior knowledge of an optimal solution. The GA operates by creating an initial number, or population, of genomes, each genome representing a single possible solution to the problem, usually randomly initialized. The *encoding* of the genome is an important factor – it is the representation of the abstract genome. This encoding can take a variety of forms, from simple bitstrings or sets of floating-point numbers to more complex structures such as trees. These genomes are then ranked according to their fitness – a measure of how well each individual genome meets the requirements set by the designer for an optimal system. Those genomes with higher fitness ratings are considered to be 'fitter', and are more likely to be selected to 'breed' with other genomes. This breeding is implemented through *crossover*, a process by which specific genes are selected from two 'parent' genomes and placed into a new genome, considered to be the 'child' of both of the parents. This results in a new genome that possesses characteristics of both of the parent genomes, without being an exact replica of either. *Mutation* can then be applied to the child genome on a probability basis – only a small portion of the offspring of a population will be mutated. How this is accomplished is totally dependent upon the form of the genome and the problem being solved. The processes of crossover and mutation are applied until there are an equal number of child genomes to parent genomes, at which point the population of parents is destroyed and replaced by the offspring. This entire process is known as a single *generation*. By selecting individual genomes with higher fitness values as being more likely to breed, it is likely that the new population of offspring after a generation will have a higher average fitness value than the population of parents. A GA can be set to run for as many generations as the designer sees necessary for the development of an optimal or near-optimal solution.

There has been a great deal of work and innovation in the field since Holland's work in 1975. Goldberg (1989) provided a more thorough explanation of the field in his book *Genetic Algorithms in Search, Optimisation, and Machine Learning*. With a better understanding of the mechanics of EC, new selection, crossover, and mutation methods have been introduced and refined, resulting in faster, more efficient and better performing GAs. One of the most important and useful developments has been in the implementation of hybrid GAs. These algorithms use a heuristic approach combined with traditional

evolutionary techniques to find appropriate solutions to the problem, often faster than a standard GA. While standard GAs are often adequate to fulfil the problem-solving needs of a system, a heuristic approach is often able to direct the development of a population of genomes towards an area of the solution space that is known to contain solutions of a higher fitness level. This often results in significantly faster development of the population. *Local search* is one of the more popular and applicable heuristics that is added to GAs. It is usually implemented as a technique used to improve a genome after breeding and mutation have taken place, retaining the improvements to the genome. There are a variety of other heuristic techniques that can be applied, and researching new methods has become very popular.

1.2 Network Topology Design

A network is composed of a number of nodes representing individual computers, memory addresses, buildings, terminals, depots, and any other sort of location, physical or abstract. These nodes can even represent smaller, self-contained networks – the only requirement is that each node is completely independent of all other nodes. These nodes are connected by links, also known as edges, representing paths between the two nodes at either end of the link. The links that are specified can be any form of connection, from fiber-optic relays to water pipes to roads. Individual links have a variety of properties such as length, capacity, reliability, and a cost. Cost is calculated from properties possessed by a link. Intuitively, links with higher values for their properties are considered to be 'better' and will therefore have a higher cost.

1.2.1 Motivations

Designing the topologies for these networks is an undertaking that can be applied to almost every aspect of modern society. City planning, telecommunication networks, and even mapping out a route for a trip across town are all tasks that seek optimal solutions to a part of the network design problem. The ability to create networks that operate efficiently, smoothly, and perhaps most important, reliably, results in improved transport and communication in almost every situation.

1.2.2 Difficulties

Unfortunately, the network design problem is not easily solved – it falls into the category of NP-hard problems. These problems are characterized by the requirement of a brute force method for obtaining a guaranteed optimal solution. Brute force methods obtain the definitive optimal solution to a problem by performing a thorough examination of all possible solutions, a time-consuming and repetitive process.

The difficulties experienced in solving NP-hard problems through brute force methods are even more pronounced when it comes to the network design problem. Given a collection of n nodes, the number of possible links for that configuration of nodes can be calculated as:

$$\frac{n(n-1)}{2}$$

Using this equation to determine the number of possible links in a network also tells us the number of possible network configurations for the specified collection of nodes. If each link has s possible states, which could correspond to presence or absence, reliability levels, or other such properties, this gives a total of:

$$s^{\frac{n(n-1)}{2}}$$

possible network configurations. The following table shows the number of possible configurations for networks containing various numbers of nodes.

n	*s = 2*	*s = 3*	*s = 4*
5	1024	59049	1048576
10	3.5E13	2.95E20	1.2E27
20	1.5E88	4.5E121	2.46E145
50	5.77E399	2.98E615	3.3E768
100	1.25E1521	5.6E2392	1.57E3011

The number of possible configurations grows exponentially through both *n* and *s*, resulting in enormous search spaces for moderate values of *n* and s that could take brute-force methods unacceptable amounts of time – months or years - to fully process, without even taking into account the time required for actually evaluating each network configuration. Because of this exponential increase in the complexity of the problem, normal optimisation methods are almost worthless, given the extreme cost.

While a trained expert on the subject has the ability to draw on experience for designing optimal topologies, the limitations of human designers usually restrict this activity to very small networks of less than 20 nodes. Without an experienced designer, even small problems are all but unsolvable through standard means. Because of these difficulties, evolutionary approaches have been the methods of choice for solving the problem of network design. The nature of evolutionary computation is very applicable in this situation, given the vast search spaces and limited available resources in relation to the size of the problem.

1.3 Goals and Methods

This project consists of a series of experiments aimed at the development of a genetic algorithm for solving the network design problem. The GA that is used incorporates several heuristic methods, described below, that are believed to have an impact, sometimes significant, on the speed of development for optimal topologies. In addition, the research is not limited to small-scale networks of fewer than 20 nodes, as many previous experiments have been. Scaling the algorithm is taken into consideration, as this is an important aspect of developing larger, more complex networks.

1.3.1 The Effectiveness of a GA on a scaled network

Although many such algorithms have been developed in past research (see section 2), almost all of these have concerned themselves with small or very small networks, those containing between 5 and 20 nodes due to the exponential increase in problem complexity as network size increases. This increase in complexity means an increase in computer processing and memory requirements, which have only been met in recent years.

Small-scale network development is very useful for the design of local area networks (LANs), and other such topologies. Given their relatively smaller solution spaces, a network design is often found that can be assumed to be optimal, or nearly so. Using the same algorithms for finding optimal or near-optimal designs for much larger networks can introduce problems, particularly in the issue of maintaining all-terminal connectivity, the network state in which every node, or terminal, is connected to every other terminal, most often through a path between a series of connected nodes. In very large networks, as in smaller networks, minimizing the cost of the network entails removing as many extraneous edges as possible. This presents a major problem in networks with hundreds or thousands of edges when the removal of a single edge could result in the network becoming disconnected. This is not so much a problem with smaller networks, as the ratio of edges that will reconnect the network to the number of total possible edges is much smaller – on a larger network with an undirected algorithm such as a GA, however, finding a link that will reconnect the network could be like trying to find a needle in a haystack.

The most important adjustment that must be made for finding near-optimal solutions in very large networks is within the fitness function. As reliability is so much more of an issue in order to ensure that there is a valid path between each pair of nodes in a large network as opposed to a small network, the penalty for unreliable or infeasible networks must be increased simply to maintain feasibility. In smaller networks this penalty occurs simply to compare two networks that are very likely to be feasible and find the more reliable of the two. In large networks, the penalty is applied in the interest of giving disconnected networks very low fitness values. An appropriate scaling of the fitness function must be found in order to attain these goals.

1.3.2 The Addition of Local Search

Local search is a technique that is used a great deal in modern GAs in order to increase the speed of the algorithm in finding an optimal solution. There has been a variety of techniques proposed, but most come down to a simple brute-force method of searching the solution space around the genome for a slightly fitter solution. The size of the area to search is often the main factor – a balance has to be struck to keep the local search effective while not overemphasizing the procedure and moving the GA too close to becoming a brute force method itself.

While it is not strictly necessary to implement local search in a GA, the benefits, for the most part, outweigh the costs. One of the shortcomings of GAs is their inability to explicitly direct their development toward promising areas of the solution space, a difficulty that implementing a local search algorithm is able to overcome. In extremely complex problems such as network design, this can make for significant increases in performance as the algorithm is guided towards networks that are similar to those found by the GA, but with higher fitness levels.

1.3.3 The Addition of Prior Knowledge

The concept of introducing prior knowledge to a GA involves giving the algorithm information about what an optimal solution looks like, either in part or in full. Naturally it would be pointless to give a full optimal solution to a GA; there would be no point to running the algorithm in the first place. When a portion of the solution is given to the GA, however, it often has the effect of speeding the search, as part of the problem has already been solved.

Prior knowledge can take a variety of forms depending on the specific GA, and more importantly, on the problem. In terms of the network design problem, prior knowledge is intuitively implemented as giving the algorithm an optimal solution for one or more network configurations of a small number of nodes – for example, solving the network design problem for a set of four nodes is a simple undertaking, which could even be done by hand in a reasonable amount of time, given that there are only six possible edges and thus s^6 possible configurations, many of which will be infeasible. The knowledge of this optimal link configuration for a set of nodes with the same specific physical arrangement can be passed to a GA that is assigned to solve a problem containing more than four nodes, and then apply the knowledge to all sets of four nodes with the same physical layout before starting the evolution of the population. In theory, this will result in the initial population being composed of genomes that are fitter than a randomly initialized genome, thus skipping the number of generations necessary for the GA to reach that point on its own.

1.4 Structure

Following the introduction of evolutionary computing and the network design problem as described in this chapter, I will be describing related work that has been done in the field, including previous methodologies and motivations. This will not be confined to the overarching design problem, but will also cover sub-topics which have been looked at outside of the specific problem. Following this I will describe in detail the implementation of the GA, including the encoding, data structures, evolutionary techniques, and fitness calculation. The fourth chapter will cover results obtained from the GA over a variety of problems, with discussions of each result as it is presented. Finally, the last chapter will be devoted to conclusions and suggestions for future work on the network design problem using heuristic and non-heuristic GAs.

2. Related Work

The network design problem is composed of a variety of smaller problems, each of which contributes to the overall complexity. Most of these have been researched heavily in their own right, as they can be applied to many problems. This chapter will begin with a description of related work on the network design problem, followed by an examination of research on the vitally important issue of network-specific reliability calculations, and finishing with a review of studies on the encoding of networks for use in a GA.

2.1 Developing Network Topologies

Large numbers of studies have been done on the network design problem, describing varying solutions, algorithms both heuristic and non-heuristic, and important considerations to be taken into account. The research in this are encompasses a wide range of specific problems that fall under the network design umbrella, and some novel approaches have been proposed. In recent years, EC has come to the forefront of the search for reasonable solutions to these problems, given its observed success in similar situations where very large solution spaces must be searched efficiently for optimal solutions.

In one of several studies by the authors, a hybrid approach is taken towards the problem of designing backbone structures for communication networks (Konak and Smith, 1999). The backbone network design problem is defined by the authors as "finding the network topology minimizing the design/operating cost of a network under performance and survivability considerations." Backbone communication networks are unconcerned with specific linking of all nodes in a network, rather basing their connectivity measures on hubs within the network – nodes that act as centrally located connectors for all nodes in close physical proximity. The study uses a local search algorithm to improve genomes within the population by applying domain-specific information. Results showed that the application of local search resulted in an algorithm that improved on previous algorithms in terms of efficiency, measured as the number of generations required for the GA to find an optimal solution. The number of nodes in a network for this study ranged between ten and twenty-three.

The same general approach is taken in another study (Deeter and Smith, 1997), with the exception that the algorithm is applied to full networks. A heuristic GA approach is used for the design of networks in an attempt to minimize cost while taking all-terminal reliability into account. Unlike many studies, the research performed here accounts for varying reliability levels for links – in other words, it allows for more than two possible values of *s*. The algorithm was tested with promising results on networks of five nodes, although the technique used is questionable on larger networks – my own reimplementation performed in the course of this project showed results suggesting that highly unlikely topologies were 'optimal' for networks of twenty or more nodes.

A study on the development of an evolutionary algorithm for "the design of hybrid fiber optic-coaxial cable networks in small urban areas" by a team of researchers at the University of Seville applies a variety of new constraints to the standard network design problem (Cortes, Guerrero, et al, 2001). Most intriguingly, the authors present an algorithm that solves the problem as efficiently as possible, but ground it in real-world situations by addressing issues that would influence the actual construction of a network, such as problems with civil works. They make the effort to minimize the network deployment costs over small urban areas, using LANs as an example of a suitable network size and structure. The case is represented in the study as a form of the Steiner problem – trying to maximize a value $f(x)$ for the value x. The network is divided into two sections – Steiner nodes, or those which are used to pass data through the network, and

terminal nodes, the destination nodes for the data.

Part of the same team of researchers used a novel approach to the problem by basing their GA on an exploitation of the Kuhn-Tucker optimality conditions (Cortes, Larraneta, et al, 2001). The Kuhn-Tucker theorem, a generalization of Lagrange multipliers, is based in nonlinear programming, stating that if a regularity condition holds while functions *f* and *h* are convex, a global minimum is a solution which satisfies the conditions of *h* for a vector of multipliers. The researchers used this theorem in designing a genetic approach dealing with topological design for the backbone of a two-level hierarchical network.

A case study was performed by members of both University of Seville teams (Cortes, Onieva, et al, 2001) which applied a decision support system from the previous work in planning telecommunication networks to a real world node layout based on the Andalusian region of southern Spain. This system took as its major consideration the economic status of the areas represented by nodes, and used this consideration in selecting properties such as hub and terminal locations. Unlike other work on the subject, this study was concerned not only with the optimal layout of the links for connecting nodes, but with the physical location of the nodes themselves. This point, as well as the real world considerations and implementations, make for an extremely interesting case study.

Evolutionary techniques were used by Philippides (1996) in an examination of their applicability to the network design problem. An important point that is made concerns developing networks that are not simply optimized for minimum cost and reliability, but also for shorter paths between nodes. This has the effect of decreasing latency in the network, making it more useful in real-world applications. Tests were limited to a single seven node configuration, reducing the credence of general performance measures, although no claims were made regarding the usefulness of the algorithms on larger, more complex arrangements of nodes.

A decomposition method based on branch and bound for the minimization of total network cost with a system reliability constraint was presented by Jan et al (1993). The algorithm presented provided excellent results on small networks, given its totally heuristic nature. Because of this nature, however, the algorithm was not applicable on large networks given the computational cost. Regardles, this study is often cited as an important step in developing the heuristic portion of a hybrid GA.

The subject of evolutionary design for larger networks was presented by Denzig and Altiparmak (1995) where a heuristic search algorithm based on a GA was developed. Its purpose, like previous studies, was to optimize the design of a network while taking into account a reliability constraint. While the algorithm developed appears sound for larger networks, it does not appear to have been tested on topologies containing more than 30 nodes, likely given the technological constraints at the time.

2.2 Calculating Network Reliability

Finding solutions to the problem of calculating network reliability is a well-established field in its own right. Its applicability extends beyond the network design problem, to encompass all areas of research into networks and graphs. Stated simply, network reliability is "the probability that every node can communicate with every other node through some (non-specified) path of arcs [edges] for a stated mission time not allowing repair" (Konak and Smith, under review).

A variety of techniques have been suggested and implemented, each with strengths and weaknesses. The simplest and fastest method is to approximate the reliability of the network by determining the upper and/or lower bounds – the maximum and/or minimum values that the reliability of the network could possibly reach. Depending on the algorithm used, these calculations can take between polynomial and exponential time. The results that they give are not guaranteed to be close enough to the actual reliability value of the network for determining whether the tested topology is optimal, but they are very useful for establishing which topologies warrant a close examination through a more exact and time-consuming method.

Jan (1993) gave an algorithm for determining the upper bound of maximum network reliability, which is used quite a few studies of the network design problem. His method derives the upper bound in terms of the connectivity level of each node in the network – i.e. how many edges are connected to it. In short, the edges of a network are placed into an indexed array, and the maximum reliability of a network is calculated as the sum of the probability of each edge in the network failing simultaneously with every edge prior to it, determined recursively. While Jan's method is robust, efficient, and easily converted to calculating a lower bound of minimum network reliability, its major shortcoming is that it only works on networks where all edges possess identical reliability values. This drawback is noted in the conclusions of the study, and suggested as a future line of work.

This is precisely what is undertaken in Konak and Smith's study "An Improved General Upper Bound for All-Terminal Network Reliability," at last report under revision for publication. Their algorithm is simply an extension of Jan's that takes into account edges with varying reliability ratings, along with a few minor changes that slightly improve the tightness of the bound on all networks.

The second method for determining network reliability is estimation, usually performed by a Monte-Carlo style process of removing random edges from the network undergoing testing one at a time subject to their probability of failure. Once all of the failed edges have been removed, the network is tested for connectivity. This entire process is repeated a set number of times, and the final network reliability is determined by the number of times the network failed, i.e. was not connected, divided by the number of times the test was performed. The chief problem with this method is that with s^E possible system states, where E is the number of possible edges in the network, it would

require the test to be repeated many, many times for even moderate values of E in order to account for even a small portion of these states. Without a suitable large number of repetitions, sampling error would be introduced in serious quantities, but when the number of repetitions is large, the algorithm's running time increases as well.

An algorithm was determined by Yeh et al (1994) that improves upon this standard "crude" Monte-Carlo method by first finding the probability of k edges failing for $k = 0$ through E, repeated a set number of times. Secondly, the algorithm determines the probability that if k edges fail in the network, the network will become disconnected, again repeated the same set number of times. These two probabilities are multiplied, and the sum is found for all k. The final reliability rating of the network is equal to this sum divided by the constant number of repetitions used in both previous equations. Because of the more refined approach to determining edge failures, this process needs fewer iterations in comparison to the crude Monte-Carlo approach, leading to lower processing times.

The final method of determining network reliability is done through backtracking through the entire network, removing all combinations of edges based on their reliability ratings and testing for network connectivity. The final exact value of the network reliability is then calculated as the number of failed networks divided by the total number of combinations of edges, s^E. Because this is a brute-force process operating on an NP-hard problem, processing times are unacceptably high for all but the smallest networks, despite the exact reliability rating that is obtained. Because of this difficulty combined with the nature of this project, this method was not heavily researched or implemented here.

