Genetic Algorithms

SEM6120 - Assignment 1

Alexander D Brown (adb9)

Contents

1 Introduction

Genetic algorithms are a biologically-inspired approach to heuristic search which mimic natural selection. Unlike many other evolutionary strategies and evolutionary programming, they are not designed to solve a specific problem, but are designed to solve the problem of optimisation which is made difficult by substantial complexity and uncertainty[2].

The complexity of the task should make it such that discovering an optimum solution is a long, maybe even impossible, task. At the same time the uncertainty needs to be reduced so that the knowledge of *available* options can be increased.

The initial design for a genetic algorithm was a method for moving from one population of chromosomes to another using a form of natural selection. This algorithm also included methods for crossover, mutation and inversion. This idea of having a large population was the distinguishing feature from any past attempts which had only considered the parent and one offspring, where the offspring was simply a mutation of the parent[3].

1.1 Evolutionary Algorithms

As their name suggests, an evolutionary algorithm applies elements from the biological theory of evolution to the problem of optimisation. These elements include:

- Reproduction
- Mutation
- Recombination
- Selection

Typically, a population of candidate solutions are generated to which a fitness function can be applied. The population is then subject to some form of evolution, and this process is repeated until a halting criteria is met.

Genetic algorithms are a type of evolutionary algorithm with a focus on the genetic evolution of solutions. Candidate solutions for genetic algorithms, known as *chromosomes* are encoded as a series of *genes*. These genes are a representation of the choices which need to be optimised for the solution and can be as simple as a single bit or as complex as a real number, depending on the problem.

There are many other forms of both evolutionary and genetic algorithms which this report with mention in later sections.

1.2 Genetic Algorithms

The basic principals of genetic algorithms are to represent candidate solutions as a population of chromosomes, from this population the fittest members can be picked out and used in the next generation and to create new member of the population through reproduction and/or mutation.

This cycle repeats with the aim to produce better performing individuals in each generation until the optimum solution is either reached or gotten close enough to that any future improvement is unnecessary or unwanted due to other constraints such as processing time. The latter of these allows a genetic algorithm to come up with a “good” solution in a reasonable amount of time.

Reproduction and mutation are an important part of genetic programming, and too of evolutionary programming. Without these parts the algorithm would quickly reach a local optimum for the initial population and would not improve past this.

1.3 Chromosome Representation

One of the key parts parts in implementing a genetic algorithm is the representation of chromosomes. This is very dependent of the problem the genetic algorithm needs to optimise and can have a knock on affect on the efficiency and accuracy of the algorithm.

Sometimes a simple solution is enough to represent the problem, binary strings are a commonly suggested approach. However sometimes more complex representations are required, potentially any data structure can be used as a chromosome but lists and trees are the common choices as they are easy to perform crossover[1] and mutation on.

As a very simple example, to maximise y in: $y = f(x)$, one could represent the value of x as a binary string, an example of which is shown in figure 1.

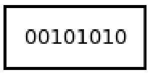

Figure 1: Chromosome representation as a binary string

1.4 Fitness Function

To fulfil the step of natural selection; the process of choosing the “best” members of a population, there needs to be a way of evaluating each chromosome, such that they can be compared to one another.

The function for doing so is known as a fitness function, which typically returns either a single number or a list of numbers, depending on the problem. Each chromosome can then be ranked in order of fitness and the top members of a population can then be selected.

As the value returned from a fitness function, with be specific to the domain it has be used within, it is necessary to rescale the fitness value to ensure uniformity when genetic algorithms are applied over several different domains at the same time.

1.5 Selection

Selection simulates the “survival of the fittest” nature of biology. However, it is beneficial not to keep some lesser performing members of the population to avoid getting trapped in local optimum. Most selection algorithms will introduce an element of randomness into the selection to deal with this.

[1]A term used instead of reproduction in genetic algorithms.

Simple methods for selection typically involve randomly selecting individuals, giving those individuals with higher fitness greater chance of being selected. While others take inspiration from biology by making members of the population complete or assign individuals genders to imitate sexual reproduction.

1.6 Crossover

Crossover is the algorithmic approach to replicating biological reproduction and is key to genetic algorithms as it will produce new, unseen, offspring based on selected members of the population.

In its simplest form crossover is the process of taking one chromosome and splicing it with another. This can be done in several ways, but the most common is to choose a random point, known as the *crossover point*. The offspring consists of all the genes from the first parent up until the crossover point, the remaining genes then come from the second parent. As shown in figure 2.

