The Microbial Genetic Algorithm

Inman Harvey

Evolutionary and Adaptive Systems Group, Centre for Computational Neuroscience and Robotics, Department of Informatics, University of Sussex, Brighton BN1 9QH, UK
inmanh@sussex.ac.uk

Abstract. We analyse how the conventional Genetic Algorithm can be stripped down and reduced to its basics. We present a minimal, modified version that can be interpreted in terms of horizontal gene transfer, as in bacterial conjugation. Whilst its functionality is effectively similar to the conventional version, it is much easier to program, and recommended for both teaching purposes and practical applications. Despite the simplicity of the core code, it effects Selection, (variable rates of) Recombination, Mutation, Elitism ('for free') and Geographical Distribution.

Keywords: Genetic Algorithm, Evolutionary Computation, Bacterial Conjugation.

1 Introduction

Darwinian evolution assumes a population of replicating individuals, with three properties: Heredity, Variation, and Selection. If we take it that the population size neither shrinks to zero, nor shoots off to infinity -- and in artificial evolution such as a genetic algorithm we typically keep the population size constant -- then individuals will come and go, whilst the makeup of the population changes over time.

Heredity means that new individuals are similar to the old individuals, typically through the offspring inheriting properties from their parents. Variation means that new individuals will not be completely identical to those that they replace. Selection implies some element of direction or discrimination in the choice of which new individuals replace which old ones. In the absence of selection, the population will still change through random genetic drift, but Darwinian evolution is typically directed: in the natural world through the natural differential survival and mating capacities of varied individuals in the environment, in artificial evolution through the fitness function that reflects the desired goal of the program designer.
Any system that meets the three basic requirements of Heredity, Variation and Selection will result in evolution. Our aim here is to work out the most minimal framework in which that can be achieved, for two reasons: firstly theoretical, to see what insights can be gained when the evolutionary method is stripped down to its bare bones; secondly didactic, to demystify the Genetic Algorithm (GA) and present a version that is trivially easy to implement.

2 GAs Stripped to the Minimum

The most creative and challenging parts of programming a GA are usually the problem-specific aspects. What is the problem space, how can one best design the genotypic expression of potential solutions or phenotypes, and the genotype-phenotype mapping, how should one design an appropriate fitness function to achieve ones goals?

Though this may be the most interesting part, it is outside the remit of our focus here on those evolutionary mechanisms that can be generic and largely independent of the specific nature of any problem. So we will assume that the programmer has found some suitable form of genotype expression, such that the genetic material of a population can be maintained and updated as the GA progresses; for the purposes of illustration we shall assume that there is a population size P of binary genotypes length N. We assume that all the hard problem-specific work of translating any specific genotype in the population into the corresponding phenotype, and evaluating its fitness, has been done for us already.

Given these premises, we wish to examine the remaining GA framework to see how it can be minimized.

2.1 Generational and Steady State GAs

Traditionally GAs were first presented in generational form. Given the current population of size P, and their evaluated fitnesses, a complete new generation of the same size was created via some selective and reproductive process, to replace the former generation in one fell swoop. A GA run starts with an initialized population, and then loops through this process for many successive generations. This would roughly correspond to some natural species that has just one breeding season, say once a year, and after breeding the parents die out without a second chance.

There are many natural species that do not have such constraints, with birth and death events happening asynchronously across the population, rather than to the beat of some rhythm. Hence the Steady State GA, which in its simplest form has as its basic event the replacement of just one individual from P by a single new one (a variant version that we shall not consider further would replace two old by two new). The repetition of this event P times is broadly equivalent to a single generation of the Generational GA, with some subtle differences.

In the Generational version, no individual survives to the next generation, although some will pass on genetic material. In the Steady State version, any one individual may, through luck or through being favoured by the selective process, survive unchanged for many generation-equivalents. Others, of course may disappear much earlier; the average lifetime of each individual will be P birth/death events, as in the generational case, but there will be more variance in these lifetimes.