2.3 Network Encoding Schemes

Much of the work done in the area of network encoding schemes draws inspiration from the encoding of neural architectures such as neural networks, which is similar to the network design problem in that it is concerned with developing ideal links and weights for a set of nodes. This problem can be solved using evolutionary techniques, and central to this process is the choice of the representation scheme that is used to encode a neural architecture in the form of a genome and to decode a genome into the corresponding neural architecture, the phenome. Many of these encoding schemes are easily translatable to the network design problem.

Miller et al (1989) gave an example of a neural encoding where a network with n nodes is represented by a connectivity matrix of dimensions $n \times n$, where each column specifies the constraints, or weights, on the connections between the node represented by that column and all other nodes. Each entry C_{ij} in the matrix indicates the nature of the constraint on the link from node j to node i, with a 0 representing an absent edge and a 1 representing the presence of a connection between the nodes. The genotype for this scheme is constructed by concatenating the rows of the matrix, resulting in a bitstring of length $n \times n$. It should be noted that this scheme is designed for a directed network, where each edge can only represent a link from one node to another; there is no connection from the second node back to the first. This differs from many of the network design representations that use an undirected network, where edges represent a path that goes in both directions.

Cellular encoding (Gruau, 1992) is an encoding process by which neural networks are translated into trees of grammar transformations, which are then optimised using genetic programming techniques. This results in a framework that shows consistency between applications, and provides for automatic network generation. However, it does not offer a clear-cut association between the encoded genome and the corresponding network phenomes – identical series of transformations at different locations in the tree can result in totally different configurations of parts of a network. Despite these difficulties, cellular encoding has been used in a great deal of research regarding neural architecture due to its concise, complete nature, which is very useful for encoding complex networks.

These two methods represent the core of most direct and indirect encoding schemes used for neural architectures, and thus the core of most schemes used in solving the network design problem. An example of the direct representation used in most research was implemented by Deeter and Smith (1997), in which all possible links are represented by a connectivity matrix, differing from Miller's scheme only in that an active link is defined by an integer corresponding to that link's reliability level. Because of their simplicity, direct encodings of this form are nearly ubiquitous in network design research – indirect encodings are extremely rare, though Philippides (1996) provided a comparison of a direct encoding using Prufer sequences to augment an adjacency matrix, and an

indirect encoding based on relative addressing for determining the destination of each link in the network. In the tests performed, the direct encoding outperformed the indirect encoding by a significant measure.

3. Implementation

The program corresponding to the research performed in this project is coded in C++ and Java 1.4. The Java portion of the programming is done on the Eclipse programming IDE using the Codefarm AI platform to allow for pre-programmed GA functions, ease of use, and advanced visualisation opportunities.

The selection process, a 2d spatial selector, is taken from the Codefarm software library, as is an elitism method that retains the best solution from each generation within the population of genomes. Each of the genome objects described in 3.2.1 is based on the generic genome provided in the library, and the crossover and mutation operators are extensions of the basic methods, re-implemented to function on the specialized encoding used in this project.

3.1 Encoding

In almost all of the previous research into the network design problem described in sections 2.1 and 2.3, genomes representing all of the edges of the network were implemented as arrays of integers, with a 0 representing an absent edge, and some positive-valued integer representing a present edge, with the exact value varying based on the reliability rating of that edge, if this property was being tested for. While this encoding was functional for small networks with a limited number of possible edges, in larger networks where the number of possible edges is increased exponentially, this results in arrays that are composed almost entirely of 0s, given the attempt by the GA to minimize the total number of edges present in the network.

For example, a 5 node network has a total of 10 possible edges, given that the edges have only two states – on and off. If this network is 2-connected, or has at least 2 edges connected to every node, the minimum number of active edges, and therefore 1s in the genome, is 5, or 50% of the genome. However, in a 400 node network, with 79800 possible edges, a 2-connected network would use a minimum of 400 edges, or .005% of the number of possible edges. As the GA is used to minimize the total cost of the network, his would lead to enormous genomes that contained miniscule percentages of 1s, an extremely inefficient implementation.

To avoid this problem in this project, genomes are implemented containing only the active edges in a network. The object-oriented nature of the languages used makes this a straightforward process, as each edge is handled as an individual object with a unique identification. The result is a genome that is only as long as the number of active edges in the network, and therefore 100% efficient, while still allowing the same functionality as the encoding described above. See section 3.2.1 for a full description of the genomes used in the encoding.

3.2 Data Structures

3.2.1 Node, Edge, and Network Genomes

Three different genomes are used in the project for full network representation. Networks are split into three parts – the set of nodes with their fixed physical locations, the variable set of edges representing the connections between nodes and their associated properties, and an encompassing network genome that contains the other two genomes. A network genome is used solely as a wrapper around a node genome and an edge genome. It is passed to the Network structure, deconstructed into its component parts, and used to create a phenotypic network for that specific genome.

Node genomes are a relatively simple collection of nodes representing the unconnected state of the network. Each gene represents a single node, and includes that node's properties, namely its unique node id and its Cartesian coordinates, which are used in calculating the distance between any two nodes, and thus the lengths of edges. In a single run of the GA, where a solution is sought for a single configuration of nodes, one static node genome is created and shared amongst all of the network genomes in the population.

Edge genomes are the primary variable components of the GA population. All of the evolutionary operators used in the project and detailed in section 3.3 are designed to work on edge genomes, as these represent every connection present in a given network topology, each gene representing a single active edge in the network. These genes contain all of the relevant data about that edge, specifically the nodes at each end of the edge and the length, cost, and reliability rating of that edge. The reliability rating can take one of three values, 85%, 90%, or 95% reliability, represented by levels 1, 2, or 3, respectively. These reliability values were chosen to stay in keeping with much of the other work on the subject (see section 2.1) where these same reliability ratings were used. The cost of the edge is scaled non-linearly according to the reliability rating in order to closer simulate real-world conditions, where the more reliable network links tend to be more expensive than less reliable links at an increase in cost that is greater than proportional to the increase in reliability.

Each edge genome also contains a unique *perc* array of floating point numbers, of equal length to the number of genes in the genome. This array is used to represent the location of each gene in the genome, expressed as a percentage. It is used in the single point and multi point crossover operators described in section 3.3.1 for allowing crossover to take place on genomes of different sizes.

In addition to being the representation of all of the edges in the network, the local search process is carried out on edge genomes, as they are the only variable part of the full composite network genome. See section 3.3.3 for a description of the local search operator.

3.2.2 Networks

The Network object in the implementation represents the phenome state of a member of the population. Each network contains a set of nodes, taken from the node sub-genome, and a set of edges, taken from the edge sub-genome. There are a variety of properties possessed by a network, each of which is used in some way in the calculation of the fitness of that phenome (see section 3.4.2). These are:

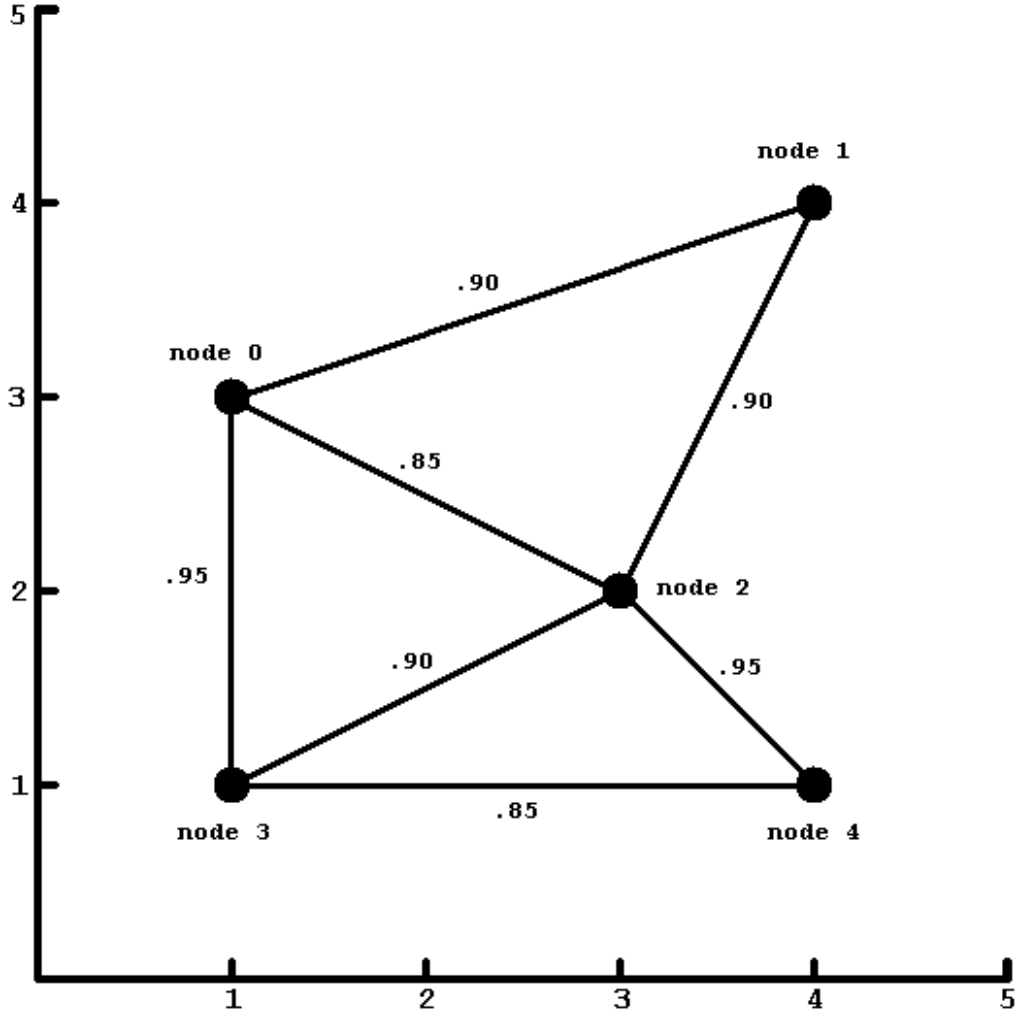
- The number of connected components in the network. In a connected network, this value will be 1. Any greater value would mean that the network was disconnected at some point.

- The k-connectivity of the network. This represents the number of edges that are connected to the least-connected node in the network. K-connectivity is used as a measure of reliability. See section 3.5.2 for a full description.

- The maximum cost of the network. When a network is fully connected, i.e. there is a direct path between any two nodes, each edge in this network is set to the maximum reliability rating, thus giving it the maximum cost. The total cost of all of these edges represents the maximum cost of the network.

- The shortest paths structure. This data structure gives the first edge on the shortest path from every node in the network to every other node, which can be used to determine the full path. See section 3.5.1 for a full description.

Network



NetworkGenome

NodeGenome

node id	0	1	2	3	4
x-coord	1	4	3	1	4
y-coord	3	4	2	1	1

EdgeGenome

prim node	0	0	0	1	2	2	3
sec node	1	2	3	2	3	4	4
reliability	2	1	3	2	2	3	1

A Network and its corresponding NetworkGenome

3.3 Data Manipulators

3.3.1 The Crossover Operators

The crossover operator is regarded by most researchers, with a few notable exceptions, as the most important piece of a GA. It is the way in which the population of genomes breed, passing the structure of the parent genomes into the children, while mixing structures enough to ensure that the population does not stagnate at a single solution. Crossover is generally implemented as a 'blind' operator, in that the chosen technique for combination does not use any information specific to the problem.

A variety of different crossover operators were implemented in order to test the viability of a number of techniques in breeding genomes. While none of the techniques can be objectively said to be 'better' than any other, in terms of the specific problem certain approaches will show results that attain an optimal or near-optimal solution faster than others.

3.3.1.1 Single Point Crossover

Single point crossover is the simplest of the crossover techniques and is included here only for a better understanding of the more complex methods, especially the adaptation to accommodate variable-length genomes. In single point crossover, each of the two parents is split into two pieces – this split often occurs at the centre of the genomes, although it can take place at any point.

```
                Parent Genomes:

   Gene:  0-1   0-4   1-3   2-3   2-4   3-4

   Gene:  0-2   0-4   1-2   1-4   2-3   2-4

                Cross Point: 3

              Crossover Procedure:

 Gene:  0-1   0-4   1-3    |    2-3   2-4   3-4

 Gene:  0-2   0-4   1-2    |    1-4   2-3   2-4

                Child Genomes:

   Gene:  0-1   0-4   1-3   1-4   2-3   2-4

   Gene:  0-2   0-4   1-2   2-3   2-4   3-4
```

When genomes can vary in length, as is the case in the algorithm presented here, single point crossover using fixed cross points becomes slightly more complicated. In order to determine the correct location for the cross point in a genome of variable length, the *perc* array described in 3.2.1 is used. As this array contains floating-point numbers representing how far into the genome each gene is located, it can be used to place a fixed cross point in the same relative spot in two genomes of unequal lengths.

```
                    Parent Genomes:

Gene:  0-1   0-3   0-4   1-2   1-3   1-4   2-3   2-4   3-4

Gene:  0-2   0-4   1-2   1-4   2-4   3-4

                    Cross Point: .25

                    Crossover Procedure:

    Gene: 0-1 0-3  |  0-4 1-2 1-3 1-4 2-3 2-4 3-4
    perc:  .1  .2  |   .3  .4  .5  .6  .7  .8  .9

    Gene:     0-2  |  0-4 1-2 1-4 2-4 3-4
    perc:      .14 |   .28 .42 .57 .71 .85

                    Child Genomes:

   Gene:  0-1   0-3   0-4   1-2   1-4   2-4   3-4

   Gene:  0-2   0-4   1-2   1-3   1-4   2-3   2-4   3-4
```

Single point crossover is obviously useful mainly on problems for which the genomes are extremely small, usually containing less than 10 genes. For larger genomes, too much of each parent genome is preserved in its current form, leading to faster convergence of the population to a single, usually non-optimal solution. This effect can only be countered by high mutation rates, which cause problems in their own right. Single point crossover was implemented but never used for this project, given the large number of genes in each genome.

3.3.1.2 Multi Point Crossover

Multi point crossover is the intuitive next step up in complexity from single point crossover. Rather than merely splitting the genome into two pieces, it is split into three or more, with the exact number of chosen crossover points designer-dependant and usually chosen based on the length of the genomes. In the implementation used here, the number of crossover points is selected for each network size based on the number of nodes divided by a constant that is used for all tests. This has the result of splitting larger

networks into more pieces, a necessity to prevent premature convergence in the population. The genome is split into equal sections in order to maintain the determinism of the method – random crossover points would severely hamper the GA process of developing fitter individuals through breeding, due to the inability of the network to retain any sort of network structure from the parents to the child genome.

```
                    Parent Genomes:

Gene:  0-1   0-3   0-4   1-2   1-3   1-4   2-3   2-4   3-4

Gene:  0-2   0-4   1-2   1-4   2-4   3-4

        Equally-spaced Crossover Points: 0.33, 0.66

                    Crossover Procedure:

  Gene: 0-1 0-3 0-4  |  1-2 1-3 1-4  |  2-3 2-4 3-4
  perc:  .1   .2   .3  |   .4   .5   .6  |   .7   .8   .9

  Gene:      0-2 0-4  |       1-2 1-4  |       2-4 3-4
  perc:      .14 .28  |       .42 .57  |       .71 .85

                    Child Genomes:

   Gene:  0-1   0-3   0-4   1-2   1-4   2-3   2-4   3-4

   Gene:  0-2   0-4   1-2   1-3   1-4   2-4   3-4
```

As can be seen from the example, multi point crossover has a much more pronounced effect of blending the parent genomes than single point crossover, making it widely used in most research. Because of its general applicability, it is used as the baseline crossover operator that the other two more specialized techniques are tested against.

3.3.1.3 Node Based Crossover

Choosing a proper crossover function for the problem being solved is one of the most important steps in the creation of a GA. The choice must be made on the basis of the structure of the genome – what works well for a certain type of genome could have little effect on another, or worse, destroy any beneficial advances that are made through the evolutionary procedure. The multi point crossover technique is similar to that used in many implementations, including most of those specified in section 2.

After experimentation on the network design problem using this form of crossover, a different approach was tried for crossing genomes. Rather than dividing the genome up into equal sections and crossing these sections to produce children, the new

implementation uses variable crossover points representing the locations on the genome where the primary node of an edge (the node with the lower id) changed. The number of crossover points used in this method is equal to the number of nodes in the network minus two, meaning that networks with more nodes will have more crossover points. Implementing the crossover operation in this fashion ensures that links attached to a single source node stay together from generation to generation, and are combined with collections of links from other nodes. Aside from the different method of setting crossover locations and resulting different number of partitions in a genome, the alternative implementation functions identically to the multi point technique.

```
                    Parent Genomes:

 Gene:  0-1   0-3   0-4   1-2   1-3   1-4   2-4   3-4

 Gene:  0-2   0-4   1-2   1-4   2-3   2-4   3-4

                  Crossover Procedure:

0-1  0-3  0-4   |   1-2  1-3  1-4   |   2-4       |   3-4

0-2  0-4        |   1-2  1-4        |   2-3  2-4  |   3-4

                    Child Genomes:

 Gene:  0-1   0-3   0-4   1-2   1-4   2-4   3-4

 Gene:  0-2   0-4   1-2   1-3   1-4   2-3   2-4   3-4
```

While it can be reasoned that this technique will result in larger collections being attached to low-valued nodes and smaller or non-existent collections attached to higher-valued nodes, in a range of tests this node-based method of crossover showed better results than the simpler multi point method.

