Packet Transmission Optimisation using Genetic Algorithms

Mark Withall, Chris Hinde, Roger Stone, and Jason Cooper

Department of Computer Science, Loughborough University, Loughborough, Leics. LE11 3TU, United Kingdom
{m.s.withall2,c.j.hinde,r.g.stone,j.l.cooper}@lboro.ac.uk

Abstract. A Genetic Algorithm (GA) is used to optimise the parameters for a sequence of packets sent over the Internet. Only the parameters that a client machine can change are used and the fitness is based on the delay time returned by the Traceroute program. The GA performance is compared to a fixed packet size with no priority used to assess the status of the network. The GA generally performed to the same level as the control settings but in some cases significant improvements were made.

Keywords. Internet Applications, Genetic Algorithms, Adaptive Control

1 Introduction

Various work has been done on using heuristic and adaptive techniques, such as Genetic Algorithms (GA), to solve networking problems (see for example [4, 6, 14]). The main focus of this work has been on areas such as network topology design, routing table construction and performance analysis. Most of these areas involve having some form of control or access to large areas of the network. This work looks at what performance increase, if any, can be gained from the perspective of a single client machine on the Internet, without having any external effect on the network other than the packets sent.

The aims of the experiments were to find good parameters for a sequence of packet transmissions to optimise the time taken to send 10,000 bytes of data (experiment 1) and minimise the delay of the slowest packet (experiment 2), over the Internet. The parameters being varied were the packet size, and hence the number of packets, and the priority settings of the packets. The optimisation was performed using a Genetic Algorithm[9] and was compared to a control setting, to monitor the state of the network, of the maximum packet size and no priority settings.

2 Genetic Algorithms

Genetic Algorithms (GA) began in the 1950s, some of the early papers being by Fraser [7, 8] and Bremermann [1]. Later on work by Holland[10–12] popularised

genetic algorithms and Holland's work is often cited as the origins of Genetic Algorithms. GAs are based on Darwin and Wallace's theories of *Natural Selection*[5, 18] and Gregor Mendel's theory of *Genetic Inheritance*[15]. A GA takes a population of possible solutions to a given problem (individuals), evaluates these solutions based on some criteria (fitness) and then genetically recombines the solutions based on the fitness of the individuals (using genetic operators) in the population to form a new generation of the population. This process is repeated until some termination criterion is met. Figure 1 gives the basic flow diagram for a GA.

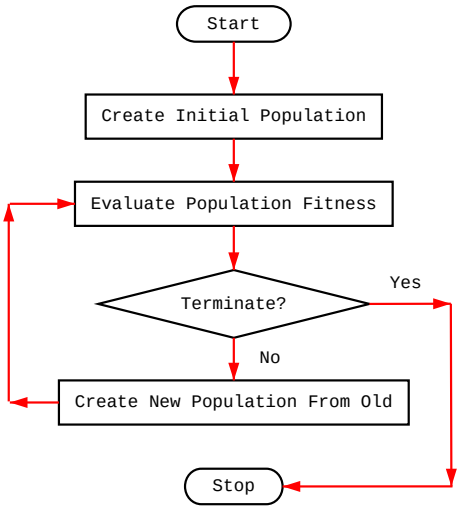


Fig. 1. Flow diagram for the basic Genetic Algorithm

2.1 Representation

The way that the individuals are represented (genome) can have a great effect on the performance of the GA. Traditionally, GA individuals are represented as binary strings, however, any representation that is appropriate for the problem can be used.

2.2 Initial Population

The initial population of a GA is important as it specifies the gene values which the GA has to work with. This gene pool can then be expanded using mutation as the GA runs. It is usual to start with a randomly generated population.

2.3 Fitness Evaluation

The fitness evaluation, for the GA, determines how good an individual from a population is at solving a given problem. The fitness is traditionally given as some integer or real number value, where the higher the value the greater the fitness. The fitness value is used to select parents for the reproduction stage of the algorithm.