There are pragmatic considerations for the programmer. The core piece of code in the Steady State version, will be smaller, although looped through P times more, than in the Generational GA. The Generational version needs to be coded so that, in effect, P birth/death events occur in parallel, although on a sequential machine this will have to be emulated by storing a succession of consequences in a temporary array; typically two arrays will be needed for this-generation and next-generation. If the programmer does have access to a cluster of processors working in parallel, then a Steady State version can farm out the evaluation of each individual (usually the computational bottleneck) to a different processor. The communication between processors can be very minimal, and the asynchronous nature of the Steady State version means that it will not even be necessary to worry too much about keeping different processors in step.

These are suggestive reasons, but perhaps rather weak, subjective and indeed even aesthetic, for favouring a Steady State version of a GA. A stronger reason for doing so in a minimalist GA is that it can exploit a very simple implementation of selection.

2.2 Selection

Traditionally but regrettably, newcomers to GAs are usually introduced to 'fitness-proportionate selection', initially in a Generational GA. After the fitnesses of the whole current population are evaluated by whatever criterion has been chosen, and specified as real values, these values are summed to give the size of the reproductive 'cake'. Then each individual is allocated a probability of being chosen as a parent according to the size of its own slice of that cake.

This method has three virtues: it will indeed selectively favour those with higher fitnesses; it has some tenuous connection with the different biological notion of fitness (although in the biological sense any numbers associated with fitness are based on observing, after the event, just how successful an individual was at leaving offspring; as contrasted with the GA approach that reverses cause and effect here); and lastly, it happens to facilitate, for the theoreticians, some mathematical theorem-proving. But pragmatically, this method often leads to unnecessary complications.

What if the chosen evaluation method allocates some fitnesses that are negative? This would not make sense under the biological definition, but where the experimenter has chosen the evaluation criteria it may well happen. This is one version of the general re-scaling issue: different versions of a fitness function may well agree in ordering of different members of the population, yet have significantly different consequences. One worry that is often mentioned is that, under fitness-proportionate selection, one often sees (especially in a randomized initial population) many individuals scoring zero, whilst others get near-zero; though this may reflect nothing more than luck or noise, the latter individuals will dominate the next generation to the complete exclusion of the former.

These issues motivated many complex schemes for re-scaling fitnesses *before* then implementing fitness-proportionate selection. But -- although it was not initially reflected in the standard texts -- many practitioners of applying GAs to real-world problems soon abandoned them in favour of rank-based methods.

2.3 Rank-based and Tournament Selection

The most general method of rescaling is to use the scores given by the fitness function to order all the members of the population from fittest to least fit; and thereafter to ignore the original fitness scores and base the probabilities of having offspring solely on these relative rankings. A common choice made is to allocate (at least in principle) 2.0 reproductive units to the fittest, 1.0 units to the median, and 0.0 units to the least fit member, similarly scaling pro rata for intermediate rankings; this is linear rank selection. The probability of being a parent is now proportional to these rank-derived numbers, rather than to the original fitness scores.

There are in practice some complications. If as is common practice P is an even number, the median lies between two individuals. The explicit programming of this technique requires some sorting of the population according to rank, which adds further complexity. Fortunately there is a convenient trick that generates a similar outcome in a much simpler fashion.

If two members of the population are chosen at random, their fitnesses compared, and the Winner selected, then the probability of the Winner being any specific member of the population exactly matches the reproductive units allocated under linear rank selection. This is easiest to visualize if one considers P such tournaments, in succession using the same population each time. Two individuals are chosen at random each time, so that each individual can expect to participate in two such tournaments. The top-ranker will win both its tournaments, for 2.0 reproductive units; a median-ranker can expect to win one, lose one, for 1.0 units; and the bottom-ranker will lose both its tournaments, for 0.0 units. A minor difference between the two methods will be more variance in the tournament case, around the same mean values.