3.3.1.4 Edge Based Crossover

The last method of crossover implemented is based on the gene mixing technique used by Philippides (1996). A new genome is formed based on the genes in the parent genomes by looking for identical genes in both genomes. If a gene is present in both parents, that gene is added to the child with a probability value *p1*. If the gene is present in only one of the parents, it is added to the child with probability *p2*, where $p2 < p1$.

```
                    Parent Genomes:

Gene:  0-1   0-3   0-4   1-2   1-3   1-4   2-4   3-4

Gene:  0-2   0-4   1-2   1-4   2-3   2-4   3-4

                Possible Child Genomes:

Gene:  0-3   0-4   1-2   1-3   1-4   2-3   2-4   3-4

Gene:  0-2   0-4   1-2   1-4   2-4   3-4
```

This technique is very likely to maintain good structures that are shared between the parent genomes, making it particularly useful as the population approaches an optimal solution, while still leaving open the possibility of extraneous genes remaining present in the children to prevent total convergence in the population. Different values for *p1* and *p2* were tried, and in the end *p1* = 1.0 and *p2* = 0.5 were selected. These values make it likely that a child genome will maintain the same length as the parents, thus relying on the mutation operator and local search, when present, to remove unnecessary edges from the network. This is not an issue, as the genomes that are initially placed into the population before the GA is started are created to have between 3 and 7-connectivity, a range the optimal solution for a network nearly always falls into.

3.3.2 The Mutation Operators

3.3.2.1 Single Point Mutation

Mutation operators are used in GAs to ensure that diversity is maintained throughout the entire run. There are a limited number of possible combinations that can be produced in a population by the crossover operator – if a particular gene is not present in any genome in a population at initialisation, there is no way for it to be introduced during the run. Mutation allows each genome in the population to be extended, reduced, or changed in various ways, subject to a probability selection. Mutation levels are generally kept very low to prevent a genome from 'jumping' in the solution space from one area to a significantly different one. Rather than being a technique for guiding the population toward an optimal solution, mutation exists solely to prevent genomes from getting stuck in local minima.

Mutation is implemented here in a variety of ways. A single point mutation operates on the principle that only one gene from the entire genome is altered. The gene to be altered is randomly selected from the genome before the mutation operator is initialised.

The actual mutation function allows for five equally probable alterations to the genome. They are as follows:

- A new edge, randomly chosen from the available (i.e. currently absent) edges is added to the genome, extending its length by one gene. This is a fairly major mutation to the genome.

- One of the edges currently in existence in the network is removed from the genome, reducing the genome's length by one gene. This is another major mutation.

- The primary node (the node with the lower id value) of the edge corresponding to the selected gene is altered. This mutation maintains the length of the genome, but alters the current network topology slightly, making it a somewhat average mutation.

- The secondary node (the node with the higher id value) of the edge corresponding to the selected gene is altered. This operation is nearly identical to the previously described mutation, existing solely to ensure that either end of an edge can be moved.

- The reliability rating of the edge corresponding to the selected gene is changed. This is a minor mutation, as it only affects the reliability value assigned to the network, leaving the physical topology unchanged.

A random, equally weighted selection of the mutation to be applied is made despite the varying levels of severity in the mutations, mainly because it is impossible to know at the time of the mutation how much a genome should be changed in order to differentiate it from the rest of the population. In addition, initial tests showed that equally weighting the probability of selecting any mutation method gave better overall results in the efficiency of the GA.

3.3.2.2 Multi Point Mutation

The multi point mutation operator is the one used by the GA implemented here – rather than being an alternative to single point mutation, it merely calls that method on multiple genes in a single genome, the exact number of which is determined based on the length of the genome. In a standard GA the mutation operator is only applied to a small percentage of the population of genomes to prevent too much variation interfering with the development enacted by the crossover operator. In this project, a value of 0.1 was chosen as the mutation rate, as tests showed this value to give the best results.

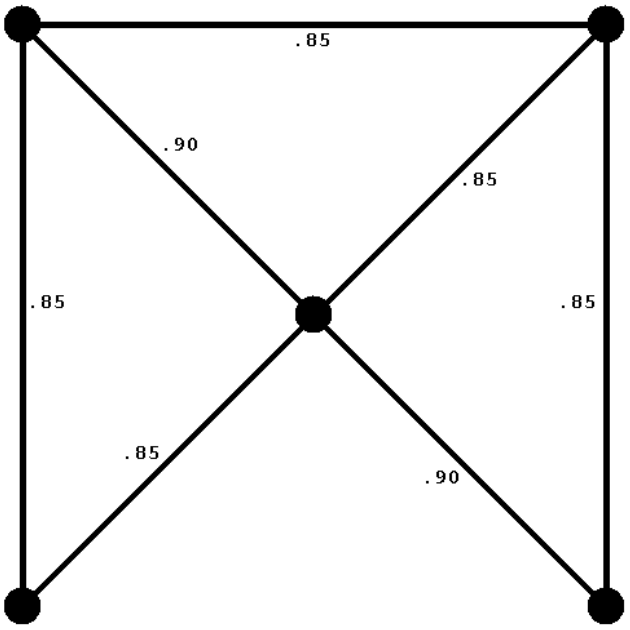
3.3.3 The Local Search Operator

Local search was implemented in this project as a form of Lamarckism through directed mutation. This method was chosen over more complex forms of local search, such as hill climbing, due to its low processing overhead. Hill climbing is feasible in small networks, where the fitness is quickly and easily found, given the relatively small number of nodes and edges. In large networks, however, searching the all of the space around a particular solution would be an extremely time-consuming and processor-intensive method of local search, given the complexity of converting a genome to a phenome, calculating the properties of the network, and finding the fitness of larger networks.

Because of this, the implementation of local search uses the pre-defined single-point mutation operator to search for better solutions. A mutation is made, and the resulting network is evaluated for its fitness. If the fitness of the altered network is greater than that of the original, the altered form is kept in the population while the original is discarded. If the fitness of the altered network is lower than that of the original, the altered form is disposed while the original is retained. This process is repeated until an altered network is of a lower fitness rating than the original, at which point the current network in the population is retained for the current generation. This technique allows for great improvements to be made to very unfit genomes, while not affecting genome with higher fitness levels very much. Because the local search operator normally only makes a few changes before trying a mutation that reduces the fitness level of the network, local search is not overemphasized, and is less likely to lead the population into local minima.

3.3.4 The Prior Knowledge Operator

Prior knowledge was implemented in the way described in section 1.3.3. Specifically, a network of five nodes was constructed using an easily recognisable physical layout, which was then run through the GA to find an optimal topology given a reliability requirement of 0.95. This reliability rating was chosen in order to prevent this sub-network from contributing too much to the cost of the encompassing larger network. The optimal network that was found by 90% of the 20 test runs is shown below.



Each occurrence of this physical layout of five nodes within the larger network is then found, with overlapping of the sub-networks prevented, and the edges corresponding to the optimal layout of the sub-network are inserted into the larger network with the same reliability levels seen in the sub-network. A small amount of leeway is provided on the locations of each of the nodes, so as not to restrict the application of the sub-network to instances in the larger network that match it exactly. This leeway is kept small to prevent interference with the optimality of the sub-network topology.

Once every occurrence of nodes matching the layout of the five node sub-network has been found and configured appropriately, the rest of the nodes in the network are randomly connected to the sub-networks and each other until the initial connectivity requirement is met. The final result is a genome that is in large part randomly generated, like a standard genome without the prior knowledge operator applied, but with a percentage of nodes already optimally configured with the proper edges. This should result in initial genomes that are higher in fitness than randomly initialised genomes, and as a result should increase the speed of the GA. Obviously choosing larger sub-networks would increase the fitness levels of the initial genomes more, but the number of matching

sub-networks in the larger networks would also likely drop proportionally.

One potential danger with the addition of prior knowledge to the initial genomes in a population is premature convergence, which occurs when the majority of the population reaches the same sub-optimal solution. This makes crossover operators much less functional, as they end up simply breeding genomes that are identical, or nearly so. Because of this, the population evolves much slower than a non-converged population, if at all. Care must be taken to ensure that prior knowledge is not overemphasised at the cost of fewer edges in the network being randomly initialised to promote diversity in the population. For this reason, application almost always takes place on networks with many more nodes than are used in the operator.

3.4 Data Evaluators

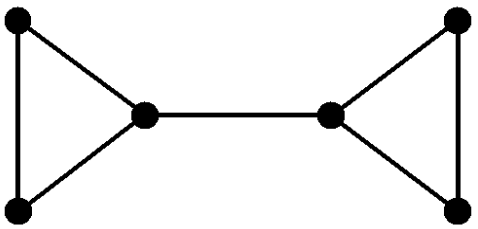
3.4.1 The Network Reliability Calculation Functions

Network reliability is calculated through a series of checks and increasingly exact estimations. Infeasible networks can be defined as those which are composed of two or more components, where a single component represents an independent connected network. Feasible networks are those which contain only one component – these networks are guaranteed to have a valid path between any two nodes.

The first check is for network feasibility, which is determined through a recursive function that steps through the entire network checking for connections between nodes. If this test for network connectivity is passed, the network is passed to the next check. If the network is found to contain more than one connected component, it is assigned a reliability rating of 0, as it is in no way useable as a solution.

The second check is for the k-connectivity of the network. It is necessary to ensure that if a certain number of links fail fails, regardless of how small that chance is, there is still at least one link running to every node, e.g. a 5-connected network can sustain up to 5 individual link failures before any one node is disconnected from the rest of the network. To ensure that this is the case for every node in the network, the k-connectivity algorithm (see section 3.5.2) is used to calculate the total network k-connectivity, which is equal to the k-connectivity of the least-connected node. If the network is found to have a total k-connectivity of at least the minimum level specified it is passed to the first reliability estimator. If the network does not meet the required level of connectivity it is assigned a reliability rating of 0, to ensure that the optimal network found by the GA meets the defined connectivity requirement.

Many previous studies have ended their network reliability calculations at this point, assuming that if a network has *k*-connectivity, it can sustain *k*-1 link failures before becoming disconnected. While it is the case that *k* links can fail before any single node becomes disconnected, it is not the case that *k* links can fail before the entire network becomes disconnected. In the image below, every node is 2-connected, but it would take only the failure of the central link to disconnect the network. This situation is fabricated solely for this demonstration, but it is easy to see how it could arise in large networks where small sections are connected to the rest of the network by only one or two links.



The first reliability estimator is a test for the upper-bound reliability of the network using the method described in section 2.2 by Konak and Smith (under revision) for determining upper-bound reliability for networks with links of different reliability ratings. This test approximates the reliability of the network to a close degree while running in polynomial time. Before initialisation of the population of genomes, the GA is given a minimum reliability rating for the optimal network – the highest potential chance that there is of a network failing. If the upper-bound approximation of the network's reliability meets or exceeds this minimum reliability requirement, the network has a chance of meeting the requirement in a more exact reliability calculation, and is passed to the final estimator. If the upper-bound approximation does not meet the requirement, the reliability rating is set as that upper-bound value – below the minimum requirement, but signifying how close the network is to meeting it.

The final reliability estimator is the variant Monte Carlo technique described by Yeh et al (1994). This estimator has been shown in several studies to provide a reliability rating that is within 0.01% of the exact reliability of the network, given enough repetitions of testing. While the method is still far more processor-intensive than an upper-bound approximation of reliability, it is also far less intensive than an exact calculation, and the extremely small degree of error makes it perfectly suitable in this situation. The exact number of repetitions required for an acceptable level of precision is based on the number of nodes in the network, a calculation outlined in the original research. Networks that have made it to this stage are passed to the method and have their reliability estimated almost exactly. The final reliability of these networks is set to the value found by this algorithm.

3.4.2 The Fitness Function

The fitness function is the defining operation of a genetic algorithm. By ranking each member of the population against the others by a measure of their fitness according to the constraints put in place by the designer, this method is responsible for almost every aspect of the population composition, as well as the functionality of the selection operator, which bases its methods upon the ranking of the genomes. For this reason, a great deal of care must be taken in designing the fitness function – most are configured specifically for the problem that the GA is trying to solve.

For this project, the fitness function has been designed to account for three properties of the network being tested. The first of these is the cost of the network, measured in terms of the edges as:

$$\sum_{i=1}^{n-1} \sum_{j=2}^{n} C_{ij}$$

where C_{ij} is the cost of the edge between nodes i and j, for $i = 1 \ldots n$ and $j = 2 \ldots n$, n being the number of nodes in the network. If there is no connection between node i and

node *j*, the cost for that edge is 0.

The second property is the average path length between any two nodes in the network. This property is included to prevent the network from forming highly unlikely 'optimal' solutions, such as ring topologies, as well as to account for latency in communication between nodes that is proportional to distance. This calculation uses the shortest path structure, and is measured as:

$$\frac{\sum_{i=1}^{n-1}\sum_{j=2}^{n} L_{ij}}{E}$$

where L_{ij} is the length of the shortest path between nodes *i* and *j*, for $i = 1 \ldots n$ and $j = 2 \ldots n$, *n* being the number of nodes in the network, and *E* being the number of active edges in the network. If there is no valid path, i.e. the network is disconnected, L_{ij} is equal to the maximum value for a *double* value in Java, 1.7976931348623157e+308.

The third part of the fitness calculation is a penalty *P* based on the reliability, or rather the unreliability of the network. The network is first tested for its reliability rating using the process described in section 3.4.1. If this rating meets the minimum value specified by the designer, the network is assigned a penalty of 0. If the reliability is below the minimum requirement, the value of *P* increases as the reliability decreases. The exact rate of increase can be tuned in order to emphasize or deemphasize network reliability. This is one of the aspects of scaling the GA to larger networks – higher reliability levels must be maintained through the run in order to ensure that the removal of a small number of edges would not disconnect the network and render it infeasible. This is not an issue with smaller networks and their limited number of potential edges, but in networks with hundreds of thousands of potential edges, it could render the network all but irreparable.

The final fitness function is therefore a combination of these three aspects of the network, and can be calculated as:

$$\left(\sum_{i=1}^{n-1}\sum_{j=2}^{n} C_{ij} \bullet \sum_{i=1}^{n-1}\sum_{j=2}^{n} L_{ij}\right) + P$$

In the implementation here, the GA libraries provided by Codefarm Inc., are not yet optimized for minimization, so the calculation is simply inverted to give a maximization version:

$$-\left(\sum_{i=1}^{n-1}\sum_{j=2}^{n} C_{ij} \bullet \sum_{i=1}^{n-1}\sum_{j=2}^{n} L_{ij}\right) - P$$

3.5 Other Algorithms

3.5.1 Shortest Path Algorithms

"A shortest path between two vertices s and t in a network is a directed simple path from s to t with the property that no other such path has a lower weight [shorter distance]*."* (Sedgewick, 2002)

There are three types of shortest path structures within every network. These are as follows:

- **Source-sink shortest path** – this is the path defined by Sedgewick above, and the most basic shortest path structure. The starting node *s* is referred to as the source, and the destination node *t* is known as the sink node.

- **Single-source shortest paths** – for any starting node *s*, the single-source shortest path is the collection of all source-sink shortest paths from the source to all other nodes in the network.

- **All-pairs shortest paths** – this is the collection of all shortest paths between all nodes in the network.

An important point about shortest paths is that distance is measured purely by the length of the edges on the path, not in any way by the number of edges in the path. Because of this, it is possible, and common, for a shortest path to cover more edges in a network than another, less direct path that may only use two or three edges. Switching delay at each node is not taken into account in this implementation, as its presence or absence is totally dependent upon the specific situation.

3.5.1.1 Floyd's Shortest Path Algorithm

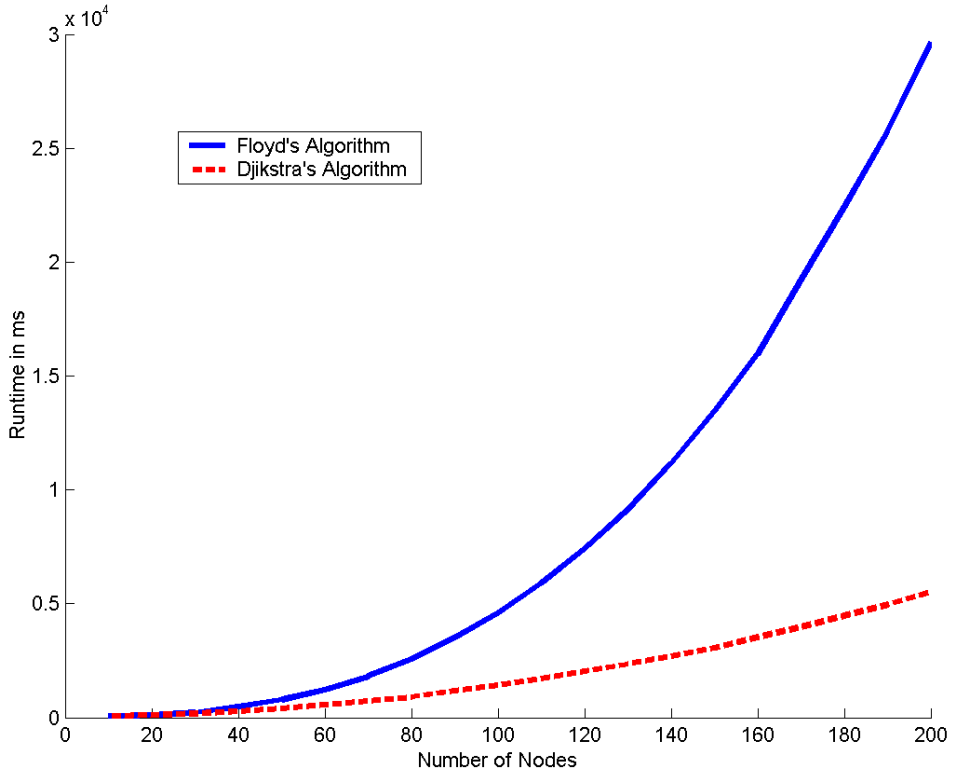
Floyd's algorithm for finding the all-pairs shortest paths in a network solves the problem directly in time proportional to V^3, where V is the number of nodes, or vertices, in the network. This method is preferred in dense networks, where the large number of edges is inconsequential to the running time of the algorithm.

The algorithm keeps two V by V matrices, one containing the length of each shortest path, the other containing the first edge on each path in the corresponding location in the matrix. The shortest path can then be found by going to each node in the matrix on the other end of the edge representing the next step on the path until the destination node is reached.

