B.Sc. (Hons) Computing Science (Data Communications)

Final Year Project Report

* * *

"CHARLEY"

A Genetic Algorithm for the Design of Mesh Networks.

* * *

Jason Hewitt

May 1995

Jason Hewitt - May 1995
BSc (Hons) Computer Science - Final Year Project
"CHARLEY": A Genetic Algorithm for the design of Mesh Networks

Abstract

This report presents a program "CHARLEY" that uses a genetic algorithm to design optimal mesh networks. Mesh networks are of importance in communication networks, where the backbone is formed from a highly connected mesh, to provide reliability in the event of a link or node failure. The traditional approach to network design is one of repeated iteration with continual refinement. The goal is to produce a network that satisfies given constraints (delay and connectivity) for a minimum cost. During the design process optimal solutions are sought for: network topology, link capacity allocation and routing. Each of these network components is treated independently of the others, though decisions made in one area will effect other aspects of the design. It is therefore possible that a network constructed from these optimal parts, may itself be sub optimal. No single procedural algorithm exists that solves this problem by considering the network as a whole.

It was proposed that the use of a genetic algorithm might be a way of designing networks whilst taking into account the inter-dependence of all aspects of design. To test this idea, a program "CHARLEY" based on a genetic algorithm was designed and implemented in C on a Sun Sparc 10 platform. The algorithm uses a set of links to represent a network. Crossover, and two mutation operators are defined to produce new networks. Starting from an initial population of candidate networks, the algorithm produces new generations by applying crossover and mutation to the fittest individuals. Fitness, in this context, is defined by a metric incorporating cost, connectivity and performance components.

The program was tested on two design problems, the first a multi-point local access network consisting of 13 nodes, the second a 6 node mesh with a required connectivity of 3 and 4. The networks produced by the program for the multi-point network were identical to those produced by a modified Esau-Williams algorithm. The algorithm produced a solution to the mesh network problem (conectivity 3) that was 5.33 % cheaper than a solution produced by the link deficit algorithm. For connectivity 4, the algorithm was unable to produce a better or equivalent solution; this was possibly due to restricting the number of generations to 400.

A paper based on this work entitled "CHARLEY: A genetic algorithm for the design of mesh networks" was submitted to the IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications GALESIA.

Acknowledgement

As with most projects, this one would not have been possible without help of a number of people. My thanks must first go to S. McKenzie for her support, encouragement throughout the project and for designing the evaluation networks. My thanks must also go to A. Soper for quietly setting me right on several points of GA design.

Finally, I must thank G. Carter for repeatedly proof reading something he did not understand, Veerle for being so understanding and Anton who was too young to understand why Daddy wanted to do anything with the computer except play Noddy.

TABLE OF CONTENTS

List of Figures

This Page left blank

Jason Hewitt - May 1995
BSc (Hons) Computer Science - Final Year Project
"CHARLEY": A Genetic Algorithm for the design of Mesh Networks

1. OVERVIEW

Finding an optimal solution to a problem is often of great economic importance. This has lead to the development of analytical techniques and algorithms that attempt to find an optimal solution for a particular problem. Genetic algorithms (GAs), first proposed by J Holland, have been used to solve a wide range of combinatorial optimisation problems [Goldberg 1989], where procedural algorithms are either non-existent or extremely complicated. Communication network design is one such example. GAs have been applied successfully to two areas of interest to the network designer, the unconstrained minimum spanning tree [Michalewicz, 1991] and the concentrator assignment problem [Routen 1994]. This report describes the design, implementation and evaluation of "CHARLEY", a GA to design mesh networks.

Communication networks are hierarchical in structure, consisting of a high speed backbone of switches with clusters of user sites connected to the switches in multi-point local access configurations. The backbone is configured as a highly connected mesh with redundant links to provide reliability in the event of a link or node failure. Local area access networks are generally configured as trees. The object of design is to produce a minimum cost network subject to performance and reliability constraints.

The traditional approach to this problem is iterative with continual refinement of the network design. To reduce the complexity of the design problem, local access and backbone design are considered independently, as are topological design, link capacity allocation and routing. However these aspects are not independent, each having an effect on all other aspects of the network. It is likely that an optimal solution for one aspect may have a detrimental effect on the whole.

This is particularly true of mesh networks, where the traditional approach is to design the network topology optimised to minimise connection costs, based on inter-node distances. Routing and capacity allocation are done subsequently. However since the connection costs depend on link capacities, as does optimal routing, the final solution may be far from overall optimality.

Genetic Algorithms work by assigning a fitness value to each member of a population. This value is derived from the individual as a whole. A genetic algorithm that produced networks which are evaluated according to a fitness function based on mean message delay, connectivity and monetary cost would remove the problem of the inter-dependence of topological design, link capacity allocation and routing. To test this idea a program based on a genetic algorithm was designed and implemented. The program's performance was evaluated by comparing the networks produced by the program with those obtained using a traditional design methodology.

2. INTRODUCTION

This report details the design, implementation, and evaluation of a computer program based on a genetic algorithm to design mesh networks.

2.1. Report readership

This report has been aimed at network designers and therefore assumes a background in the design and performance analysis of networks. It is not necessary for the reader to have an understanding of genetic algorithms as they covered in some depth in the body of the text.

2.2. Summary of the report

The report is divided into four main chapters; introduction, design, implementation and evaluation. The introduction details the problem domain, the objectives and discusses previous work in this field. The design chapter is split into six sections which record the "DNA" encoding, genetic operators, fitness function, initial population set up and selection procedure and program interface used. The design chapter also introduces the fundamental concepts of genetic algorithms where appropriate.

The implementation chapter takes the design details and adds the program specific details. This chapter also contains the user documentation.

The evaluation chapter introduces the test bed of network design problems, discuses the performance of the algorithm and draws conclusions from the results. The report ends by outlining areas for further research.

2.3. The problem domain

The problem is to design and implement a GA to evolve minimum cost networks that satisfy a given set of constraints. The constraints are as follows:

1) The number of nodes in the network (N).

2) A cost matrix (C) where C_{ijs} is the cost of a link of capacity s connecting node i to node j. The cost of an individual link is dependent on both link speed and distance between the two nodes it connects.

3) A traffic matrix (T) where T_{ij} denotes the mean number of messages of a particular length between node i and node j. Exponential distributions are assumed for both message length and inter arrival times.

4) The minimum number of node disjoint paths (K) between each node in the network.

5) The maximum acceptable mean message end-to-end delay (t_{max}) in seconds.

GAs work by encoding a trial solution to the problem (a candidate network) into a linear representation. Two genetic operators are applied to this representation, crossover and mutation during the mating process, to derive new and hopefully better solutions to the problem. To decide which solutions are mated, each candidate is evaluated according to a fitness function.

Therefore the design phase of this project must identify the following:

1) A suitable "DNA" representation of a network.

2) How the crossover operator is to be applied to the representation.

3) Which mutations are appropriate for the problem domain and how they are to be applied.

4) How networks are analysed for fitness.

Once the algorithm has been implemented, the solutions it produces must be evaluated against those produced by traditional network design techniques. This issue must be addressed by developing a test bed of network design problems, together with solutions, for use in the evaluation stage of the project.

2.4. Project scope

The project sub-divides into three main sections; Design, implementation and evaluation.

Design

The design phase of the project will determine; 1. How a network representation can be encoded as "DNA". 2. What genetic operators are required and which are appropriate. 3. How a generated network can be evaluated for fitness.

Implementation

To implement a genetic algorithm using the design produced in the first section.

Evaluation

A comparative study between the solutions to network design problems produced by the GA and traditional methods, on several network design problems.

2.5. Aims and objectives

The aims of this project were to:-

1) Produce a program based on a genetic algorithm that evolves network designs which satisfy a given set of network design specifications.

2) Measure the performance of the genetic algorithm program against traditional techniques and present the findings.

3) Critically examine the findings of this project and indicate directions for further study.

Approach taken

To meet the aims indicated the project was broken into three stages.

The design phase considered the issues involved in applying a genetic algorithm to network design. The implementation phase created a computer program which embodied the results of the design phase. The evaluation phase tested the performance of the program in designing networks.

A critical review was performed of the project, resulting in highlighting areas that require further study.

2.6. Previous work

An examination of the current computer literature indicates that genetic algorithms are being applied to many traditionally difficult optimisation problems. Several researchers have investigated the use of GAs to solve parts of the network design problem. Z Michalewicz in 1991 [Michalewicz 1991] describes the successful application of a GA to a minimum spanning tree problem involving 100 nodes. This is of importance in local area access design. L Davis in 1989 [Davis 1989] evaluated the performance of a GA against two heuristic algorithms for the link capacity allocation problem. The paper concludes that the solutions produced by the GA were significantly better, if the GA is given enough computer time. T Routen in 1994 [Routen 1994] compares the solutions produced by a neural network and a GA to the concentrator assignment and local node access problem. The GA performed better than the neural network in both problem domains.

