Improving genetic algorithms' efficiency using intelligent fitness functions

Improving the Efficiency of Genetic Algorithms Using Intelligent Fitness Functions

Jason Cooper and Chris Hinde

Department of Computer Science
Loughborough University
Loughborough
LE11 3TU
UK
j.l.cooper@lboro.ac.uk
c.j.hinde@lboro.ac.uk

Abstract. Genetic Algorithms are an effective way to solve optimisation problems. If the fitness test takes a long time to perform then the Genetic Algorithm may take a long time to execute. Using conventional fitness functions Approximately a third of the time may be spent testing individuals that have already been tested. Intelligent Fitness Functions can be applied to improve the efficiency of the Genetic Algorithm by reducing repeated tests. Three types of Intelligent Fitness Functions are introduced and compared against a standard fitness function The Intelligent Fitness Functions are shown to be more efficient.

Keywords : Genetic Algorithms

1 Introduction

Genetic Algorithms (GA) [5] are based on Darwin's theory of evolution [2]. They were invented in the 1950s, some of the early papers being by Fraser [3, 4] and Bremermann [1]. Later on work by Holland[6–9] popularised genetic algorithms and Holland's work is often cited as the origins of Genetic Algorithms. A GA consists of a population of individuals, each individual represents a possible set of parameters for the algorithm. Each individual is tested and assigned a level of fitness depending on how well they solve the problem. The fitness levels are then used to decide which individuals should be used to produce the next generation. The better the fitness of an individual the more likely they are to produce the next population.

The next generation is produced using two genetic operators after selecting the parents. Parents are selected based on their fitness level. The higher their fitness the more likely they are to be chosen as parents. The first operator is crossover where two individuals swap part of their genes. The second is mutation where an part of an individuals genes are changed. The combination of these operators with parent selection and a fitness function enables the GA to evolve better solutions.

2 Need for efficiency

In a GA the majority of the time maybe spent fitness testing the individuals. If a fitness test for an individual takes 1 minute to run then a fitness test of 6000 individuals will take approximately 4 days and 4 hours.

One method which may increase the number of generations that are produced is to reduce the population size. Slow fitness functions associated with small population sizes can lead to problems with premature convergence and lack in diversity of the population.

A test of 6000 individuals does not mean that 6000 locations in the search space have been examined. Individuals with the same genomes keep appearing. The graph shown in figure 2 shows how many fitness test have been performed against how many unique tests have been performed (how much of the search space has been searched). The graph shown in figure 2 is the results of running a GA on the test problem and the parameters described in section 4.

The graph in figure 2 shows that on hard problems it is easy for a GA to spend a third of its time fitness testing individuals it has already tested before. This is not an obvious result considering that in the example the size of the search space is 2^{40} so the chances of randomly duplicating an individual are very small. The reason that there are so many duplicates produced in the GA is because they are not randomly created each time. The next generation is based on the previous generation which was based on the one before. It is this fact that directs the search of the GA but it is also this that causes individuals to appear multiple times.

The graph in figure 2 shows that if the fitness test takes 1 minute to test an individual then the GA would have wasted approximately one and a half days. In the case of a fitness test that was evolving transmission strategies for sending data over a network a simple fitness test could take a week or more. The GA would have wasted over 38 years of time. To make the GA more efficient it needs to reduce the number of fitness tests it is repeating.

If a fitness test takes a long time to run it may be possible to use an approximation for the fitness test which will take less time to test an individual but not give accurate results. A survey of approximations was produced by Yaochu Jin [10]. Rasheed, Ni and Vattam [11] have used approximations to speed up the run time of GA's. Unfortunately there are problems that have no known approximations that might be used; real world network transmissions is just one of them.

3 Intelligent Fitness Functions Concept

The standard fitness function in a GA takes an individual as the parameters for an algorithm and evaluates the result. The better the result the better the fitness. Intelligent fitness functions have memory; they can calculate an individuals fitness based on the information stored from previous evaluations. There are two types of memory that an intelligent fitness function can use :

– Short Term Memory
– Long Term Memory

The short term memory is cleared at the start of each new generation and so can only store information about the current generation. The long term memory never gets cleared, but is not able to store every piece of information about previous generations. The fitness function needs to decide what to keep and what to overwrite as it has a limited memory.

An example of the use of short term memory is to check whether an individual is a duplicate of one encountered earlier in the current generation. If it is it returns a low fitness level to stop premature convergence and to increase the diversity of the population.