Because of its large time requirements on sparse networks, as most of the configurations examined here are, Floyd's algorithm was only used for small networks of less than ten nodes, where the ratio of edges to nodes is large.

3.5.1.2 Djikstra's Shortest Path Algorithm

Djikstra's algorithm for finding the all-pairs shortest paths in a network is simply an extension of his algorithm for finding the source-sink shortest path between a pair of nodes. It is implemented using a priority queue based on a *d*-ary heap, and can find all the shortest paths in a network in time proportional to $VE \log_d V$, where $d = 2$ if E, the number of edges in the network, is less than twice the number of nodes, and $d = E / V$ in all other cases. Because it uses the value of E in the equation, it can show poor performance on densely-connected networks – on the networks used here, however, it outperforms Floyd's algorithm by a considerable margin.



The source-sink algorithm functions by first placing the source node onto the shortest path, then building the path one edge at a time by adding nodes to the path in order of their distance to the starting node. This algorithm operates in time proportional to $E \log V$, hence the all-pairs shortest paths algorithm's $VE \log V$ running time. Because of its greatly improved performance on large, sparse graphs, Djikstra's algorithm was used for nearly all of the experiments performed here.

3.5.2 K-Connectivity Algorithm

The algorithm to calculate the k-connectivity of a network is simple, yet vitally important to almost all aspects of the GA. especially the fitness function. It provides a way for any program using the Network class to determine the degree of any specific node in the network, and using this functionality, to determine the k-connectivity of the entire network. The degree of a node is defined as the number of edges connected to that node. The k-connectivity of the network is then determined by finding the node with the least number of edges emanating from it.

The implementation is a simple one, merely iterating through each edge present in the network and incrementing the degree of each node connected to the current edge. This simplicity means that the algorithm runs in linear time proportional to E. However, because the algorithm has to be run every time an edge is added, removed, or has one of its connected nodes changed, the k-connectivity algorithm is run almost every time there is a mutation, and almost constantly during the local search process.

4. Results

The GA was tested on a variety of networks from ranging in size from 10 to 200 nodes in order to determine its effectiveness at scaling from small networks to very large ones. For each network size, 5 random node configurations were generated, each of which was tested 10 times for each combination of search techniques (see below). The average performance of the GA on each of the networks was averaged, and the average performance for each of the 5 networks was averaged to give the final results for each network size. Population size was 64 genomes, mutation rate was 0.1, and crossover rate was 1.0.

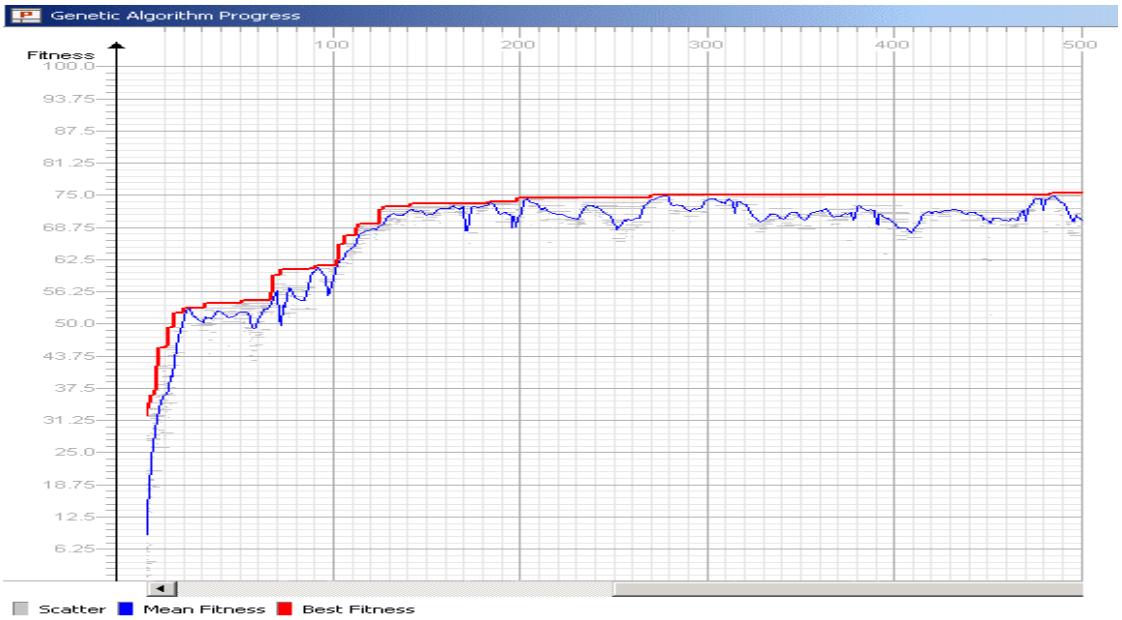
All three forms of crossover – multi point, node based, and edge based – were tested on the small and medium sized networks (10 – 50 nodes). Only the two better performing crossover techniques, node based and edge based were used for testing the large networks (100 – 200 nodes), due to their significant improvements over standard multi point crossover in the smaller networks. All networks were tested with and without local search enabled; in addition, the large networks were tested with both local search and prior knowledge enabled.

While only the first 500-700 generations are shown in the images, this was done mainly to conserve space, since past these points only small improvements were seen made to the populations in most tests. Each test was run for at least 2500 generations, by which point all of the GAs seemed to have found an optimal solution.

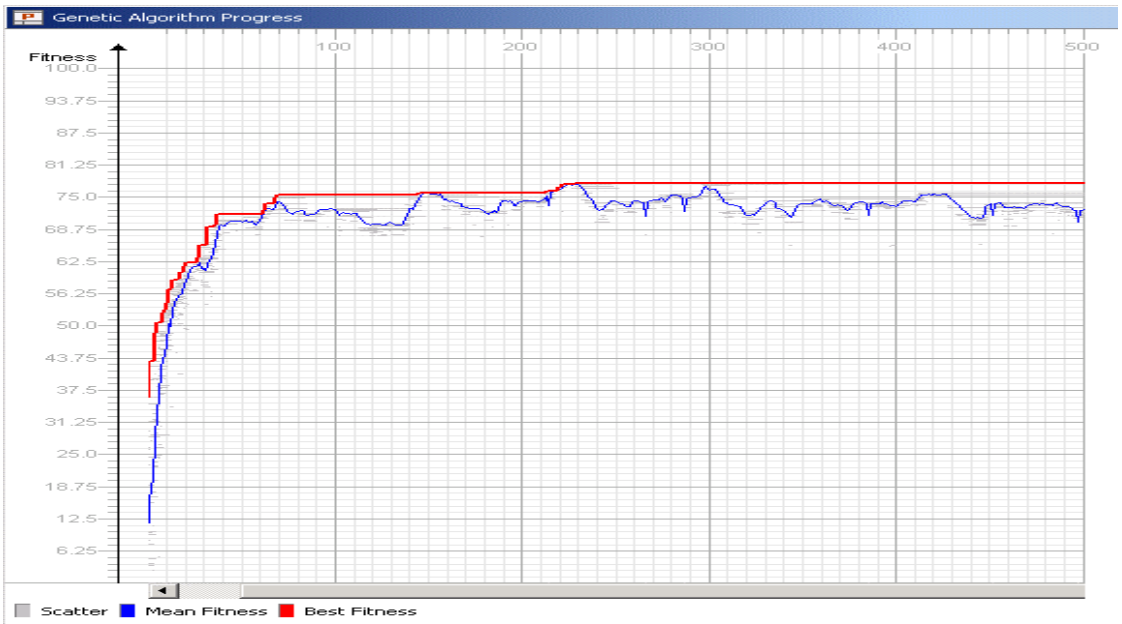
It should be noted that the values for the fitness rating are not indicative of any definable measurement, but are included merely to offer a comparison of the relative fitness of networks as the size is increased. In other words, a 100 node network with a fitness rating of 80 has approximately the same reliability level, cost to maxcost ratio, and average path length proportional to network size as a 10 node network with the same rating. In addition, the size of the area required to store a network increases based on the number of nodes in the network. Network costs and average path lengths increase because of this, so the cost to maxcost ratio is included as an indication of how well the best found solution compares to the fully-connected network.

4.1 Small Networks (10 nodes)

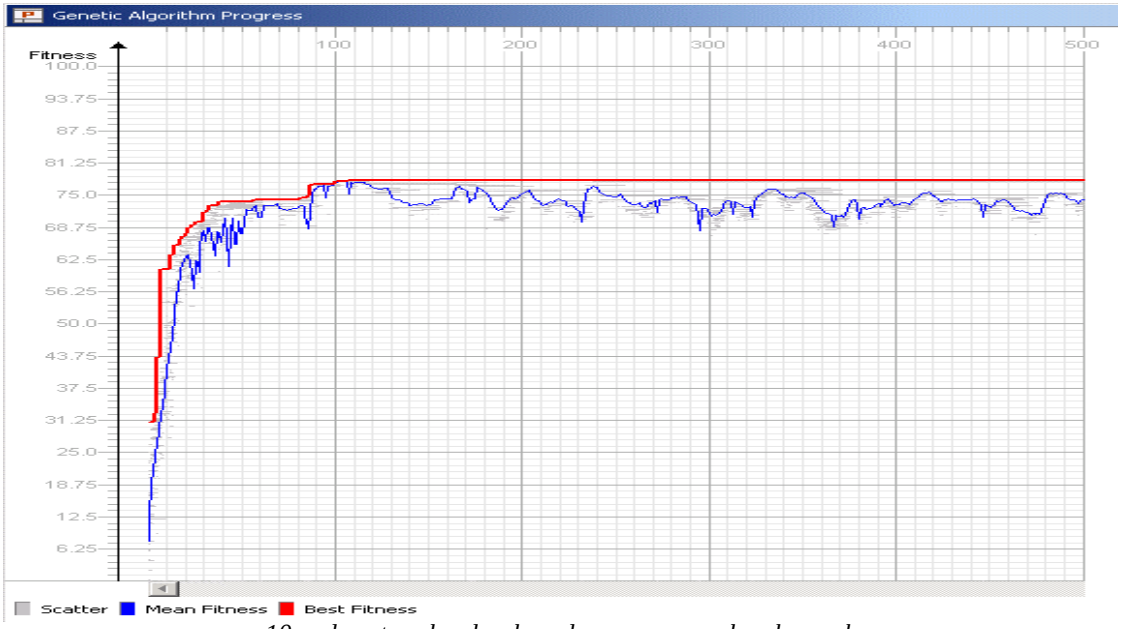
10 node networks were the smallest tested by the GA for this project. Each of the crossover operators was used, with and without local search implemented.



10 node network, multi-point crossover, no local search



10 node network, node-based crossover, no local search

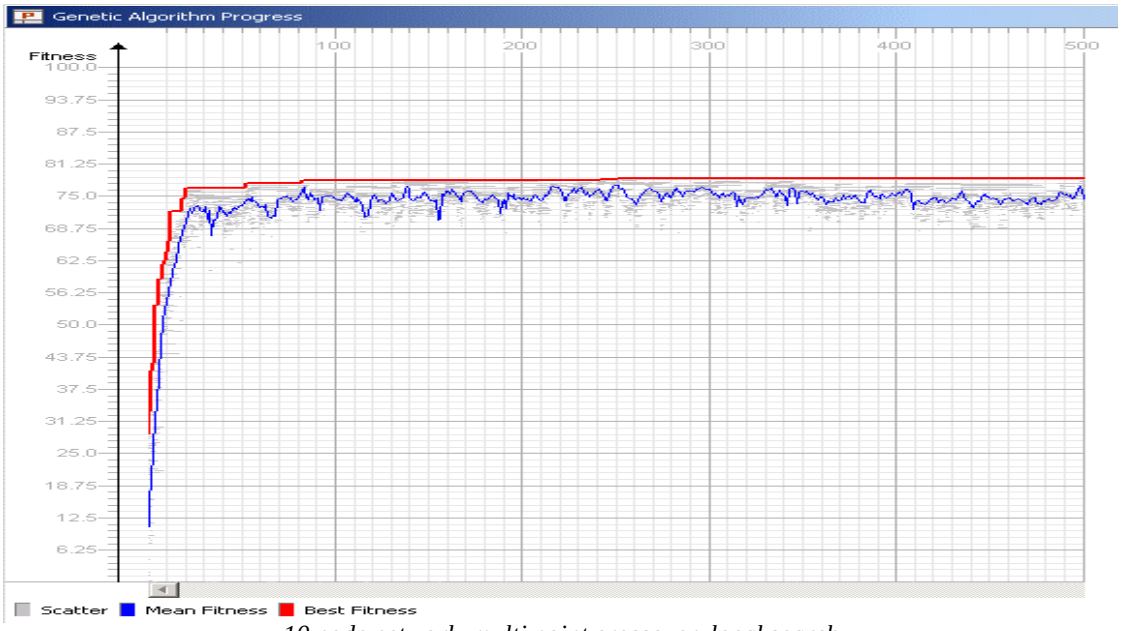


10 node network, edge-based crossover, no local search

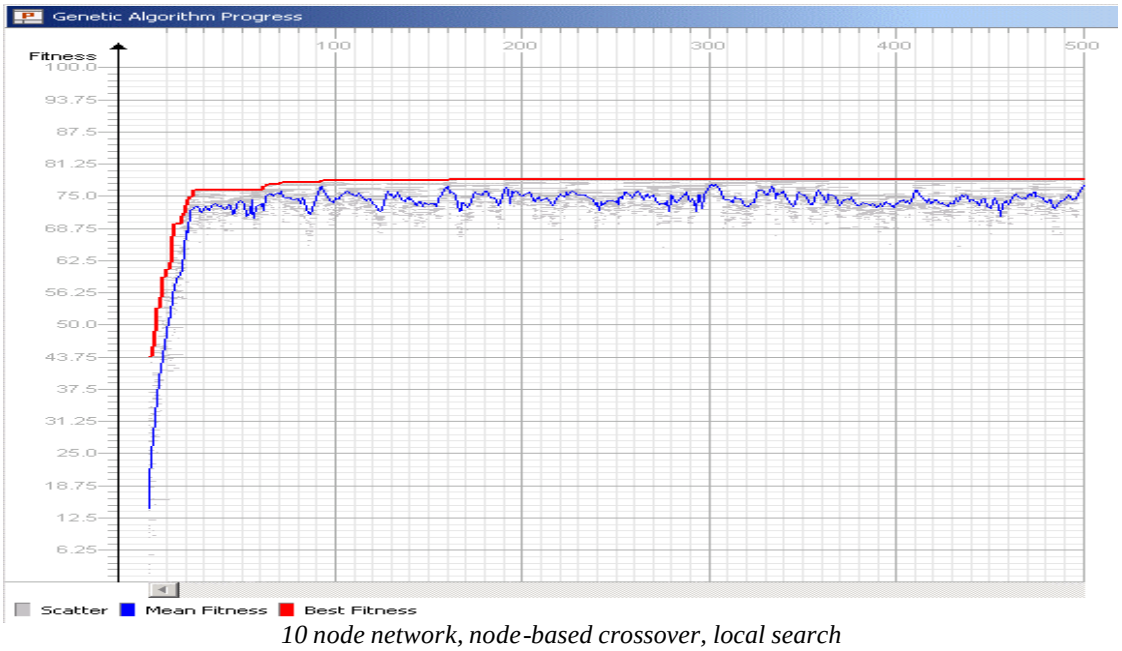
Without local search implemented, the most obvious effect that can be seen between the runs of the GA is the small but noticeable improvement that node based and edge based crossover offer over standard multi point. While almost all of the tests using multi point crossover eventually reached the same optimal level as the others, these tests took up to 50% more generations to do so.

Node based crossover shows a smoother motion for the average fitness levels than the other two techniques, due to its nature of working with collections of nodes connected to the same node, thus keeping cost and reliability measures of various physical sections of the network static. This is unlike the other two methods, where edges can be individually added to or removed from any area of the network at any time, independent of the other edges around it.

