Genetic Algorithms

Helen Harman,
Aberystwyth University,
heh14@aber.ac.uk,
Student Number: 110007212

October 29, 2015

1 Introduction

Biology has been the inspiration for many computational methods. This includes work on neural networks, swarm robotics and genetic algorithms (GAs). In this report we will look into the current research on GAs.

1.1 GA Overview

GAs can be defined in the following ways :

- "a programming technique that mimics biological evolution as a problem-solving strategy" [4]
- "an optimization searching algorithm based on the theory of evolution and the genetic mutation theory" [3]

Taking inspiration from how biology has evolved over many generations, the inputs to a function, called a fitness function, are evolved until a optimal solution is achieved. This can be applied to may computational problems including data mining, game playing and image processing. [4] An optimal solution may be one which takes the fewest CPU cycles and/or results in the smallest error.

1.2 History

The initial concept of GAs was introduced by biologists in the late 1950s to early 1960s [4]. Their aim was to mimic biological systems to learn more about how they have evolved over time. These ideas were later adopted by computer scientists, who applied then to different computational problems.

GAs took inspiration from Darwin's theory of evolution. If biological systems can evolve to solve complex problems, then programs should be able to evolve to solve problems.

Work on evolutionary programming in 1966 was performed using finite state machines [5]. Later on ideas where taken from evolutionary programming and hill climbing algorithms to develop GAs.

In recent years work has been done on combining GAs with other AI concepts. For example work has been done on training neural networks using GAs. Using this technique the weights in a neural network can be evolved to find the optimal weightings. [6]

In order to understand the current research on applying and improving GAs, we first must introduce the key concepts of GAs.

1.3 Key Concepts

The inputs to GAs are often encoded into binary, though strings and numerical values can also be used. The use of binary allows the simple evolution of the population through the flipping of bits. Strings can be manipulated through the changing of individual letters; and numerical values can be changed using mathematical operations. [4]

These inputs are passed into a fitness function which gives a quantitative measure of how optimal the input is. This is then used to determine how the population should be evolved and if the best solutions has been achieved.

In order for the population to evolve genetic operations are applied. There are three main types of genetic operators. These are selection, mutation and crossover.

Mutation causes an individual to evolve. This tries to prevent individuals getting stuck in local search areas. Mutation often involves comparing a randomly generated number to a probability that a change will occur. This causes small random changes between generations. [2]

Crossover combines multiple individuals together. In the case of binary representation the different bits are combined to create a individual for the next generation. There are multiple methods that can be used to combined the individuals. For example uniform crossover, which has an equal probability of the bit from each parent being chosen.

A selection of the previous generation is retained within the new generation. This tends to be waited towards the proportion of the population which performed the best. There are a wide range of different selection algorithms. This includes elitist selection, tournament selection and steady-state selection.[4] Which one to use often depends on the situation.

Different mutation, crossover and selection operations have been used in the three papers that will be discussed in section 2.

2 Research Undertaken

Although GAs alone provide a powerful concept for computing solutions, a lot of recent research combines GAs with alternative AI approaches. In this section we look at several papers which have expanded on the standard GA. The firsts paper, "Intelligent bionic genetic algorithm (IB-GA) and its convergence" [3], tries to solve some of the issues encountered when using GAs. "Comparison of nearest point algorithms by genetic algorithms" [2] applies GAs to the testing of algorithms. Finally we will discuss "Hybrid Genetic Algorithm for Cloud Computing Applications" [9] which tries to solve the scheduling of resources in cloud computing through the use of a Multi-agent GA.

2.1 Intelligent bionic genetic algorithm (IB-GA) and its convergence [3]

The IB-GA tries to improve upon the commonly used GA by addressing commonly occurring problems with convergence. It begins by looking through the history of GAs in order to show where their work fits. They go on to suggest an alternative mutation, selection and crossover operations.