An example of long term memory to store copies of the best few individuals so far with their fitness. Then when it is asked to fitness test an individual it checks to see if it has tested it before and if it has it simply returns the fitness level it had last time it tested it. This will reduce the number of tests that are performed multiple times by the GA.

4 Test Problem

The problem that is used to test the intelligent fitness functions is one that has been designed as hard for a GA to solve. An individuals is mapped into and array, i, which consists of "dim" eight bit integers. The array is then used to calculate the distance from a point C, coordinates c_k, using equation 1. The result of equation 1 is then used in equation 2 to calculate a fitness value for the individual.

$$dist = \sqrt{\sum_{k=1}^{dim} (i_k - c_k)^2} \tag{1}$$

$$Fitness = \cos\left(2 * \pi * \frac{dist}{rad} * amp * \left(2^{-\frac{dist}{ahl}}\right) + height * \left(2^{-\frac{dist}{hhl}}\right)\right) \tag{2}$$

The default values used for the tests were the following
$dim = Number\ of\ Dimensions = 5$
$rad = radius = 50$
$height = 10.0$
$amp = amplitude = 5$
$hhl = height\ half\ life = 200$
$ahl = amplitude\ half\ life = 200$

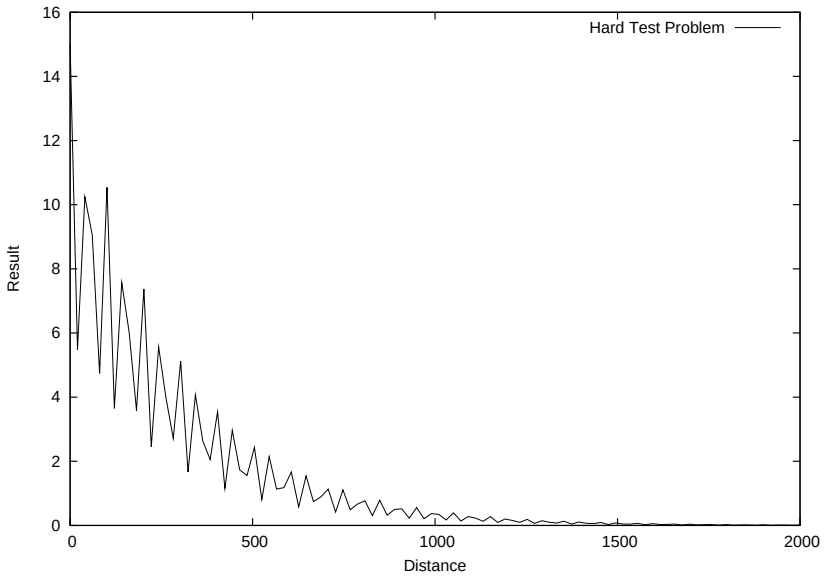

Fig. 1. A graph of the hard test problem with one dimension

5 The Fitness Functions

5.1 Standard Fitness Function

The standard fitness function is a normal GA's fitness function that calculates the fitness of each individual in the current generation.

5.2 Intelligent Fitness Function

Three intelligent fitness functions were tested. One had a short term memory which was used to discover if the current individual being tested has already been tested in this generation. If it has then the individual was given a fitness value of 0.

One had a long term memory which was used to discover if it had a record of testing the individual in a previous generation. If the individual has been tested before then the fitness value in memory for that individual is used. If there is no record found then it tests the individual and if it does well enough it stores it in the long term memory for later reference.

The final intelligent fitness function had both a long term memory and a short term memory. The short term memory had precedence over the long term memory.

6 Result

All the standard fitness function and the intelligent fitness function reached the same level of fitness at the same number of generations.

On graphs in figures 2 to 11 the closer the two lines the more efficient the GA. The graph shown in figure 2 shows the efficiency of the standard fitness function. The graph shown in figure 3 shows the efficiency of the Intelligent fitness function with short term memory. The graph shown in figures 4, 6, 8 and 10 shows the efficiency of the intelligent fitness function with long term memory of sizes 100, 500, 1000 and 2000 individuals. The graph shown in figures 5, 7, 9 and 11 shows the efficiency of the intelligent fitness function with short term memory and long term memory of sizes 100, 500, 1000 and 2000 individuals.

The graph in figure 2 shows that approximately one third of tests carried out by a standard fitness function fitness have already been carried out previously by it.