2.4 Reproduction

Parent selection is used to determine which individuals from a population will be used to create the next generation of individuals. Genetic operators are used to manipulate the genes of the selected parents to create new individuals for the next generation. There are two main genetic operators: *crossover* and *mutation*.

Crossover: This consists of combining the genes from two or more parent individuals to create a new individual. This can take the form of picking a random point on the genome and using the genes before this point from one parent and those after from the other. Multiple points can be chosen and used in the same way. At the most extreme, a decision can be made for every gene for which parent to take it from.

Mutation: This consists of changing one or more genes in the individual to a new random value.

2.5 Termination Conditions

The two main methods of terminating a GA are to pre-specify a number of generations to run the algorithm for or to run until a member of the population reaches a specific fitness level.

2.6 Variables

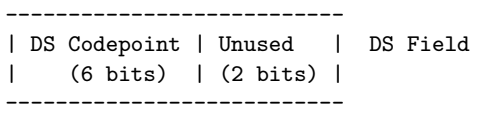
Both the population size and the probability of a mutation can be instantiated with different values. The values set can greatly affect the performance of the GA.

3 Experimental Procedure

The following subsections describe the particular GA being used for the experiments. The first subsection describes the representation being used for the individuals in the population. The second subsection describes the method of fitness evaluation for the individuals in each experiment. The third subsection describes the genetic manipulation operators being used and other parameters for the GA. Finally, the hardware and software used for testing is summarised.

3.1 Problem Representation

There are two parameters being varied in this experiment: the packet size and the priority. As two parameters would be too few to be usefully varied, the values are represented in binary. The packet size is represented as an 11-bit value (0–2047), however, the packet size is restricted to being between 500 and 1460 bytes in length, plus a 40 byte header. If the packet size is less than 500 it is rounded up and if greater than 1460 it is rounded down. The priority is represented as an 8-bit value which directly maps to the 8 bits for priority in the packet header, shown in Figure 2. Therefore, the representation of each individual is a 19-bit binary string which means there are 524288 possible settings being searched through, although some will be functionally the same.



Bits 0-2 define class Bits 3-5 define relative priority with class

[Format from http://www.itec.uni-klu.ac.at/~hellwagn/RN-QoS/rn2-kap5-DiffServ.pdf]

Fig. 2. The DiffServ priority octet

3.2 Fitness Evaluation

The fitness evaluation for the GA was conducted by recording the delay time for each packet of a 10,000 byte message being sent over the Internet. For experiment 1, the total time taken (sum of packet delays) to send the 10,000 bytes was used as the fitness value. For experiment 2, the time taken (delay) for the slowest packet was used as the fitness value. The data was sent to an IP address in Japan, as the path was quite a slow one. The source and destination IP addresses are given in Figure 3. To test the effectiveness of a transmission strategy takes a week or two. The method used here will not tell us the effectiveness of the individual except at the time of the test. The reason for using this method is that it is quicker and that it will show if there are improvements to be made at times. If it shows that there are times where the GA performs better than the control then it would be worth running a GA using a partial fitness function [2] to evolve transmission strategies.

The data was sent using the Traceroute program with the following command line switches:

```
traceroute -q 1 -n -S 28 -t <priority> <address> <packetsize>
```

The `-q` switch specifies how many packets to send, the `-n` switch keeps output concise, the `-S` switch specifies only to print the information for the destination address and not intermediate servers and the `-t` switch specifies the priority.

