Chapter: Genetic Programming

1 Genetic Programming

Genetic programming is an EA that explores a program space. Genetic programming was proposed by Koza. Genetic programming is viewed as an extension of genetic algorithms. In GP an individual is a computer program and the hope is that through evolution of the population of programs, fitter programs can be evolved until a program that provides an (near) optimal solution is generated.

The basic approach of a GP algorithm is to initially create a population of randomly generated programs. Each program is constructed from building blocks needed to solve the problem GP is being applied to. The fitness of each randomly generated program is then evaluated. If a specified termination criteria is not met good programs are then selected to act as parents for the generation of new programs. New programs are generated by applying genetic operators to parent programs and their fitness is evaluated. The process of selecting good programs and applying genetic operators to them is repeated until a stopping criteria is met and the best program is outputted. Therefore, unlike a GA which searches for a solution to a problem at hand in a solution space GP conducts a search in a program space for a program to solve a problem at hand.

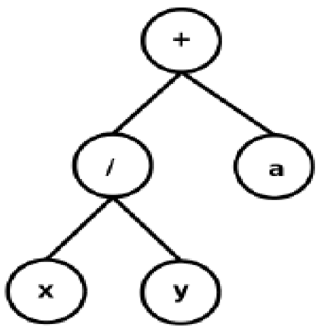

Figure 1:

Figure 1.1.4 is an example of a syntax tree. Whose executable expression is as follows.

$$f = (x/y) + a \tag{1}$$

In GP, programs (individuals) are traditionally represented as syntax trees which can be converted to their corresponding executable expressions usually in prefix notation. Each node of the tree is considered to be a gene. The internal nodes of a syntax tree are known as functions while the external nodes are known as terminals. Functions and terminals are constructed from elements of a function and a terminal set respectively. A function set constitutes of application specific operators and a terminal set constitutes of inputs to the GP program for the problem being considered. Functions and terminals sets are problem dependant primitives from which GP programs are constructed. The operators in a function set are defined with an arity, which represents the number of inputs a specific operator requires. The function set may include the following:

- arithmetic operators {+, -, *, / }.
- mathematical functions { sine, cosine, sqrt }.
- logical operators { AND, OR, NOR, NOT }.
- equality operators { <, >, <=, >=, == }.
- user defined domain specific operators.

There are two important properties that a function set and terminal set must satisfy, these are the closure property and sufficiency property. The closure property requires that any output from any of the functions / terminals must be a valid input for all the other functions. To meet this property the functionality of some operators are modified. An example of this is the arithmetic divide operator. Its functionality is modified to enable division by zero to be performed. The common modification is to have it return a 1 or 0 when a division by 0 operation occurs. This is known as protected divide. The sufficiency property requires that elements of the function and terminal sets should be able to represent the solution.

1.1 GP Algorithm

Algorithm 1 Genetic Programming

```
1: Create an initial population of programs
2: Execute each program and establish the fitness
3:   while termination condition not met do
4:     Select fitter programs to participate in reproduction
5:     Create new programs using genetic operators and update the population
6:     Execute each new program and establish the fitness
7:   end while
8: return best program
```

1.1.1 Initial Population Generation

Like most EAs, initial population generation of GP programs is performed randomly. The number of programs generated is specified by a user-defined *population size* parameter. To create a program, a function for the root node is randomly selected from the function set and the rest of the tree is recursively constructed with the leaves of any new nodes filled until the *maximum tree depth* is reached. Maximum tree depth is defined as the number of nodes between the end node of a tree and the root node. The value of this parameter is user-defined. Three three methods for initial tree generation namely, *full*, *grow* and *ramped half-and-half.* The full method constructs trees in such a manner that all the nodes up to a depth of (*maximum tree depth - 1*) are functions and a depth equal to the maximum tree depth are terminals. The grow method creates trees of variable length. Nodes between the root and (*maximum tree depth - 1*) may randomly be assigned as functions or terminals and those at the maximum tree depth set as terminals. The ramped half-and-half method combines the full and grow method.