The aim of the mutation operation is to change the population while keeping individuals from a large area of the search space to avoid converging on a local optima. The IB-GA, described by the paper, uses a compound mutation strategy. It does this by attempting to retain a population which has a large distribution across the solution space. A random number is used to determine if a mutation should take place. To calculate how much the individual should be mutated by the random number is passed to a mutation criteria function. The mutation is then applied to the individual.

Each individual consist of a list of real numbers. When using arithmetic crossover a random amount of each value each parent contains is combined(ie. $X' = r(X)+(1-r)Y$, $Y' = r(Y) + (1 - r)X$ [3]). All values are changed which leads to a lack of local search being performed. The paper suggests controlling the amount of crossover that happens by splitting the values into segments, exchanging alternative value, performing arithmetic crossover and then joining the segments back together. The amount of crossover performed is dependent on the number of segments.

The paper uses a variation on elitist selection. Using the standard elitist selection, the current elite individuals may be based around a local optima. If only these elite are kept then the GA may converge on the local optima. The IB-GA aims to keep a diverse population. As well as taking into consideration the results from the fitness function, the distance between the individuals is taken into consideration during selection.

To prove that the IB-GA will always converge on the optimal value Markov chain theory is used. The state in the Markov chain is the current population. The final state will contain the optimal solution. [8]

The results show a comparison of the IB-GA with two other GAs. Both of these use common genetic operators. One encodes the input as binary, whereas the other uses real numbers. The comparison has been performed using two different fitness functions. The convergence value and time is discussed. All given results show that IB-GA is a more effective search method.

2.1.1 Critique

The paper gives a good introduction to GAs and allows the reader to understand the background to the problem they are trying to address. It describes the history of GAs and the current issues being experienced. This allows the paper to clearly get across the importance of solving issues with convergence.

In many AI algorithms deciding when to converge is often problematic. Do we stop after a certain number of iterations or when a small error is achieved? It is important to research into methods of achieving quick global optima convergence.

Due to real numbers allowing for more accuracy than binary numbers, real numbers have been used as inputs. However, this causes a theoretically infinite search space. The Markov chain has been used to prove than an optimal solution will be reached. In practise I am not convinced that this will always happen. Unless every state has been check we can not be sure the global optimal has been reached. The use of real numbers also makes the genetic operations more complex.

Rather than just flipping a bit, a value needs to be manipulated in some way. In commonly used mutation this can be done using a equation. A random number can be generated to determine the amount of mutation. Changing the value too much will cause local search spaces to be missed; and changing it too little will not allow a large distribution of the search space to be covered. As discussed in section 2.1 the paper uses a "compound mutation strategy". The mutation criteria function is built from several equations. Therefore this adds even more complexity.

There are multiple parameters being used, although this add flexibility, they require fine tuning for each problem. The mutation uses a probability value; the crossover requires the number of segments the individuals should be split into; and the selection requires the percentage of the population that should be kept. All of these parameters are likely to have a large effect on the evolution that takes places. GAs can be used to solve the fine tuning of parameters. Therefore, requiring a lot of parameters can cause the problem that GAs are designed to solve.

The drawbacks of using this approach were not discussed in the paper. Throughout the paper an positive view of the IB-GA was shown. Showing situations where the IB-GA fails would be useful to allow future work to build on this. I would not be surprised to see a paper disproving some of the statements they have made.

It can be difficult to show the reliability of results for non-

deterministic algorithms. In the paper the experiments are run 10 times and the average is calculated. This has allowed fairer results to be shown. A good detailed overview of the experiment results has been given, which proves evidence that, in the given situation, IB-GA performs better.

The IB-GA should be applied to alternative fitness functions should be evaluated. The two fitness functions being used are just mathematical equations. The IB-GA could have been applied to situations where other GAs are being used. This would give a comparison to real situations where the IB-GA could be applied. It is not clear if the author has selected fitness functions with a bias towards the IB-GA's performance.

