Table of Contents

1. Synopsis 2
2. Genetic Algorithms 3
   2.1 Initialization Operator
   2.2 Selection Operator
   2.3 Crossover Operator
   2.4 Mutation Operator
   2.5 Replacement Operator
3. Genetic Programming and Genetic Algorithms 5
4. Results 8

Genetic Programming

1. Synopsis

Genetic programming (GP) is an automated methodology inspired by biological evolution to find computer programs that best perform a user-defined task. It is therefore a particular machine learning technique that uses an evolutionary algorithm to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task. The first experiments with GP were reported by Stephen F. Smith (1980) and Nichael L. Cramer (1985), as described in the famous book Genetic Programming: On the Programming of Computers by Means of Natural Selection by John Koza (1992).

Computer programs in GP can be written in a variety of programming languages. In the early (and traditional) implementations of GP, program instructions and data values were organized in tree-structures, thus favoring the use of declarative languages that naturally embody such a structure (an important example pioneered by Koza is Lisp). Nonetheless, other forms of GP have been suggested and successfully implemented, such as the simpler linear representation, which suits the more traditional imperative languages [see, for example, Banzhaf et al. (1997)]. The commercial GP software Discipulus, for example, uses linear genetic programming combined with machine code language to achieve better performance.

GP is very computationally intensive and so in the 1990s it was mainly used to solve relatively simple problems. However, more recently, thanks to various improvements in GP technology and to the well-known exponential growth in CPU power, GP has started delivering a number of outstanding results. At the time of writing, nearly 40 human-competitive results have been gathered, in areas such as quantum computing, electronic design, game playing, sorting, searching and many more. These results include the replication or infringement of several post-year-2000 inventions, and the production of two patentable new inventions.

Developing a theory for GP has been very difficult and so in the 1990s genetic programming was considered a sort of pariah amongst the various techniques of search. However, after a series of breakthroughs in the early 2000s, the theory of GP has had a formidable and rapid development. So much so that it has been possible to build exact probabilistic models of GP (schema theories and Markov chain models) and to show that GP is more general than, and in fact includes, genetic algorithms.

Genetic Programming techniques have now been applied to evolvable hardware as well as computer programs.

The aim of this project is to evolve programs (or their representations) to solve the following problem:

Greatest of Three numbers: *Evolve a computer program that can find the greatest of three numbers*.

2. Genetic Algorithms

Genetic Algorithms and their robustness in searching huge search spaces are the motivation behind genetic programming. Genetic algorithms model the evolutionary process to evolve solutions to problems. Natural evolution evolves fit individuals that can tackle their environment. Genetic algorithms evolve fit solutions that can tackle their respective problems. Genetic Algorithms borrow from natural evolution certain *operators,* which are described in the following paragraphs.

Genetic Algorithms differ from other search strategies in the sense that they search using a population of individuals unlike other search methods that use only one solution at a time. This makes genetic algorithms less prone to getting stuck at local optima. By searching using a population of individuals, genetic algorithms perform a *schema search* that implicitly processes $O(n^3)$ solutions in the search space while explicitly evaluating only *n* solutions explicitly. The operators used by genetic algorithms are described below..

2.1 Initialization operator

It creates a population of random individuals that form a sampling of the search space for further evolution by other operators. In biological terms, it creates the *primordial ooze* that is the basis for all other evolution.

2.2 Selection Operator

It selects two individuals from the population based on their fitness. It implements the naturally occurring *survival of the fittest* idiom. The selected individuals are then used to create new individuals using the crossover operator. Several selection strategies like *roulette wheel, tournament, boltzmann and ranking* are widely used.

2.3 Crossover Operator

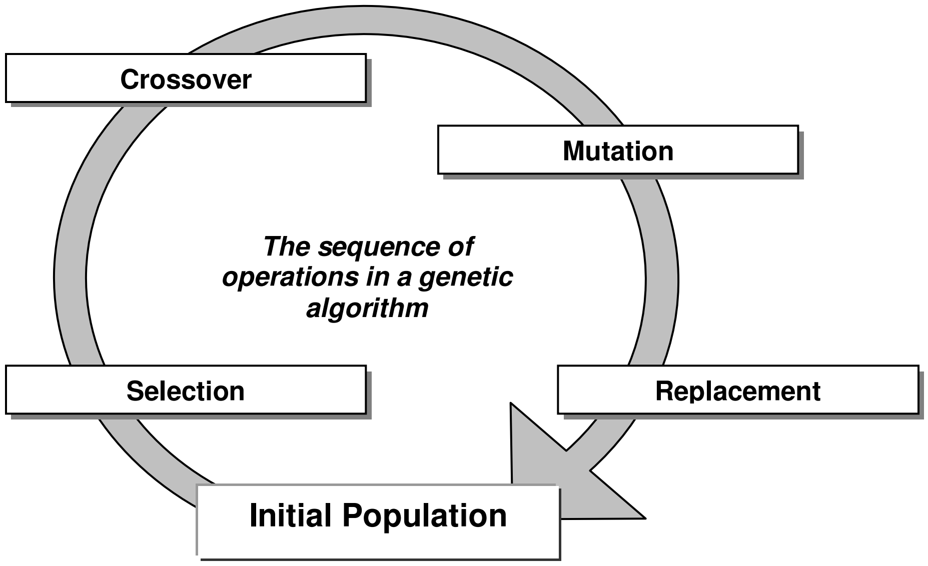
It combines parts of two different solutions to produce new solutions, in an attempt to create better solutions. During this process, the children inherit most characteristics from their parents.

