CONCEPT FORMATION AND DECISION TREE INDUCTION USING THE GENETIC PROGRAMMING PARADIGM

John R. Koza
Computer Science Department
Stanford University
Stanford, California 94305 USA
E-MAIL: Koza@Sunburn.Stanford.Edu PHONE: 415-941-0336

ABSTRACT

This paper describes the application of the recently developed "genetic programming" paradigm to the problem of concept formation and decision tree induction.

1. INTRODUCTION AND OVERVIEW

This paper describes the recently developed "genetic programming" paradigm which genetically breeds populations of computer programs to solve problems. In genetic programming, the individuals in the population are hierarchical compositions of functions and arguments of various sizes and shapes.

2.BACKGROUND ON GENETIC ALGORITHMS

Genetic algorithms are highly parallel mathematical algorithms that transform populations of individual mathematical objects (typically fixed-length binary character strings) into new populations using operations patterned after (1) natural genetic operations such as sexual recombination (crossover) and (2) fitness proportionate reproduction (Darwinian survival of the fittest). Genetic algorithms begin with an initial population of individuals (typically randomly generated) and then iteratively (1) evaluate the individuals in the population for fitness with respect to the problem environment and (2) perform genetic operations on various individuals in the population to produce a new population. John Holland of the University of Michigan presented the pioneering formulation of genetic algorithms for fixed-length character strings in *Adaptation in Natural and Artificial Systems* (Holland 1975).

3. BACKGROUND ON GENETIC PROGRAMMING PARADIGM

Entire computer programs can be genetically bred to solve problems in a variety of different areas of artificial intelligence, machine learning, and symbolic processing (Koza 1989, 1990a, 1990b). This new genetic algorithm paradigm has been successfully applied to example problems in several different areas, including (1) machine learning of functions, (2) planning, (3) automatic programming, (4) sequence induction, (5) pattern recognition, (6) symbolic "data to function" regression, symbolic "data to function" integration, and symbolic "data to function" differentiation, (7) symbolic solution to functional equations (including differential equations with initial conditions, integral equations, and general functional equations), (8) empirical discovery, (9) simultaneous architectural design and training of neural networks, and (10) game-playing (e.g. finding a minimax strategy for a differential pursuer-evader game and finding a minimax strategy for a discrete game represented by a game tree in extensive form).

In this recently developed "genetic programming" paradigm, the individuals in the population are compositions of functions and terminals appropriate to the particular problem domain. The set of functions used typically includes arithmetic operations, mathematical functions, conditional logical operations, and domain-specific functions. Each function in the function set must be well defined for any element in the range of every other function in the set. The set of terminals used typically includes inputs (sensors) appropriate to the problem domain and various constants. The search space is the hyperspace of all possible compositions of functions that can be recursively composed of the available functions and terminals. The symbolic expressions (S-expressions) of the LISP programming language are an especially convenient way to create and manipulate the compositions of functions and terminals described above.

The basic genetic operations for the genetic programming paradigm are fitness proportionate reproduction and crossover (recombination). Fitness proportionate reproduction is the basic engine of Darwinian reproduction and survival of the fittest and operates for genetic programming paradigms in the same way as it does for conventional genetic algorithms. The crossover

operation for genetic programming paradigms is a sexual operation that operates on two parental LISP S-expressions and produces two offspring S-expressions using parts of each parent. In particular, the crossover operation creates new offspring S-expressions by exchanging sub-lists (sub-trees) between the two parents. Because entire sub-lists are swapped, this genetic crossover (recombination) operation produces syntactically and semantically valid LISP S-expressions as offspring regardless of which point is selected in either parent.

For example, consider the following two parental LISP S-expressions:

```
(OR (NOT D1) (AND D0 D1))
(OR (OR D1 (NOT D0)) (AND (NOT D0) (NOT D1)))
```

Suppose that the second point of the first parent (i.e. the NOT function) is randomly selected as the crossover point of the first parent and that the sixth point of the second parent (i.e. the AND function) is randomly selected as the crossover point of the second parent. The two offspring resulting from crossover are shown below:

```
(OR (OR D1 (NOT D0)) (NOT D1))
(OR (AND (NOT D0) (NOT D1)) (AND D0 D1)).
```