It would be interesting to apply the IB-GA to training neural networks. Often with techniques, like backpropagation, the convergence is slow and sometimes does not converge. Using the IB-GA should allow the training to converge with optimal weights.

The paper is clearly structured with each genetic operation in separate sections. Though parts of the paper are difficult to read due to the technical nature, it is well written and a lot of information has been included.

Overall the paper has met the its aims and present a GA that improves upon the convergence of commonly used GAs. The paper is well written and gives a good grounding for future work in applying the IB-GA to different situations.

2.2 Comparison of nearest point algorithms by genetic algorithms [2]

Testing takes a long time to manually perform so it is important to find ways of automating the process. Comparison of nearest point algorithms by genetic algorithms[2] describes a method of testing using GA. The paper uses GAs to evaluate and compare the efficiency of three nearest point algorithms.

The paper starts being introducing testing and the concept of white, grey and black box testing. The fitness functions used require some knowledge of the inner workings of the algorithms; therefore, grey box testing is used. The introduction describes the work others have done on applying evolutionary learning techniques to testing the performance of algorithms.

The testing is performed on three different nearest point algorithms. This includes k-nearest neighbour which uses a k-d tree; and two strip base methods. In strip based methods each strip of points is inserted into a hash table, called a skip list. The two strip based algorithms use different methods of setting the number of strips that should be used.

A fitness function has been assigned to each of the nearest point algorithms. These fitness functions count the number of times certain operations are performed. Example operations include: the number of times the algorithm jumps between nodes; the number of nodes that have been inserted; and the modifications and moves within the skip lists.

Often a Gaussian distribution of points is used as input to test nearest neighbour algorithms. This paper uses floating point numbers within a boundary box, which are distributed over a 2D Gaussian curve. Rather than inputting these manually, which would take a considerable amount of time, a "density function" is used. This means that the nearest neighbour arguments can be calculated using five values. In order for the genetic operators to work these have been encoded into a single 28 bit binary number. In the experiments, these five inputs are chosen at random using a Bernoulli distribution. [7]

Truncation selection is used to decided which individuals should be used to create the next generation. The population is ordered based on the results of the fitness function. A percentage of the fittest individuals are truncated from the list to create the next generation. [1]

Uniform crossover has been chosen as the method of combining individuals. This is a commonly used method and has been described in section 1.3 of this report. Mutation is performed using probability. Each bit has a small probability of being flipped.

Each of the fitness functions have been evaluated 200 times using a population of 500 individuals which are generated randomly. The mean, standard deviation and skewness of the results of evaluated for each of the fitness functions. The mean results for the three fitness functions is also compared.

The following four methods of comparing the execution time have been used :

- The BCET and WCET of the algorithms.
- A pairwise comparison of the BCET and WCET. The difference between two algorithms BCET and WCET times are found and summed.
- The case where one algorithm out performs the other with the largest margin is found. This is performed in both directions, ie where algorithm 1 out performs algorithm 2, and where algorithm 2 out performs algorithm 1.
- Finally the difference between these margins is calculated and compared, ie. the difference between the maximum amount algorithm 1 out performed algorithm 2, and the maximum amount algorithm 2 out performed algorithm 1 is calculated.

Through these comparisons they have shown that one of the strip-based methods is more optimal, than the others, in the majority of cases.

2.2.1 Critique

The testing of algorithms is an important part of the software development process. Unfortunately it is often forgotten about or poorly completed due to developers finding it slow and laborious. If a task in going to be performed multiple times and can be automated, it should be

automated. Therefore, I think that this paper is trying to address an important issues.

As computers are becoming more and more powerful, developers often become less concerned with the computational complexity of their algorithms. Nevertheless, it may be required to processing large amounts of data, running on low powered devices or run along side other programs. Therefore, it is important to find algorithms with a low computational time.

As input a 2D Gaussian distribution is created using five values. These values include the standard deviations, expectation values and a correlation coefficient. This allows for different distributions of points to be tested, as well as different numbers of points. Testing how the algorithms perform under different situations is important and shows that the input is not biased towards one of the algorithms.

