What is a genetic algorithm?

Concisely stated, a genetic algorithm (or GA for short) is a programming technique that mimics biological evolution as a problem-solving strategy. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a *fitness function* that allows each candidate to be quantitatively evaluated. These candidates may be solutions already known to work, with the aim of the GA being to improve them, but more often they are generated at random.

The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, of course, most will not work at all, and these will be deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem.

These promising candidates are kept and allowed to reproduce. Multiple copies are made of them, but the copies are not perfect; random changes are introduced during the copying process. These digital offspring then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. Those candidate solutions which were worsened, or made no better, by the changes to their code are again deleted; but again, purely by chance, the random variations introduced into the population may have improved some individuals, making them into better, more complete or more efficient solutions to the problem at hand. Again these winning individuals are selected and copied over into the next generation with random changes, and the process repeats. The expectation is that the average fitness of the population will increase each round, and so by repeating this process for hundreds or thousands of rounds, very good solutions to the problem can be discovered.

As astonishing and counterintuitive as it may seem to some, genetic algorithms have proven to be an enormously powerful and successful problem-solving strategy, dramatically demonstrating the power of evolutionary principles. Genetic algorithms have been used in a wide variety of fields to evolve solutions to problems as difficult as or more difficult than those faced by human designers. Moreover, the solutions they come up with are often more efficient, more elegant, or more complex than anything comparable a human engineer would produce. In some cases, genetic algorithms have come up with solutions that baffle the programmers who wrote the algorithms in the first place!

Methods of representation

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form that a computer can process. One common approach is to encode solutions as binary strings: sequences of 1's and 0's, where the digit at each position represents the value of some aspect of the solution. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only and often "is intuitively closer to the problem space" (Fleming and Purshouse 2002, p. 1228).

This technique was used, for example, in the work of Steffen Schulze-Kremer, who wrote a genetic algorithm to predict the three-dimensional structure of a protein based on the sequence of amino acids that go into it (Mitchell 1996, p. 62). Schulze-Kremer's GA used real-valued numbers to represent the so-called "torsion angles" between the peptide bonds that connect amino acids. (A protein is made up of a sequence of basic building blocks called amino acids, which are joined together like the links in a chain. Once all the amino acids are linked, the protein folds up into a complex three-dimensional shape based on which amino acids attract each other and which ones repel each other. The shape of a protein determines its function.) Genetic algorithms for training neural networks often use this method of encoding also.

A third approach is to represent individuals in a GA as strings of letters, where each letter again stands for a specific aspect of the solution. One example of this technique is Hiroaki Kitano's "grammatical encoding" approach, where a GA was put to the task of evolving a simple set of rules called a context-free grammar that was in turn used to generate neural networks for a variety of problems (Mitchell 1996, p. 74).

The virtue of all three of these methods is that they make it easy to define operators that cause the random changes in the selected candidates: flip a 0 to a 1 or vice versa, add or subtract from the value of a number by a randomly chosen amount, or change one letter to another. (See the section on Methods of change for more detail about the genetic operators.) Another strategy, developed principally by John Koza of Stanford University and called *genetic programming*, represents programs as branching data structures called trees (Koza et al. 2003, p. 35). In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one subtree with another.