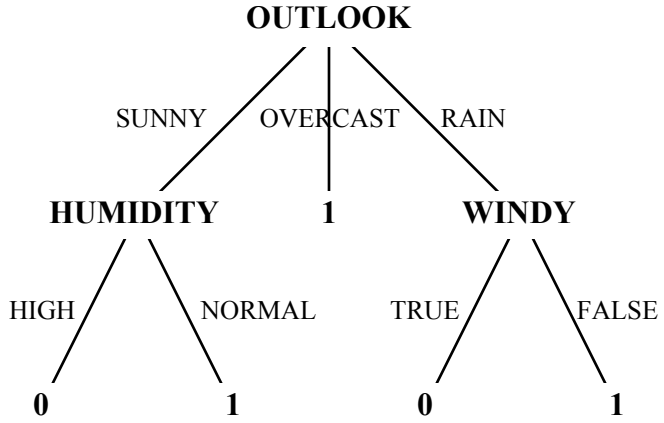
4. CONCEPT FORMATION

Quinlan (1986) initiated development of a particularly effective family of hierarchical classification systems for inducing a decision tree from a limited number of training case examples. In ID3 (and various other systems of the ID3 family), the goal is to partition a universe of objects into classes. Each object in the universe is described in terms of various attributes. The system is first presented with a set of training case examples which consist of the attributes of a particular object and the class to which it belongs. The system then generates a decision tree which hopefully can then be used to classify a new object correctly into a class using the attributes of the new object. The external points (leaves) of the decision tree are the eventual class names. The internal points of the decision tree are attribute-based tests which have one branch emanating from the decision point for each possible outcome of the test.

The induction of such decision trees for classifying objects can be approached by genetically breeding LISP S-expressions for performing this

task. In particular, the set of terminals is the set of class names. The set of functions is the set of attribute-based tests. Note that this set of attribute-based tests are always assumed to be given and available for solving induction problems via decision trees of the ID3 family. Notice that ID3 is similar to the genetic programming paradigm in that the set of functions is given. Each function has as many arguments as there are possible outcomes of that particular test. When a particular object is presented to the LISP S-expression (i.e. the decision tree), each function in the S-expression tests one attribute of the object and returns the particular one of its arguments designated by the outcome of the test. If the designated argument is an terminal, the function returns the class name. When the S-expression is fully evaluated in LISP's usual left-oriented depth-first way, the S-expression as a whole thus returns a class name. That is, the S-expression is a decision tree that classifies the new object into one of the classes.

To demonstrate the technique of genetically inducing a decision tree, we apply this approach to the small training set of 14 objects presented in Quinlan (1986). In Quinlan's problem, each object has four attributes and belongs to one of two classes ("positive" or "negative"). The attribute of "temperature", for example, can assume the possible values hot, mild, or cool. Humidity can assume the values of high or normal. Outlook can assume values of sunny, overcast, or rain. Windy can assume values of true or false. The decision tree presented by Quinlan as the solution for this problem is shown below:



If, for example, the OUTLOOK of a particular object is sunny and the

HUMIDITY is high, then that object is classified into class 0 (negative).

In order to genetically induce the decision tree, each of the four attributes in this problem is converted into a function. For example, the function "temperature" operates in such a way that, if the current object has a temperature of "mild," the function returns its second argument as its return value. The other attributes in this problem, namely "humidity", "outlook", and "windy", are similarly converted to functions. The function set for this problem is therefore F = {TEMP, HUM, OUT, WIND} with 3, 2, 3, and 2 arguments, respectively. The set of terminals for this problem is T = {0, 1} since there are two classes. A population size of 300 was used.

In one run, the LISP S-expression

```
(OUT (WIND 1 0) (WIND 1 1) (HUM 0 1)
```

emerged on the 8th generation with a maximal fitness value of 14 (i.e. it correctly classified all 14 training cases). Since (WIND 1 1) is equivalent to just the constant atom 1, this S-expression is equivalent to the decision tree presented in Quinlan (1986) using ID3.

REFERENCES

Holland, John H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press 1975.

Koza, John R. "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs." In *Proceedings of the 11th International Joint Conference on Artificial Intelligence.* San Mateo: Morgan Kaufman 1989.

Koza, John R. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Stanford University Computer Science Department Technical Report STAN-CS-90-1314. 1990a.

Koza, John R. Evolution and co-evolution of computer programs to control independently-acting agents. *Proceedings of Conference on Simulation of Adaptive Behavior.*. Cambridge, MA: MIT Press. 1990b.

Quinlan, J. Induction of decision trees. *Machine Learning* 1(1), 81-106, 1986.

CONCEPT FORMATION AND DECISION TREE INDUCTION USING THE GENETIC PROGRAMMING PARADIGM

John R. Koza
Computer Science Department
Stanford University
Stanford, CA 94305 USA
E-MAIL: Koza@Sunburn.Stanford.Edu PHONE: 415-941-0336

KEYWORDS: Genetic algorithm, genetic programming paradigm, concept formation, decision trees, ID3, hierarchies, LISP programming language