Rather than improving on GAs, like in [3], this paper shows an application of a GA. This GA uses commonly used genetic operations. The paper does not discus why they chose the GA they did. For example why they chose Truncation selection over Tournament selection. It would be interesting to see how different GAs perform when applied to the same situation.

The GA approach is compared to random search. In the conclusion they state that "random search was found as efficient in some cases"[2]. It does not discus the improvements or failure modes of the GA. More specifically we don't know if the GA is getting stuck within a local search area or converging on a local optimal. They mention that more testing needs to be performed but do not say what this testing should be.

Comparison against other testing methods could be performed. During the introduction multiple methods have been introduced, including past methods which use GAs. These methods are not mentioned again. A very detailed comparison to random search is performed, but additional comparisons to other algorithms would be beneficial.

The work done in the paper is a good starting point for future work. It is well written and they have met their aims stated in the introduction. Perhaps applying the work done in [3] would give more optimal results.

2.3 Hybrid Genetic Algorithm for Cloud Computing Applications [9]

One of the main complications in cloud computing is how to balance the work load across the available resources. Hybrid Genetic Algorithm for Cloud Computing Applications[9] combines concepts from multi agent algorithms with GAs to address these load balancing issues.

In the past GAs have been used in cloud computing, but have issues with "imperfect convergence, slow convergence, and no convergence" [9]. GAs can often converge on a local optimal rather than a global optimal. As a stopping criteria is often used, this criteria may never be reach. This paper uses a Multi-Agent GA (MAGA) to try and solve these issues.

The individuals (agents) in the MAGA are connected in a grid formation. The crossover operation is performed using neighbouring agents. The selection is also performed by comparing an agents results with its neighbours results. A mutation operation and self-learning is also performed by the agents.

The fitness function calculates the average load across the physical hosts. Each physical host contains one or more Virtual Machines (VMs); and each VM contains one or more user group. Each group contains several parameters which allow the memory load and the CPU load to be calculated. The CPU and memory load are summed, with weights, to get the load for a group. The average load for a VM can then be calculated, followed by the average load per host.

The inputs into the fitness function have been encoded into a list of binary values. A list item is used for each user group. This list position is used to as a reference to a particular user group. The binary number indicates which VM the group is using.

The MAGA has been compared to a GA using a optimization function. The MAGA is also compared to a Min_min function. This comparison shows the the CPU usage, memory usage and the single-point of failure rate. This report will discus this in more detail in the critique section (section 2.3.1).

2.3.1 Critique

In the introduction related work has been discussed in order aim to show the aim of the paper. However, they state that "scholars have proposed a variety of improved genetic algorithms" but they do not give further details of this. It would be of use to see what improvements other have made, and there drawbacks. This would show how they have expanded on the research into using GAs and the importance of their work.

The paper is poorly written. It is filled with grammar mistakes, spelling mistakes and repetition of points that have already been made. For example they have used "CUP" rather than CPU. This makes the paper harder to read and understand; and is likely to put other researchers off building on their work.

The description of what a user group is is stated twice. The first time it is described they have said "parameters from a single user include". One of the parameters, "ReqPerHrPerUser", is the average number of users online per hour. This clear shows that the list of parameters is not for a single user. The same parameters are also stated when talking about a user group.

These parameters have been poorly named. For some of the parameters 'Req' clearly means request; but "ReqPerHrPerUser" is not the number of requests, it is the number of users. When talking about the parameters they use the index of the parameter (eg. "The forth param-

eter"). These indexes do not always match the given order.

The paper does not give enough information on the genetic operations that are being performed. It mentions them by name but does not give further details. It also does not mention what effect the self-learning has on the agents. Additional research revels what the operations are likely to be doing. However, the paper should describe them in relation to the work they are performing.

