Genetic programming as a means for programming computers by natural selection

JOHN R. KOZA
Computer Science Department, Stanford University, Stanford, CA 94305, USA

Many seemingly different problems in machine learning, artificial intelligence, and symbolic processing can be viewed as requiring the discovery of a computer program that produces some desired output for particular inputs. When viewed in this way, the process of solving these problems becomes equivalent to searching a space of possible computer programs for a highly fit individual computer program. The recently developed genetic programming paradigm described herein provides a way to search the space of possible computer programs for a highly fit individual computer program to solve (or approximately solve) a surprising variety of different problems from different fields. In genetic programming, populations of computer programs are genetically bred using the Darwinian principle of survival of the fittest and using a genetic crossover (sexual recombination) operator appropriate for genetically mating computer programs. Genetic programming is illustrated via an example of machine learning of the Boolean 11-multiplexer function and symbolic regression of the econometric exchange equation from noisy empirical data.

Hierarchical automatic function definition enables genetic programming to define potentially useful functions automatically and dynamically during a run, much as a human programmer writing a complex computer program creates subroutines (procedures, functions) to perform groups of steps which must be performed with different instantiations of the dummy variables (formal parameters) in more than one place in the main program. Hierarchical automatic function definition is illustrated via the machine learning of the Boolean 11-parity function.

Keywords: Genetic programming, genetic algorithm, crossover, hierarchical automatic function definition, symbolic regression, Boolean 11-multiplexer, econometric exchange equation, Boolean 11-parity

1. Introduction and overview

Computer programs are among the most complex and intricate structures created by human beings. They are usually written line by line by applying human knowledge and intelligence to the problem at hand. Writing a computer program is usually difficult. Indeed, one of the central questions in computer science (attributed to Arthur Samuel in the 1950s) is

How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?

In the natural world, complex and intricate structures do not arise via explicit design and programming or from the application of human intelligence. Instead, complex and successful organic structures evolve over a period of time as the consequence of Darwinian natural selection and the creative effects of sexual recombination (genetic crossover) and mutation. Complex structures evolve in nature as a consequence of a fitness metric applied by the problem environment because structures that are more fit in grappling with their environment survive and reproduce at a higher rate.

The question arises as to whether an analogue of natural selection and genetics can be applied to the problem of creating a program that enables a computer to solve a problem. That is, can complex computer programs be created, not via human intelligence, but by applying a fitness measure appropriate to the problem environment?

Such a process of genetical breeding of computer pro-
programs might start with a primordial ooze consisting of a population of hundreds or thousands of randomly created computer programs of various randomly determined sizes and shapes. In such a process, each program in the population would be observed as it tries to grapple with its environment—that is, to solve the problem at hand. A value would then be assigned to each program, reflecting how fit it is in solving the problem at hand. We might then allow a program in the population to survive to a later generation of the process with a probability proportionate to its observed fitness. Additionally, we might also select pairs of programs from the population with a probability proportionate to their observed fitness and create new offspring by recombining subprograms from them at random. We would apply the above steps to the population of programs over a number of generations.

Anyone who has ever written and debugged computer programs and has experienced their brittleness, highly nonlinear, and perversely unforgiving nature will probably be understandably skeptical about the proposition that the biologically motivated process sketched above could possibly produce a useful computer program. However, in this article we present a number of examples from various fields supporting the surprising and counterintuitive notion that computers can indeed be programmed to find a function which solves, or approximately solves, a problem.

2. Background on genetic algorithms and genetic programming

John Holland's pioneering book *Adaptation in Natural and Artificial Systems* described how the evolutionary process in nature can be applied to artificial systems using the genetic algorithm operating on fixed-length character strings (Holland, 1975). Holland demonstrated that a population of fixed-length character strings (each representing a proposed solution to a problem) can be genetically bred using the Darwinian operation of fitness proportionate reproduction and the genetic operation of recombination. The recombination operation combines parts of two chromosome-like fixed-length character strings, each selected on the basis of their fitness, to produce new offspring strings. Holland established, among other things, that the genetic algorithm is a near optimal approach to adaptation in that it maximizes expected overall average payoff when the adaptive process is viewed as a multi-armed slot machine problem requiring an optimal allocation of future trials given currently available information. The genetic algorithm has proven successful at searching non-linear multidimensional spaces in order to solve, or approximately solve, a wide variety of problems (Goldberg, 1989; Davis, 1987; 1991; Davidor, 1991; Michalewicz, 1992). Recent conference proceedings provide an overview of current work in the field (Schaffer, 1989; Forrest, 1990; Belew and Booker, 1991; Rawlins, 1991; Mayer and Wilson, 1991; Schwefel and Manner, 1991; Langton et al., 1992; Whitley, 1992).

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate a coded chromosomal representation of the problem. The representation scheme can therefore severely limit the window by which the system observes its world. On the other hand, the use of fixed-length character strings has permitted Holland and others to construct a significant body of theory as to why genetic algorithms work. Much of this theoretical analysis depends on the mathematical tractability of the fixed-length character strings as compared with mathematical structures that are more complex and comparatively less susceptible to theoretical analysis. The need for increasing the complexity of the structures undergoing adaptation using the genetic algorithm has been reflected by considerable work over the years in that direction (Smith, 1980; Cramer, 1985; Holland, 1986; Holland et al., 1986; Wilson, 1987a,b; Fujiki and Dickinson, 1987; Goldberg et al., 1989).

For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program (i.e. a hierarchical composition of primitive functions and terminals) of indeterminate size and shape, as opposed to character strings whose size has been determined in advance. It is difficult, unnatural, and overly restrictive to attempt to represent hierarchies of dynamically varying size and shape with fixed-length character strings.

Genetic programming provides a way to find a computer program of unspecified size and shape to solve, or approximately solve, a problem. The book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza, 1992a) describes genetic programming in detail. A videotape visualization of applications of genetic programming can be found in the *Genetic Programming: The Movie* (Koza and Rice, 1992a). See also Koza (1992b).

3. Overview of genetic programming

Genetic programming continues the trend of dealing with the problem of representation in genetic algorithms by increasing the complexity of the structures undergoing adaptation. In particular, the individuals in the population in genetic programming are hierarchical compositions of primitive functions and terminals appropriate to the particular problem domain. The set of primitive functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-