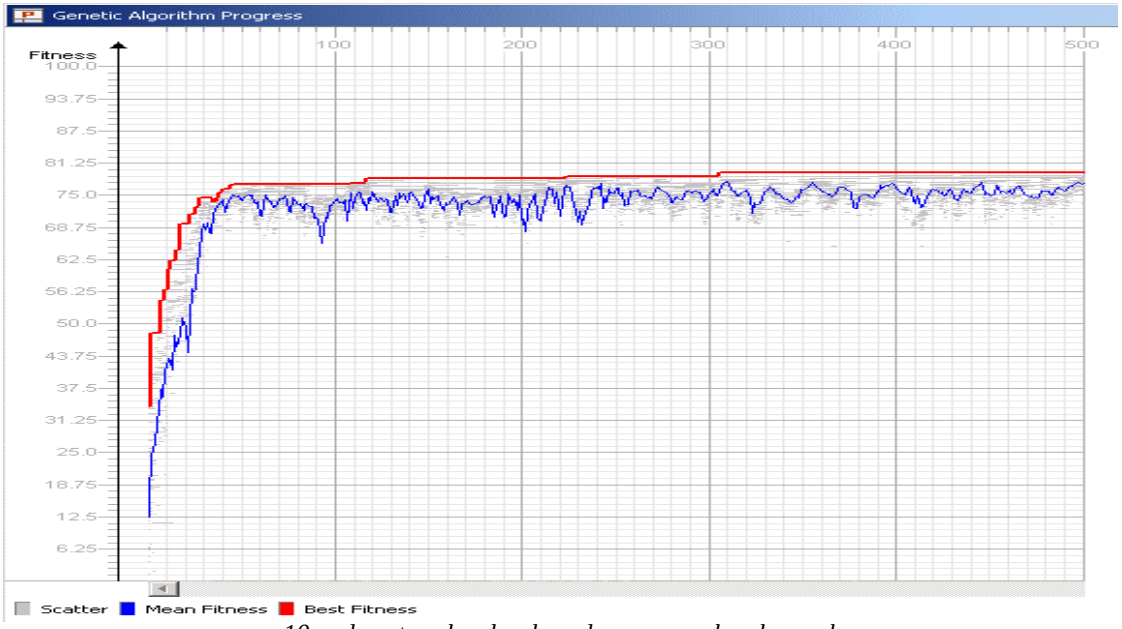
4.1.1 The Addition of Local Search



10 node network, multi-point crossover, local search

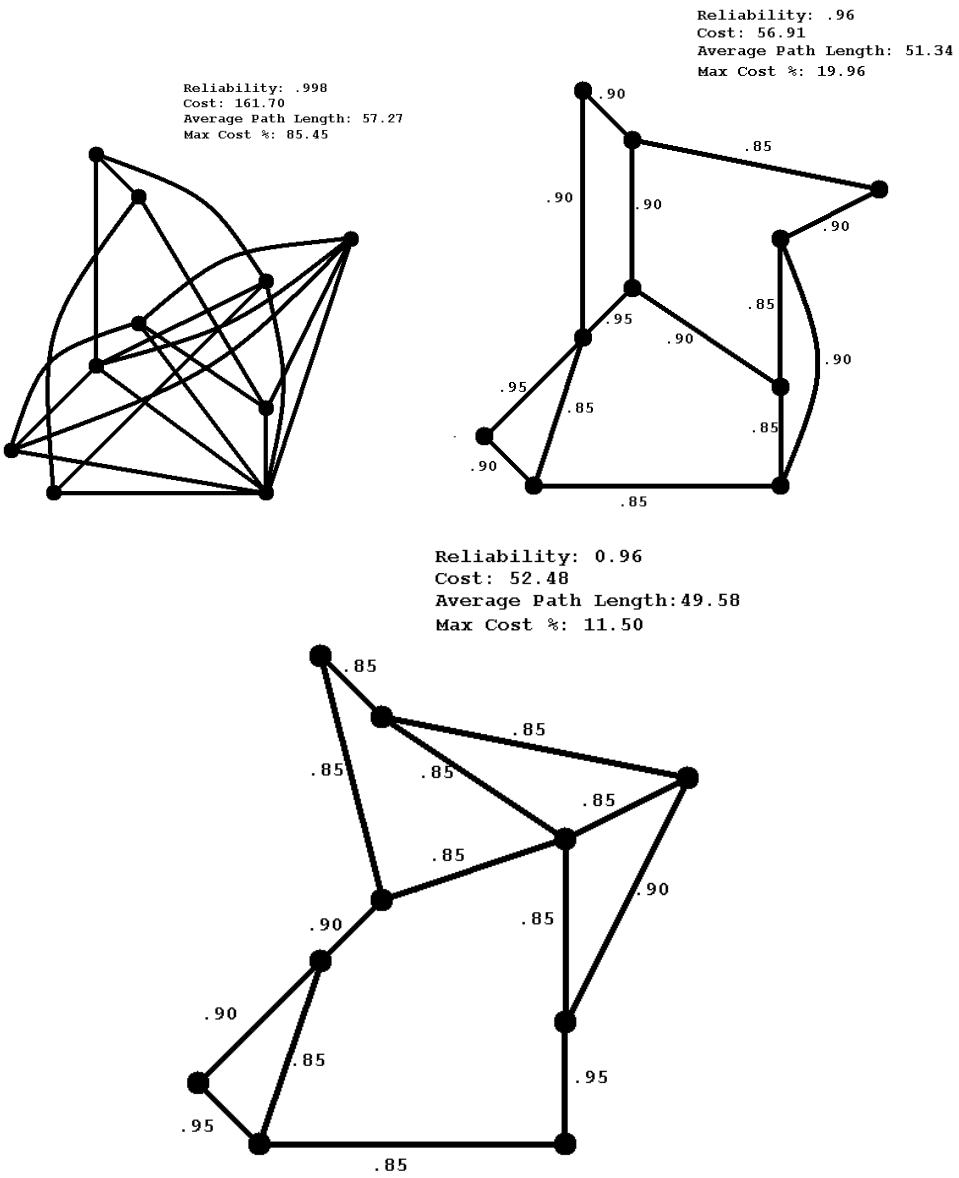


10 node network, node-based crossover, local search



10 node network, edge-based crossover, local search

With local search implemented, the form of crossover that is used becomes almost inconsequential, with each test demonstrating a steep climb in the first 50 generations, followed by minor increases when local search stops having a major effect on the population.



3 stages of network development

These images show three stages in the development of an optimal network topology for one of the tested random 10 node configuration with a minimum reliability rating of 95%. The first network is the phenome for one of the randomly initialised genomes of the initial population. While it has an extremely high reliability rating based on the large number of edges, its cost to maxcost ratio is extremely high, giving it a low overall fitness rating. In the second network, the number of total edges has been trimmed down considerably at the cost of the reliability rating – the rating is still above the minimum, however. A few extraneous edges remain, such as the one on the far right represented by a curved line. The final optimal network, which was found by almost every test, is shown in the third image. Mutation has shifted the edges slightly and adjusted the reliability levels, allowing the network to maintain short paths between

nodes while still trimming out all unnecessary edges.

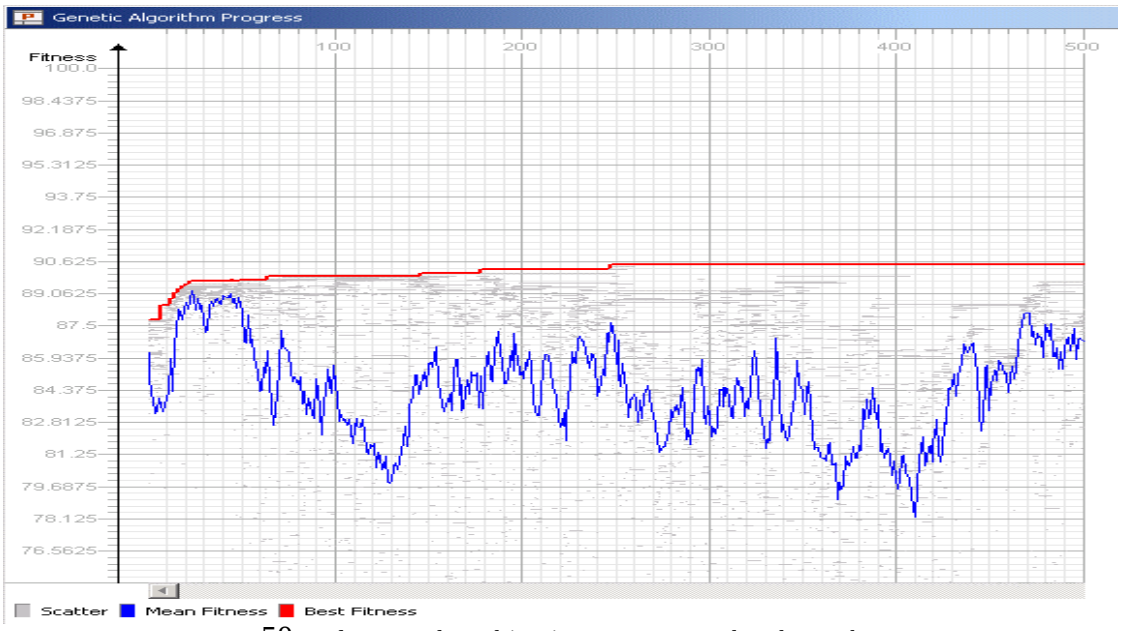
The average optimal network found for a 10 node network configuration with a minimum reliability requirement of 95% had the following properties.

K-Connectivity:	2
Reliability:	95%
Average Path Length:	48.23
Cost:	49.71
Cost / MaxCost ratio:	10.19%

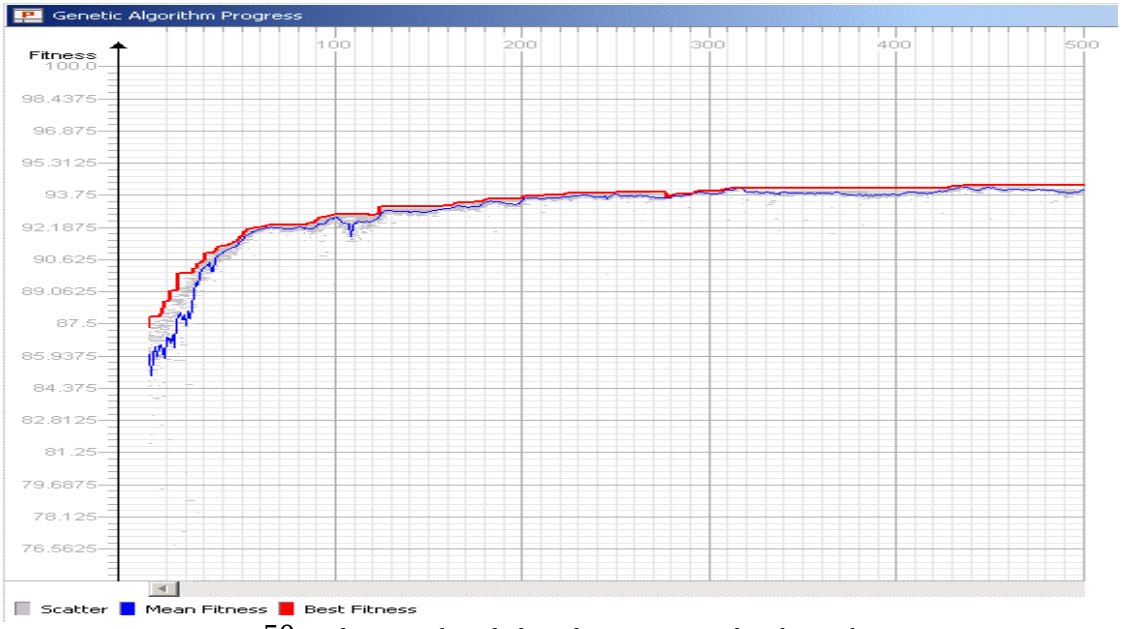
4.2 Medium-sized Networks (20 - 50 nodes)

The first scaling of the algorithm to accommodate larger networks takes place here. Scaling techniques are applied, such as increasing the reliability rating in order to lessen the chances of infeasible networks being developed that are difficult to repair through a GA. Penalty modifiers are increased, effectively making it more of a priority for the GA to develop reliable networks. The k-connectivity of the initial randomly generated networks is increased as well, from 3 to 5, to ensure that the population does not enter an inescapable downward spiral if too many edges are removed near the beginning of the development.

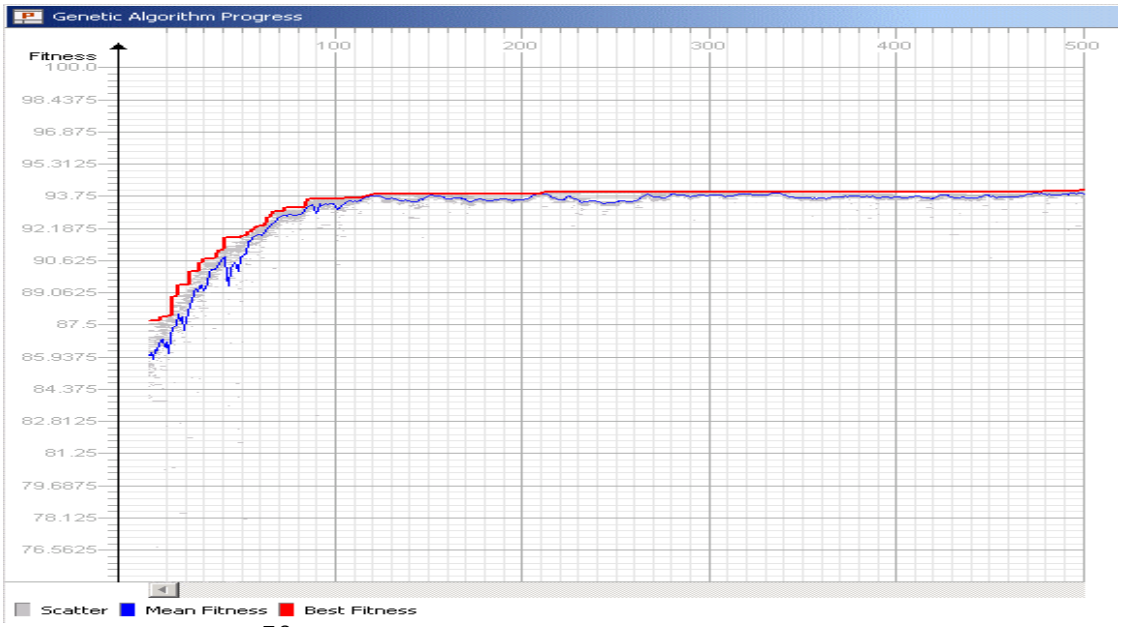
Only 50-node network results are shown here, as 20-node networks show the same patterns, only less pronounced. See Appendix A for full 20-node networks results.



50-*node network, multi-point crossover, no local search*



50-*node network,node-based crossover, no local search*

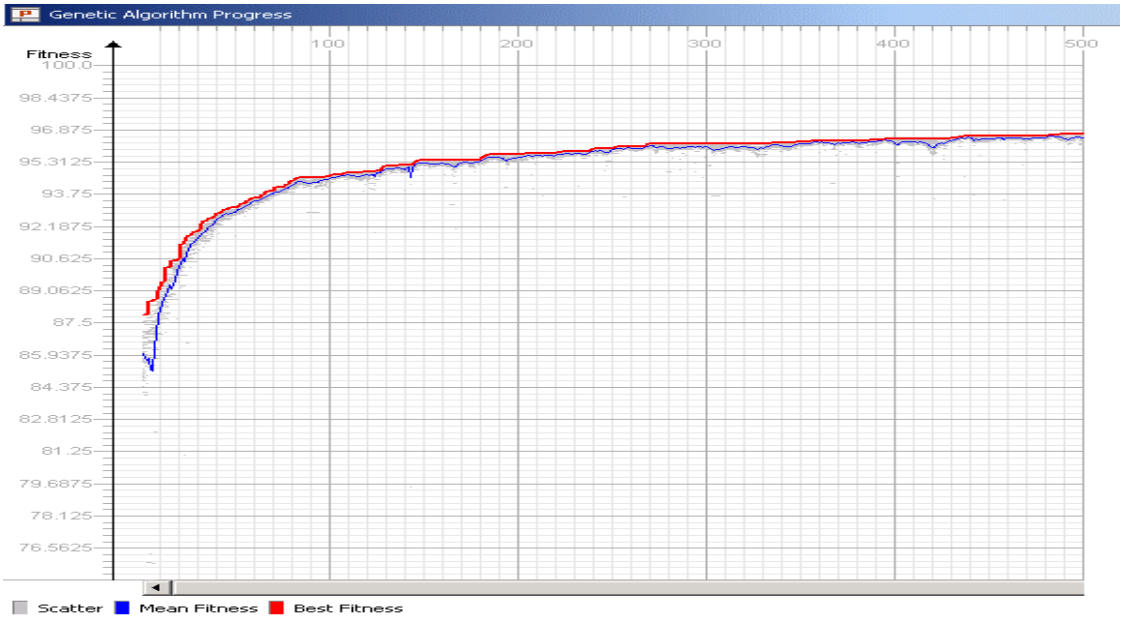


50-*node network, edge-based crossover, no local search*

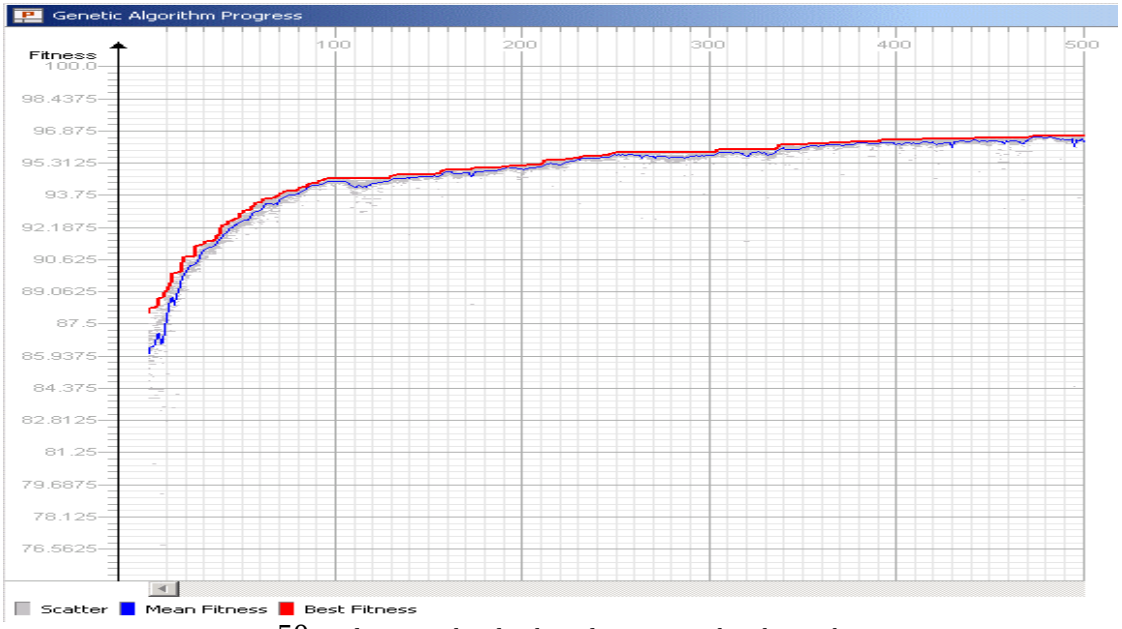
Easily the most noticeable effect here is the much poorer performance of multi point crossover when compared to the other two forms. Although the population is much more diverged in the tests using multi point crossover, this does not appear to be having an effect, whereas the much more converged populations of the node based and edge based crossover runs give better results. This could be caused by the size of the solution space, and the fact that very different solutions could easily return similar fitness ratings.

4.2.1 The Addition of Local Search

50-*node network, multi-point crossover, local search*



50-*node network, node-based crossover, local search*



50-*node network, edge-based crossover, local search*

Again, abysmal results are seen from standard multi point crossover, this time with the local search operator enabled. Very good networks are obtained from both of the other forms of crossover, with neither clearly showing an advantage over the other, and both finding the same likely optimal network. Multi point crossover, on the other hand, never found a solution with anywhere near as high a fitness level as the other two, and appeared to stagnate by the 400th generation. This exemplifies the need for a problem-specific crossover function for this problem. The node based and edge based crossover techniques are both designed to work specifically with genomes based on networks, and are therefore able to perform necessary functions such as retaining structure through crosses, making feasibility more likely. Because of the poor performance shown by multi point crossover in tests up to this point, these were the last tests for which it was used.