Tournament selection can be extended to tournaments of different sizes, for instance choosing the fittest member from a randomly selected threesome. This is an example of *non-linear* rank selection. From both pragmatic and aesthetic perspectives, Tournament Selection with the basic minimum viable tournament size of two has considerable attractions for the design of a minimalist GA.

2.4 Who to Breed, Who to Die?

Selection can be implemented in two very different ways; either is fine, as long as the end result is to bias the choice of those who contribute to future generations in favour of the fitter ones. The usual method in GAs is to focus the selection on who is to become a parent, whilst making an unbiased, unselective choice of who is to die. In the standard Generational GA, every member of the preceding generation is eliminated without any favouritism, so as to make way for the fresh generation reproduced from selected parents. In a Steady State GA, once a single new individual has been bred from selected parents, some other individual has to be removed so as to maintain a constant population size; this individual is often chosen at random, again unbiased.

Some people, however, will implement a method of biasing the choice of who is removed towards the less fit. It should be appreciated that this is a *second* form of selective pressure, that will compound with the selective pressure for fit parents and potentially make the combined selective pressure stronger than is wise. In fact, one can generate the same degree of selective pressure by biasing the culling choice towards the less fit (whilst selecting parents at random) as one gets by the conventional method of biasing the parental choice towards the more fit (whilst culling at random).

This leads to an unconventional, but effective, method of implementing Tournament Selection. For each birth/death cycle, generate one new offspring with random parentage; with a standard sexual GA, this means picking both parents at random, but it can similarly work with an asexual GA through picking a single parent at random. A single individual must be culled to be replaced by the new individual; by picking two at random, and culling the *Loser*, or least fit of the two, we have the requisite selection pressure. One can argue that this may be closer to many forms of natural selection in the wild than the former method.

Going further, we can consider a yet more unconventional method, that combines the random undirected parent-picking with the directed selection of who is to be culled. Pick two individuals at random to be parents, and generate a new offspring from them; then use the same two individuals for the tournament to select who is culled -- in other words the weaker parent is replaced by the offspring.

It turns out that this is easy to implement, is effective. This is the underlying intuition behind the Microbial GA, so called because we can interpret what is happening here in a different way -- Evolution without Death!

2.5 Microbial Sex

Microbes such as bacteria do not undergo sexual reproduction, but reproduce by binary fission. But they have a further method for exchanging genetic material, *bacterial conjugation*. Chunks of DNA, plasmids, can be transferred from one bacterium to the next when they are in direct contact with each other. In the conventional picture of a family tree, with offspring listed below parents on the page, we talk of vertical transmission of genes ‘down the page’; but what is going on here is horizontal gene transfer ‘across the page’. As long as there is some selection going on, such that the ‘fitter’ bacteria are more likely to be passing on (copies of) such plasmids than they are to be receiving them, then this is a viable way for evolution to proceed.

We can reinterpret the Tournament described above, so as to somewhat resemble bacterial conjugation. If the two individuals picked at random to be parents are called A and B, whilst the offspring is called C, then we have described what happens as C replacing the weaker one of the parents, say B; B disappears and is replaced by C. If C is the product of sexual recombination between A and B, however, then ~50% of C’s genetic material (give or take the odd mutation) is from A, ~50% from B. So what has happened is indistinguishable from B remaining in the population, but with ~50% of its original genetic material replaced by material copied and passed over from A. We can consider this as a rather excessive case of horizontal gene transfer from A (the fitter) to B (the weaker).

3 The Microbal GA

We now have the basis for a radical, minimalist revision of the normal form of a GA, although functionally, in terms of Heredity, Variation and Selection, it is performing just the same job as the standard version.

