Comparing different Genetic Algorithms through solving Steiner Networks

Gustav Sennton
910616-1293
CDATE3 KTH

Supervisor: Michael Minnock

May 21, 2012

Abstract

The purpose of this paper is to inspect the behavior of different genetic algorithms. Some different characteristics, of selection and reproduction of individuals, are implemented in a genetic algorithm and the results are then compared to see if some characteristics are more important than others. More specifically the reproduction methods mutation and crossover are compared and the selection methods elitism selection and biased random selection are compared. The problem, which the genetic algorithms are tested on, is called the Steiner network problem. The results indicates that using elitism selection together with mutations is the best method when solving trivial problems. For more difficult problems mixing mutations and crossovers and using biased random selection seems to be the best alternative though this result is not as certain as the fast convergence of elitism-mutation algorithms for trivial problems.

Sammanfattning

I denna rapport undersöks olika sorters genetiska algoritmer. Olika sorters selektion och fortplantning av individer implementeras i en genetisk algoritm. Resultaten av användandet av dessa olika implementationer jämförs sedan för att se hur mycket de olika egenskaperna påverkar algoritmen. Fortplantningmetoder som tas upp är mutationer och blandningar av olika individer medan de selektionmetoder som tas upp är elitist-selektion och partisk slumpselektion. Problemet som den genetiska algorimen testas på kallas Steinernätverk. Enligt de resultat som tas fram är elitistselektion kombinerat med mutationer den bästa metoden för att lösa simpla problem. När det kommer till svårare problem verkar det bäst att använda både mutationer och blandningar av individer tillsammans med partisk slumpselektion.

Contents

1 Introduction **1**

2 Overview **2**

3 Background **3**
3.1 Genetic Algorithms 3
3.2 Steiner networks 5

4 Approach **7**

5 Implementation **8**
5.1 Overview 8
5.2 Individual representation 8
5.3 Fitness 9
5.4 Reproduction methods 10
5.5 Selection methods 11
5.6 Replacement 11

6 Results **13**

7 Discussion **16**

8 Conclusions **18**

9 References **19**

Chapter 1

Introduction

One of the tools an engineer uses is reusing, that is, using already known concepts to skip redoing old work. Many such concepts are not created solely by humans; some concepts are barely interpretations of the properties of nature. One part of computer science, where concepts from nature are used, is Genetic algorithms. Those kinds of algorithms are based on the evolution theory, i.e. the concept of how organisms evolve through surviving and reproducing.

To think that the development of something as complex as a human started with some single-celled organism is incredible. This development is the result of billions of years of evolution. Genetic algorithms is a way to try to mimic those results; the basic idea of genetic algorithms is to start with some random (and thus probably unusable) individuals which are to be interpreted as solutions to a specific problem. These individuals are then evolved until they reach a state where they actually can be used to solve the problem in a good way (preferably in the optimal way).

Genetic algorithms have some features which are highly attractive. For example several solutions to a problem are handled at the same time. Some of the benefits of this is that the solutions can be improved in parallel and that the solutions can be combined.

The purpose of this paper is to compare some of the features of genetic algorithms. More specifically to compare the two reproduction methods mutation (changing a single individual) and crossover (combining several individuals) and the two selection methods elitism selection and biased random selection.

Chapter 2

Overview

In the Background section of this paper the basics of genetic algorithms and Steiner networks are explained. Some examples of different genetic algorithms are given and also some examples of solutions to Steiner networks are shown.

The Approach section explains the different tests done in this paper and why those tests are being done.

The different parts of the implementation used are explained in the Implementation section – this includes the representation of individuals, the different selection methods and the reproduction methods.

In the Results section of this paper the results, of running the different genetic algorithms with two different Steiner network problems, are shown. The first problem is a simple problem chosen to show the faster convergence of some algorithms and the other is a more difficult problem chosen to show the better performance of more sophisticated algorithms.

The results are then discussed in the Discussion section and some suggestions, on how to improve the work done in this paper, are presented.

Chapter 3

Background

One way of solving problems, where finding the optimal answer is very difficult, is using heuristics – algorithms that are fast but which might never find the optimal solution as opposed to, for example, exhaustive search which always finds the best solution but needs a lot of time. Genetic algorithms are one kind of heuristics where aspects of Darwin's evolution theory are used to improve numerous random solutions to find one optimal solution. Problems, which can be solved using genetic algorithms, are called Combinatorial Optimization problems. A combinatorial optimization problem is a problem where the goal is to find the best element in a set of elements. An example of a combinatorial optimization problem is the Traveling Salesman problem.