The GAs presented in these three papers have several common features. All use non-standard encoding necessitating the use of specially modified crossover and mutation operators. All the papers conclude that the GAs produce solutions that are better or equivalent to those produced by traditional design techniques.

Several reports concerning genetic algorithms have been produced by undergraduates at the university of Greenwich. Michael Suttling in 1993 [Suttling 1993] examines the use of GAs in the field of job scheduling. Although a very different optimisation problem, the report contains a discussion (which the interested reader is directed to) of the history of genetic algorithms and the problems they have been applied to.

3. PROGRAM DESIGN

This chapter of the report details the decisions made in the design of "CHARLEY". The basic approach adopted is that outlined by Michalewicz [Michalewicz 1991]. The design process is divided into five stages; genetic representation, initial population creation, fitness function, genetic operators and the parameters required by the GA.

3.1. Mapping to DNA

Goldberg states that a successful "DNA" coding must satisfy two criteria. The coding should be selected so that short strings of "DNA" are relevant to the underlying problem. Secondly the coding should represent the minimum "alphabet" that permits a natural expression of the problem.

Commonly the problem is encoded as a string of binary digits. This encoding has several attractions. Binary digits may be grouped in variety of patterns to represent the minimum alphabet of the problem, with short sequences forming building blocks that are relevant to the underlying problem. Genetic operators (crossover and mutation) are easily applied to the representation. Several authors [Davis 1989] have proposed that "DNA" composed of higher level components than the binary digit can be used where they are appropriate for the problem domain.

In the network design problem a representation is required that will map to the building blocks of computer networks. The basic building block of a computer network is the link. The links define both the topology and performance of a network. The effect nodes have on performance is disregarded to simplify the problem. A network can be represented as a set of link structures (containing source, destination and speed).

The following segment of "DNA" encodes the network shown;
((A,B,1200)(A,C,600)(E,F,2400)(C,E,300)(B,D,9600)(D,E,4800)(B,C,300)
(B,F,4800)(E,F,300)).

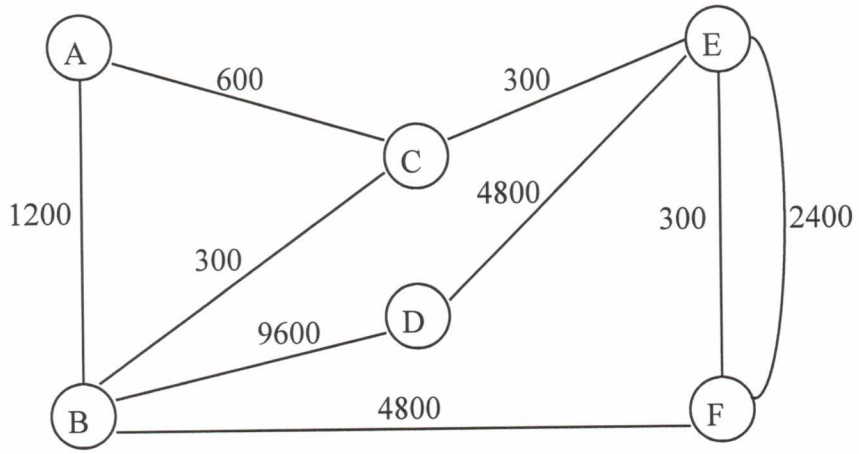


Fig. 1

This representation has the advantage of flexibility, it allows duplicate links (important in designing mesh networks), the natural expression of a network and a variable number of links. This flexibility ensures that the GA's search space is not constrained by the representation.

However, this representation does not conform to Goldberg's second requirement that short stretches of "DNA" must form building blocks that are relevant to the underlying problem. This is achieved by sorting the links in the network according to the nodes that they connect. The nodes in the network are arranged in an arbitrary order to form a node list. The links are then sorted into order according to which connected node appears highest in the node list. The previous network would be encoded as follows; (node list order A,B,C,D,E), node sets are indicated by [].
([(A,C,600)(A,B,1200)][(B,C,300)(B,D,9600)(B,F,4800)][(C,E,300)]
[(D,E,4800)][(E,F,300)(E,F,2400)]). The links within a node set are unordered.

This representation satisfies both of the design criteria. Short sequences of structures represent groupings of links associated with a node, the building blocks that form the network. The link structure is a minimum alphabet for the network design problem.

3.2. Initial population generation

There are two types of GA, hard and soft. Hard GAs only consider candidate solutions that do not violate any constraints. Solutions that do violate the constraints are either rejected or modified in such a way as to remove the violation. Soft GAs allow candidate solutions to violate constraints but penalise those that do. Two methods exist for producing an initial population for a hard GA. The first creates a population, using heuristic algorithms to build the candidate solutions so that no constraints are violated. The second method generates the individuals at random, discarding any that violate the constraints.

Both of these methods are unsuitable for the network design problem. The first requires networks to be designed using traditional techniques, the very thing the use of a GA was intended to avoid. The second method is unsuitable as networks that satisfy the constraints are few and far between in the large number of possible networks. The GA would spend a considerable amount of time producing random networks only to reject them. The network design GA must therefore be a soft algorithm. The creation of the initial population is therefore a trivial exercise of creating a number of networks containing links chosen at random from all possible links. The number of links in the random networks was arbitrarily chosen to be the minimum number of links required to satisfy the connectivity constraint (see equations 1 and 2 paragraph 3.3.3).

3.3. Fitness function

So that the GA can evolve solutions that are closer to the required optimum, the "better" solutions in a population are preferentially selected to form the next generation. The better solutions in a network are identified by a fitness function. This function, when applied to a solution, derives a value indicating how close this particular solution is to the optimum. The paradox of determining how close a possibly optimum solution is to an unknown optimum is solved by identifying the attributes of a "good" solution. If the optimisation problem were to maximise the function $f(x) = x^2 \{0 \leq x \geq 30\}$ then an appropriate fitness function would be $f_{FIT}(x) = x^2$; the larger the value returned by the function the fitter the solution.

Where the optimisation is in respect to a single criterion the fittest solutions are simply those that return the largest or smallest value for the optimisation criteria. However, many problems have several criteria that must be optimised. In these, what defines an optimal solution is far from clear. Network design must optimise performance, connectivity and cost. As the absolute values of these criteria differ greatly, each is normalised to lie in the range 0-1, 1 being the least fit.

3.3.1. Network Performance

Required network performance is usually stated in terms of an acceptable delay [Schwartz 1977]. A delay can be expressed in a variety of ways; however, mean message end to end delay (mean delay) has several advantages. It is unambiguous, a network that returns a low value will perform better than one with a high mean delay. The value is also dependant on the whole network. To calculate the delay penalty ($D_{penalty}$), the following function is used.

$$D_{Penalty} = \min[0,(\frac{D_{Mean} - D_{acceptable}}{D_{Mean}})]$$

If the average delay is less than the acceptable delay, the returned value is zero.

The calculation of mean end to end delay in mesh networks requires routing decisions to be made. Routing is needed to calculate the level of traffic on a particular link. Only static routing strategies can be used; an adaptive strategy although more realistic would make the calculations unwieldy. It was decided to use the Dijkstra forward search algorithm [Tannenbaum 1988] to determine which routes messages in the network

would use. This assumes that a network that gives good performance with a fixed routing strategy will also perform well if an adaptive strategy is used.

Mean message end to end delay is a suitable measure of performance for complete networks but is not ideal when used to evaluate disjoint and saturated networks. In disjoint networks not all the nodes are connected; saturated networks on the other hand have routes between the nodes but the capacity of one or more links in a route is insufficient to carry the traffic load. These types of networks are likely to be produced by the GA during the evolutionary process and the performance function must still identify which networks are better.

The problem of saturated networks is solved by employing a link utilisation of 0.999 in the M/G/1 delay formula [Schwartz 1977] to give a virtually infinite delay on the saturated link , so that the delay penalty tends to 1.

If no route exists between two nodes it is impossible to calculate the mean message delay between the nodes. Therefore, traffic that attempts to flow between two disjoint nodes (blocked traffic) is not accounted for in the delay penalty calculation. To account for the blocked traffic, a metric (the traffic penalty) is added to the delay penalty. The traffic penalty is calculated using the formula shown.

$$T_{Penalty} = \frac{\sum T_{Blocked}}{T_{Total}}$$

The traffic that flows along saturated routes is also included in the blocked traffic term. This ensures that networks containing saturated links are penalised twice. This mirrors disjoint networks being penalised twice, once in the blocked traffic term and again in the connectivity penalty. If a network is so badly disjointed that no traffic can flow, the delay penalty term is set to 1.

The performance penalty is calculated as the sum of the delay penalty ($D_{penalty}$) and the traffic penalty ($T_{penalty}$). Therefore the performance of a network ranges between 0 and 2. Zero indicates networks where the required mean delay is satisfied and the network contains routes between all nodes with sufficient capacity to carry the traffic. Networks with a large number of disjoint or saturated routes that do not satisfy the acceptable delay constraint return higher values, up to 2.