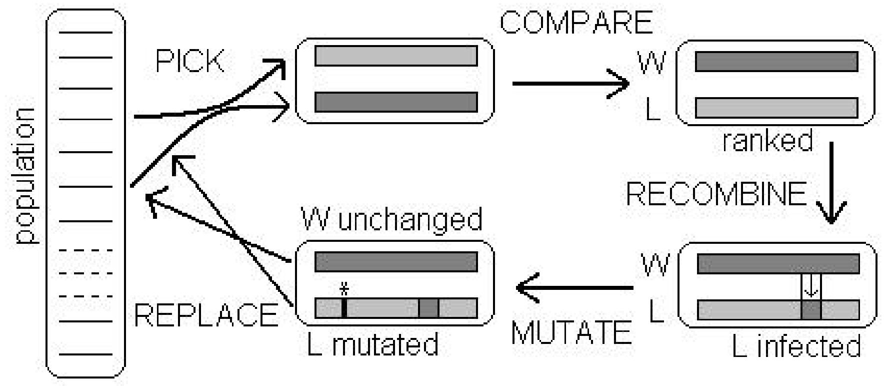

Fig. 1. The genotypes of the population are represented as a pool of strings. One single cycle of the Microbial GA is represented by the operations PICK (two at random), COMPARE (their fitnesses to determine Winner = W, Loser = L), RECOMBINE (where some proportion of Winner's genetic material 'infects' the Loser), and MUTATE (the revised version of Loser).

This is illustrated in Fig. 1. Here the recombination is described in terms of 'infecting' the Loser with genetic material from the Winner, and we can note that this rate of infection can vary. In bacterial conjugation it will typically be rather a low percentage that is replaced or supplemented; to reproduce the typical effects of sexual reproduction, as indicated in the previous section, this rate should be ~50%. But in principle we may want, for different effects, to choose any value between 0% and 100%. In practice, for normal usage and for comparability with a standard GA, the 50% rate is recommended. The simplest way of doing this, equivalent to Uniform Recombination, is with 50% probability independently at each locus to copy from Winner to Loser, otherwise leaving the Loser unchanged at that locus.

From a programming perspective, this cycle is very easy to implement efficiently. For each such tournament cycle, the Winner genotype can remain unchanged within the genotype-array, and the Loser genotype can be modified (by 'infection' and mutation) *in situ*. We can note that this cycle gives a version of 'elitism' for free: since the current fittest member of the population will win any tournament that it participates in, it will thus remain unchanged in the population -- until eventually overtaken by some new individual even fitter. Further, it allows us to implement an effective version of geographical clustering for a trivial amount of extra code.

3.1 Trivial Geography

It is sometimes considered undesirable to have a panmictic population, since after many rounds of generations (or, in a Steady State GA, generation-equivalents) the population becomes rather genetically uniform. It is thus fairly common for evolutionary computation schemes to introduce some version of geographical distribution. If the population is considered to be distributed over some virtual space, typically two-dimensional, and any operations of parental choice, reproduction, placing of offspring with the associated culling are all done in a local context, then this allows more diversity to be maintained across sub-populations. Spector and Klein [3] note that a one-dimensional geography, where the population is considered to be on a (virtual) ring, can be as effective as the 2-D version, and demonstrate the effectiveness in some example domains. If we consider our array that contains the genotypes to be wrap-around, then we can implement this version by, for each tournament cycle: choose the first member A of the tournament at random from the whole population; then select the next member B at random from a deme, or sub-population that starts immediately after A in the array-order. The deme size D, <= P, is a parameter that decides just how local each tournament is.