The average optimal network found for a 20 node network configuration with a minimum reliability requirement of 97.5% had the following properties.

K-Connectivity:	3
Reliability:	97.7%
Average Path Length:	221.39
Cost:	212.52
Cost / MaxCost ratio:	5.0%

The average optimal network found for a 50 node network configuration with a minimum reliability requirement of 95% had the following properties.

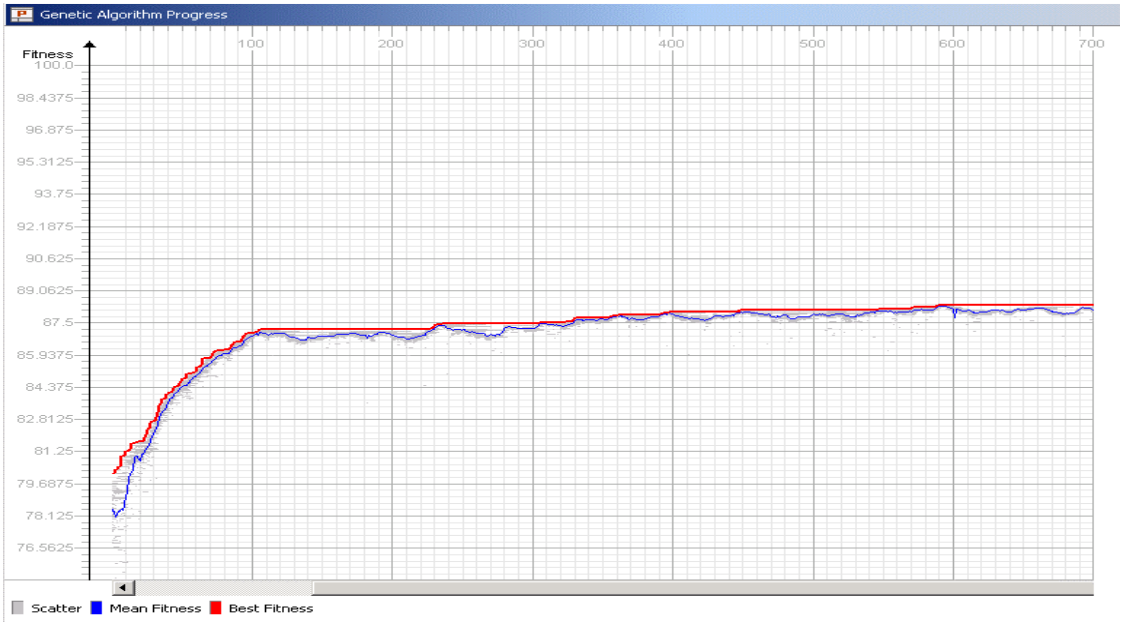
K-Connectivity:	5
Reliability:	98.1%
Average Path Length:	1365.77
Cost:	1781.19
Cost / MaxCost ratio:	3.9%

4.3 Large Networks (100 - 200 nodes)

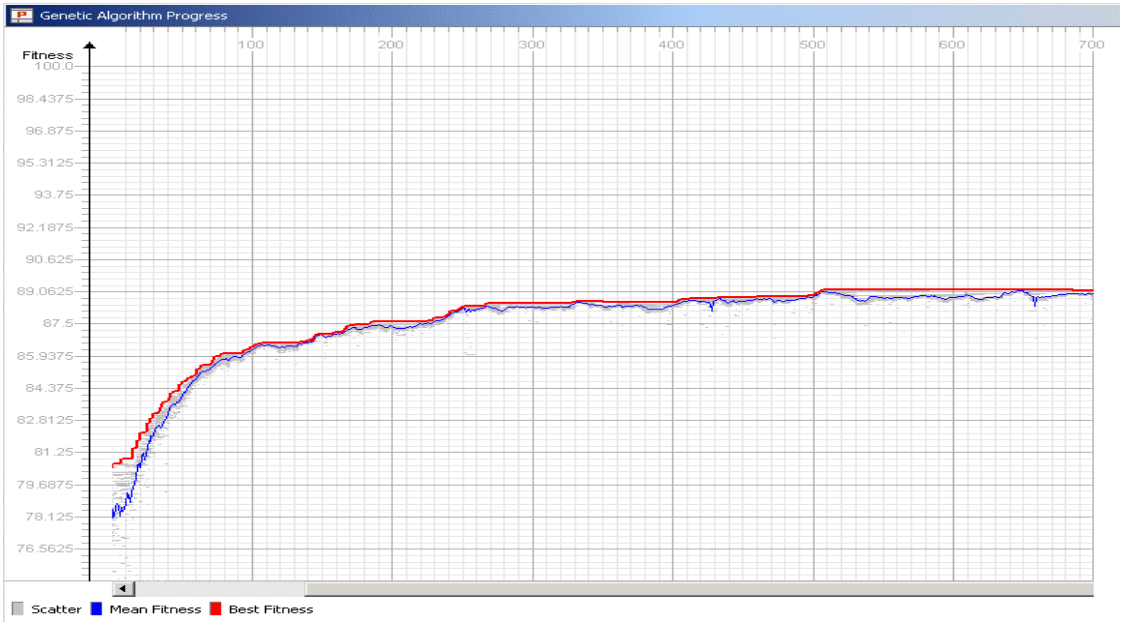
More scaling along the previous lines was required to develop networks with 100 or more nodes. Penalty modifiers were increased again, as well as the minimum reliability rating – without these increases, average fitness levels tended to grow at far slower rates, due to the higher ratings of less reliable networks and their corresponding spread through the population. The initial k-connectivity is increased again, from 5 to 7, again ensuring that the population manages to reach a stable range of network configurations before too many edges are removed and some networks become infeasible.

The prior knowledge operator is enabled for the first time for both 100 node and 200 node networks – at smaller sizes there was too much chance of the information given dominating the network and causing premature convergence in the population.

As before, only the larger of the two tested network sizes is shown here, due to similarities. See Appendix 1 for 100 node network graphs.



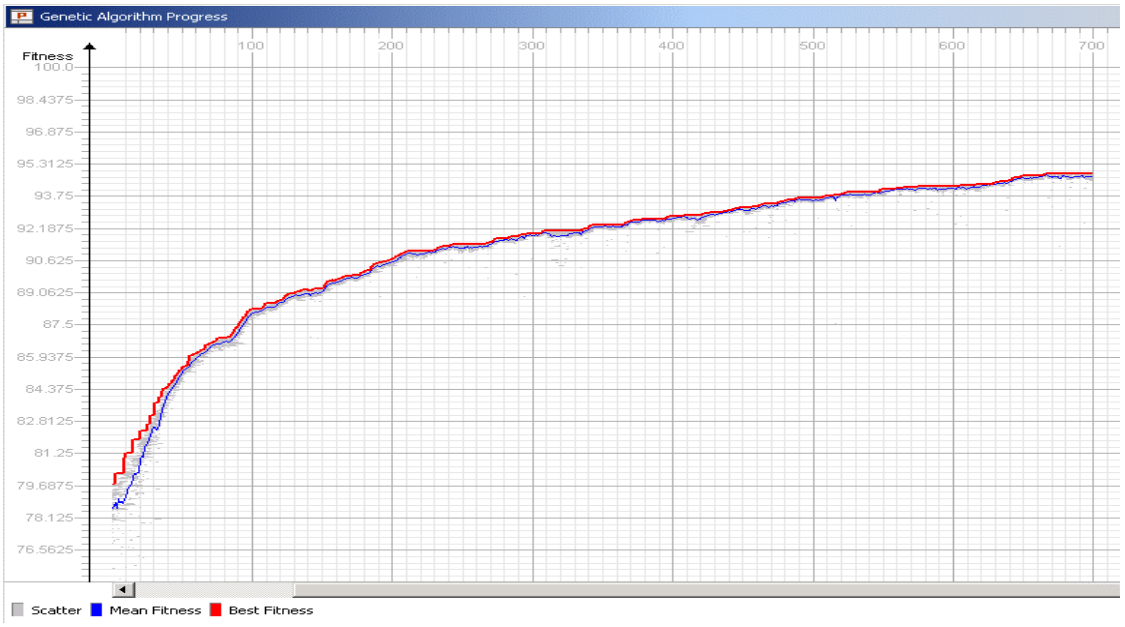
200-*node network, node-based crossover, no local search*



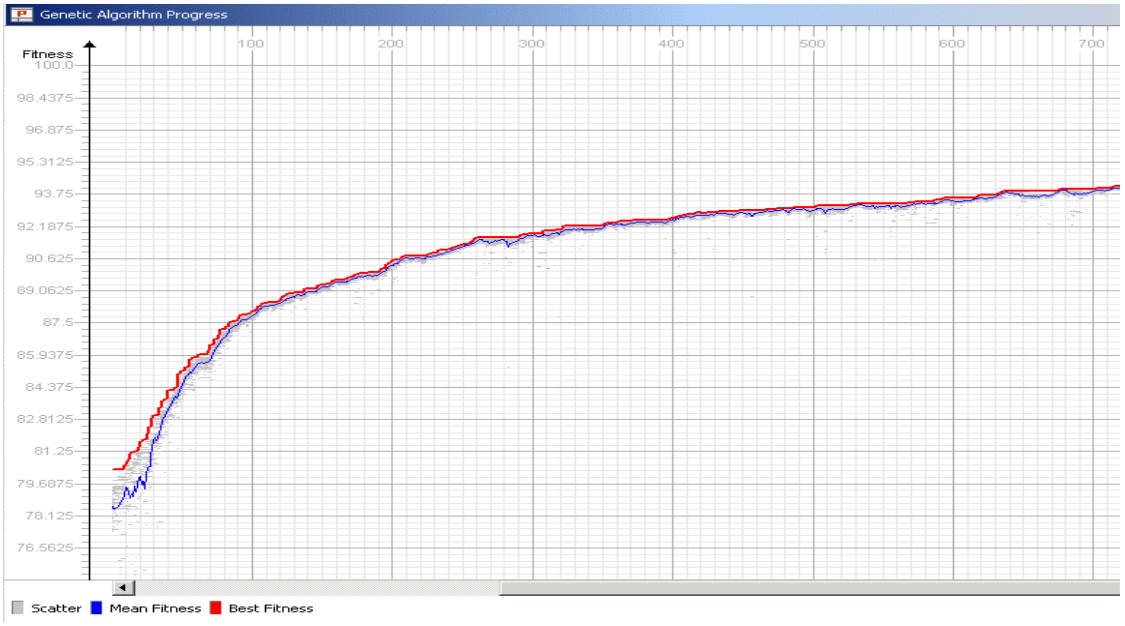
200-node network, edge-based crossover, no local search

The results seen for 200 node networks are encouragingly similar to those of smaller networks. While the final best fitness rating is lower than that of smaller networks, this is understandable as initial genome fitness levels are lower, as well as the hugely increased complexity of the larger networks. Edge based crossover appears to perform better on average for the first 1000 generations, but both techniques reached the same general level by 2500 generations.

4.3.1 The Addition of Local Search



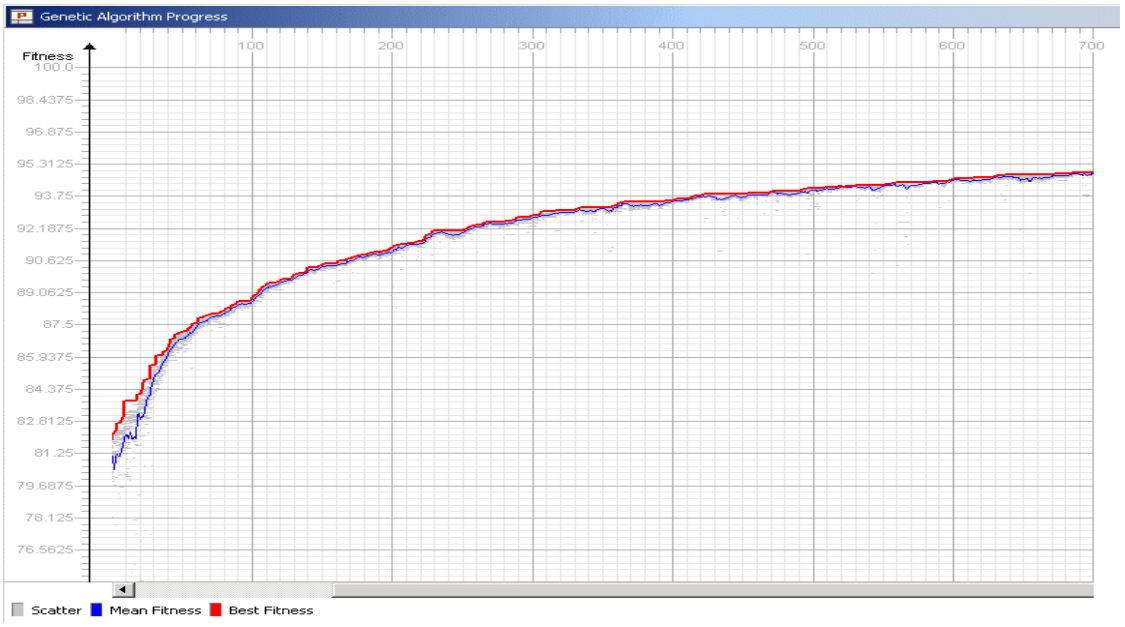
200-*node network, node-based crossover, local search*



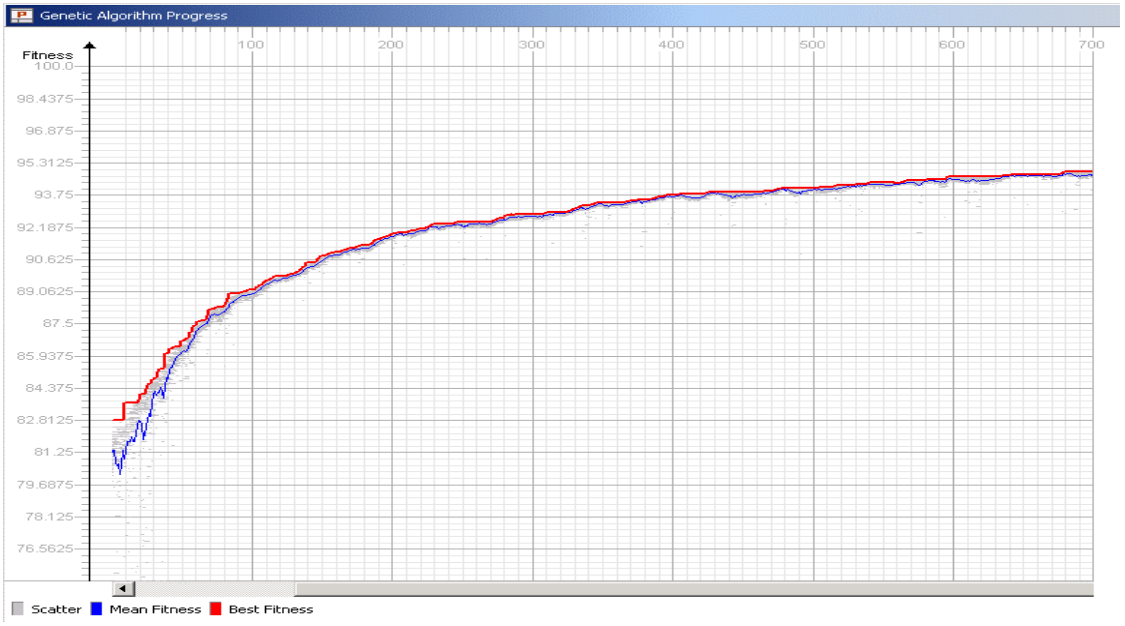
200-node network, edge-based crossover, local search

Excellent results are seen with the enabling of the local search operator – population fitness levels increase at a much faster rate than without local search, and a more optimal solution is converged to by the 2500th generation. Despite the strong appearance of convergence from nearly the start of the run, the effect is the same as with the medium-sized networks, with solutions with similar fitness levels being spread across the solution space. Here, node-based crossover offers slightly better results for the first 1000 generations, but as before, this advantage is negligible in the long run.

4.3.2 The Addition of Prior Knowledge



200-*node network, node-based crossover, local search, prior knowledge*



200-node network, edge-based crossover, local search, prior knowledge

With the addition of the prior knowledge genetic operator, immediate results are seen on the fitness levels of the initial genomes, which are no long purely randomly generated. Despite this advantage, the overall performance of the GA does not seem to be affected in any great way from its performance using only local search in addition to standard techniques. The application of prior knowledge appears to result in initial genomes of fitness levels that are reached after approximately 50 generations by a GA

using only prior search in addition to standard operators, effectively reducing the total number of generation required to reach an optimal solution.

The average optimal network found for a 100 node network configuration with a minimum reliability requirement of 99% had the following properties.

K-Connectivity:	5
Reliability:	99.8%
Average Path Length:	1147.02
Cost:	2389.11
Cost / MaxCost ratio:	2.3%

The average optimal network found for a 200 node network configuration with a minimum reliability requirement of 99% had the following properties.

K-Connectivity:	4
Reliability:	99.7%
Average Path Length:	5411.39
Cost:	9183.43
Cost / MaxCost ratio:	1.02%

5. Conclusions and Future Work

5.1 Conclusions

As expected, the node based and edge based crossover techniques performed much better than the more general multi point crossover, due to their specific design to deal with network based genomes. A semblance of phenotypical structure can be maintained in the genome through these techniques, and this benefit showed in the results, with both of the specialised operators outperforming the multi point crossover operator in every test. This performance was amplified as the size of the network was increased, until multi point crossover become almost useless in omparison to node and edge based crossover for very large networks. This effect shows the need for careful design of even the standard operators of a GA for this problem. Effects are not nearly as dramatic for small networks, which is likely why this issue has not been specifically addressed in previous work, but can be clearly seen on networks of greater than 20 nodes.

