Short Running Title:

Genetic Modeling

Send communications to:
John R. KOZA
Computer Science Department, Stanford University,
Stanford, CA 94305 USA
EMAIL: Koza@Sunburn.Stanford.Edu **PHONE:** 415-941-0336. **FAX:** 415-941-9430.

A GENETIC APPROACH TO ECONOMETRIC MODELING

John R. KOZA

Computer Science Department, Stanford University
Stanford, CA 94305 USA

ABSTRACT

An important problem in economics is finding the mathematical relationship between the empirically observed variables measuring a system. In many conventional modeling techniques, one necessarily begins by selecting the size and shape of the model. After making this choice, one usually then tries to find the values of certain coefficients required by the particular model so as to achieve the best fit between the observed data and the model. But, in many cases, the most important issue is the size and shape of the model itself.

Finding the functional form of the model can be viewed as searching a space of possible computer programs for the particular computer program which produces the desired output for given inputs. This most fit computer program can be found via a recently developed genetic programming paradigm originally developed for solving artificial intelligence problems. This new genetic programming paradigm genetically breeds populations of computer programs in a Darwinian competition using genetic operations, such as crossover (sexual recombination).

In this paper, we illustrate the process of discovering a model by rediscovering the well-known multiplicative (non-linear) "exchange equation" $P=\frac{MV}{Q}$ relating the money supply, price level, gross national product, and velocity of money in an economy.

KEYWORDS

Modeling; Non-linear models; Empirical discovery; Regression; Genetic algorithms; Genetic programming paradigm; Darwinian natural selection; Sexual recombination

INTRODUCTION AND OVERVIEW

An important problem in economics and other areas of science is finding the mathematical relationship between the empirically observed variables measuring a system. In many conventional modeling techniques, one necessarily begins by selecting the size and shape of the mathematical model. Because the vast majority of available mathematical tools only handle linear models, this

choice is often simply a linear model. After making this choice, one usually then tries to find the values of certain coefficients and constants required by the particular model so as to achieve the best fit between the observed data and the model.

But, in many cases, the most important issue is the size and shape of the mathematical model itself. That is, one really wants first to find the functional form of the model that best fits observed empirical data, and, only then, go on to find any constants and coefficients that happen to be needed.

We suggest that finding the functional form of the model can be viewed as being equivalent to searching a space of possible computer programs for the particular individual computer program which produces the desired output for given inputs. That is, one is searching for the computer program whose behavior best fits the observed data. Computer programs offer great flexibility in the ways that they compute their output from the given inputs.

The most fit individual computer program can be found via a recently developed genetic programming paradigm originally developed for solving artificial intelligence problems. This new genetic programming paradigm genetically breeds populations of computer programs in a Darwinian competition using genetic operations. The Darwinian competition is based on the principle of survival and reproduction of the fittest. The genetic crossover (sexual recombination) operator is designed for genetically mating computer programs so as to create potentially more fit new offspring programs. The best single individual computer program produced by this process after many generations can be viewed as the solution to the problem.

In this paper, we illustrate the process of formulating and solving problems of modeling (which may also be called empirical discovery, symbolic regression, or symbolic function identification) with this new genetic programming paradigm. In particular, the problem of modeling requires finding a function, in symbolic form, that fits given numeric data points representing some observed system. Finding such an empirical model for a system can also be used in forecasting future values of the state variables of the system.

In this paper, we focus on the simple non-linear econometric "exchange equation" $P=\frac{MV}{Q}$ relating the price level, gross national product, money supply, and velocity of money in an economy.

We claim that the process of solving the problems of the type described above can be reformulated as a search for a most fit individual computer program in the space of possible computer programs. In our research we use the LISP programming language so in particular, the search space is the hyperspace of LISP "symbolic expressions" (called S-expressions) composed of various terms (called atoms in LISP) along with standard arithmetic operations, standard programming operations, standard mathematical functions, and various functions peculiar to the given problem domain.

For example, the standard arithmetic function of multiplication is relevant when we are attempting to discover the econometric "exchange equation" $P=\frac{MV}{Q}$. In general, the objects that are manipulated in our attempts to build computer programs are of four types. These objects include functions of various number of arguments, such as multiplication mentioned above; variable atoms; constant atoms, such as 0, 1, etc.; and control structures such as If-Then-Else, Do-Until, etc.