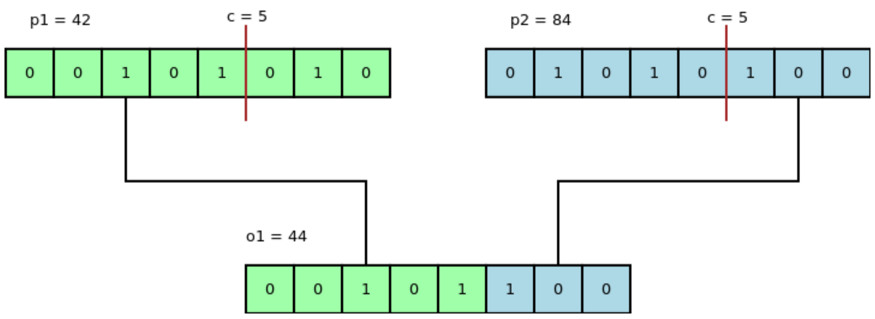

Figure 2: Crossover of parents $p1$ and $p2$ at crossover point c, where $c = 5$, resulting in offspring $o1$.

More complex methods revolve around the idea of having multiple crossover points and eventually it becomes more efficient to use a binary mask over the two parents, as shown in figure 3

This can lead to problems if the representation of chromosomes is complex; if, for example, each gene represents a node in the travelling salesman problem then crossover can produce routes which are invalid as nodes may be duplicated due to this process.

1.7 Mutation

Another method of getting variation into a population is mutation, the random change of genes within a chromosome. In genetic algorithms this is typically replicated by flipping a single bit in the chromosome.

As with crossover, mutation must be handled carefully with complex chromosome representations as it may lead to invalid chromosomes. The rate of mutation is usually kept low to avoid large amounts of variance.

1.8 Termination

The final problem with genetic algorithms is the problem of deciding when a result is good enough. Usually the implementation has some form of termination

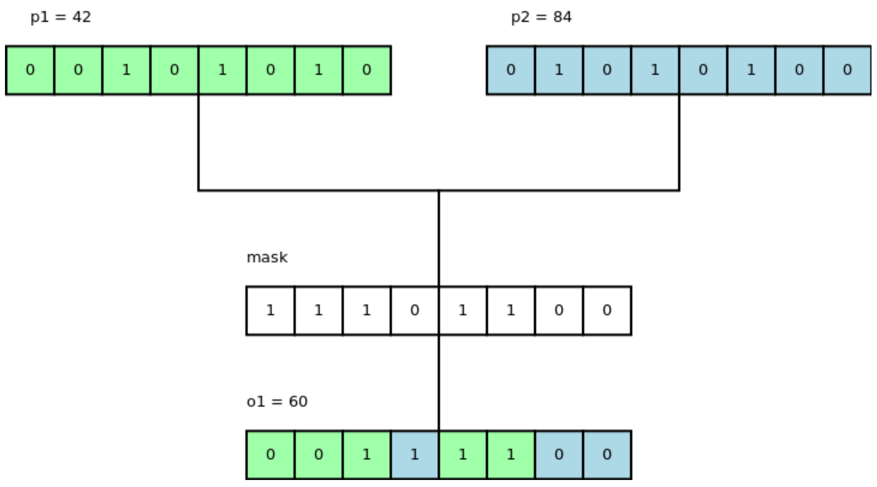

Figure 3: Crossover of parents $p1$ and $p2$ using a binary mask, resulting in offspring $o1$

criteria, this may be a time limit or number of generations or may be something more complex like when the algorithm no longer improves in fitness greatly.

Termination criteria is very specific to the problem domain. Genetic algorithms are very good at giving an approximate answer and typically bad at finding the exact optimum solution.

2 Previous Research into Selection Schemes

2.1 Comparing Selection Schemes

There have been studies into the effects of different selection algorithms on genetic algorithms. Goldberg and Deb[1] compared four methods of selection; proportional reproduction, ranking selection, tournament selection and the GENITOR algorithm.

They analysed each selection scheme in terms of:

1. growth ratio; the expected ratio for the members of the best class to the umber of members of the population.
2. Takeover time; the approximate number of generations it takes to converge, this gives an idea of how long it would take before mutation, rather than crossover, becomes the primary method for exploring the search space.
3. Time complexity; the standard big O measure for time complexity.