3.3.2. Network connectivity

For a network to be reliable it must contain redundant routes between nodes. These routes are then used to carry traffic in the event of a failure. When designing a network there must be the minimum required number of paths (k_{req}) between node pairs. The number of node disjoint paths between a pair of nodes is called the node connectivity (k_{node}). A mesh network with k=3 has a minimum k_{node} of three between all pairs of nodes. A k=3 network would remain viable in the event of any two failures.

The connectivity penalty ($K_{penalty}$) is calculated by summing the differences between k_{req} and k_{node} for all node pairs. If the node connectivity is greater than k_{req} the node connectivity is set to k_{req}. Normalisation, so that $K_{penalty}$ lies in the range 0-1 is performed by dividing by the total number of paths in a network containing n nodes.

$$K_{Penalty} = \frac{\sum \max[0, k_{req} - k_{node}]}{P_{Total}}$$

The total number of paths in a network containing n nodes is given by;

$$P_{total} = \frac{k_{req} \times n(n-1)}{2}$$

3.3.3. Network cost

The cost penalty of a network ($C_{penalty}$) is simply the sum of the cost of the links used in the network. The normalisation of this term so that it lies in the range 0-1 is problematic. To determine the lower limit on network cost requires knowledge of the optimum solution and the upper limit is theoretically infinite. A workable solution is to calculate the cost of the worst theoretical network (C_{worst}) that satisfies the connectivity constraints and normalise the cost with respect to this value.

C_{worst} is calculated by multiplying the cost of the most expensive link possible by the minimum number of links required to satisfy the connectivity requirement. The minimum number of links for a network containing n nodes is given by equation (1) for $k_{req}=1$ and (2) for $k_{req}>1$.

$$links = n - 1 \quad (1)$$

$$links = \frac{n \times k_{req}}{2} \quad (2)$$

The cost penalty is then calculated using, where L_{ij} is the set of links forming the network;

$$C_{penalty} = \frac{\sum L_{ij}}{\max(C_{ijs}) \times links}$$

The cost penalty can therefore range from 0 for inexpensive to infinite for more expensive networks but for reasonable networks, those not containing a large number of redundant links, the penalty term will generally be less than 1.

3.3.4. Network fitness

The fitness value is used to determine which networks are fitter and therefore have a higher probability of being used to form the next generation. The obvious method of combining the three penalty terms is simply to sum them together, the lower the value obtained the fitter the network. Traditionally this value is used, either deterministically or stochastically, to calculate the number of offspring derived from an individual that will appear in the next generation [Brindle 1981].

Unfortunately, simply summing the penalty terms produces a fitness function that erroneously regards low cost networks containing some saturation the "best" if the increase in the performance penalty is outweighed by a decrease in the cost penalty. To overcome this problem the fitness function must be modified to distinguish "good" and "bad" networks correctly.

Non-saturated and non-disjoint networks are "better" than saturated and disjoint ones regardless of the cost of the networks. It is of no benefit to have a network, however cheap, if it is unable to carry the network traffic at the required speed. Therefore the fitness function should compare networks using the performance term as a primary key and the sum of the connectivity and cost penalties as a secondary key. This method evaluates networks primarily on performance. Networks with equivalent performance are compared on their respective costs.

Although the fitness function now correctly identifies "good" and "bad" networks it no longer produces a suitable value for calculating the number of derived offspring. Baker in 1985 [Baker 1985] proposed a nonparametric method of assigning the number of offspring to an individual. The method ranks individuals in the population according to the fitness function. The number of offspring is then allocated according to the individuals rank. This approach disassociates the value produced by the fitness function from allocation of offspring allowing the modified fitness function to be used. It also has the benefit of maintaining the genetic diversity of a population from one generation to another, the importance of which is discussed later.

3.4. Genetic Operators

All GAs possess two genetic operators; crossover and mutation. During mating, crossover is used to exchange genetic information between individuals, to form an individual that inherits characteristics from both of the parents. All individuals in the population are given a chance to form the next generation but a selection procedure is used to favour the fitter solutions. Mutation is a random alteration applied to the "DNA" of an individual and is used to introduce new genetic information into the population.

3.4.1. Crossover

During crossover two stands of "DNA" line up and at a random point the "DNA" is broken into two. The tails formed are then swapped over and rejoined to create two new strands. As shown below.

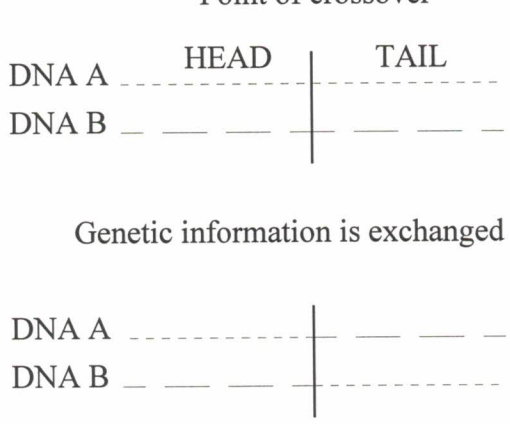
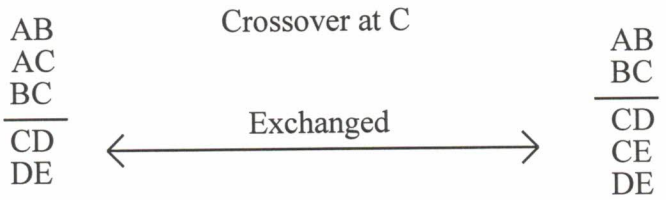


Fig. 2

This traditional approach to crossover is easy to apply to string representations of the problem domain. Where more complex representations have been used the crossover operator has been modified to retain the functionality that genetic information is exchanged between two solutions but not necessarily the way in which this occurs [Michalewicz 1991, Routen 1994 etc.]. However, the selected representation allows a traditional approach.

Crossover is accomplished by treating the network representation as a list of node sets. This list is then split at a random group set boundary into a head and tail. The exchange of information is performed by swapping the tail of a list with the tail from a similarly split list. The effect of the crossover operator on two networks, together with the networks produced, is shown in Fig 3.

Representation treated as a list



Forms two new networks

Fig. 3

As the selection of the node set boundary where crossover is to take place is random, several complications can occur. It is possible that one, or both networks do not posses the boundary selected. In this case all of the network not containing the node boundary is treated as the tail and the head is an empty set. This results in the two networks AB and null C producing the two networks AC and null B. If neither of the two networks contain the selected crossover point then two copies of the original networks are produced. Any null networks resulting from a crossover are discarded.

3.4.1.1. Mating selection

So that the GA can evolve better candidate solutions, the fitter individuals in a population must be mated together. The mating selection strategy employed must ensure that beneficial genetic information is not prematurely lost from the population. This happens when individuals with low fitness values are prevented from mating or exceptionally fit individuals flood the next generation with their genetic information. Many different scaling methods are used [Forest 1985] to bring fairness to the mating selection whilst favouring fit individuals.

As solutions are ranked according to fitness, all that is required to maintain diversity is to select individuals from the population using a probability density function to return the rank of an individual with a given probability. The Reeves rank based scheme [Reeves 1993] uses the function shown to determine the probability of selecting a rank (R) with a population size of M and the individuals sorted into ascending order of fitness.

$$p(R) = \frac{2R}{M(M+1)}$$

To increase a population's inertia to the effects of a super individual, a percentage of individuals selected for mating do not undergo crossover but are copied unchanged, apart from any random mutation, into the next generation. This ensures that the next generation contains a significant proportion of genetic information from the previous generation. The percentage of selected individuals that are simply copied was investigated by De Jong [De Jong 1975] who concluded that 40% was a suitable level.

3.4.2. Mutation

The crossover operator ensures that generated offspring share characteristics of both parents. The possible range of characteristics exhibited by the offspring is limited to the range of genetic information in the initial population. A Mutation operator is used to introduce new genetic information into a population, preventing the GA from converging prematurely on a sub-optimal solution.

The level of mutation that is applied to candidate solutions is generally in the order of 1% [De Jong 1975]. If the mutation rate is too high then promising segments of "DNA" are frequently destroyed before they have a chance of combining to produce better solutions. This prevents the GA from converging. If the mutation rate is too low then the introduction of new genetic information into the population will be too slow to prevent the GA from converging prematurely.

Some authors have used the mutation operator as a way of adding heuristics into the GA [Davis 1989]. The mutation operator is biased so that favourable modifications of the candidate solution are more likely to happen. This increases the speed at which the GA converges to a solution. However if this technique is applied too liberally there is a danger that the GA will rapidly converge to the same solution as that produced by the heuristic algorithm alone.

The crossover operator has a tendency to rapidly increase the number of links and therefore the cost of a network. To counter this upward pressure on cost the mutation operators are biased to reduce the cost of a network.

GAs have used a wide variety of mutation operators [Suttling 1993]. However, as the fitness of a network solution is independent of the order of the links, only mutation operators that change the contents of the set of links representing a network can meaningfully be applied.

