Evolutionary Computation and Genetic Algorithms

William H. Hsu
Kansas State University, USA

INTRODUCTION

A **genetic algorithm** (**GA**) is a method used to find approximate solutions to difficult search, optimization, and machine learning problems (Goldberg, 1989) by applying principles of evolutionary biology to computer science. Genetic algorithms use biologically-derived techniques such as inheritance, mutation, natural selection, and recombination. They are a particular class of evolutionary algorithms.

Genetic algorithms are typically implemented as a computer simulation in which a population of abstract representations (called *chromosomes*) of candidate solutions (called *individuals*) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but different encodings are also possible. The evolution starts from a population of completely random individuals and happens in generations. In each generation, multiple individuals are stochastically selected from the current population, modified (mutated or recombined) to form a new population, which becomes current in the next iteration of the algorithm.

BACKGROUND

Operation of a GA

The problem to be solved is represented by a list of parameters, referred to as *chromosomes* or genomes by analogy with genome biology. These parameters are used to drive an evaluation procedure. Chromosomes are typically represented as simple strings of data and instructions, in a manner not unlike instructions for a von Neumann machine. A wide variety of other data structures for storing chromosomes have also been tested, with varying degrees of success in different problem domains.

Initially several such parameter lists or chromosomes are generated. This may be totally random, or the programmer may seed the gene pool with "hints" to form an initial pool of possible solutions. This is called the *first generation pool*.

During each successive generation, each organism is evaluated, and a measure of quality or *fitness* is returned by a fitness function. The pool is sorted, with those having better fitness (representing better solutions to the problem) ranked at the top. "Better" in this context is relative, as initial solutions are all likely to be rather poor.

The next step is to generate a second generation pool of organisms, which is done using any or all of the genetic operators: selection, crossover (or recombination), and mutation. A pair of organisms is selected for breeding. Selection is biased towards elements of the initial generation which have better fitness, though it is usually not so biased that poorer elements have no chance to participate, in order to prevent the solution set from converging too early to a sub-optimal or local solution. There are several well-defined organism selection methods; roulette wheel selection and tournament selection are popular methods.

Following selection, the *crossover* (or *recombination*) operation is performed upon the selected chromosomes. Most genetic algorithms will have a single tweakable *probability of crossover* (P_c), typically between 0.6 and 1.0, which encodes the probability that two selected organisms will actually breed. A random number between 0 and 1 is generated, and if it falls under the crossover threshold, the organisms are mated; otherwise, they are propagated into the next generation unchanged. Crossover results in two new child chromosomes, which are added to the second generation pool. The chromosomes of the parents are mixed in some way during crossover, typically by simply swapping a portion of the underlying data structure (although other, more complex merging mechanisms have proved useful for certain types of problems.) This process is repeated with different parent organisms until there are an appropriate number of candidate solutions in the second generation pool.

The next step is to mutate the newly created offspring. Typical genetic algorithms have a fixed, very small *probability of mutation (P_m)* of perhaps 0.01 or less. A random number between 0 and 1 is generated; if it falls within the P_m range, the new child organism's chromosome is randomly mutated in some way, typically by simply randomly altering bits in the chromosome data structure.

These processes ultimately result in a second generation pool of chromosomes that is different from the initial generation. Generally the average degree of fitness will have increased by this procedure for the second generation pool, since only the best organisms from the first generation are selected for breeding. The entire process is repeated for this second generation: each organism in the second generation pool is then evaluated, the fitness value for each organism is obtained, pairs are selected for breeding, a third generation pool is generated, etc. The process is repeated until an organism is produced which gives a solution that is "good enough".

A slight variant of this method of pool generation is to allow some of the better organisms from the first generation to carry over to the second, unaltered. This form of genetic algorithm is known as an *elite selection strategy*.

MAIN THRUST OF THE CHAPTER

Observations

There are several general observations about the generation of solutions via a genetic algorithm:

- GAs may have a tendency to converge towards local solutions rather than global solutions to the problem to be solved.
- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hypermutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*).
- As time goes on, each generation will tend to have multiple copies of successful parameter lists, which require evaluation, and this can slow down processing.
- Selection is clearly an important genetic operator, but opinion is divided over the importance of crossover verses mutation. Some argue that crossover is the most important, while mutation is only necessary to ensure that potential solutions are not lost. Others argue that crossover in a largely uniform population only serves to propagate innovations originally found by mutation, and in a non-uniform population crossover is nearly always equivalent to a very large mutation (which is likely to be catastrophic).
- Though GAs can be used for global optimization in known intractable domains, GAs are not always good at finding optimal solutions. Their strength tends to be in rapidly locating *good* solutions, even for difficult search spaces.

Variants

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The basic algorithm performs crossover and mutation at the bit level.

Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

Efficiency

Genetic algorithms are known to produce good results for some problems. Their major disadvantage is that they are relatively slow, being very computationally intensive compared to other methods, such as random optimization.

Recent speed improvements have focused on speciation, where crossover can only occur if individuals are closely-enough related.