The local search heuristic demonstrated exceptionally good results on all networks, making it all but a requirement for future work on the problem. This can be inferred from the size of the solution space, and the results seen here verify that a directed heuristic greatly improves the performance of the GA. The effects are even more apparent on large networks, where more optimal results are found for large networks in shorter periods of time.

The prior knowledge operator showed slight benefits to the development of large networks, though less than expected. It would be best used in situations where only a limited number of generations can be processed, possibly due to time, processing, or memory constraints on extremely complex problems. Although its effect is only seen in the first 200 – 500 generations, this could justify its use, especially if a more beneficial application can be determined through extensive testing.

One of the results obtained that is most indicative of the effectiveness of the GA is how the cost to maxcost ratio is shown to approximately halve for each consecutive network size that was tested. This is the best indication that the GA is finding optimal or near-optimal solutions for each node configuration, as the maxcost of a network approximately quadruples as the number of nodes doubles, while the minimum cost only doubles. The approximate halving of the cost to maxcost ratio as networks double in size shows that if the smaller, more easily solved networks' solutions are near-optimal, the solutions found for the larger, more complex networks must be near-optimal as well.

Final conclusions can be made that a specially designed GA with appropriate operators and heuristics can be used to solve the network design problem for extremely large networks. As mentioned in the introduction, a major reason for the dearth of research into this area is a heretofore unavailable amount of processing power and memory shortage, as well as the added complexity of large networks and the generality

required for an algorithm that is able to solve the problem for both small and large networks. Design of such algorithms is an in-depth procedure involving workable heuristics that must be tested for applicability to the problem. With the necessary background work, GAs can and will be extended to scale to the design of larger networks

5.2 Future Work

Future work on the subject of solving the network design problem through evolutionary techniques should likely take the form of specialized GAs for large and very large networks. Specific operators that were not examined here but that could show benefits include network splitting, where many smaller networks are solved for optimality before being connected to form a larger network, and fine-tuning small areas of a larger network through another form of local search.

Without taking into account extremely specific cases, the network design problem has been all but exhausted for networks of fewer than twenty nodes. The GA presented here, with its scalability between small and large networks is merely a first step on the road to developing efficient, reliable networks on scales seen in the real world, and it is hoped that the insights provided here will be of use and inspiration for new ideas in this problem field.

6. Bibliography and References

1. Ackley, D., and Littmann, M., "A Case for Lamarckian Evolution." *Artificial Life III*, (Langton, C., editor), Addison-Wesley, 1994, 487-509.

2. Codefarm Software Ltd., "The Codefarm AI Platform: A Technical Whitepaper." http://www.codefarm.com/papers/platform-wp1.pdf (Registration required)

3. Cortes, P., Guerrero, F., Canca, D., and Garcia, J., "An Evolutionary Algorithm for the Design of Hybrid Fiber Optic-Coaxial Cable Networks in Small Urban Areas." *Proceedings of the 6th International Work-Conference on Artificial and Natural Networks, Vol 1*, Granada, Spain, June 2001, 749-756.

4. Cortes, P., Onieva, L., Larraneta, J., and Garcia, J., "Decision Support System for Planning Telecommunications Networks: A Case Study Applied to the Andalusian Region." *Journal of the Operational Research Society, Vol 52*, March 2001, 283-290.

5. Cortes, P., Larraneta, J., Onieva, L., Garcia, J., and Caraballo, M., "Genetic Algorithm for Planning Telecommunication Networks." *Applied Soft Computing Journal, Vol 1*, 2001, 21-33.

6. Deeter, D. and Smith, A., "Heuristic Optimization of Network Design Considering All-Terminal Reliability." *Proceedings of the 1997 Annual reliability and Maintainability Symposium*, Philadelphia PA, January 1997, 194-199.

7. Dengiz, B. and Altiparmak, F., "A Genetic Algorithm Approach to Optimal Topological Design of All-Terminal Networks." *Intelligent Engineering Systems Through Artificial Neural Networks, Vol 5* (C. H. Dagli, M. Akay, C. L. P. Chen, B. R. Fernandez and J. Ghosh, editors), ASME Press, 1995, 405-410.

8. Goldberg, D., *Genetic Algorithms in Search, Optimisation, and Machine Learning*. Addison-Wesley, 1989.

9. Gruau, F., "Automatic Definition of Modular Neural Networks." *Adaptive Behaviour, Vol 3*, 151 – 183.

10. Gruau, F., "Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm." *Ph.D. thesis*, ENS Lyon – Université Lyon I, 1994.

11. Holland, J., *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

12. Jan, R., "Design of Reliable Networks." *Computers and Operations Research, Vol 20, No 1*, 1993, 25-34.

13. Jan, R., Hwang, F., Cheng, S., "Topological Optimization of a Computer Network Subject to a Reliability Constraint." *IEEE Transactions on Reliability, Vol 42, No 1*, 1993, 63-70.

14. Kitano, H., "Designing Neural Networks using Genetic Algorithms with Graph Generation System." *Complex Systems, Vol 4*, 1990, 461-476.

15. Konack, A. and Smith, A., "A Hybrid Genetic Algorithm Approach for Backbone Design of Communication Networks." *Proceedings of the 1999 Congress on Evolutionary Computation,* Washington D.C., IEEE, 1999, 1817-1823.

16. Konack, A. and Smith, A., "An Improved General Upper Bound for All-Terminal Network Reliability," under revision at IIE Transactions. http://citeseer.nj.nec.com/495815.html

17. Li, C., Bettati, R., Zhao, W., "Static Priority Scheduling for ATM Networks." *Proceedings of the Real-Time Systems Symposium*, San Francisco, CA, Dec 1997, 264-273.

18. Miller, G., Todd, P., and Hegde, S.. "Designing Neural Networks using Genetic Algorithms." *Proceedings of the Third International Conference on Genetic Algorithms*, (Schaffer, J., editor), 379-384.

19. Philippides, A., "An Exploration of Evolutionary Techniques as applied to the Network Design Problem." *Knowledge-Based Systems MSc Dissertation*, University of Sussex, 1996.

20. Ross, B., "A Lamarckian Evolution Strategy for Genetic Algorithms." *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, (Chambers, L., editor), CRC Press, 1999, 1-16.

21. Schiffmann, W., "Encoding Feedforward Networks for Topology Optimization by Simulated Evolution." University of Hagen (Germany), 2000, http://citeseer.nj.nec.com/392557.html

22. Sedgewick, R., *Algorithms Part 5: Graph Algorithms, Third Edition*, Addison-Wesley, 2002.

23. Sobol', I., *A Primer for the Monte Carlo Method*. CRC Press, 1994.

24. Stepniewski, S. and Keane, A., "Topology Design of Feedforward Neural Networks by Genetic Algorithms." *Parallel Problem Solving from Nature, Vol 4* (Voight, H., Ebeling, W., Rechenberg, I, and Schwefel, H., editors), Springer, Berlin, 1996, 771-780.

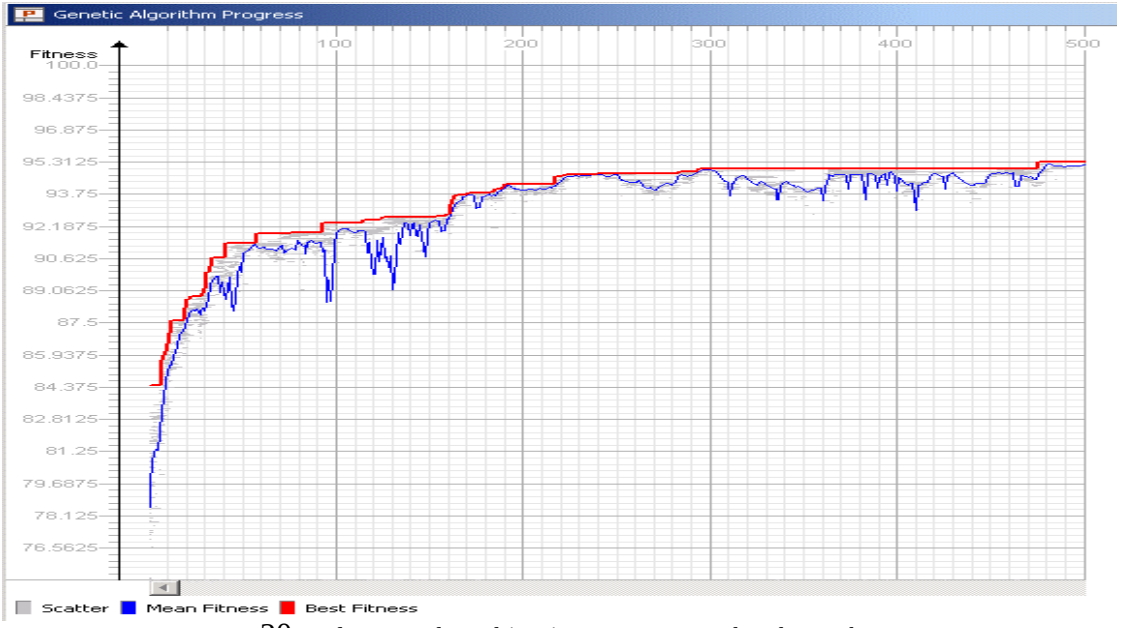
25. Whitley, D., "Modelling Hybrid Genetic Algorithms." *Genetic Algorithms in Engineering and Computer Science* (Winter, G., P'eriaux, J., Gal'an, M., and Cuesta, P.,

editors), 1995, 191-201.

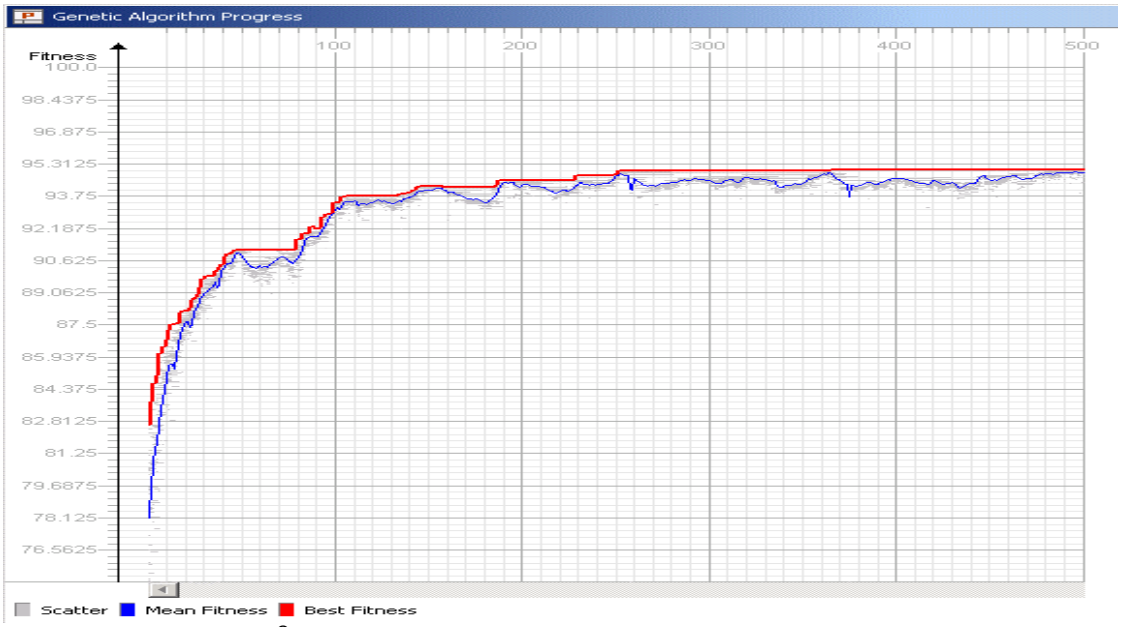
26. Yeh, M., Lin, J., and Yeh, W., "A New Monte Carlo Method for Estimating Network Reliability." *Proceedings of the 16 th International Conference on Computers and Industrial Engineering*, March 1994, 723-726.

7. Appendices

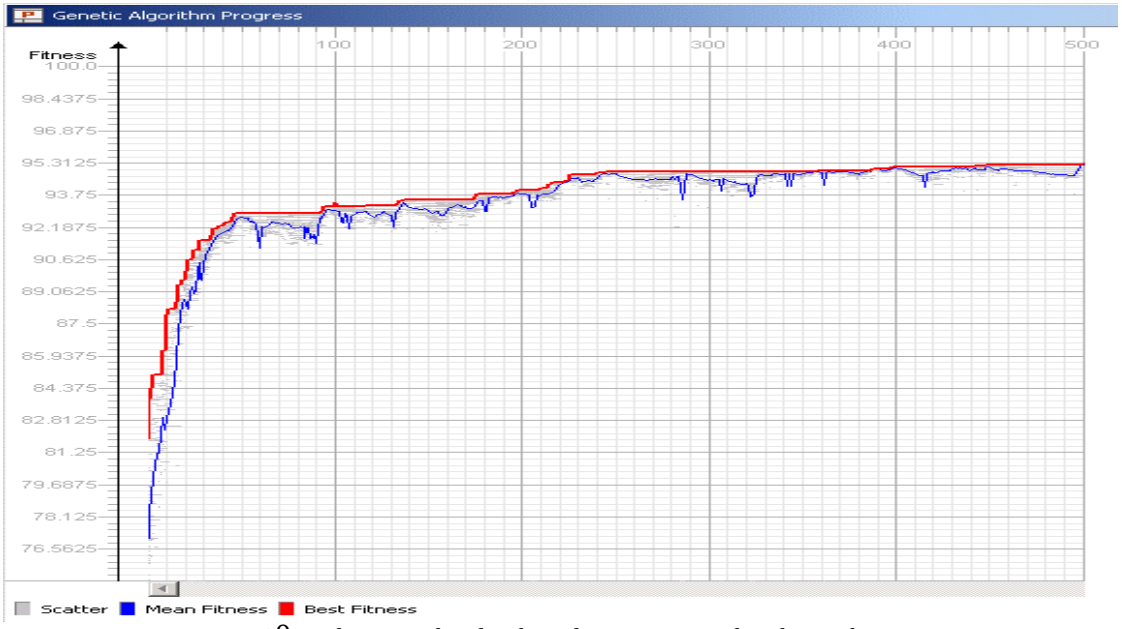
Appendix 1: Additional Output



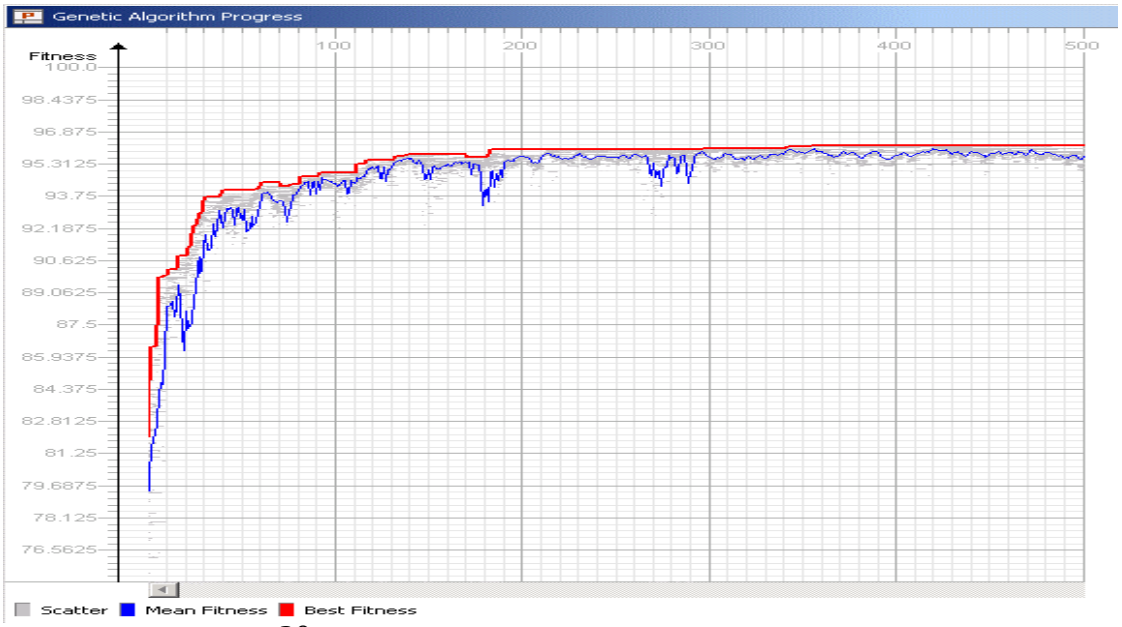
20-*node network, multi-point crossover, no local search*



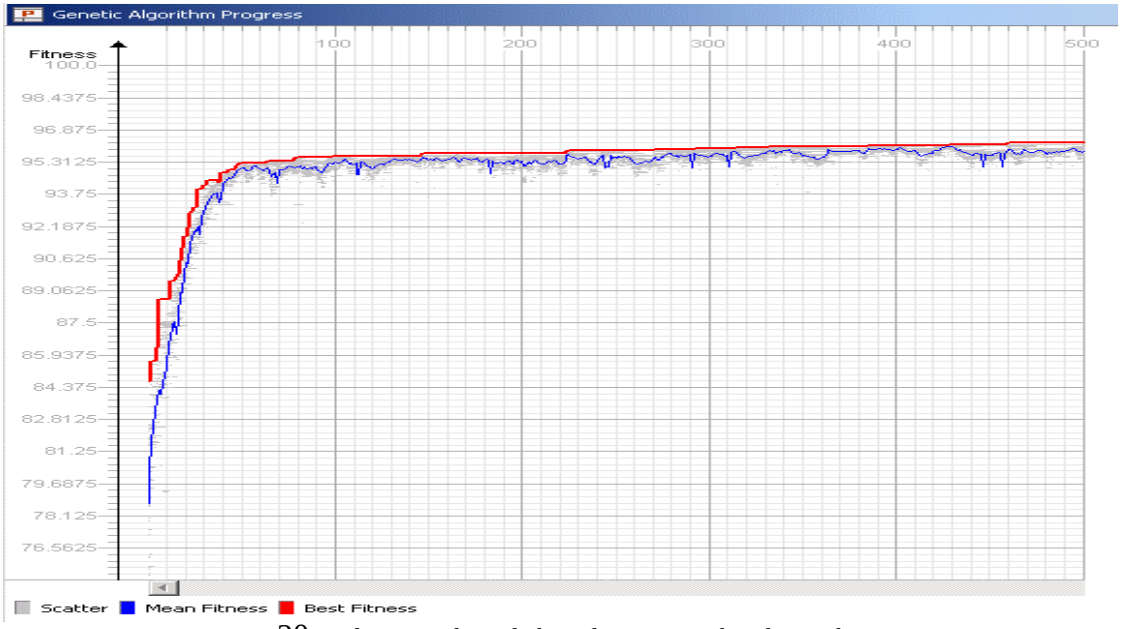
20-*node network, node-based crossover, no local search*