Linear ranking and binary tournament (a tournament with a pool size of two individuals) were found to have similar growth rates

Takeover times all converged in around $O(log\ n)$ generations. Most interestingly the time complexities had a large range, from $O(n^2)$ to $O(n)$, with tournament selection being an $O(n)$. This is particularly interesting as tournament selection is a very easy algorithm to make parallel, this fits in with genetic algorithms nicely as they improve drastically from parallel implementation.

2.2 Genetic Drift

Rogers and Prügel-Bennett[4] defined a method of analysing selection schemes using genetic drift, a term borrowed from biology, which describes the change in frequency of an allele (gene variation) through random sampling of the population.

Genetic drift is a phenomenon observed in genetic algorithms due to the nature of selection and, unlike analysis methods like convergence time, leads to a exact analytical solution. Though previous attempts to calculate genetic drift were often approximations or difficult to generalise to other cases, according to the authors.

In genetic algorithms, genetic drift is a measure of the change in fitness variation within a population and is worked out using probabilistic analysis of the selection scheme.

To analyse genetic drift, the variation of population fitness needs to be calculated. Standard variance (σ^2) is applied to the initial population of P members. Each member (α) has fitness F_α. Therefore the variance is defined as:

$$\sigma_F^2 = \mathrm{E}[F^2] - (\mathrm{E}[F])^2 \tag{1}$$

Calculating this out becomes:

$$\sigma_F^2 = \frac{1}{P}\sum_{\alpha=1}^{P} F_\alpha^2 - \left(\frac{1}{P}\sum_{\alpha=1}^{P} F_\alpha\right)^2 \tag{2}$$

Applying a selection scheme to this population and drawing a new population of P individuals means there will be n_α copies of a population member F_α. The variance of the new population is given as:

$$\sigma_F'^2 = \frac{1}{P}\sum_{\alpha=1}^{P} n_\alpha F_\alpha^2 - \left(\frac{1}{P}\sum_{\alpha=1}^{P} n_\alpha F_\alpha\right)^2 \tag{3}$$

To get an average case the authors averaged over all the possible ways of performing selection. To do this they looked at neutral selection, where the probability of selection is due to random, neutral, occurrences and not due to any form of fitness of the individual. This allows n_α to be independent of F_α. By working through this and simplifying further to use the fact that P is kept constant, they derive:

$$\mathrm{E}[\sigma_F'^2] = \frac{P - \mathrm{E}[n^2]}{P-1}\sigma_F^2 \tag{4}$$

The change in population fitness variance for any selection scheme can be found by calculating $\mathrm{E}[n^2]$, the expected square of the number of times any population member is selected.

By applying the variance in the number of times any individual is simply:

$$\sigma_n^2 = \mathrm{E}[n^2] - (\mathrm{E}[n])^2 \tag{5}$$

Substituting this into equation 4, the following equation is found, and is used as the basis for their results:

$$\mathrm{E}[\sigma_F'^2] = \left(1 - \frac{\sigma_n^2}{P-1}\right)\sigma_F^2 \tag{6}$$

3 Evaluation of Research

The results from Goldberg and Deb[1] should be taken with some scepticism as they are formulated from the pure mathematics of the selection techniques and not from running experiments and the authors even mention that it is only designed as a simple method to better understand the *expected* behaviour and suggest looking at better methods, citing the inherently noisy nature of genetic algorithms.

The work of Rogers and Prügel-Bennett[4] also runs into this inherent problem of genetic algorithms being a form of subsymbolic learning, which is very difficult to apply probabilistic methods to, due to the randomness used within them.

The authors attempt to deal this by using results from experiments averaged over 100,000 runs. Again though, a lot will depend on the nature of the problem, the representation of chromosomes and even the way in which the selection scheme is implemented. Like with the work of Goldberg and Deb, this only provides an indication of the performance of a selection scheme, rather than a full view. It does, however, seem to give some useful insights into different selection schemes and may help rule out less useful selection schemes.

References

[1] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93, 1991.

[2] John H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence.* MIT Press, April 1992.

[3] Melanie Mitchell. An introduction to genetic algorithms, 1996.

[4] A. Rogers and A. Prugel-Bennett. Genetic drift in genetic algorithm selection schemes. *Evolutionary Computation, IEEE Transactions on*, 3(4):298–303, November 1999.