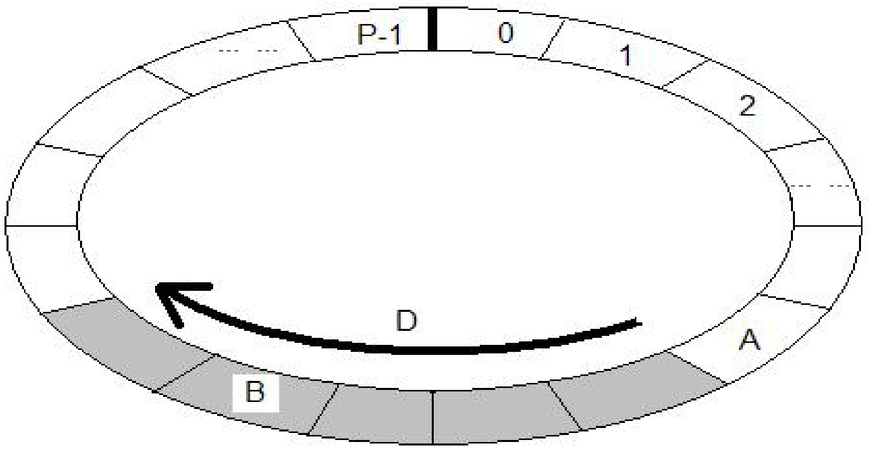

Fig. 2. The genotypes of the population are represented as geographically distributed on a ring, numbered from 0 to P-1. For a tournament, A is picked at random from the whole population; then B is picked at random from the deme (here of size D=5) immediately following A.

3.2 Program Code

The core part of the code for a Microbial GA is here given in pseudo-code based on C. We shall assume that there is a population size `P` of binary genotypes length `N`, stored in an array of the form `gene[P][N]`, that has been suitably initialized at random for the start of the GA; we have a function `eval(i)`, that returns some real value for the ith member of the population. This code represents a single tournament, that is repeated as many times as is considered necessary. We assume a pseudo-random number function `rnd()` that returns a real number in the range [0.0,1.0). `REC` is the recombination or 'infection' rate (suggested value 0.5), and `MUT` is the (per locus) mutation rate. `D` is the deme size.

This minimal GA routine incorporates Rank Selection, (variable rates of) Recombination and Mutation, (variable size) Demes, and Elitism, in just a few lines:-

```
void microbial_tournament(void) {
  int A,B,W,L,i;
  A=P*rnd();                        // Choose A randomly
  B=(A+1+D*rnd())%P;                // B from Deme, %P..
  if (eval(A)>eval(B)) {W=A; L=B;} // ..for wrap-around
  else {W=B; L=A;}                  // W=Winner L=Loser
  for (i=0;i<N;i++) {               // walk down N genes
    if (rnd()<REC)                  // RECombn rate
      gene[L][i]=gene[W][i];        // Copy from Winner
    if (rnd()<MUT)                  // MUTation rate
      gene[L][i]^1;                 // Flip a bit
  }
}
```

4 Discussion

The original version (without demes) of this minimal Microbial GA has been widely used and taught at Sussex for over a decade [1], but only previously discussed in published form briefly in [2], in the context of Evolutionary Robotics. Another application in that field has been the Embodied Evolution of [4]. It is here presented in full for the first time with added Trivial Geography [3] providing demes.

We do not claim that it is more effective than the standard GA; indeed its underlying functionality is basically the same. It lends itself well to distributed, asynchronous applications. The minimalism makes it easy to teach and to implement. The re-interpretation in terms of bacterial conjugation opens up new perspectives and possibilities, including that of varying the rates of recombination or 'infection'.

References

1. Harvey, I.: The Microbial Genetic Algorithm. Unpublished report (1996)
2. Harvey, I.: Artificial Life, a Continuing SAGA. In Gomi, T. (ed.) ER 2001. LNCS 2217, pp. 94-109. Springer, Heidelberg (2001)
3. Spector, L., Klein, J.: Trivial Geography in Genetic Programming. In: Yu, T., Riolo, R.L., Worzel, B. (eds.) Genetic Programming Theory and Practice III, pp. 109-124. Boston, MA: Kluwer Academic Publishers (2005)
4. Watson, R.A., Ficci, S.G., Pollack, J.B.: Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems, 39(1), pp. 1-18 (2002)