They use a "Agent Grid" as a layout for the individuals. How they form this grid is not disused. Perhaps this would be clearer with more background knowledge on multi-agent systems; however, more information should be included. As neighbourhood competition and "neighbourhood orthogonal crossover" is being performed this layout is likely to have a high impact on the evolution of the population.

The fitness function has been described using a large amount of detail. It has been broken down into CPU load and memory load for each group; allowing the mathematical equations to be easily understood. A step by step explanation of the algorithm is also provided. When authors breakdown down their work in this way I find it becomes easier to grasp.

No consideration has been given to none deterministic behaviour of the MAGA. Experiments should be run multiple times in order to receive reliable results. The algorithm should also be tested on different numbers of VMs and user groups. This would show how well the MAGA performs under different situations.

The graphs are poorly presented. The graphs all appear to have been cropped, causing them to become impossible to interpret. Most the titles and units are not visible and half the plotted lines are not visible. The description of them is brief giving the read little information on how well the MAGA has actually performed.

The MAGA has been compared to a GA. However, no information on the GA being used has been given. The results show that MAGA is considerably better for the each of the runs performed. Though I can not prove it, especially with the lack of information given, I believe their to be a bias towards the MAGA. In most cases I do not think such a large difference would be seen. Also, this comparison is shown randomly in the middle of the paper rather than in the "Simulation Experiment Result and Analysis" section.

I selected this paper due to their interesting sounding abstract and conclusion. The idea of combining Multi-agents and GAs seems like a useful concept. After reading the paper, I am disappointed in the lack of information on the MAGA and the quality in presenting the results.

3 Summary

The first two papers reviewed were well written and gave a good detailed explanation of the GA being used. The third paper was not written to as higher standard. The lack of quality of the third paper, has probably made me more sceptical about the research performed, than I was when critiquing the first two papers.

The first two papers were from the journal Expert Systems with Applications, whereas the third was from IEEE Asia-Pacific Services Computing Conference. This means the two journal papers had a similar style and quality. Whereas the conference paper was written in a very different style and was more focused on what the GA has been applied to, rather than the actual implementation of the GA. I believe that all three papers have added important concepts to their research areas, and future work should be performed to expand on their ideas.

References

[1] Khalid Jebari and Mohammed Madiafi.
Selection methods for genetic algorithms.
International Journal of Emerging Sciences, 3(4), 2013.

[2] Janne Koljonen.
Comparison of nearest point algorithms by genetic algorithms.
Expert Systems with Applications, 38(8):10303 – 10311, 2011.

[3] Fachao Li, Li Da Xu, Chenxia Jin, and Hong Wang.
Intelligent bionic genetic algorithm (ib-ga) and its convergence.
Expert Systems with Applications, 38(7):8804 – 8811, 2011.

[4] Adam Marczyk.
Genetic algorithms and evolutionary computation.
Avaliable: `http://www.talkorigins.org/faqs/genalg/genalg.html`, 2004.
Accessed: [06 October 2015].

[5] Thomas M. Mitchell.
Machine Learning.
McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[6] David J Montana and Lawrence Davis.

Training feedforward neural networks using genetic algorithms.
In *IJCAI*, volume 89, pages 762–767, 1989.

[7] Eric Weisstein.
Bernoulli distribution.
MathWorld A Wolfram Web Resource. Avaliable: `http://mathworld.wolfram.com/BernoulliDistribution.html`, 2015.
Accessed: [06 October 2015].

[8] Hongwei Zhang.
Review of markov chain theory.
Avaliable: `http://www.cs.wayne.edu/~hzhang/courses/7290/Lectures/14\%20-\%20Review\%20of\%20Markov\%20Chain\%20Theory.pdf`, 2015.
Accessed: [19 October 2015].

[9] Kai Zhu, Huaguang Song, Lijing Liu, Jinzhu Gao, and Guojian Cheng.
Hybrid genetic algorithm for cloud computing applications.
In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 182–187, Dec 2011.