In this recently developed new genetic algorithm paradigm, the individuals in the genetic population are compositions of functions (i.e. LISP S-expressions). In particular, the individuals in the population are LISP S-expressions created from compositions of functions and atoms appropriate to the particular problem domain. The set of functions used typically includes arithmetic operations,

mathematical functions, conditional logical operations, and functions appropriate to the problem domain at hand. The set of atoms used typically includes various constants and particular inputs appropriate to the problem domain. The search space is the hyperspace of all possible LISP S-expressions that can be recursively composed of the available functions and atoms. The crossover operation appropriate for mating two parents from this hyperspace of LISP S-expressions creates new offspring S-expressions by exchanging sub-trees (i.e. sub-lists) between the two parents. The results of this process are inherently hierarchical.

As will be seen, the LISP S-expression required to solve the problem described above will emerge from a simulated evolutionary process using a new genetic programming paradigm.

In each case, this simulated evolutionary process will start with an initial population of randomly generated LISP S-expressions composed of functions and atoms appropriate to the problem domain.

The fitness of each individual LISP S-expression in a population at any stage of the process will be measured in a simple, natural, and consistent way. Simply stated, fitness will measure how well the individual performs in the particular problem environment. In particular, fitness will be measured by the sum of the squares of the distances (taken for all the environmental cases) between the point in the solution space created by the S-expression for a given set of arguments and the correct point in the solution space. The closer this sum is to zero, the better the S-expression.

Predictably, the initial random individual S-expressions will have exceedingly poor fitness. Nonetheless, some individuals in the population will be somewhat more fit than others. And, in the valley of the blind, the one-eyed man is king.

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.

At each stage of this highly parallel, locally controlled, and decentralized process, the state of the process will consist only of the current population of individuals. Moreover, the only input to the algorithmic process will be the observed fitness of the individuals in the current population in grappling with the problem environment.

As will be seen, this 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.

The solution produced by this process at any given time can be viewed as the entire population of disjunctive alternatives (typically with improved overall average fitness), or, more commonly, as the single best individual in the population at that time ("winner take all").

The inherently hierarchical character of the computer programs is an essential aspect of the genetic programming paradigm. The dynamic variability of the computer programs that are developed along the way to a solution is also an essential aspect of the genetic programming paradigm. In each case, it would be difficult and unnatural to try to specify or restrict the size and shape of the eventual solution in advance. Moreover, the advance specification or restriction of the size and shape of the solution to a problem narrows the window by which the system views the world and might well preclude finding the solution to the problem.

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

• natural genetic operations such as sexual recombination (crossover) and

• 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). Holland established, among other things, that the genetic algorithm is a mathematically 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.

In this work, Holland demonstrated that a wide variety of different problems in adaptive systems (including problems from economics, game theory, pattern recognition, optimization, and artificial intelligence) are susceptible to reformulation in genetic terms so that they can potentially be solved by the highly parallel mathematical genetic algorithm that simulates Darwinian evolutionary processes and naturally occurring genetic operations on chromosomes.

Genetic algorithms differ from most iterative algorithms in that they simultaneously manipulate a population of individual points in the search space rather than a single point in a search space. The current increasing interest in genetic algorithms stems from their intrinsic parallelism, their mathematical near optimality in solving problems, and the availability of increasing powerful computers. An overview of genetic algorithms can be found in Goldberg's *Genetic Algorithms in Search, Optimization, and Machine Learning* (1989). Recent work in genetic algorithms and genetic classifier systems can be surveyed in Davis (1987), Davis (1991), Schaffer (1989), and Belew and Booker (1991).

Representation is a key issue in genetic algorithm work because genetic algorithms directly manipulate the coded representation of the problem and because the representation scheme can severely limit the window by which the system observes its world. Fixed length character strings present difficulties for some problems — particularly problems in artificial intelligence where the desired solution is hierarchical and where the size and shape of the solution is unknown in advance. The structure of the individual mathematical objects that are manipulated by the genetic algorithm can be more complex than the fixed length character strings. For example, Holland's classifier system (Holland 1986) is a cognitive architecture into which the genetic algorithm is embedded so as to allow adaptive modification of a population of string-based if-then rules (whose condition and action parts are fixed length binary strings).

Marimon, McGrattan, and Sargent (1990) have applied genetic classifier systems to describe the emergence of a commodity in a simulated trading environment as a medium of exchange among artificially intelligent agents . Holland (1990) discusses the global economy as an adaptive system.