3.4.2.1. "Addel" mutation

The "addel" operator either removes a random link from the set of links that define a network, or adds a link into the network selected at random from all possible links (C_{ijs}). This mutation operator is biased in favour of removing links from a network. 75% of "addel" mutations result in a link being removed.

3.4.2.2. Link mutation

The link mutation operator works by replacing a random link in a network with one selected at random from C_{ijs}. This operator is biased in favour of replacing expensive links in a network with cheap ones. 75% of the exchanges only take place if the replacement link is cheaper than the one already in the network. 25% of "link" mutations happen regardless of the relative costs.

3.5. Parameters

The parameters required by the GA include; the population size, number of generations that the algorithm is to run for and the probabilities of applying the different genetic operators. The values of these parameters will be established by experimentation. The network design section records the required network connectivity and information regarding nodes and links. For each node, the links that it may use together with the volume and destination of the traffic it produces is recorded. For each link, the capacity and cost is recorded. The GA assumes that all links are full duplex and network traffic is symmetrical to simplify the calculations.

4. IMPLEMENTATION

This chapter details the implementation of a program "CHARLEY" containing the GA developed in chapter 3. The chapter is divided into five sections; program overview, data structures, algorithms, testing and the user documentation.

"CHARLEY" was implemented in ANSI C for portability. Initially the program ran on a MSDOS platform. Due to lengthy execution times and restricted memory the program was ported to a SPARC 10 platform, using Linux (an Intel hosted UNIX clone) as a stepping stone. The full source code (both C and associated header files) is available on the enclosed diskette.

4.1. Program overview

The program was implemented as a batch process. The program starts by reading an initialisation file containing the parameters for the GA and the network design constraints. As this file is read a database containing the information is assembled in memory. Once the file has been read, the GA is applied for the required number of generations. During program execution a record of the best solution produced so far is maintained. If a better solution is produced the details of the solution are printed and the current best updated. After the GA has finished the program prints details of the best network generated.

4.2. Data structures

The program makes extensive use of linked lists. Linked lists were chosen for their flexibility and efficient use of memory. Memory usage was important as a GA uses large amounts to store populations and intermediate solutions during crossover, mutation etc. Program efficiency can also be improved by storing all required data in main memory.

4.2.1. Network implementation

Linked lists were the natural choice for implementing the network representation. A link was stored in the following C structure;

```
typedef struct link_struct
        {
        char struct_id[20];

        /* which nodes the link connects */
        struct node_struct * node1;
        struct node_struct * node2;

        /* the speed of this link */
        unsigned long speed;

        /* the mean number of messages over this link */
        float no_messages;

        /* the mean message length over this link */
        float message_len;

        /* the link delay of this link */
        float delay;

        /* the current state of this link */
        unsigned state;

        /* the cost of the link */
        unsigned long cost;

        /* pointer to the next and previous links in the network */
        struct link_struct * prev;
        struct link_struct * next;

        }LINK;
```

The structure contains the elements of the link proposed in paragraph 3.1. The two node pointers point to node structures containing the details of a particular node. The use of pointers to store the source and destination nodes allows a link to reference node information, held in the program database, directly. The speed of the link is expressed in bits per second. The last two pointers point to the previous and next link structures in this list. The struct_id field was used during program debugging and testing (all structures used by the program contain this field). The remaining fields are used during calculations of connectivity, link delay and network cost.

To form a representation of a network, the link structures were connected to form a doubly linked list [Ammeraal 1988]. The links are sorted within the list according to which nodes they connect (see paragraph 3.1).

This implementation has the advantage that crossover can be achieved by redirecting the prev and next pointers (see paragraph 4.3.2). However, the implementation has several drawbacks. Linked lists can only be searched linearly, degrading the programs performance. The use of the linked list also entails extra work in modifying standard algorithms for connectivity and routing, most of which represent a network as a conectivity matrix.

4.2.2. Network constraints database

The program builds a database of network constraints from the initialisation file. The basic structure of the database is a linked list of nodes. The list of nodes is constructed from node structures.

```
typedef struct node_struct
        {
        char struct_id[20];

        /* name of the node */
        char * node_name;

        /* flag of the current state of the node */
        unsigned state;

        /* number of hops this node is from a destination */
        unsigned hops;

        /* how many links this node is allowed */
        unsigned no_links;

        /* pointer to the list of allowed links */
        struct link_struct * allowed_links;

        /* pointer to the list of traffic records */
        struct traffic_struct * traffic;

        /* pointer to the next and prev nodes in this list */
        struct node_struct * prev;
        struct node_struct * next;

        }NODE;
```

Each node structure contains a pointer to a string containing a unique label identifying this node, a pointer to a list of allowed links and traffic records. The other fields are used during calculation of connectivity and routing. The use of a label allows nodes in the network to be identified using meaningful names. The nodes are linked together to form a doubly linked list in the same way as the link structures. The order of the node list is used to sort the links into node sets. This list is also used extensively by algorithms to ensure that all nodes in the network have been processed.

For each node, a list of links that it may use to connect to other nodes in the network is constructed. This list uses the same link structure as described in paragraph 4.2.1. The program assumes all links are full duplex so that a link that is recorded as connecting A to B will appear in both the A, and B allowed link lists.

A node also maintains a list of traffic structures as shown;

```
typedef struct traffic_struct
        {
        char struct_id[20];

        /* which nodes the traffic flows between */
        struct node_struct * node1;
        struct node_struct * node2;

        /* the mean number of messages */
        float no_messages;

        /* the mean message length */
        float message_len;

        /* pointer to the next and previous traffic records */
        struct traffic_struct * prev;
        struct traffic_struct * next;
        }TRAFFIC;
```

The structure records the nodes between which this traffic flows, the number of messages sent per second and the length of the messages. All traffic is assumed to be bi-directional i.e. the volume of traffic flowing from A to B is the same as that flowing from B to A. The structure also records the mean message mean length and number produced per second for a particular class of traffic. If several classes of traffic are used, the program calculates a mean message length weighted according to the frequencies of the different classes of traffic. The delay calculations used in the fitness function assume exponential message lengths and inter arrival times.

4.3. Program algorithms

The program uses five main algorithms, two used to evolve networks and three associated with the fitness function. The GA uses crossover and the Reeves selection procedure. The three algorithms used by the fitness function calculate the number of independent paths between pairs of nodes, network routing and mean end to end message delay.

4.3.1. The GA algorithm

The GA is applied to the contents of a population to produce new population that hopefully contains fitter solutions than the preceding population. The GA starts by sorting the current population into order according to fitness. Members of this population are then selected according to the Reeves selection procedure. The selected members then mate or are copied before being subjected to a random mutation. The fitness of the offspring is calculated and they are inserted into the next generation. The GA may be summarised as follows:

1) Sort the population into descending order of fitness

2) For the required number of offspring

2.1) Select two networks from the population according to the Reeves selection procedure.

2.2) Perform a crossover on the networks with a probability of 60 %

2.3) Subject the produced offspring to "addel" and "link" mutation according to the respective mutation frequencies in the initialisation file.

2.4) Calculate the fitness values of the generated networks and place them into the new population.

Endfor

3) Swap the new population with the current population.

4.3.2. The crossover algorithm

The crossover operator splits the linked list of the network into two, the head and tail. The next and prev pointers are then rearranged to point to a tail from another network. The process can be summarised as follows.

1) Select a random node from the node list (node X).

2) Find and remember the first link in network A that connects to node X (link A).

3) Find and remember the first link in network B that connects to node X (link B).

4) If network A contains a connection to node X:

Split network A at link A to form head A and tail A.

Else

Set tail A to be the whole of network A.

Endif

Repeat from 4 for network B

5) If there is a head A:

Reform the links so that tail B is connected to head A forming the new network A.

Else

Make the tail B the new network A

Endif

Repeat from 5 for network B

4.3.3. The Reeves selection procedure

The Reeves selection procedure uses the probability density function shown to give the probability of selecting a particular rank (r).

$$P(r) = \frac{2r}{M(M+1)}$$

The function was rearranged to produce the cumulative probability function (see appendix 1).

$$r = \frac{1+\sqrt{1+4C(r)M(M+1)}}{2}$$

Individuals were selected from the population using this function. The rank (r) returned being rounded up. Random numbers (C(r)) where generated using the C runtime function "rand", seeded from the system clock.

4.3.4. Independent paths algorithm

Several algorithms exist in the literature for calculating network connectivity. These algorithms calculate the connectivity of a complete network. To calculate the connectivity between node pairs (start and end node) a directed recursive path search was employed. To direct the search so that shortest paths are found first, the nodes are labelled with their distance in hops from the end node

The algorithm determines the connectivity between a start and end node by repeatedly trying to find independent paths between them. Each path found is recorded (one being added to the node pair connectivity). Paths are found using the following recursive function; beginning at the start node an unused neighbouring node closest to the end node is selected. If the selected node is the end node the path is recorded and the nodes and links used are removed from the network. If there are no unused neighbouring nodes the recursion unravels one level and a new neighbouring node selected. If there is an unused neighbour this node is made to be the start node and the path finding function is called recursively to find a path between the new start and the end nodes.