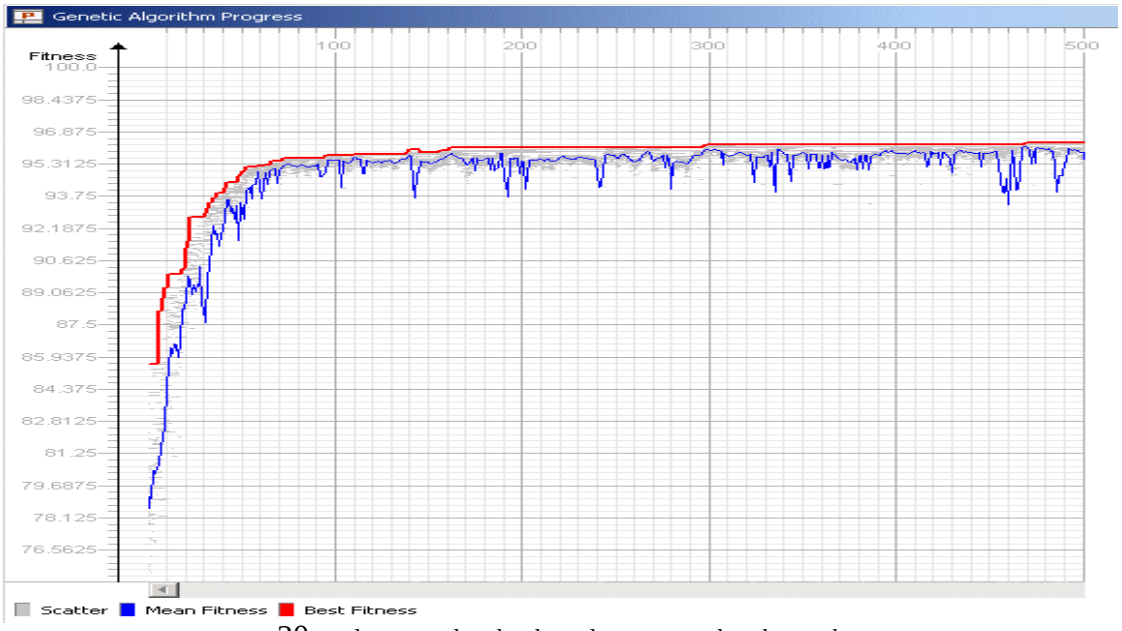
20-*node network, edge-based crossover, no local search*



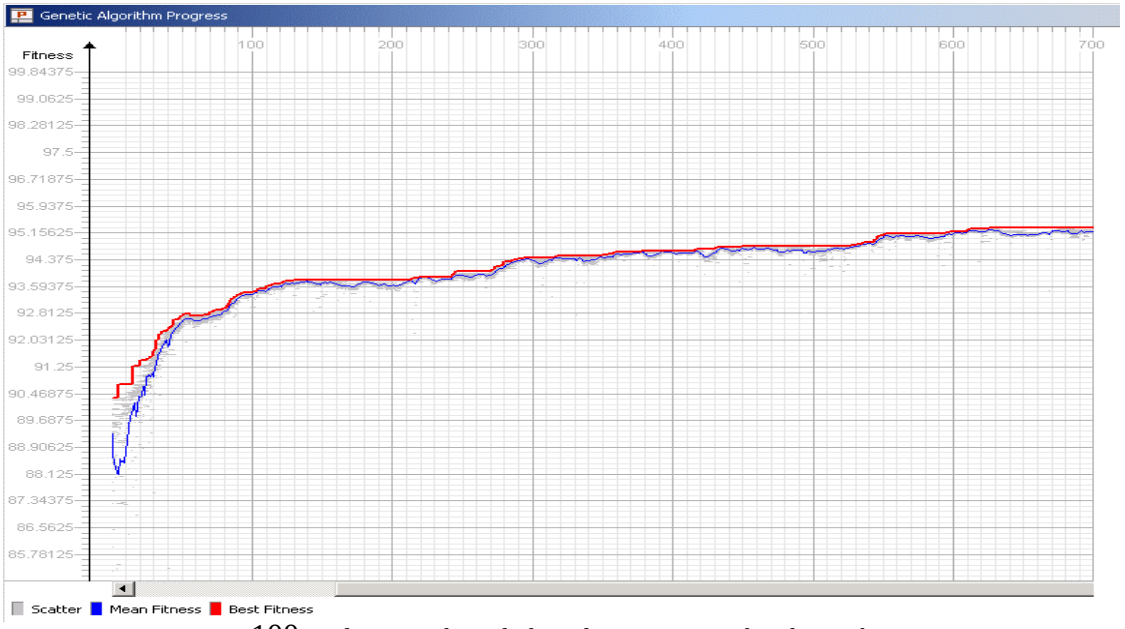
20-*node network, multi-point crossover, local search*



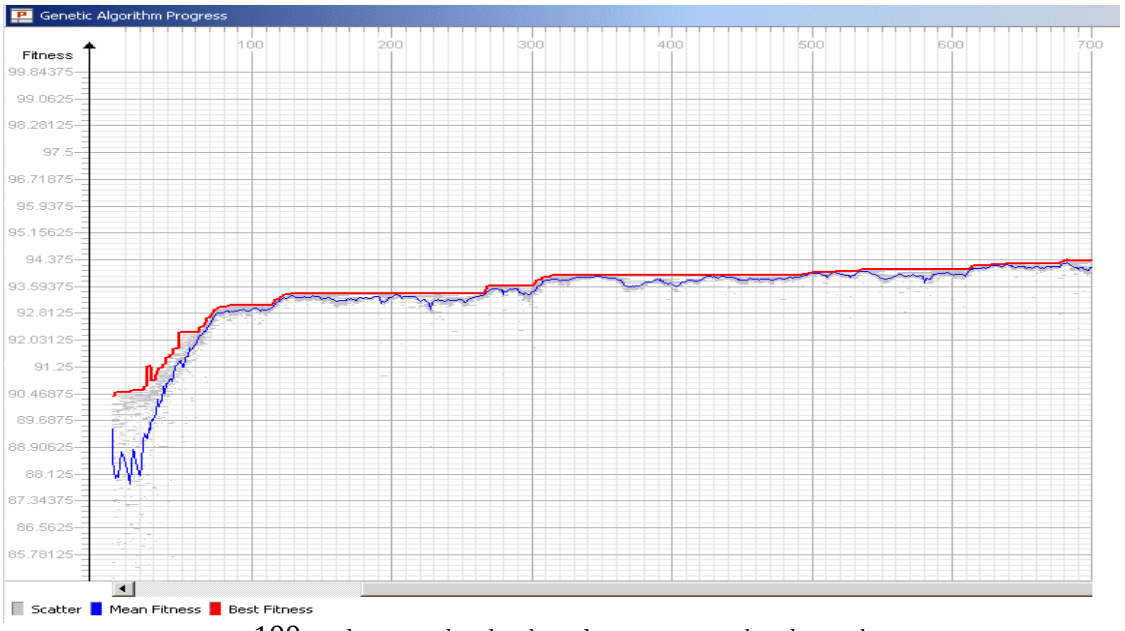
20-*node network, node-based crossover, local search*



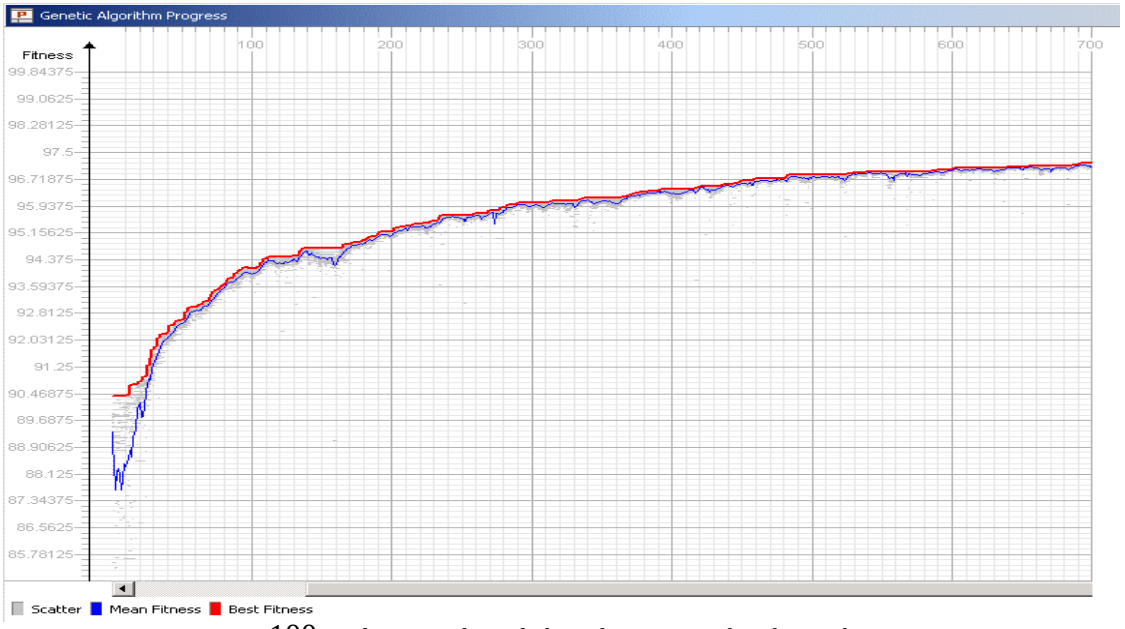
20-*node network, edge-based crossover, local search*



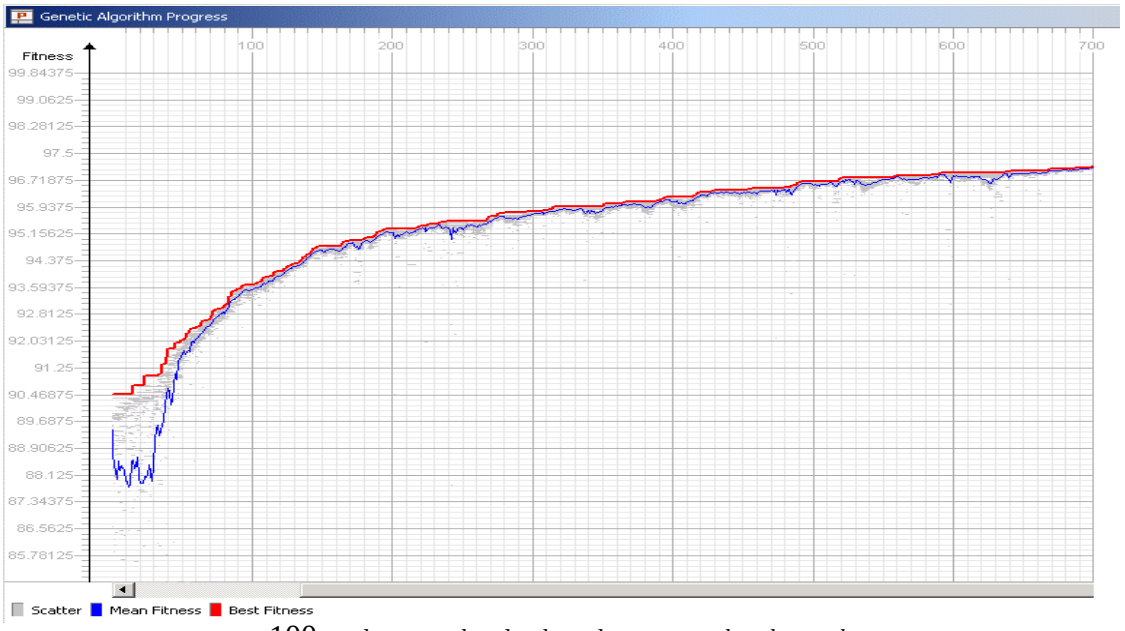
100-*node network, node-based crossover, no local search*



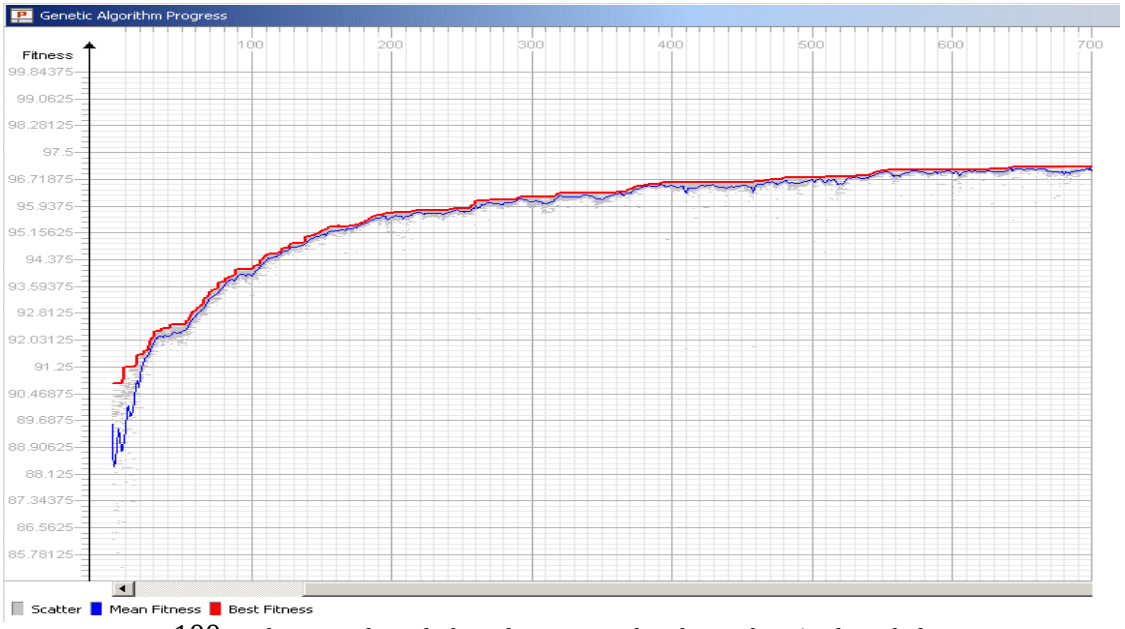
100-*node network, edge-based crossover, no local search*



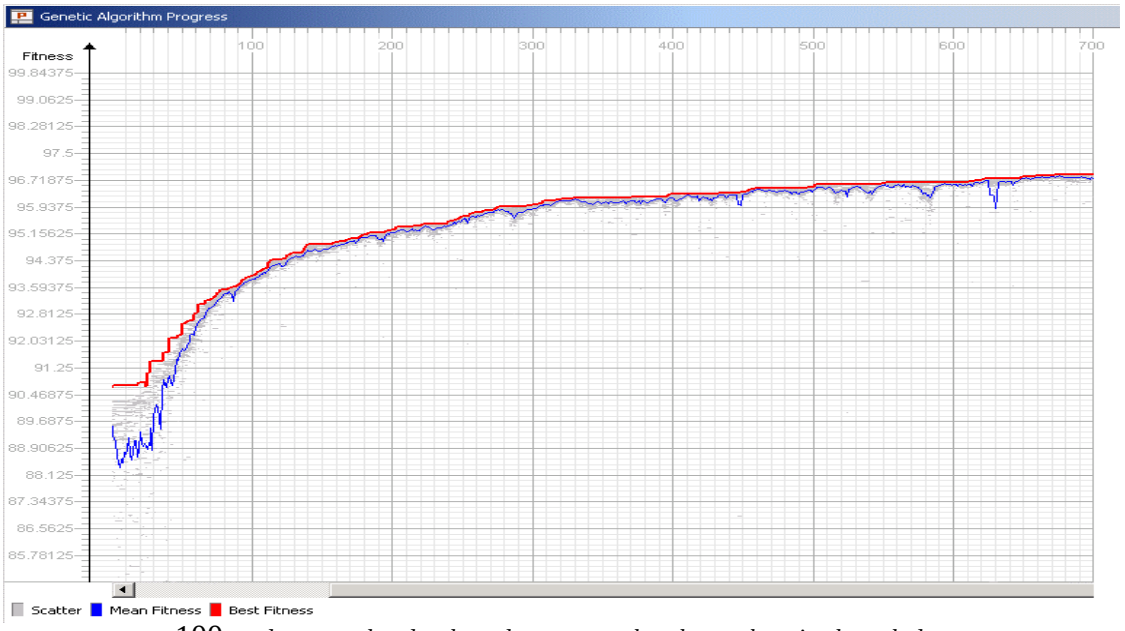
100-*node network, node-based crossover, local search*



100-*node network, edge-based crossover, local search*



100-*node network, node-based crossover, local search, prior knowledge*



100-*node network, edge-based crossover, local search, prior knowledge*

Appendix 2: Program Source Code

It must be noted that several core GA classes are the copyrighted material of Codefarm, Inc., and are protected from reproduction here under a non-disclosure agreement. For the most part, the classes not shown here are core GA classes whose operation is not specific to this project, rather being designed as generally as possible so as to be functional for any GA-solved problem. The operation of most of these operators, the main two of which are selection (a spatial 2d neighbour process) and population management (asynchronous), is apparent, and can be easily inferred with a basic knowledge of evolutionary computing.

Source code available on request. Email dbratton@node99.org with inquiries.