BACKGROUND ON THE GENETIC PROGRAMMING PARADIGM

The recently developed genetic programming paradigm described in this paper continues the above trend in the field of genetic algorithms towards increasing the complexity of the structures undergoing adaptation. In the new genetic programming paradigm, populations of entire computer

programs are genetically bred to solve problems.

We have recently shown that 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, 1991d). 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 every element in the range of any other function in the set. The set of terminals used typically includes inputs 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. These S-expressions in LISP correspond directly to the "parse tree" that is internally created by most compilers.

We have recently shown that entire computer programs can be genetically bred to solve problems in a variety of different areas of artificial intelligence, machine learning, and symbolic processing. Specifically, this recently developed genetic programming paradigm has been successfully applied to example problems in several different areas, including

- machine learning of functions (e.g. learning the Boolean 11-multiplexer function),
- planning (e.g. navigating an artificial ant along an irregular trail, developing a robotic action sequence that can stack blocks in a specified order) (Koza 1990c),
- automatic programming (e.g. solving pairs of linear equations, solving quadratic equations for complex roots, and discovering trigonometric identities),
- optimal control (e.g. centering a cart and balancing a broom on a moving cart in minimal time by applying a "bang bang" force to the cart) (Koza and Keane 1990a, Koza and Keane 1990b),
- pattern recognition (e.g. translation-invariant recognition of a simple one-dimensional shape in a linear retina),
- sequence induction (e.g. inducing a recursive procedure for generating sequences such as the Fibonacci and Hofstadter sequences and simple chaotic sequences),
- symbolic "data to function" regression, integration, differentiation, and symbolic solution to general functional equations (including differential equations with initial conditions, and integral equations),
- empirical discovery (e.g. rediscovering Kepler's Third Law), and
- emergent behavior (e.g. discovering a computer program which, when executed by all the ants in an ant colony, produces interesting overall "emergent" behavior) (Koza 1991a).
- concept formation and decision tree induction (Koza 1991c),
- finding minimax strategies for games (e.g. differential pursuer-evader games, discrete games in extensive form) by both evolution and co-evolution (Koza 1990b),
- simultaneous architectural design and training of neural networks.

DESCRIPTION OF THE GENETIC PROGRAMMING PARADIGM

In this section we describe this new genetic programming paradigm in greater detail.

The Structures Undergoing Adaptation

The structures that undergo adaptation in the genetic programming paradigm are hierarchically structured computer programs. This is in contrast to the one-dimensional linear strings (whether of fixed or variable length) of characters (or other objects) used in the conventional genetic algorithm.

In order to be able to successfully manipulate and modify entire computer programs using operations patterned after genetic operations appearing in nature, we must work in a computer programming language that is unusually flexible. The LISP programming language (frequently used in artificial intelligence and symbolic processing applications) is especially well-suited to our needs here. LISP is especially suitable for complex compositions of functions of various types, handling hierarchies, recursion, logical functions, self-modifying computer programs, self-executing computer programs, iterations, and structures whose size and shape is dynamically determined (rather than predetermined in advance). The LISP programming language is especially appropriate when the structures to be manipulated are hierarchical structures. Moreover, *both* programs and data have the same form in LISP.

Thus, the structures that undergo adaptation in the genetic programming paradigm are Common LISP computer programs. LISP computer programs are called "symbolic expressions" (that is, S-expressions). Since both programs and data have the same form in LISP, it is easy to modify a computer program and then execute it.

In the LISP programming language, everything is expressed in terms of "functions" operating on some arguments. In LISP S-expressions, the function appears just inside an opening (left) parenthesis and is then followed by its arguments and a closing (right) parenthesis. Thus, for example, (+ 1 2) calls for the function of addition (+) to be applied to the arguments 1 and 2. In other words, the LISP S-expression (+ 1 2) is equivalent to "1+2" in ordinary mathematics and evaluates to 3. In LISP, any argument can itself be an S-expression. For example, (+ 1 (* 2 3)) calls for the addition function to be applied to the argument 1 and the argument (* 2 3). That is, the addition function is to be applied to 1 and the result of applying the multiplication function (*) to the arguments 2 and 3. The result is 7. The LISP programming language has "functions" which perform all of the operations found in other programming languages.