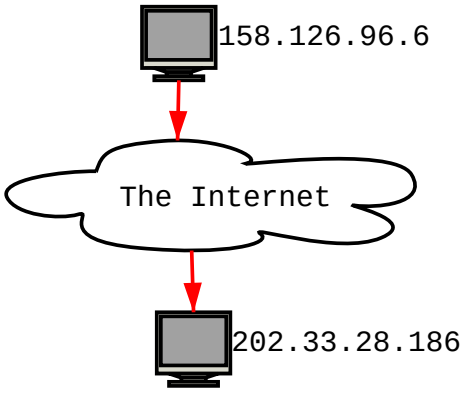


Fig. 3. Source and destination addresses used for the experiment

The shorter the time, for both experiments, the better. This value is inverted and normalised so that the higher the value the fitter the individual, for easier selection at the reproduction stage.

In addition to the GA tests, a control was run with no priority and a packet size of 1500 bytes (including header). This control was run once for every individual in the population i.e. ten individual runs and ten control runs.

3.3 Genetic Operators and Parameters

The GA uses two types of genetic operator. The first is a crossover operator, where two parents are chosen and some genes from one parent and some from the other are used to create the new individual. The crossover is uniform i.e. performed at every point in the genome[16]. The second genetic operator is mutation, where one gene value is randomly changed to a new random value. All new individuals are created by the crossover of two parent individuals with some probability of mutation in the new individual.

The population size used for the experiment was 10 individuals and the mutation rate was a probability of 1 gene in 100. The population was initialised randomly.

3.4 Hardware and Software

The GA was written in Perl version 5.6.1[17] and used Traceroute version 6.0 GOLD. The experiments were run on a Sun Sparc 4 with a Debian Linux operating system. The machine was connected to the Internet over a 10Mb/s switched LAN and to the SuperJanet network.

4 Results

The following sections give the results for the two experiments. The aim of experiment 1 was to minimise the total delay to send 10,000 bytes of data over

the Internet and the aim of experiment 2 was to minimise the delay for the slowest packet in the same transmission.

4.1 Experiment 1

The graph in Figure 4 shows the best control at each generation against the best evolved individual at each generation.

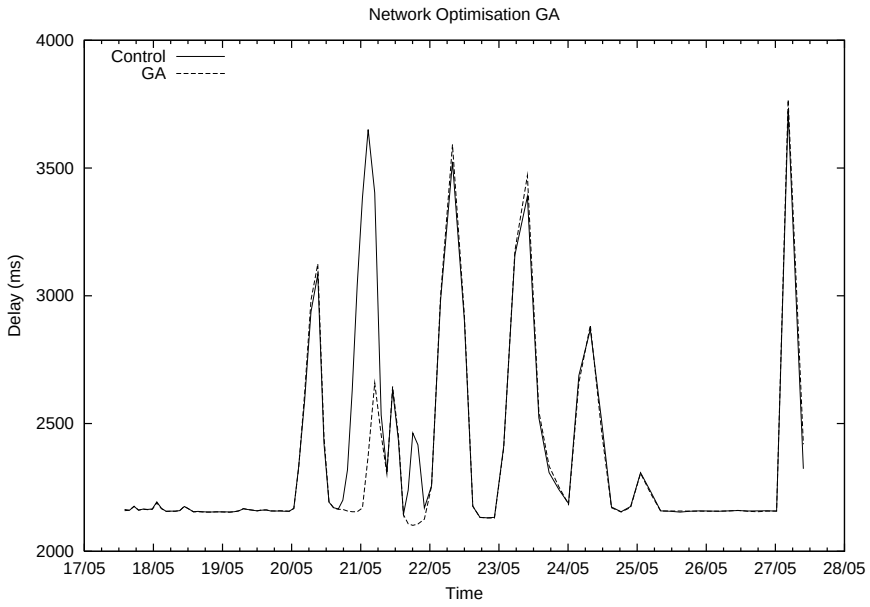


Fig. 4. Delay against time for the best control and best GA solution, at each generation, for experiment 1

For most of the experiment, the best GA individual and the best control performance are virtually identical. The GA quickly learns that the largest packet sizes perform best. There are two main occasions where the GA performs significantly better than the control. These occur at around generation 3000 (∼00:00 21/05) and generation 4440 (∼18:00 21/05). The priority settings at these points are generally 199 (class 6, priority 1) and 207 (class 6, priority 3) in the first instance and 227 (class 7, priority 0) and 243 (class 7, priority 4) in the second. The result seems logical as the higher priority classes would be likely to receive better service. However, it is uncertain why the GA can not maintain this performance.