1.1.2 Fitness Function

The effectiveness of a GP program is measured using a fitness function which is normally problem dependant. Each program is applied to a training set (fitness cases) and the result is assigned as the fitness of the program. Fitness can be measured in a number of ways. For example, it can be a measure of how close a program output is to a desired outcome such as the accuracy rate in classification or a measure of how quick a GP program is at providing a solution

1.1.3 Selection

As in other evolutionary algorithms selection in GP is biased towards fitter individuals.The two commonly used selection methods are tournament selection and fitness proportionate selection.

1.1.4 Genetic Operators

Crossover, mutation and reproduction are the commonly used genetic operators. In crossover, two programs selected as parents exchange code to create two new programs while in mutation a random change is made to a selected parent program.

- **Subtree Crossover**
  An exchange of subtree branches between parents. Two parents are selected from the population using a selection method. A random crossover point is selected on each parent. Subtrees rooted at the randomly selected crossover points are known as crossover fragments. The two fragments are exchanged from one parent to the other thus creating two offspring.

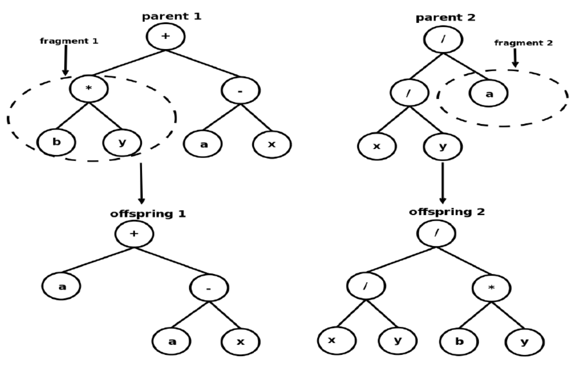

Figure 2:

The size of the offspring must not exceed a user-defined *offspring depth* limit. Those that do must be pruned.

- **Mutation**
  Mutation is an important operator for increasing diversity in a population during evolution. The two commonly used mutation types are *Grow* and *Shrink* mutation.

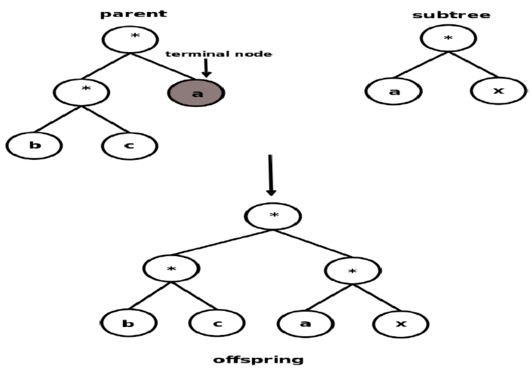

Figure 3:

Shrink mutation is an operator that replaces a randomly selected subtree with a randomly created terminal node. Grow mutation works by randomly selecting a terminal and replacing it with a subtree. Figure 3 is an illustration of grow mutation. Grow mutation has the effect of increasing the size of a tree. The resultant offspring needs to conform to the specified offspring depth in a similar manner to the offspring resulting from crossover. Therefore, a parameter known as *mutation depth* is specified for a GP system. The purpose of this parameter is to control the size of a subtree created by a mutation operator. The population size of the new generation is the same size as the initially randomly generated population. Part of the new population is evolved by crossover and the other part by mutation. The number of individuals created by crossover and those created by mutation are determined by the application rates parameters and they should sum up to the population size. The application rates are known as the *crossover rate* and the *mutation rate.*

1.1.5 Population Replacement

The most frequently used population replacement methods are generational and steady-state. In steady state the population is updated in such a manner that one offspring replaces a member of the current population based on fitness.

1.1.6 Termination

Two termination conditions are usually used in GP, a maximum number of generations or a problem specific solution is met. The maximum number of generations is a user-defined parameter and is specified before a run.

1.2 Applications of GP

The flexibility of GP enables it to be applied to a wide range of real-world problems.

- cyber-security
- bio-informatics
- medical domain
- text mining

amongst a number of problems domains.

1.2.1 References

Koza , J. R. Genetic programming as a means for programming computers by natural selection. Statistics and computing 4, 2 (1994), 87–112.

Langdon, W. B., & Poli, R. (2013). Foundations of genetic programming. Springer Science & Business Media.