2.4 Mutation Operator

It changes certain aspects of a solution slightly to improve diversity and bring variety in the gene pool. In nature mutation arises due to copying errors. Mutation generally takes place with very low probability.

2.5 Replacement Operator

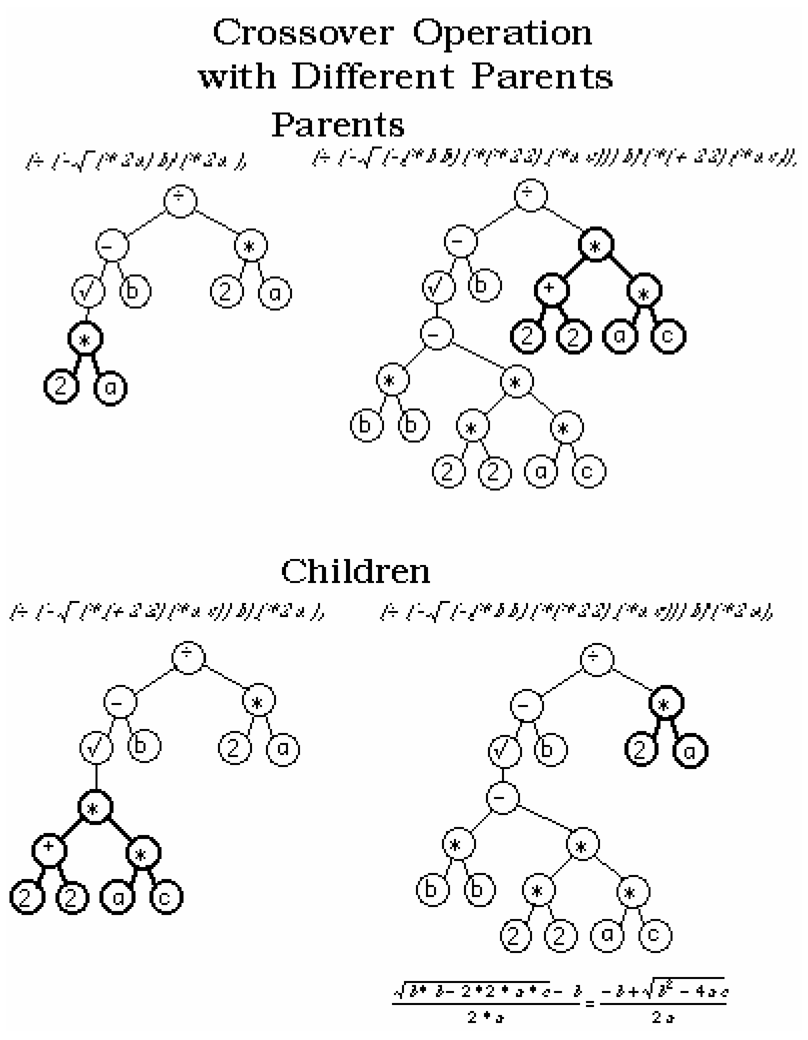
It decides which solutions are removed from the population to make room for the children.