3.1 Genetic Algorithms

Every genetic algorithm has a population of individuals; every individual represents a possible solution to the problem at hand. An individual can be represented in many different ways depending on the problem to solve, the most common way is, according to Sivanandam and Deepa (2008, p.43), to represent an individual as a bit-string. This sort of representation is easy to implement because for example mutating the bit-string can be done through simply flipping some bits.

A genetic algorithm spans over a number of generations – a step from one generation to the next is called an iteration. According to Sivanandam and Deepa (2008, p.30) every iteration of a genetic algorithm consists of four steps; Selection, Reproduction, Evaluation and Replacement. Evaluation is simply a matter of computing the fitness of the individuals of a population and Replacement is the act of choosing which individuals, of the old population, to replace with new ones. Selection and Reproduction are described in the sections below.

Selection

Selection is the method of selecting the individuals which will reproduce. Selection, just like Replacement, is done to improve the total fitness of the population from one generation to the next, i.e. to improve the solutions of the algorithm. One example of a selection method is; merely selecting the fittest individuals of a population, this is called elitism selection. Another example is to assign a bias to each individual, based on that individual's fitness, and then filter the population randomly, using the individuals' biases. In this way the fittest individuals will have an advantage while the less fit individuals still have a small chance to survive until the next generation. The biased random selection method is by Sivanandam and Deepa (2008, p.47) called a roulette wheel selection.

It is important to note that the biases of a random selection method must be chosen with care since choosing small biases will make the selection algorithm totally random while choosing large biases will make the selection method similar to elitism selection.

Reproduction

Reproduction is when the individuals breed to create new offspring. The reproduction method of a genetic algorithm is one of the most important parts of the algorithm since this method determines how new solutions are created. There are two reproduction methods: mutating (changing) a single individual and crossing (combining) multiple individuals. Mutations and crossovers can be done in many different ways. When using bit-strings one of the simpler ways to perform a mutation is, as stated above, to flip some bits of the string. Some examples of crossover, used with bit-strings, are mentioned by Sivanandam and Deepa (2008, p.51); two of those examples are single-point crossover and multi-point crossover. Single-point crossover means choosing one point in the representation of an individual. The part to the left of that point is then copied from the first parent's representation to that of the child. Then the right part of the second parent's representation is copied to the right part of the child's representation to create a whole new representation. An example of single-point crossover is shown in Figure 3.1. A multi-point crossover is similar to a single-point crossover but, instead of one point, several points are chosen so that several different sections of the representation are taken from the two different parents. An example of multi-point (two-point) crossover is shown in Figure 3.2.

An important concept when discussing genetic algorithms, or heuristics in general, is the term "fitness landscape" which is, according to Mitchell (1999, p.6) "a representation of the space of all possible genotypes along with their fitnesses" (a genotype here means an individual's representation). The term fitness landscape is important because knowing the topology of the

Parent 1: 100101
Parent 2: 010100
Child: 100100

Figure 3.1: A single-point crossover

Parent 1: 1001011001
Parent 2: 0101000111
Child: 1001000001

Figure 3.2: A two-point crossover

fitness landscape, corresponding to a specific problem, helps constructing better algorithms for that problem. For example when solving a minimization problem, with a fitness landscape which has lots of local minima, the methods used must be able to handle a population with great diversity so that alternative solutions can be presented to overcome the ones stuck in local minima.

3.2 Steiner networks

The Steiner network problem is, according to Agrawal, Klein and Ravi (1995, p.440) a problem where you want to connect a set of nodes in a graph so that the total length of connections is as small as possible. The connections between nodes of the network does not have to be straight lines; you could set up imaginary connection points, in the network, which the connections can then pass through. In figures 3.3, 3.4 and 3.5 examples of solutions to Steiner network problems are shown. The original network nodes are colored black and the imaginary connection points are colored red. In Figure 3.3 and Figure 3.4 two different solutions to the same problem are shown; the solution shown in Figure 3.4 is better than that in Figure 3.3 since the total length of the connections in Figure 3.4 is smaller. In Figure 3.5 a solution, containing two inner connection points, to a second network is shown.