The graph in figure 3 shows that an intelligent fitness function with short term memory can evolve more generations in the same number of fitness tests as a standard fitness function.

Figure 4 shows that a small long term memory enables the GA to evolve for a more generations than a standard fitness function over the same number of fitness tests performed.

The graph in figure 5 shows that an intelligent fitness function with a short term memory and a small long term memory is almost the same as one with just a short term memory.

The graphs in figures 6, 8 and 10 shows that as the size of the long term memory is increased the efficiency of the GA is also increased.

The graphs in figures 7, 9 and 11 shows that as the size of the long term memory is increased in an intelligent fitness function, with long term memory and short term memory, then the efficiency of the GA increases better than in an intelligent fitness function with just long term memory.

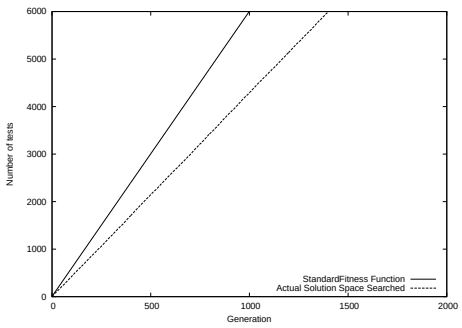

Fig. 2. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40

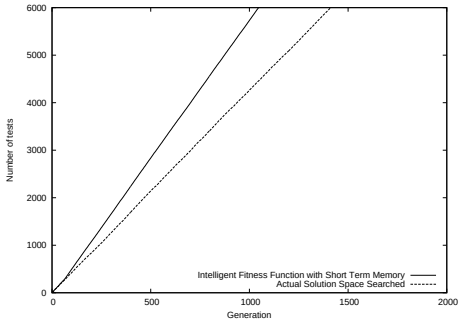

Fig. 3. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a short term memory

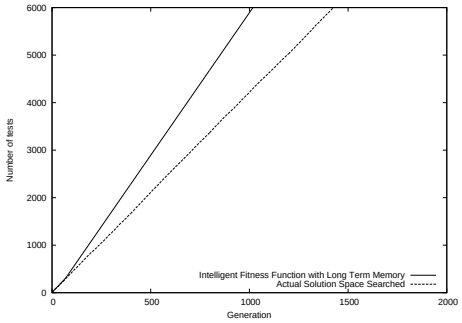

Fig. 4. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a long term memory of 100 individuals

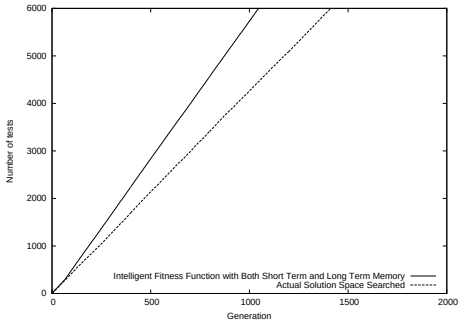

Fig. 5. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a short term memory and a long term memory of 100 individuals

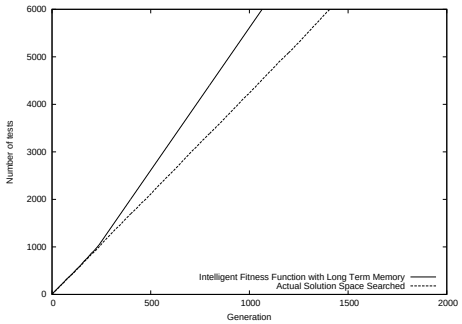

Fig. 6. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a long term memory of 500 individuals

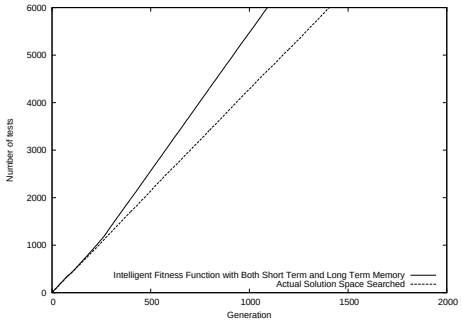

Fig. 7. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a short term memory and a long term memory of 500 individuals

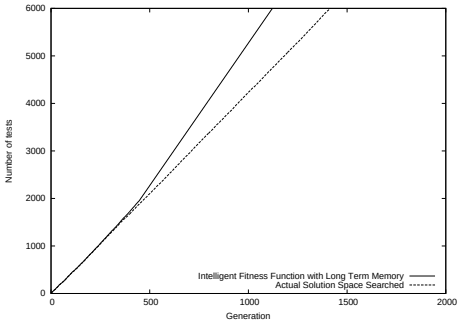

Fig. 8. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a long term memory of 1000 individuals