4.2 Experiment 2

The graph in Figure 5 shows the best control at each generation against the best evolved individual at each generation.

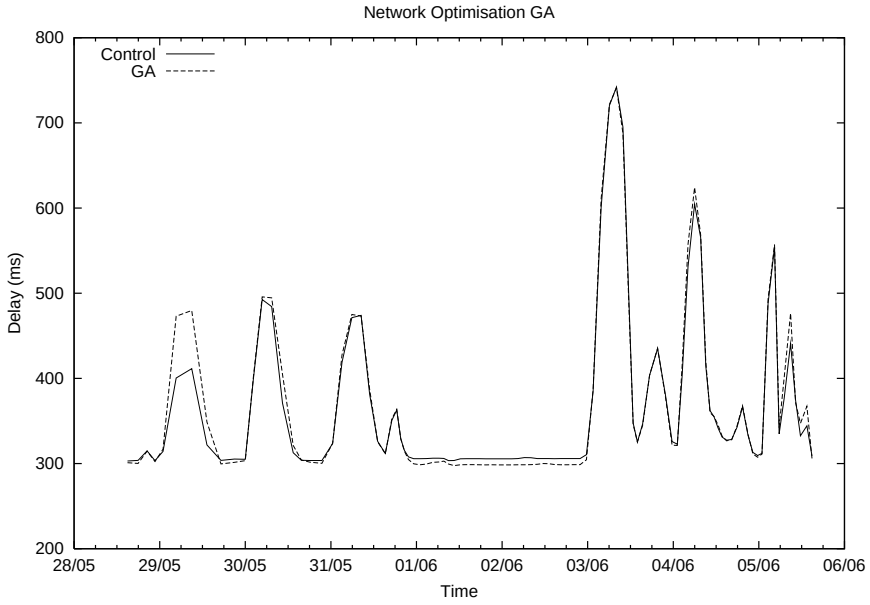


Fig. 5. Delay against time for the best control and the best GA solution, at each generation, for experiment 2

The results for experiment 2 are quite different to those in experiment 1. The GA in most cases manages to keep up with the performance of the control when the network is busy but when the network is quiet it can outperform the control. The results show that the GA uses a large packet size when the network is busy. This seems reasonable as sending less packets decreases the chance that one of them will be held up in the congestion. When the network was quiet, the GA used a small packet size. This gives enough of a performance improvement to beat the control with a fixed maximum packet size when there is less chance of a packet being held up in traffic. The priority settings being used in the three main areas of improved performance were 33 (class 1, priority 0, at ∼20:00 29/05), 120 (class 3, priority 6, at ∼19:00 30/05) and 123 (class 3, priority 6, at ∼00:00 02/06). It seems as if the main performance improvement for the GA in this situation is caused by the decrease in packet size rather than the priority setting.

5 Summary and Conclusions

For experiment 1, the GA performed, in general, at the same level as the control settings. However, for two periods the GA showed a significant improvement over the control by using high priority classes. Unfortunately, this performance could not be maintained. For experiment 2, the GA performed well during the periods of low network congestion but only to the level of the control settings or worse during high congestion periods.

In conclusion, the GA managed to adapt to different conditions on the network, and different fitness requirements, quite well but was inconsistent in its ability to outperform the control settings' performance. Although external factors play the major role in determining transmission delays the experiments have shown that some improvements can be made from the perspective of an individual machine. Further work is required to find if any consistent improvements can be obtained over a simple fixed parameter setting.