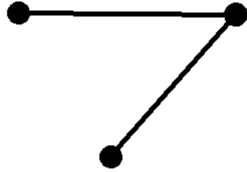

Figure 3.3: A simple, and suboptimal, solution to a Steiner network problem

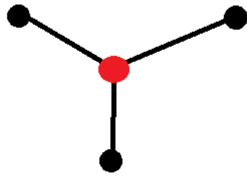

Figure 3.4: A (close to optimal) solution to a Steiner network problem. The solution contains one imaginary connection point colored red

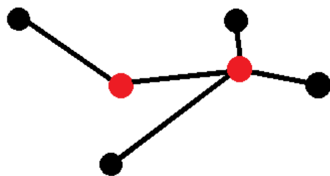

Figure 3.5: Another (sub-optimal) solution to a Steiner network problem. The solution contains two imaginary connection points colored red

Chapter 4

Approach

There are two selection methods used in this paper; elitism selection and biased random selection (roulette wheel selection). These two methods should have a difference in performance in problems with different fitness landscapes; because of this the algorithms implemented are tested on two problems which should be different in this aspect. The first problem is a simple 3-node problem and the second one is a more difficult 5-node problem. Thus the hypothesis is that the elitism selection method is of better use in the simple 3-node problem while the biased random selection should perform better in the difficult 5-node problem.

To be able to compare mutations and crossovers both mutations and crossovers use a two-point method in this implementation. The number of bits changed by the two-point method is changed during testing since crossovers probably(this is tested) are better when changing many bits and mutations probably are better when changing fewer. This is because mutating too many bits of an individual will probably change the individual too much to be effective. Crossover on the other hand should perform better than mutating when the number of bits changed is high since crossover mixes two good solutions.

One important parameter tested in this paper is the size of the population – for example when using only crossovers the diversity of the initial population is crucial.

Chapter 5

Implementation

5.1 Overview

As stated in the Background section the step of going from one generation to the next, in a genetic algorithm, is called an iteration. In every iteration of this implementation half of the population is chosen to reproduce to create the next generation. When the reproduction has been done the offspring produced replaces the individuals that were not chosen for reproduction(and possibly some of the ones that were chosen too) and thus a new generation is created.

5.2 Individual representation

In the implementation used in this paper every individual has three different variables. The first variable is an integer representing the number of imaginary connection points used by the individual. The second variable is an array of inner connection points (i.e. positions in two dimensions which can be represented as tuples of x,y-values), every x- and every y-value is an integer which can be represented as a bit-string. The third variable is a bit-string representing the connections between different imaginary connection points and also between those points and network nodes. Every bit in the bit-string represents the existence of a connection. The maximum number of connection points is fixed and the number of points represented in the point-array is always the same; the maximum. The minimum number of points is always one – having zero points in an individual does not make sense in this implementation because then there would be no connections between any network nodes.

Figure 5.1 shows an example of an individual. The maximum number of connection points of the individual is two and there are three network nodes in the network. The positions of the connection points are represented as x- and y-values of the form (x,y). In the bit-string representing the individ-

ual's connections the connection from one point to itself is also represented. This does not add anything since a connection from one point to itself has length zero but it is there for ease of implementation. Every such connection therefore works as a dead bit – changing it will not change the solution.

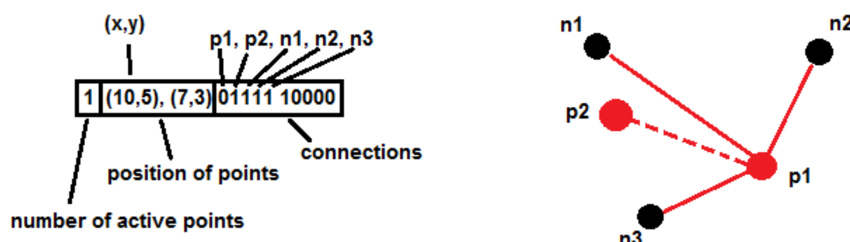

Figure 5.1: A representation of an individual and the graph corresponding to that representation. The positions of the network nodes are not specified in the representation and could therefore be placed anywhere. The connection between the two connection points is dotted since the second point (p2) is inactive, meaning that its connections does not add to the total length of the individual's connections.

