GENETIC PROGRAMMING

Definition

In artificial intelligence, **genetic programming (GP)** is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task.

Genetic programming (GP) is a specialization of **genetic algorithms** where each individual is a computer program.

Genetic Programming

It is a machine learning technique used 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 roots of GP begin with the evolutionary algorithms first utilized by Nils Aall Barricelli in 1954 as applied to evolutionary simulations but evolutionary algorithms became widely recognized as optimization methods as a result of the work of Ingo Rechenberg in the 1960s and early 1970s - his group was able to solve complex engineering problems through evolution strategies (1971 PhD thesis and resulting 1973 book).

Also highly influential was the work of John Holland in the early 1970s, and particularly his 1975 book. The first results on the GP methodology were reported by Stephen F. Smith (1980) and Nichael L. Cramer (1985),. In 1981 Forsyth reported the evolution of small programs in forensic science for the UK police. John R. Koza is a main proponent of GP and has pioneered the application of genetic programming in various complex optimization and search problems.

GP is very computationally intensive and so in the 1990s it was mainly used to solve relatively simple problems. But more recently, thanks to improvements in GP technology and to the exponential growth in CPU power, GP produced many novel and outstanding results in areas such as quantum computing, electronic design, game playing, sorting, searching and many more. These results include the replication or development of several post-year-2000 inventions. GP has also been applied to evolvable hardware as well as computer programs. There are several GP patents listed in the web site.

Developing a theory for GP has been very difficult and so in the 1990s GP was considered a sort of outcast among search techniques.

But 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).

Chromosome representation

GP evolves computer programs, traditionally represented in memory as tree structures. Trees can be easily evaluated in a recursive manner. Every tree node has an operator function and every terminal node has an operand, making mathematical expressions easy to evolve and evaluate. Thus traditionally GP favors the use of programming languages that naturally embody tree structures (for example, Lisp; other functional programming languages are also suitable).

Non-tree representations have been suggested and successfully implemented, such as the simpler linear genetic programming which suits the more traditional imperative languages. The commercial GP software Discipulus, uses AIM, automatic induction of binary machine code to achieve better performance. MicroGP uses a representation similar to linear GP to generate programs that fully exploit the syntax of a given assembly language.

Genetic operators

Crossover

The main operators used in evolutionary algorithms such as GP are crossover and mutation. Crossover is applied on an individual by simply switching one of its nodes with another node from another individual in the population. With a tree-based representation, replacing a node means replacing the whole branch. This adds greater effectiveness to the crossover operator. The expressions resulting from crossover are very much different from their initial parents.

The following code suggests a simple implementation of individual deformation using crossover:

```
individual. Children[randomChildIndex] = otherIndividual.Children[randomChildIndex];
```

Mutation

Mutation affects an individual in the population. It can replace a whole node in the selected individual, or it can replace just the node's information. To maintain integrity, operations must be fail-safe or the type of information the node holds must be taken into account. For example, mutation must be aware of binary operation nodes, or the operator must be able to handle missing values.

A simple piece of code:

```
individual. Information = randomInformation;
```

or

```
individual = generateNewIndividual;
```

Meta-Genetic Programming

Meta-Genetic Programming is the proposed meta learning technique of evolving a genetic programming system using genetic programming itself. It suggests that chromosomes, crossover, and mutation were themselves evolved, therefore like their real life counterparts should be allowed to change on their own rather than being determined by a human programmer. Meta-GP was proposed by Jürgen Schmidhuber in 1987; it is a recursive but terminating algorithm, allowing it to avoid infinite recursion.

Critics of this idea often say this approach is overly broad in scope. However, it might be possible to constrain the fitness criterion onto a general class of results, and so obtain an evolved GP that would more efficiently produce results for sub-classes.

This might take the form of a Meta evolved GP for producing human walking algorithms which is then used to evolve human running, jumping, etc. The fitness criterion applied to the Meta GP would simply be one of efficiency.

For general problem classes there may be no way to show that Meta GP will reliably produce results more efficiently than a created algorithm other than exhaustion. The same holds for standard GP and other search algorithms, of course.

Source:

http://www.juliantrubin.com/encyclopedia/electronics/genetic_algorithm.html