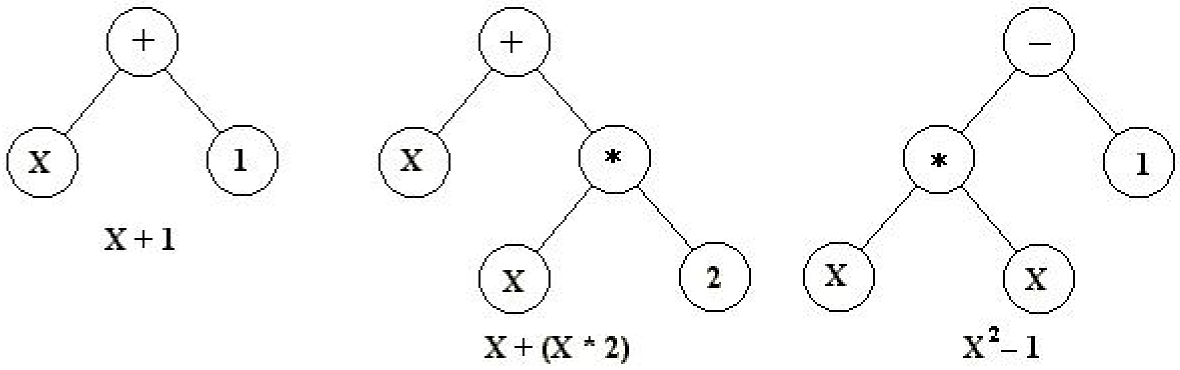

Figure 1: Three simple program trees of the kind normally used in genetic programming. The mathematical expression that each one represents is given underneath.

It is important to note that evolutionary algorithms do not need to represent candidate solutions as data strings of fixed length. Some do represent them in this way, but others do not; for example, Kitano's grammatical encoding discussed above can be efficiently scaled to create large and complex neural networks, and Koza's genetic programming trees can grow arbitrarily large as necessary to solve whatever problem they are applied to.

Methods of selection

There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation, but listed below are some of the most common methods. Some of these methods are mutually exclusive, but others can be and often are used in combination.

Elitist selection: The most fit members of each generation are guaranteed to be selected. (Most GAs do not use pure elitism, but instead use a modified form where the single best, or a few of the best, individuals from each generation are copied into the next generation just in case nothing better turns up.)

Fitness-proportionate selection: More fit individuals are more likely, but not certain, to be selected.

Roulette-wheel selection: A form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness.

(Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual "owns" the section on which it lands each time is chosen.)

Scaling selection: As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.

Tournament selection: Subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce.

Rank selection: Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking rather than absolute differences in fitness. The advantage of this method is that it can prevent very fit individuals from gaining dominance early at the expense of less fit ones, which would reduce the population's genetic diversity and might hinder attempts to find an acceptable solution.

Generational selection: The offspring of the individuals selected from each generation become the entire next generation. No individuals are retained between generations.

Steady-state selection: The offspring of the individuals selected from each generation go back into the pre-existing gene pool, replacing some of the less fit members of the previous generation. Some individuals are retained between generations.

Hierarchical selection: Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more computationally expensive fitness evaluation.

Methods of change

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation. There are two basic strategies to accomplish this. The first and simplest is called *mutation*. Just as mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code.

The second method is called *crossover*, and entails choosing two individuals to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. Common forms of crossover include *single-point crossover*, in which a point of exchange is set at a random location in the two individuals' genomes, and one individual contributes all its code from before that point and the other contributes all its code from after that point to produce an offspring, and *uniform crossover*, in which the value at any given location in the offspring's genome is either the value of one parent's genome at that location or the value of the other parent's genome at that location, chosen with 50/50 probability.

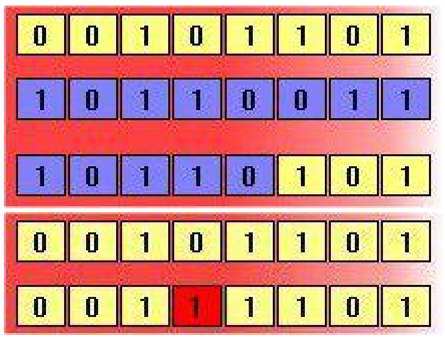

Figure 2: Crossover and mutation. The above diagrams illustrate the effect of each of these genetic operators on individuals in a population of 8-bit strings. The upper diagram shows two individuals undergoing single-point crossover; the point of exchange is set between the fifth and sixth positions in the genome, producing a new individual that is a hybrid of its progenitors. The second diagram shows an individual undergoing mutation at position 4, changing the 0 at that position in its genome to a 1.