In the implementation used in this paper there are never any direct connections between two fixed network nodes – there is always an imaginary connection point linking different network nodes together. The algorithm can still have inner connection points on the same positions as fixed network nodes which means that there can still, in a sense, be direct connections between the positions of the fixed network nodes.

5.3 Fitness

In every genetic algorithm an individual represents some solution to a problem. In this case the problem at hand is a Steiner network problem and the parameters which define an individual are therefore the inner connection points of a solution and their connections to each other and to the fixed network nodes. The goal of the Steiner network problem is to create networks with as small total length of connections as possible. The fitness of an individual should therefore directly depend on the total length of the individual's connections. Thus the individual with the best fitness has the lowest total length of connections and therefore represents the optimal solution. If an individual has not connected all the nodes of the network to each other the fitness of that individual is set to a large default value so that the individual is considered unfit.

5.4 Reproduction methods

The different algorithms implemented use either mutations, crossovers or both as reproduction methods. If both are used the distribution is random and uniform, i.e. the difference between the number of mutations and the number of crossovers is small.

Mutation

Each mutation can change only one of the three variables in an individual's representation; the number of connection points, the positions of the points or the connections between the points.

There are only three ways in which the integer, representing the number of points, can be mutated – the number can be increased by one, decreased by one or unchanged. The number is only unchanged if the mutation would make the integer go out of bounds, i.e. decrease to zero or increase above the maximum number of connection points allowed.

Since all the imaginary connection points are represented by values in an array which has a fixed size, changing the number of points only means using a greater or smaller part of that array. For example; an individual having a maximum of five imaginary connection points also has an array, of size five, holding those points. Increasing the integer, representing the number of connection points used by that individual, will not add a new random point but rather let the individual access the next point in the point array. Thus there can be inactive connection points within the representation of an individual. Those inactive points can still be changed through reproduction methods and there can therefore be delays in the algorithm before changed points are activated – changes being done during one generation can go unseen for several generations.

When it comes to mutating the positions of the connection points the x- and y-value of a connection point are both considered to be bit-arrays – when performing a mutation either an x-value or an y-value can be changed through flipping a number of its bits. The x- and y-values have a fixed maximum size which prevents them from mutating to a value far away from the network nodes. For example a value could have a maximum size of seven bits – then the value holds an integer between zero and 127. In this implementation every x- and y-value is represented by ten bits meaning that every value is an integer between 0 and $2^{10} - 1 = 1023$. The search space of the positions of this algorithm has therefore a size of $1024^2 = 1048576$ different positions.

The last variable which can be mutated is the bit-string representing connections between different points and also between connection points and network nodes. Every bit represents a connection between two points or one point and a network node. Mutating the bit-string is done through

flipping a number of the string's bits which is equivalent to removing and/or adding some connections.

Crossover

In this implementation a crossover is quite similar to a mutation – there are only two main differences.

First off, when changing the number of points of an individual, instead of randomly choosing whether to increase or decrease the number of points, by one, a crossover copies the value of the first parent and then increases or decreases that value, by one, depending on the number of points the second parent contains. Thus if the first parent has seven active connection points while the second parent has only three active points the child will have $7 - 1 = 6$ active points.

Secondly, instead of flipping bits of the bit-strings representing the positions or the connections of connection points, the bit-strings are first copied from the first parent and then a number of bits are changed to the corresponding bits of the second parent. This is similar to a two-point crossover as is shown in Figure 3.2 on page 5.

5.5 Selection methods

Two different selection methods are implemented – elitism selection and biased random selection(roulette wheel selection). The implementation of elitism selection is done through simply sorting the individuals depending on their length. Biased random selection is done through giving every individual a new variable called *luck*. Every individuals fitness is then calculated as $length * luck$ instead of just *length*. Depending on how random this *luck*-variable is this selection method can be classified as anything from random selection to elitism selection. In the implementation explained in this paper *luck* is a random integer between 100 and 200 meaning that the most unlucky ($luck = 200$) individual must have a length of half the size as that of the luckiest ($luck = 100$) individual to be just as fit. In both the implementation of elitism selection and that of biased random selection used in this paper the number of individuals selected to reproduce is half the size of the population.