4.3.5. Network routing

Routing in the network is determined using the Dijkstra forward search algorithm [Tannenbaum 1986]. The algorithm was modified to cater for networks containing node pairs connected by several links of different capacities. The "best" route between two nodes was defined as the route with the highest mean link capacity . The algorithm finds the routes from a source node to all other nodes in the network and is summarised below.

1) Set all nodes as not having a route.

2) Record the route and capacity of the best route (highest capacity) between the source node and its neighbouring nodes. Routes in this case are simply source node - destination node.

3) Mark the source node as having been processed.

4) Select a node (w) that has a route and is not marked as having been processed.

5) For all neighbouring nodes to w replace the currently recorded route with a route going through w if the route has a higher mean link capacity.

6) Mark node w as having been processed.

7) Repeat from step 4 until there are no more unprocessed nodes with a route.

4.3.6. Mean end to end message delay

The program calculates the mean end to end message delay using the M/G/1 delay formula. First, traffic flow along each link is calculated using the routing information produced by the Dijkstra forward search algorithm. The link delay (D_i) is calculated using the M/G/1 formula shown below for all links in the network. Where C_i is the capacity of the link in bit per second, L_i is the average message length on the link and N_i is the number of messages flowing per second.

$$D_i = \frac{1}{C_i / L_i - N_i}$$

Finally the mean message end to end delay is calculated.

The algorithm may be summarised as follows.

1) For all traffic in the network.

    1.1) For all links in the route used by this traffic.

        1.1.1) Calculate average length and number of messages flowing along this link.

    Endfor

Endfor

2) For all links in the network

    2.1) If link is saturated

        Calculate the link delay using a link utilisation of 0.999 C_i/L_i in the M/G/1 formula. Mark the link as saturated.

    Else

        Calculate the link delay.

    Endif

Endfor

3) For all traffic in the network.

3.1) If no route exists for the traffic.

Add the traffic to the blocked traffic term of the fitness function.

Else

Find the route used by this traffic.

Multiply the number of messages by the sum of the link delays of the links used by this route and add the result to the total message delay.

3.1.1) If one or more of the links used by this route are saturated.

Add the traffic to the blocked traffic term of the fitness function.

Endif

Endif

Endfor

4) Divide the total message delay by the total number of messages in the network to give the mean message delay.

The M/M/1 formula assumes that the mean message length is exponentially distributed. A mean message length weighted according to message frequency is not exponentially distributed. Therefore the M/M/1 formula will produce an inaccurate results for the link delays. However, as the delay calculation is only used in the fitness function to compare like with like, this inaccuracy is insignificant.

4.4. Program testing

An incremental approach was adopted for program testing. As each module was completed it was subjected to both white and black box testing. For white box testing, the module was executed under a debugger so that the program steps could be traced and the contents of memory examined. The modules were tested using a range of networks containing features to ensure that the major loops and different execution branches were exercised. In this way the programs execution was compared to that expected.

Black box testing was used to test a module's ability to cope with many different networks (some highly disjoint and saturated, others highly connected). Functions where written that generated random links, nodes and traffic. The random networks were then processed by the module under test. The produced output was compared with the expected output. Failure of a module to process a particular network frequently resulted in a program crash. The module was then subjected to white box testing using the network that had caused the crash as the input. In this way the modules were refined until they were able to process correctly any network, including those with no links, no nodes and no traffic.

As the program makes extensive use of dynamically allocated memory, two testing strategies were employed to combat memory creep and memory overwrites. Memory creep was identified by examining the amount of memory assigned to the executing program. Once the program has been running for some time the amount of memory allocated to it should remain fairly constant. If the amount does not remain constant, then memory is being allocated from the memory pool but is not returned to it. To find the module causing the memory creep, each allocation and deallocation was recorded in a log file. A program was written to read the log file checking that each allocation is paired with a deallocation. An unpaired allocation indicates which module was failing to deallocate the memory.

This process also identified modules that deallocated memory prematurely, leading to a memory overwrite. The struct_id string included in each structure was also used to identify memory overwrite errors. The string makes it possible to identify the structures contained in a memory dump even if the structure has been partially overwritten. This helps to identify the module in which the structure is being prematurely deallocated.

4.5. User documentation

"CHARLEY" executes as a batch process, reading program parameters from an initialisation file and sending its output to the default output device. The program is invoked from the command line giving the filename of the initialisation file as an argument. The name of an optional log file can be given as a second argument. The log file records details of the program execution and is of most use during debugging. This section describes the contents of the program initialisation file and interpretation of the output produced.

4.5.1. The initialisation file

The initialisation file is a text file containing a list of initialisation parameters. The parameters are grouped into two sections; those used by the genetic algorithm and those describing the network design constraints. Each of the sections is further subdivided. Each sub-division is identified by a title enclosed in square brackets. Parameters belonging to this sub-division are listed under the title indented by one tab. Parameters are listed as parameter name, an equals sign and the value.

4.5.1.1. Genetic algorithm parameters

This parameter group contains the parameters required by the GA; it is sub-divided into genetics and mutation. Genetics contains two parameters, the required size of the population and the number of generations that the program will execute for. Mutation contains the percentage of offspring that are subjected to an "addel" or "link" mutation. This value can range from 1 to 100 percent. An example is shown below.

```
[GENETICS]
        population_size=400
        no_generations=400

[MUTATION]
        addel=10
        link=10
```

4.5.1.2. Network parameters

The network parameter group is sub-divided into 4 sections; network, nodes, links and traffic. The network section contains the global network constraints, connectivity requirement as an integer greater than 0 and the acceptable mean message delay in seconds; fractional delays are acceptable. An example is shown below.

```
[NETWORK]
        connectivity_req=1
        acceptable_delay=0.5
```

The nodes section lists the names used to identify the nodes in the network. All the text after the equals sign and before the end of the line is used as the identifier. No restrictions are placed on the strings used but exactly the same string must be used to identify the "from" and "to" nodes in the link and traffic sections. An example of the nodes section is shown below.

```
[NODES]
        node_name=London
        node_name=Ashford
        node_name=Chatham
        node_name=Dover
        node_name=Hastings
```

The links section records the cost and capacity of all the links that may be used to construct the network. As all links are assumed to be full duplex, if a link connecting nodes A and B is listed it is not necessary to record the same link connecting node B and A. For each link the "from" and "to" nodes are recorded using the identifiers listed in the nodes section. The speed of the link is given in bits per second.

The cost of a link is arbitrarily the monetary cost of the link. However in a network design problem, the cost value would normally contain both the installation, running costs and any other "costs". Whatever function is used to derive "costs" the costs of all links must be directly comparable. An example of the links section follows.

```
[LINKS]
        from=London
        to=Ashford
        speed=1200
        cost=568
        from=London
        to=Chatham
        speed=4800
        cost=268
        from=London
        to=Dover
        speed=9600
        cost=344
        from=London
        to=Hastings
        speed=1200
        cost=275
```

The traffic section lists the traffic in the network. For each type of traffic the "from" and "to" nodes are recorded again using the identifiers listed in the nodes section. The traffic record also records the number of messages of this type generated in one second. The mean message lengths are recorded in bits. An example follows.

```
[TRAFFIC]
        from=Ashford
        to=London
        no_messages=0.0694
        message_len=960
        from=Ashford
        to=London
        no_messages=0.0694
        message_len=19200
        from=Chatham
        to=London
        no_messages=0.0764
        message_len=960
```

4.5.2. Program output

The program prints the output produced to the standard output device. This allows the output to be redirected by the operating system to a file, printer or the default console device. For each generation the program prints the mean, best and worst fitness values in the population. If a population contains a better network, the links forming the network are listed. An example is shown below.

```
#Network consists of 12 links
#Link between Margate  and Chelmsford  of speed 1200, costing 290
#No messages 0.152800 Mess len 2619.267116 Link delay 3.275001
#Link between London  and Folkstone  of speed 1200, costing 268
#No messages 0.090300 Mess len 12190.831754 Link delay 122.93116
#Link between Colchester  and Tonbridge  of speed 1200, costing 287
#No messages 0.048600 Mess len 6176.790542 Link delay 6.864565
#Link between Chelmsford  and Ashford  of speed 1200, costing 146
#No messages 0.252800 Mess len 3167.848378 Link delay 7.936130
#Link between Canterbury  and Ashford  of speed 1200, costing 431
#No messages 0.145900 Mess len 6173.214673 Link delay 20.623581
#Link between Cambridge  and Chatham  of speed 1200, costing 172
#No messages 0.091700 Mess len 10905.473878 Link delay 54.536087
#Link between Tonbridge  and Hastings  of speed 1200, costing 338
#No messages 0.145700 Mess len 6167.852096 Link delay 20.467813
#Link between Southend  and Hastings  of speed 1200, costing 273
#No messages 0.145900 Mess len 6173.214673 Link delay 20.623581
#Link between Hastings  and London  of speed 9600, costing 475
#No messages 0.626300 Mess len 6863.317644 Link delay 1.294598
#Link between Hastings  and Dover  of speed 1200, costing 215
#No messages 0.091700 Mess len 10905.473878 Link delay 54.536087
#Link between Chatham  and London  of speed 9600, costing 468
#No messages 0.190300 Mess len 7880.272967 Link delay 0.972827
#Link between Ashford  and London  of speed 9600, costing 768
#No messages 0.537500 Mess len 5768.572600 Link delay 0.887556
#Factors Performance 0.000000 Conn 0.000000 Dup 0.000000 Cost 4.590000
fitness 4.590000
#and has a cost of 4131.000000
```

For each link the two nodes it connects, the speed and the cost of the link is listed. The volume of traffic on the link is recorded as are the mean message length and the mean number of messages on the link per second. The calculated link delay is also printed. The penultimate line of the network description prints the normalised components of the fitness function (The values of the individual components have been multiplied by a factor of 10 to improve their readability). The last line reports the non-normalised cost of the network.