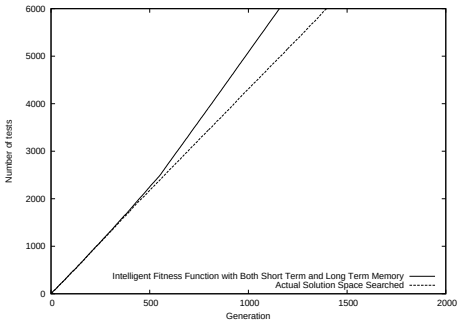

Fig. 9. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a short term memory and a long term memory of 1000 individuals

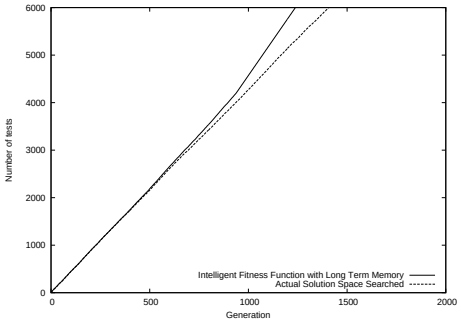

Fig. 10. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a long term memory of 2000 individuals

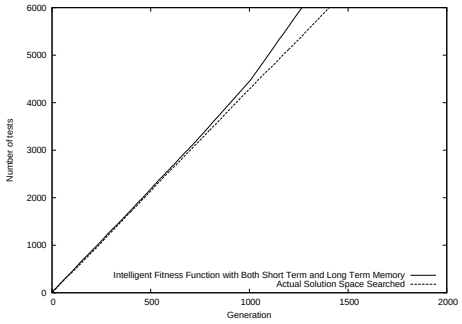

Fig. 11. Fitness tests performed and solution space searched when run over 10000 generations, with a population size of 6, a genome length of 40 with a short term memory and a long term memory of 2000 individuals

7 Conclusion

When using a GA which has a fitness function which takes a long length of time to run the GA's efficiency can be improved by the use of intelligent fitness functions. The use of a short term memory on its own in an intelligent fitness function makes a small difference to a GA's efficiency. The effect on a GA by an intelligent fitness function with long term memory increases as the size of the long term memory increases. When an intelligent fitness function has both long term memory and short term memory then the efficiency of the GA is increased more than using either of the memories on their own.

Acknowledgements

The authors of this paper would like to acknowledge the support of Nortel Networks, both financially and intellectually.

References

1. Bremermann H.J. (1962). Optimization through evolution and recombination. in [12]. pp. 93-106.
2. Darwin, C. The Origin Of Species. Oxford University Press, Walton Street, Oxford. OX2 6DP, UK. Based on: On The Origin Of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life. Second Edition, London 1859. First Published 24 Nov 1859.
3. Fraser A.S. (1957a). Simulation of Genetic Systems by Automatic Digital Computers 1, Introduction. Australian J. of Biol.Sci., Vol. 10, pp. 484-491.
4. 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.

5. Goldberg D.E., 1989, Genetic Algorithms in Search, Optimization, and Machine Learning. ISBN 0-201-15767-5.
6. Holland, J. H. Adaption in Natural and Artificial Systems. A Bradford Book, The MIT Press. ISBN 0-262-08213-6.
7. Holland J.H. (1973). Genetic Algorithms and the Optimal Allocation of Trials. In SIAM Journal on Computing, 2(2):88-105, June.
8. Holland J.H. (1975). Adaption in Natural and Artificial Systems. MIT Press.
9. Holland J.H. (1992). Adaption in Natural and Artificial Systems. MIT Press, Second Edition.
10. Yaochu Jin. Fitness Approximation in Evolutionary Computation - A Survey. Approximation and Learning In Evolutionary Computation Workshop, GECCO 2002. Pg 3 – 4.
11. Khaled Rasheed, Xiao Ni, Swaroop Vattam. Comparison of Methods for Using Reduced Models to Speed Up Design Optimization. Approximation and Learning In Evolutionary Computation Workshop, GECCO 2002. Pg 17 – 20.
12. Yovits M.C., Jacobi G.T. & Goldstein G.D. (1962). Self Organizing Systems. Spartan Books, Washington D.C.