5.6 Replacement

Since the selection methods of this implementation both select half of the population, for reproduction, each selected individual needs to produce two individuals for the next generation. This is done through using a reproduction method(i.e. mutation or crossover) on each individual twice(to create

two new individuals) and then choosing the two individuals with the best fitness out of the old individual and the two new ones.

Chapter 6

Results

The only difference between the test shown in Figure 6.1 and that shown in Figure 6.2 is the size of the populations used. The values shown in those two figures are average total lengths taken over the different tests. I.e. every algorithm is run *Tests* number of times and during every run the best solution (the fittest individual) from every generation is saved. The length of individuals from different test runs are then added and divided by *Tests*. In Figure 6.3 and in Figure 6.4 the results shown are not averages over the different tests but rather the best solutions found in those tests. This is because taking the average over the solutions gave the same results in most of the different algorithms. The maximum number of bits, in an individual, which can be changed through mutation or crossover is also changed when it comes to the five-node problem. The maximum number of bits changed is two in the three-node problem and ten in the five-node problem.

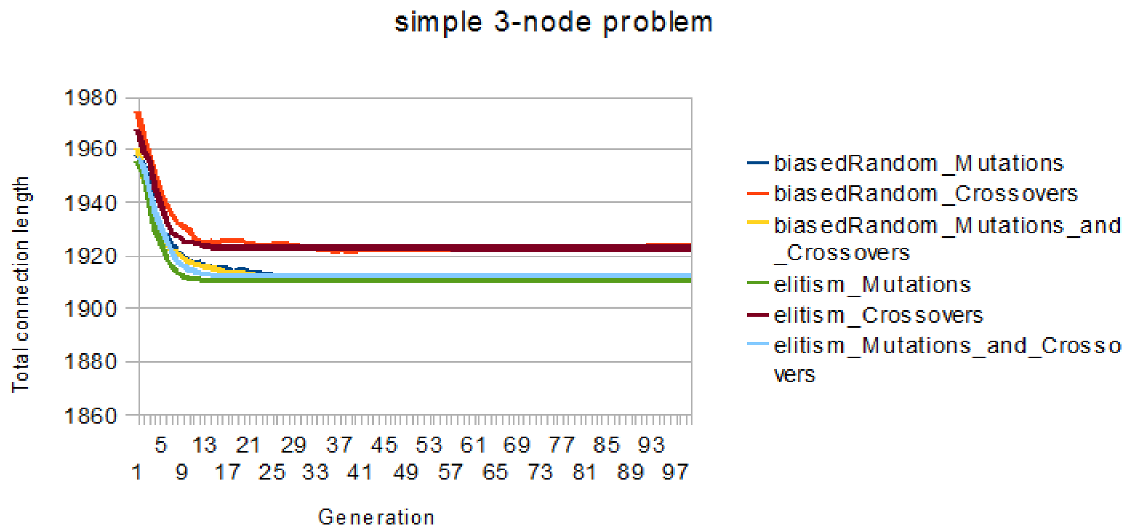

Figure 6.1: A diagram showing the results of using different genetic algorithms to solve a simple Steiner network with three nodes. Population size: 100, Generations: 100, Tests: 100