Alternative approaches to the problem include a *Nested Evolution Strategy*, which has been shown to be good at many adaptive problems. *Genetic Programming* could be used to produce a parameter control program to adjust the packet settings based on the performance of previous packets sent (and perhaps even other known facts about the network being used). This approach would probably be an improvement of the simple GA, which is just reacting blindly to the changing conditions of the network without memory of previous network changes. The disadvantage to this approach is that a reasonable complete fitness evaluation would take about a week per individual (or even more) and therefore it would take a very long time to evolve good solutions.

Partial fitness functions [2] is a promising approach which uses an individuals fitness combined with the age of the individual, this allows the fitness testing to continue to completion but also uses early results to permit promising individuals to breed. A study using evaluations of over a week shows that the most effective policy depends on the destination and route taken. For example, priority settings on one route which produce good transmission times fail to replicate across a range of routes but do replicate across limited sets of routes. This indicates that the optimum strategy depends on the destination and route chosen [3].

6 Acknowledgements

Thanks to Nortel for their support, both intellectual and financial, during the project.

References

1. Bremermann H.J. (1962). *Optimization through evolution and recombination.* in [19]. pp. 93-106.
2. Cooper J.L. & Hinde C.J. (2003). *Improving the performance of Genetic Algorithms Using Partial Fitness Functions.* Submitted to GECCO 2003.

3. Cooper J.L. Withall M.S. Hinde C.J. & Stone R.G. (2003). *Investigation into the effects of varying the parameters of packets travelling across the Internet.* Loughborough University Department of Computer Science Internal Report no. 1070.
4. Corne D. Smith G. & Oates M. (2000). Telecommunications Optimization: Heuristic and Adaptive Techniques. John Wiley and Sons Ltd.
5. Darwin C. (1996). The Origin of Species. Oxford University Press. First published 1859.
6. Di Caro G. & Dorigo M. (1998). *AntNet: Distributed Stigmergetic Control for Communications Networks.* In Journal of Artificial Intelligence Research, 9:317-365.
7. Fraser A.S. (1957a). Simulation of Genetic Systems by Automatic Digital Computers 1, Introduction. Australian J. of Biol.Sci., Vol. 10, pp. 484-491.
8. Fraser A.S. (1957b). Simulation of Genetic Systems by Automatic Digital Computers 2, Effects of Linkage on Rate of Advance under Selection. Australian J. of Biol.Sci., Vol. 10, pp. 492-499.
9. Goldberg D.E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Addison-Wesley Publishing Co. Inc.
10. Holland J.H. (1973). *Genetic Algorithms and the Optimal Allocation of Trials.* In SIAM Journal on Computing, 2(2):88-105, June.
11. Holland J.H. (1975). Adaption in Natural and Artificial Systems. MIT Press.
12. Holland J.H. (1992). Adaption in Natural and Artificial Systems. MIT Press, Second Edition.
13. Langdon W.B.,(2002). Proceedings of the Genetic and Evolutionary Computation Conference 2002,Morgan Kaufmann.
14. Liang S. Zincir-Heywood A.N. & Heywood M.I. (2002). *Intelligent Packets for Dynamic Network Routing using Distributed Genetic Algorithm.* In [13]. pp. 88-96.
15. Mendel G. (1865). Experiments in Plant Hybridization.
16. Syswerda G. (1989). *Uniform Crossover in Genetic Algorithms.* In Schaffer D., editor, Proceedings of the Third International Conference on Genetic Algorithms, 2-9. Morgan Kaufmann.
17. Wall L. Christiansen T. & Schwartz R.L. (1996). Programming Perl. O'Reilly & Associates, Inc., Second Edition.
18. Wallace A.R. (1858). *On the Tendency of Varieties to Depart Indefinitely From the Original Type.* In Journal of the Proceedings of the Linnean Society: Zoology 3(9):53-62.
19. Yovits M.C., Jacobi G.T. & Goldstein G.D. (1962). *Self Organizing Systems.* Spartan Books, Washuington D.C.