5. RESULTS

Optimum values for: population size, number of generations and mutation rates were determined by experimentation. Once optimum values for the various parameters were identified, the effect of node list order on program performance was investigated. The program's performance was evaluated by comparing evolved solutions with those produced using traditional methods. A test bed of three design problems was used.

5.1. Network design test bed

Three test network design problems were used, a multi drop network (k=1) and two mesh networks (K=3,4). The same basic problem was used for k=3 and k=4. The initialisation files used to describe the multi drop and the mesh networks are shown in appendix 4 and are included on the enclosed diskette (test1.ini, test2.ini, test3.ini).

In the multi drop network (consisting of 13 nodes) all traffic is directed to a central node. Links of 1200 bps are available between all nodes. Links of 9600 bps could be used to connect some of the nodes to the central node. Two types of network traffic are allowed, with mean sizes of 960 and 19200 bits. The cost matrix and message frequencies are given in appendix 2.

The network produced as a solution (appendix 3) for the multi-drop design problem had a cost of 4143 and a mean message delay of 39.8 sec. The solution was produced using the Esau-Williams heuristic [Ahuja 1985] adapted to take account of varying link speeds.

The mesh network contained 6 nodes. Only one class of traffic with a mean message length of 512 bits flowed between all node pairs. Links of 2.4, 4.8, 9.6 and 19.2 kbps were available for all connections. The cost matrix and message frequencies are given in appendix 2.

Solutions to the mesh network (appendix 3) were produced for conectivities of 3 and 4 using the link deficit (heuristic) algorithm [Tannenbaum 1986], with minimum hop, minimum cost routing and using the M/M/1 delay formula [Schwartz 1977]. The solution for k=3 had a cost of 825 and a delay of 0.180 sec. The solution for k=4 had a cost of 797 and a delay 0.270 sec.

5.2. Optimum parameters

To determine the optimum values for population size, number of generations and the mutation operators, two experiments were performed using the multi drop network design problem.

5.2.1. Population size and number of generations

To determine the optimum values for population size and number of generations, the program was run with population sizes and number of generations set to; 100, 200 and 400. Each run was repeated three times. The average best fitness values were then plotted against generation number to produce the graph shown.

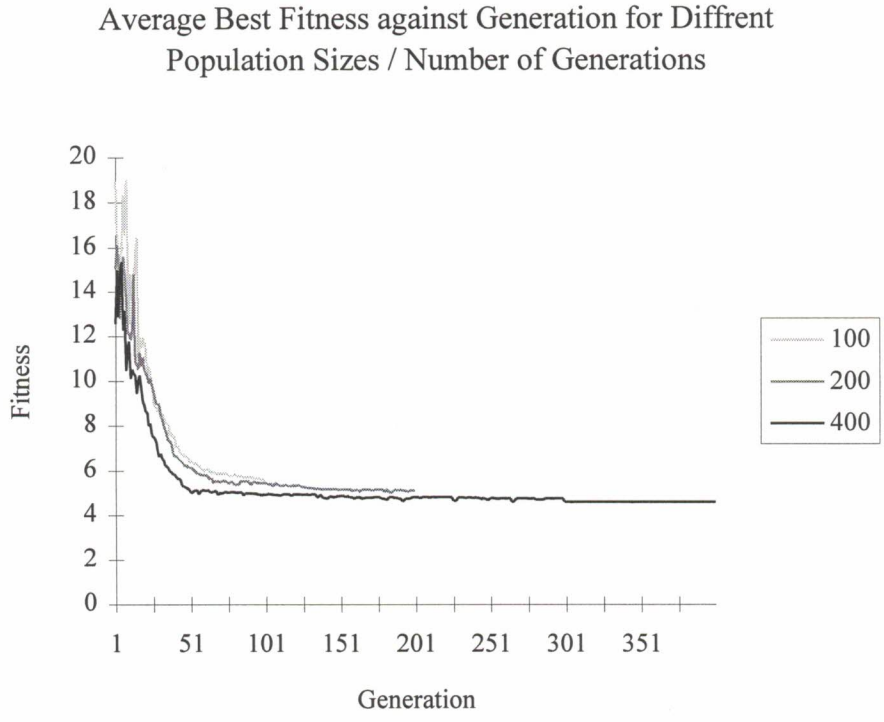


Fig. 4

This graph shows that the larger the population size the faster the GA converges to a solution and that the best final solutions are produced by longer run times.

To produce the solutions to the design problems, a population size of 400 and 400 generations were used.

5.2.2. Genetic operator frequency

To determine the optimum levels of mutation frequency, a similar strategy to that used to find the population size and number of generations was employed. The program was run 4 times using a frequencies of 1, 10, 20, and 40 percent for both "link" and "addel" mutations. A population size of 100 and 100 generations were used. Again, each run was repeated 3 times and the average best fitness values were against generation to produce the graph shown.

Average Best Fitness against Generation for Different Mutation Rates

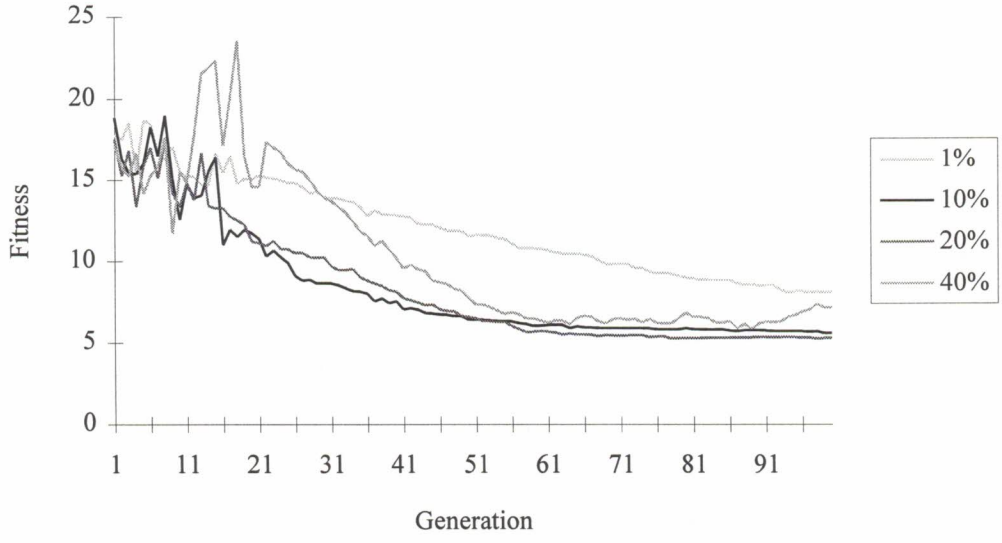


Fig. 5

The graph shows that a mutation rate of 10% produces the fastest rate of convergence. A rate of 10% was used to produce the network solutions.

5.2.3. Node list ordering

The order of nodes in the node list is maintained during the execution of the program. The list is used in the program to determine the order of links in the "DNA" representation. To determine if node order has any effect on the performance of the program, solutions to the multi-drop network were produced using two different node orders. Each node order was run three times and the mean best fitness values were plotted against generation to produce the graph shown.