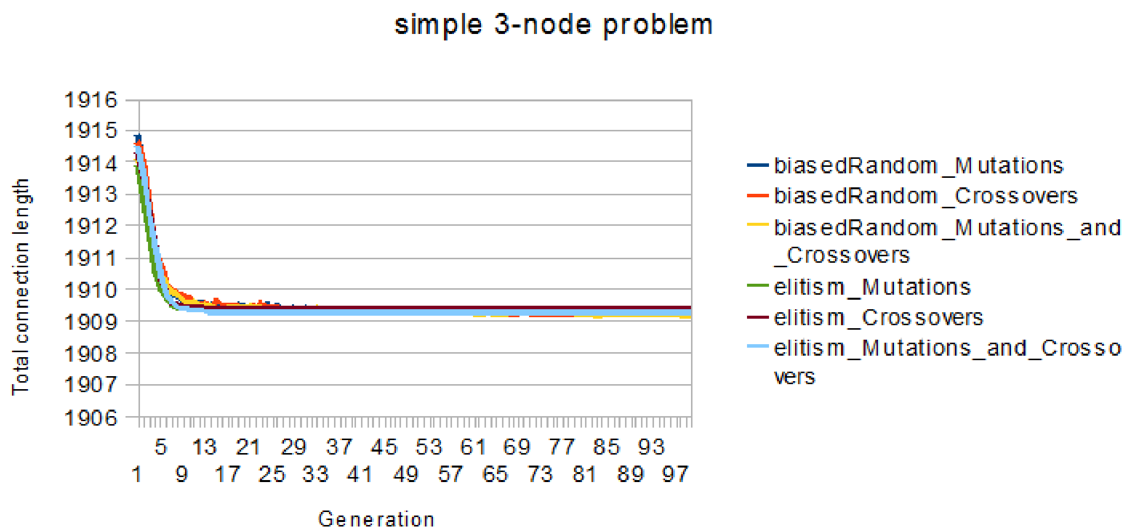

Figure 6.2: A diagram showing the results of using different genetic algorithms to solve a simple Steiner network with three nodes. Population size: 1000, Generations: 100, Tests: 100

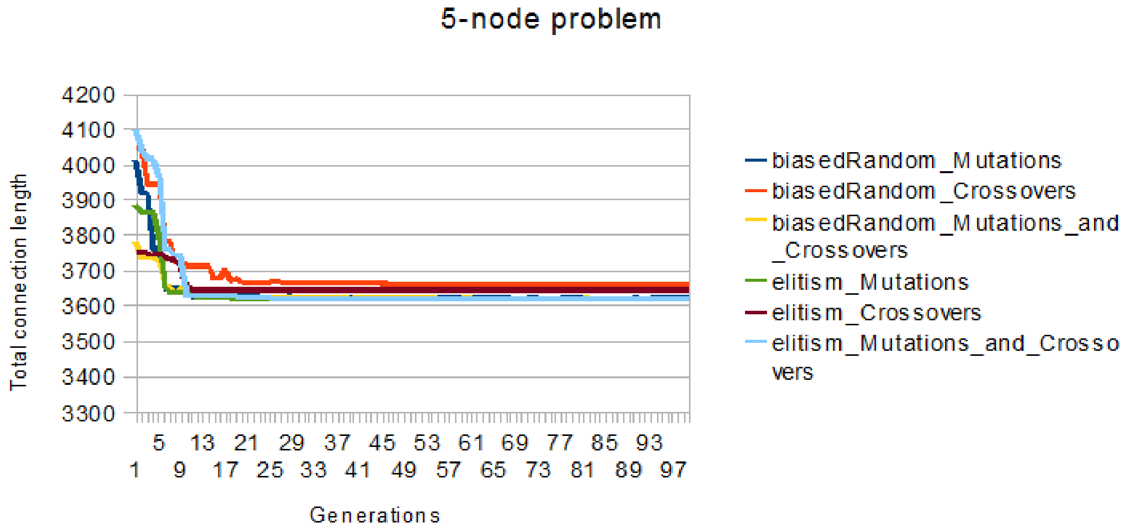

Figure 6.3: A diagram showing the results of using different genetic algorithms to solve a Steiner network containing five nodes. Population size: 100, Generations: 100, Tests: 100

