Submitted to Workshop on Integrating Symbolic and Neural Processes to be held on Sunday July 29 in Boston in conjunction with AAAI-90

ABSTRACT

INTEGRATING SYMBOLIC PROCESSING INTO GENETIC ALGORITHMS

John R. Koza
Computer Science Department
Stanford University
Margaret Jacks Hall
Stanford, CA 94305
PHONE: 415-941-0336
FAX: 415-941-9430
E-MAIL: Koza@Sunburn.Stanford.edu

ABSTRACT
INTEGRATING SYMBOLIC PROCESSING INTO GENETIC ALGORITHMS

Although genetic algorithms, like neural networks, are seemingly inappropriate for handling symbolically oriented problems, recent work in the fields of both the genetic algorithm and neural network argues otherwise. This presentation will discuss a group of seemingly different problems from symbolic processing, artificial intelligence, and machine learning which can be solved using genetic algorithms if the appropriate representation scheme and appropriate modifications to the repertoire of genetic operations are adopted.

The approaches used in applying genetic algorithms to such symbolic problems may shed light on the problem of applying neural networks to symbolic problems.

Many problems from symbolic processing appear to be inappropriate candidates for solution via genetic algorithms because they, in effect, require discovery of a computer program that produces some desired output value when presented with particular inputs. However, with an appropriate representation scheme and appropriate modifications of the traditional genetic operations, it is possible to genetically search the space of possible computer programs for a most fit individual computer program. In this new “genetic programming” paradigm, populations of computer programs (LISP symbolic expressions) are genetically bred using the Darwinian principle of survival of the fittest and using a genetic crossover (recombination) operator appropriate for genetically mating computer programs.

Depending on the terminology of the particular field of interest, the “computer program” may be called a robotic action plan, an optimal control strategy, a decision tree, an econometric model, a game-playing strategy, the state transition equations, the transfer function, or, perhaps merely, a composition of functions. Similarly, the “inputs” to the “computer program” may be called sensor values, state variables, independent variables, attributes of an object, or, perhaps more prosaically, the arguments to a function.

The methods for applying genetic algorithms to symbolic problems are illustrated using examples from the areas of

- function learning
- robotic planning
- symbolic function identification
- symbolic regression

- symbolic integration and differentiation
- symbolic solution of differential equations
- game-playing, sequence induction
- empirical discovery and econometric modeling
- concept formation
- automatic programming
- pattern recognition
- time-optimal control.

Problems of the type described above can be expeditiously solved only if the flexibility found in computer programs is available. This flexibility includes the ability to perform alternative computations conditioned on the outcome of intermediate calculations, to perform computations on variables of many different types, to perform iterations and recursions to achieve the desired result, and to define and subsequently use computed values and sub-programs.

The process of solving these problems can be reformulated as as a search for a most fit individual computer program in the space of possible computer programs composed of various terms (atoms) along with standard arithmetic operations, standard programming operations, standard mathematical functions, and various functions peculiar to the given problem domain. Four types of objects are manipulated as we build computer programs, namely, functions of various number of arguments; variable atoms;

constant atoms; and control structures such as If-Then-Else, Do-Until, etc.

As will be seen, the LISP S-expression required to solve each of the problems described above will emerge from a simulated evolutionary process. This process will start with an initial population of randomly generated LISP S-expressions composed of functions and atoms appropriate to the problem domain. Then, a process based on the Darwinian model of reproduction and survival of the fittest and genetic recombination will be used to create a new population of individuals. In particular, a genetic process of sexual reproduction among two parental S-expressions will be used to create offspring S-expressions. At least one of two participating parental S-expressions will be selected in proportion to fitness. The resulting offspring S-expressions will be composed of sub-expressions ("building blocks") from their parents. Finally, the new population of offspring (i.e. the new generation) will replace the old population of parents and the process will continue.

As will be seen, this highly parallel, locally controlled, and decentralized process will produce populations which, over a period of generations, tend to exhibit increasing average fitness in dealing with their environment, and which, in addition, can robustly (i.e. rapidly and

effectively) adapt to changes in their environment.