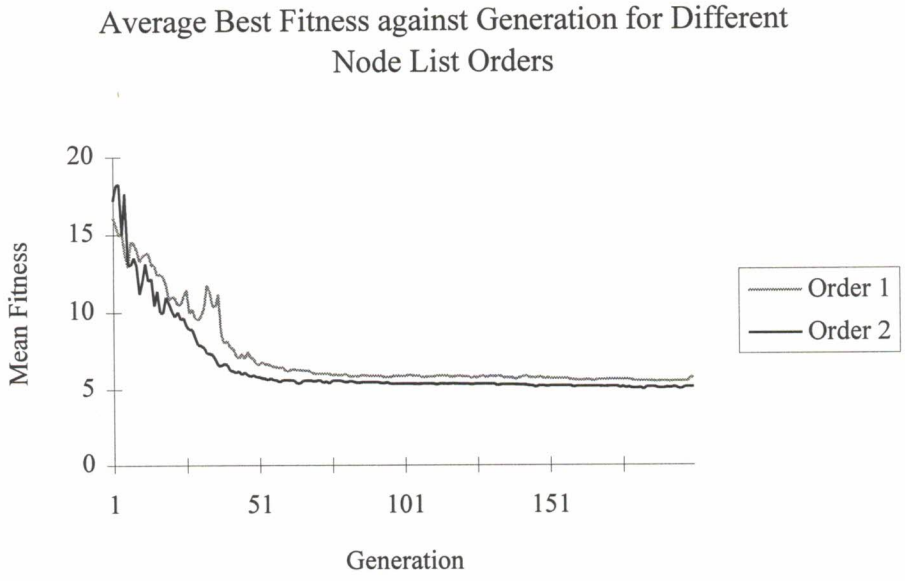


Fig. 6

The graph shows that node order possibly has an effect on the rate of convergence and the solution finally produced. The node order 2 was used to produce solutions to the network design problems and is show in the multi-drop network initialisation file (appendix 4).

5.2.4. Multi drop network

To obtain a solution to the multi-drop network problem, the program was run three times using a population size of 400 and 400 generations. The acceptable delay was set to 200 seconds so that the program would produce solutions similar to the Esau-Williams algorithm which concentrates on finding solutions where no link is saturated. The initialisation file used is shown in appendix 4. The best network produced by the program is shown below.

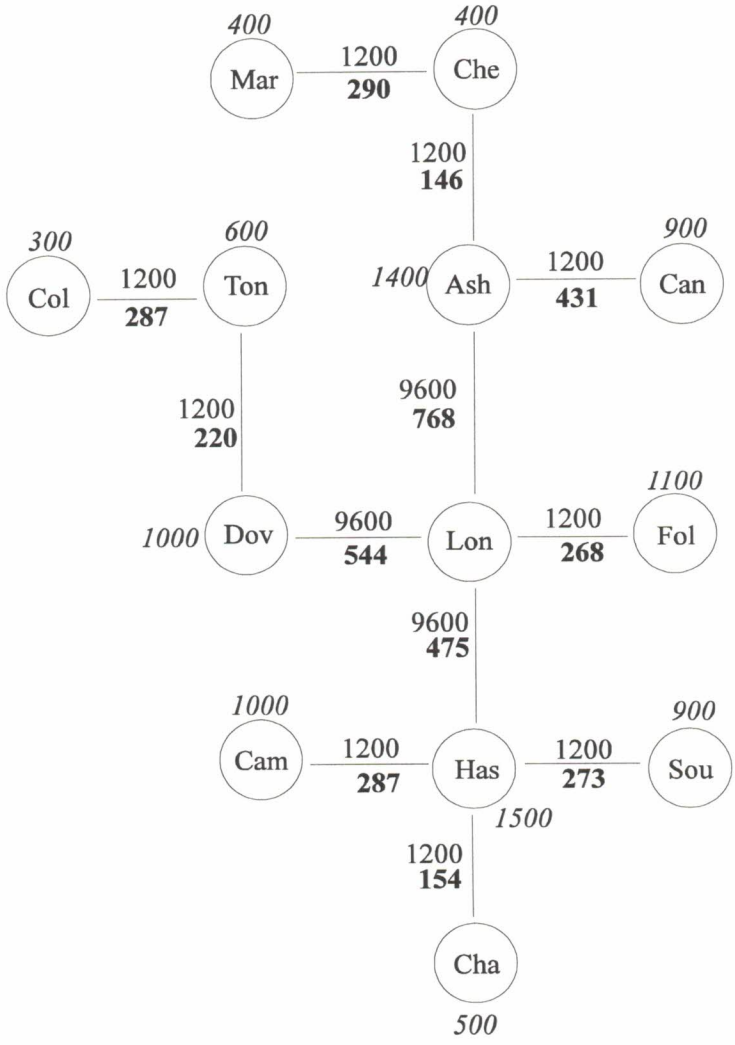


Fig. 7

For each node in the network the volume of traffic in bits per second is shown next to the node in italics. For each link the cost shown in bold and the capacity of the link in bits per second is recorded. The solution was produced by two of the three runs after 192 and 261 generations respectively. The network satisfies all the design constraints, has a cost of 4143 and mean message delay of 39.8 sec.

The output for the three runs of the program are on the enclosed diskette as 1net1.txt, 1net2.txt and 1net3.txt.

5.3. Mesh network k=3

To produce a solution to the mesh network design problem (k=3) the program was again run three times using same population size as the multi-drop problem. The acceptable delay was set to 1 second. The initialisation file used is shown in appendix 4. The best network produced is shown below.

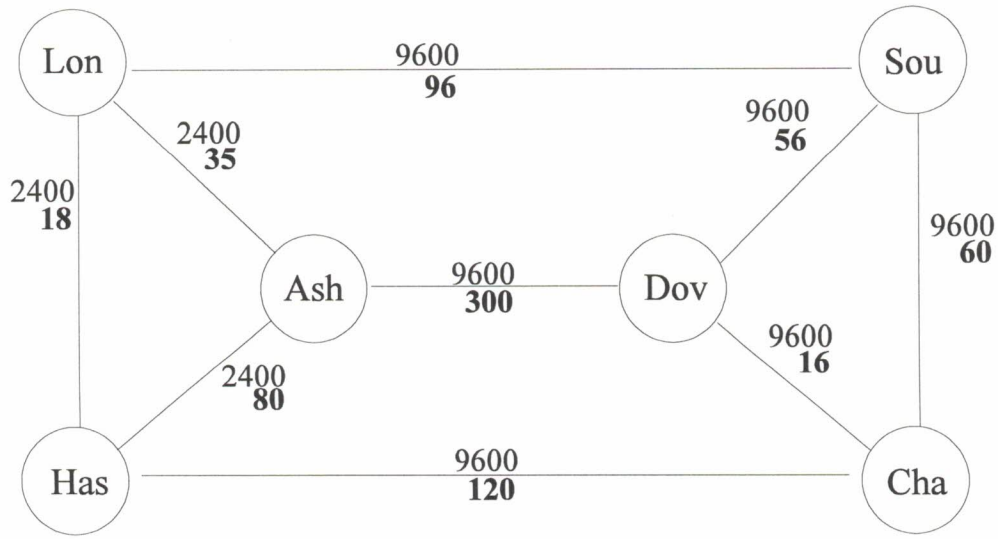


Fig. 8

The network uses the following routing strategy:

Lon - Cha via Sou.
Lon - Dov via Sou.
Lon - Ash via Sou, Dov.
Ash - Cha via Dov.
Ash - Sou via Dov.
Dov - Has via Cha.
Has - Sou via Cha.

All other routes are direct. The network, produced by generation 107 has a mean message end to end delay of 0.773 seconds and a total cost of 781. The output from the three runs is included on the enclosed diskette (2net1.txt, 2net2.txt and 2net3.txt).

5.4. Mesh network k=4

For the mesh network design problem k=4, the k=3 initialisation file was used simply changing the required connectivity. As before the program was run three times. The best network produced is shown below.

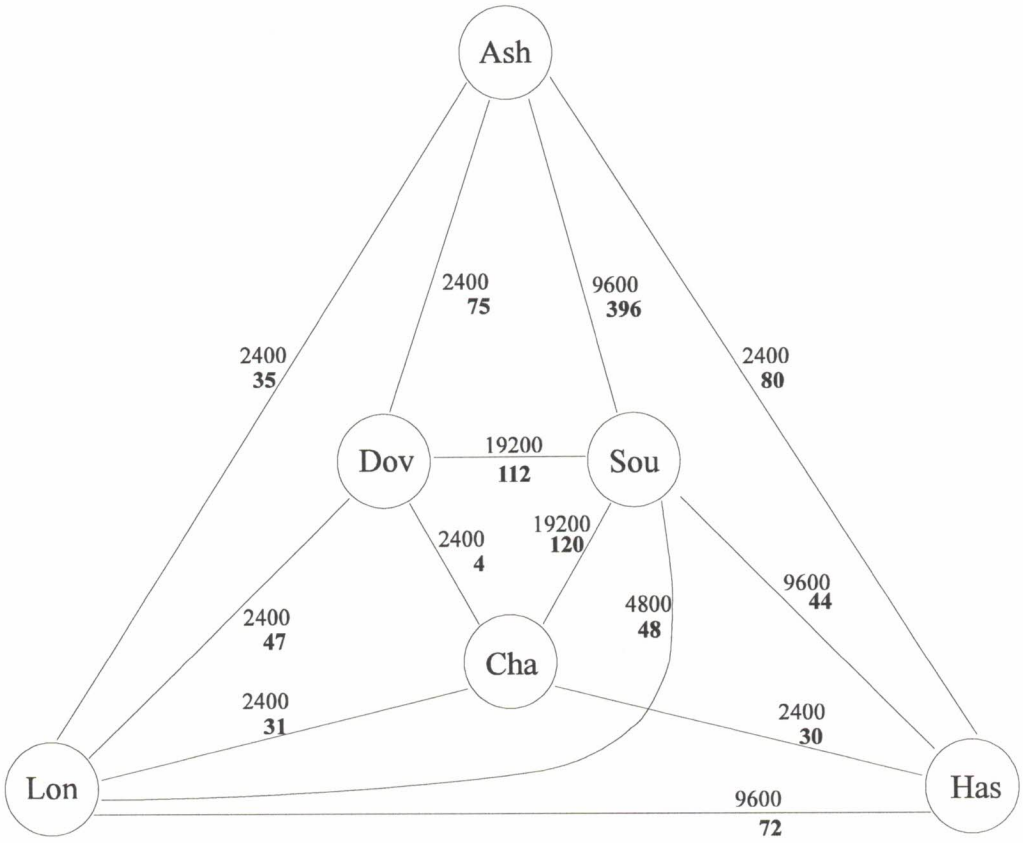


Fig. 9

The network uses the following routing strategy, all other routes are direct:

Lon - Ash via Sou, Dov.
Lon - Cha via Sou.
Lon - Dov via Sou.
Ash - Cha via Sou.
Ash - Dov via Sou.
Ash - Has via Sou.
Cha - Dov via Sou.
Cha - Has via Sou.
Dov - Has via Sou.