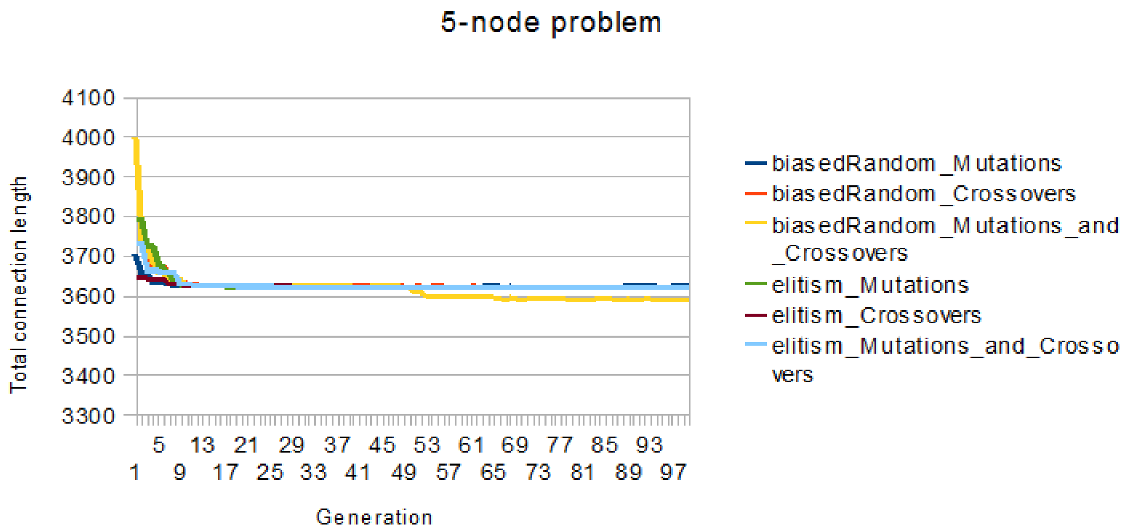

Figure 6.4: A diagram showing the results of using different genetic algorithms to solve a Steiner network containing five nodes. Population size: 1000, Generations: 100, Tests: 100

Chapter 7

Discussion

Figure 6.1 shows the results of running the genetic algorithm on a very simple three-node problem, the problem and its (close to) optimal solution are shown in figure 3.4 on page 6. The results of using only crossovers for this problem are clearly visible in figure 6.1 where both the algorithms which use crossovers show worse results than the other algorithms. The cause of this is probably the small populations used – the size of the populations is only 100 and there is probably not enough diversity in such a small population to achieve good results from a crossover-algorithm. Another observation that can be made from Figure 6.1 is the better convergence of the elitism-algorithms, than the biased random-algorithms, around rounds 5-25. Elitism algorithms should be better suited for problems with few local minima while the biased random algorithms should be better suited for more complex problem where many different types of solutions should be tested. That is probably why elitism works better in this simple problem.

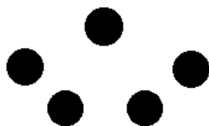

Figure 7.1: The placement of the nodes of the five-node problem tested in this paper.

The five-node problem shown in figure 7.1 is a little more complex than the three-node problem. Just as in the three-node problem the pure crossover-algorithms perform a little worse than the rest when the population size is small, as can be seen in figure 6.3. In figure 6.4 the only algorithm standing out from the rest is the one which uses the biased random selection and both mutations and crossovers as reproduction. As stated in the Approach section crossover should be more effective when the number of bits changed during reproduction is high since the point of having crossover

is combining parts of several good solutions. This does not seem to be the case in the five-node problem, which is a bit surprising. Both the crossover-algorithms are still worse than all the others when a small population is used, even though the number bits changed during reproduction is higher.

One problem with the implementation used in this paper can be found when activating an inactive imaginary connection point. Before the connection point is activated there must be connections between the connection points and all the fixed network nodes of the network. Then when the inactive point is activated, and if that point itself is connected to some node, there will be some redundancy in the connections since the newly activated connections are not needed. This will give the new individual a greater length than its parent and the individual's fitness will be worse than that of its parent. Using the fitness of an individual's parents to support the individual could, in this case, increase the diversity of the population. This is one of the improvements which can be made to the algorithms used in this paper and this shows how there are other features which should be tested before drawing any final conclusions considering the differences between different genetic algorithms.

Chapter 8

Conclusions

To get a better view of the different methods used in this paper more parameters should have been implemented and tested. For example the implementation could have involved letting the individuals' fitness depend on their parents or letting the number of bits, changed during reproduction, decrease over time.

One parameter that seems to be very important is the size of the population used, which is pretty natural – at least when it comes to crossover. Using only crossover is a method that should be avoided, using only crossovers might suffice when working with huge populations but in the cases tested in this paper using only crossover is either worse than everything else or, at best, as good as the rest.

According to the tests done in this paper simple methods like elitism selection and using mutations are best when solving simple problem while using both mutations and crossovers and using biased random selection seems to be the best option when it comes to more difficult problems.

Chapter 9

References

- Sivanandam, S.N. and Deepa, S.N., 2008. *Introduction to Genetic Algorithms*. New York: Springer Berlin Heidelberg
- Mitchell, M. 1999. *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: The MIT Press
- Agrawal, A., Klein, P. and Ravi, R., 1995. When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks. *SIAM Journal on Computing*, Vol. 24, No. 3, p.440-456