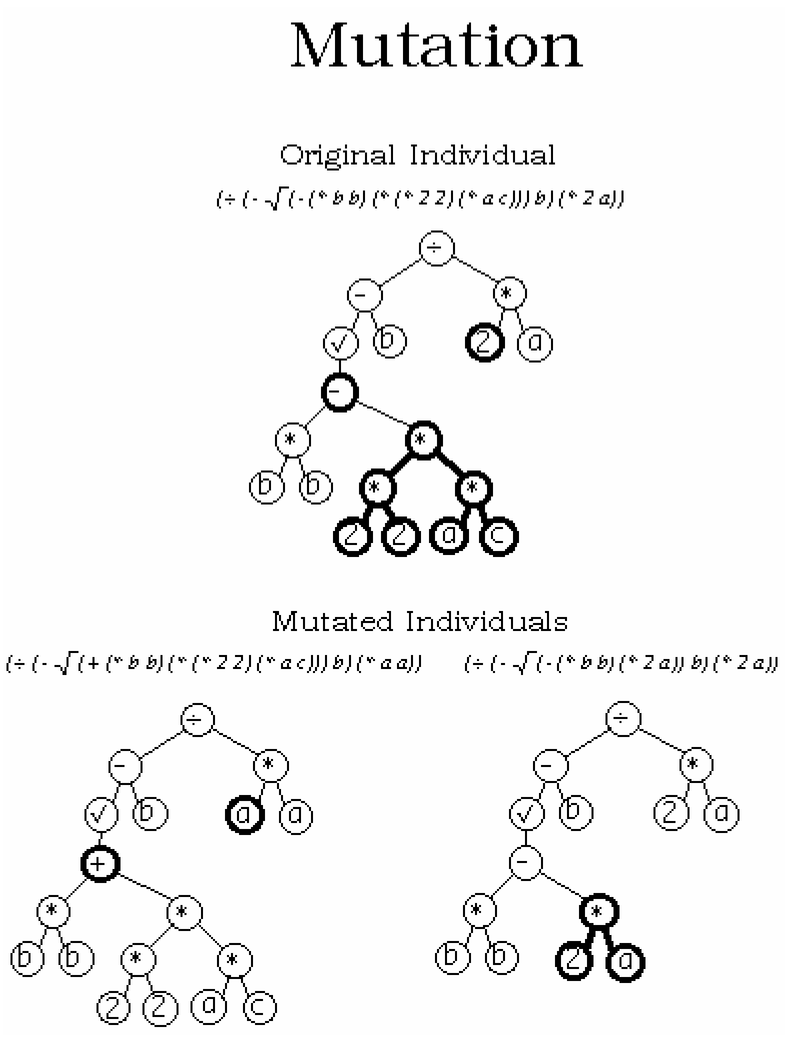
3. Genetic Programming and Genetic Algorithms

Genetic programming also uses the same operators as in genetic algorithms. Genetic programming differs from genetic algorithms only in the way solutions are represented. In genetic programming the solutions are represented as trees (parse trees / expression trees) and are of variable length. In genetic algorithms the solution representation consists mostly of strings or linear data structures that are usually fixed length. In this sense genetic programming forms a superset of genetic algorithms. The invention of genetic programming is generally attributed to John R. Koza (consulting professor, Dept. of Electrical Engg., Stanford University).

The initial random population consists of randomly generated expression trees that are height constrained. Fitness is directly proportional to the error between the outputs and the actual target outputs. The following diagrams demonstrate the crossover and mutation operations in genetic programming.


Crossover Operation
with Different Parents
Parents
(÷ (- √ (* 2 a) b) (* 2 a))
(÷ (- √ (- (* b b) (* (* 2 2) (* a c))) b) (* (+ 2 2) (* a c)))
Children
(÷ (- √ (* (+ 2 2) (* a c)) b) (* 2 a))
(÷ (- √ (- (* b b) (* (* 2 2) (* a c))) b) (* 2 a))

Crossover operation for genetic programming. The bold selections on both parents are swapped to create the offspring or children. (The child on the right is the parse tree representation for the quadratic equation).


Mutation
Original Individual
(÷ (- √(- (* b b) (* (* 2 2) (* a c))) b) (* 2 a))
Mutated Individuals
(÷ (- √(+ (* b b) (* (* 2 2) (* a c))) b) (* a a))
(÷ (- √(- (* b b) (* 2 a)) b) (* 2 a))

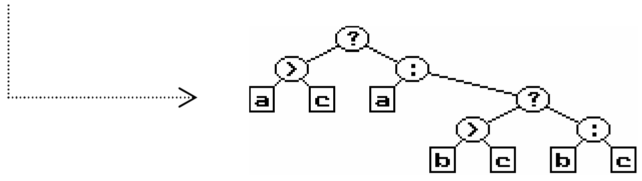
Two different types of mutations. The top parse tree is the original agent. The bottom left parse tree illustrates a mutation of a single terminal (2) for another single terminal (a). It also illustrates a mutation of a single function (-) for another single function (+). The parse tree on the bottom right illustrates a the replacement of a subtree by another subtree.

5. Results

For the greatest of 3 numbers problem:

GP evolved the following expression (prefix representation of expression tree):

1. ?b?&b?>cbc>ab?>?bba?cabcabc
2. ?&ab?&b?>cbc>ab?>?bbaacabc
3. ??>bac>ca?c?b?&a>bcbcca?ba?a>a?cbbb
4. ??>bac>ca?c?b?&a>bcbcca???>ababbca?a>a?cbbb
5. ??>bac>ca?c?b?&a>bcbcca?>>ac>>ca&bca??babbc
6. ?b?&b?>cbc>ab?>?bba?babcabc
7. ?>b?aa>ca?&a>bc?>babac?&a??cbccba?ccc
8. ?>?&&b&abb?>bab?aaccc?>babac
9. ?&>ba?&bc>bcbb?>acac
10. ?>aca?>bcbc (shortest)



Most solutions are bloated; an obvious disadvantage of GP. The parts that do not affect the final value of the expression called 'introns' are the cause for the bloat.