The network produced by generation 345, has a mean message end to end delay of 0.560 seconds and a total cost of 1094. The output from the three runs is included on the enclosed diskette (3net1.txt, 3net2.txt and 3net3.txt).

5.5. Discussion

This section discusses and explains the results produced for the five experiments.

5.5.1. Optimum parameters

The speed of convergence to a solution is dependant on population size. The larger the population the faster the convergence. As large populations are more likely to contain "fitter" genetic information, the GA will produce better solutions in fewer generations, increasing the rate of convergence. As the GA converges asymptotically to a final solution, longer program runs will produce better final solutions.

To obtain the best solutions, large population sizes and high number of generations should be used. However, using a population size of 400 and 400 generations requires a program execution time of between 1 to 2 hours. The performance of the GA must therefore be balanced with acceptable run times.

The optimal mutation rate of 10% is much higher than the 1% proposed by De Jong. However, the value of 10% is the frequency with which the program "attempts" to apply a mutation operator. As the "link" mutation is biased in favour of introducing cheaper links into the network, a percentage of mutation "attempts", which would result in the introduction of expensive links are ignored. The actual mutation rate of the "link" operator decreases during the program run as the number of networks composed of cheap links increases.

A high mutation rate is required to introduce large amounts of new genetic material so that areas of the search space, not contained in the initial population, are examined. Due to the large search space of the network design problem, the initial population of 400 networks contains only a small percentage of the total number of possible networks. At too high a mutation rate (40%), the population is prevented from converging to a stable solution by the frequent destruction of "good" networks.

5.5.2. Node ordering

The experiment suggests that node order has an effect on both convergence rate and the final solution produced. However as this result is based on only three observations it is unlikely that the difference is statistically significant.

5.5.3. Multi drop network

The solution produced by the program for the multi drop network problem is identical to that produced by the modified Esau-Williams algorithm (see appendix 3).

5.5.4. Mesh network k=3

The solution produced for the mesh network design problem k=3 costs 5.3% less than the solution produced using the link deficit algorithm (see appendix 3). The network has a mean end to end message delay of 0.773 seconds. This is higher than the 0.180 seconds for the traditional solution. Although the mean delay is higher, it is still within the required acceptable delay of one second. The GA is able to trade some performance for cost savings.

5.5.5. Mesh network k=4

The GA performed poorly on this problem, the best network produced costing 37.3% more than the solution produced by the link deficit algorithm. The network also had a higher mean end to end message delay.

The network was produced after 345 generations. This possibly indicates that the GA was given insufficient time to adequately investigate the increased search space.

6. EVALUATION

This section examines the design, implementation and testing of "CHARLEY" giving particular emphasis to areas for future study.

6.1. Design

The "DNA" representation used satisfies both the Goldberg criteria. However, grouping of the links into node sets dramatically reduces the number of possible crossover sites. This makes the crossover operator repeatably try the same combinations. The representation should be examined for ways in which the number of crossover points could be increased.

The use of biased mutation operators should be examined with a view to removing the bias completely. This would remove any danger of the GA prematurely converging on a sub-optimal solution.

The crossover operator has the tendency to rapidly increase the number of links in a network and therefore the cost. The crossover operator also produces offspring which have very different fitness values from the parents. The crossover operator should be modified so that the number of links in and the fitness of the offspring is similar to that of the parents.

The fitness function successfully identifies "good" and "bad" networks

The mesh networks evolved contain a number of links that carry the majority of the traffic; the remaining links, required to satisfy the connectivity requirement, are filled by low cost, low capacity links. In many cases these links would have insufficient capacity to carry the network traffic in the event of a link or node failure. The fitness function should be modified to reflect a network's ability to continue operating in the event of a failure.

The use of a connectivity requirement to indicate a network's acceptable reliability is too prescriptive. Ideally, the GA should be allowed to determine the required connectivity to produce a network with an acceptable reliability. A reliability requirement would be stated for each message type, together with the reliability of links and nodes. A modified fitness function would then calculate a reliability penalty based on the required message reliability and the reliability of the routes in the network.

6.2. Implementation

The use of doubly linked lists to implement the network representation and network constraints database has several advantages. However, the inherently inefficient searching degrades overall program performance. The use of data structures that can be searched more efficiently should be examined.

The algorithms used by the program, particularly the directed path search and the modified Dijkstra forward search algorithm, are generally inefficient. The efficiency of these algorithms could be easily improved, decreasing program execution times. Shorter run times would enable more extensive testing of the program to be performed.

6.3. Results

Due to the long run times, the optimal parameters used to produce the solutions to the network design problems are based on relatively few observations (n=3). The long run times also precluded a more thorough investigation of the effect of different mutation rates and node ordering. The experiments should be repeated to increase the number of observations, and an appropriate statistical test performed to reveal if any observed difference is statistically significant.

The program's performance in producing networks should also be tested on a larger number of network design problems containing larger numbers of links and nodes.

7. CONCLUSION

The design of networks, and in particular mesh networks, is a complex process. To reduce the complexity of the design process, optimal solutions for different network components (routing, link capacities and topology) are sought. The final solution is constructed from these optimal components. Although the components are treated as being independent, they are clearly dependant on each other. Decisions made to optimise one component may have a detrimental effect on the other components. The use of a GA in the design process allows networks to be evaluated as a single entity.

To test this proposal, a program "CHARLEY" was designed, implemented and its performance against traditional design techniques tested, using three network design problems, a multi drop network, and two mesh networks k=3 and k=4. The solutions produced look promising with the program performing as well as, and, in the case of the mesh network (k=3), better than traditional algorithms. The poor performance on the mesh network (k=4) was probably due to an insufficient number of generations being processed. The success of the program confirms the validity of using GAs to design networks.

Future work should concentrate on improving the efficiency of the program, investigating the effects of the various parameters on program performance and solving larger and more demanding design problems. Ultimately, the program should be modified so that networks are evolved that satisfy a reliability constraint instead of connectivity.

This page left blank.

REFERENCES

Ahuja V. 1985, Design and Analysis of Computer Communication Networks. McGraw-Hill.

Ammeraal L. 1988, Programs and Data Structures in C. John Wiley and Sons.

Baker J E. 1987, Reducing bias and Inefficiency in the selection algorithm. Genetic Algorithms and their Applications: Proceedings of the second International Conference on Genetic Algorithms.

Brindle A. 1981, Genetic Algorithms for Function Optimisation. Unpublished doctoral dissertation, University of Alberta, Edmonton.

Davis L. 1989, Optimising Network Link Sizes with Genetic Algorithms. Modelling and Methodology, Knowledge systems Paradigms.

De Jong K. 1975, An analysis of the behaviour of a class of genetic adaptive systems (Doctoral dissertation, University of Michigan) Dissertational abstracts International 36(10), 5140b. (University Microfilms No. 76-9381).

Forest S. 1985, Documentation for PRISONERS DILEMMA and NORMS programs that use the genetic algorithm. Unpublished manuscript, University of Michigan, Ann Arbor.

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

Michalewicz Z. 1991, A step towards Optimal topology of Communications Networks. Data structures and Target classification, SPIE Vol. 1470.

Reeves C. 1993, Genetic Algorithms in Modern Heuristic Techniques for Combinatorial Optimisation (Ed Reeves, C) pp 151-196, Blackwell scientific Publications, London.

Routen T. 1994, Genetic Algorithm and Neural Network approaches to local access Network design. Proceedings of 2nd IEEE International Workshop on Modelling Analysis and Simulation.

Schwartz M. 1977, Computer Communication Network Design and Analysis, Prentice-Hall.

Suttling M. 1993, Application of Genetic Algorithms to Operations Systems Scheduling. Unpublished manuscript, Final year report, University of Greenwich, London, UK.

Tannenboum A S. 1988, Computer Networks. Prentice-Hall.