As a specific example, consider the well-known econometric "exchange equation" $P=\frac{MV}{Q}$, which relates the money supply M, price level P, gross national product Q, and the velocity of money V of an economy. In particular, suppose we are given the 120 quarterly values (from 1959:1 to 1988:4) of four econometric time series.

- GNP82 is annual rate for the United States Gross National Product in billions of 1982 dollars.

- GD is the Gross National Product Deflator (normalized to 1.0 for 1982).

- FYGM3 is the monthly interest rate yields of 3-month Treasury bills, averaged for each quarter.

- M2 is the monthly value of the seasonally adjusted money stock M2 in billions of dollars, averaged for each quarter.

In attempting to rediscover the "exchange equation" using genetic programming paradigm, we might use the function set F = {+, -, *, %, RLOG, EXP, GNP82, GD, FM2, FYGM3}. The first four functions in this function set are arithmetic operations. RLOG and EXP are mathematical functions. GNP82, GD, FM2, and FYGM3 are functions of time.

Note that the protected division operation % produces a result of one if division by zero is attempted. Note that the protected logarithm function RLOG is the logarithm of the absolute value and is equal to zero for an argument of zero. These definitions allow arbitrary compositions of the functions in the function set.

The actual long-term historic postwar value of the M2 velocity of money in the United States is relatively constant and is approximately 1.6527 (Hallman et. al. 1989, Humphrey 1989). Thus, a "correct" solution for the price level P is terms of M, V, and Q is the multiplicative (non-linear) relationship (1),

$$P = \frac{MV}{Q} \tag{1}$$

or, alternately (2).

$$GD(T) = \frac{(M2(T) * 1.6527)}{GNP82(T)} \qquad (2)$$

Thus, in LISP, one correct LISP S-expression for prices in terms of the "exchange equation" would be

```
(% (* FM2 1.6527) GNP82).
```

Any LISP S-expression can be depicted graphically as a rooted point-labeled tree in a plane whose internal points are labeled with functions, whose external points (leaves) are labeled with atoms, and whose root is labeled with the function appearing just inside the outermost left parenthesis. The tree corresponding to the LISP S-expression above for the "exchange equation" is shown in Fig. 1.

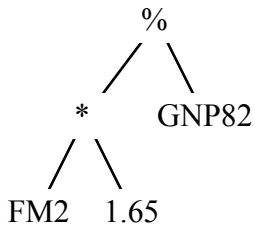

Fig. 1 The exchange equation represented parsimoniously as a tree.

In this graphical depiction, the 2 internal points of the tree are labeled with functions (% and *). The 3 external points (leaves) of the tree are labeled with atoms. The root of the tree is labeled with the function appearing just inside the outermost left parenthesis of the LISP S-expression (i.e.division %). Note that two lines emanate from the multiplication function * and the division function %. because they each take two arguments. Note also that no lines emanate from the atoms at the external points (leaves) of the tree.

The Environment

The environment is a set of cases which provides a basis for evaluating particular S-expressions. For example, for the "exchange equation", the environment is set of 120 cases listing, for each quarter between 1959:1 and 1988:4, the values of GNP82, FM2, and FYGM3 along with the associated value of GD.

The Fitness Function

Each individual in a population is assigned a fitness value as a result of its interaction with the environment. Fitness is the driving force of Darwinian natural selection and genetic algorithms.

The "raw fitness" of any LISP S-expression is the sum of the squares of the distances (taken over all the environmental cases) between the point in the solution space (which is real-valued here) returned by the individual S-expression for a given set of arguments and the correct point in the solution space. In particular, the raw fitness r(h,t) of an individual LISP S-expression h in the population of

size M at any generational time step t is (3),

$$r(i,t) = \sum_{j=1}^{N_e} |S(i,j) - C(j)|^2 \quad (3)$$

where V(h,j) is the value returned by S-expression h for environmental case j (of N_e environmental cases) and where S(j) is the correct value for environmental case j.

The closer this sum of distances is to zero, the better the S-expression.

Thus, the raw fitness of an individual LISP S-expression for the "exchange equation" problem is computed by accumulating, over each of the 120 values of time T from 1959:1 to 1988:4, the sum of the squares of the differences between the actual value of GD and whatever value the individual LISP S-expression produces for that time.

Each raw fitness value is then adjusted (scaled) to produce an adjusted fitness measure a(h,t). The "adjusted fitness" value is (4),

$$a(i,t) = \frac{1}{(1+r(i,t))} \quad (4)$$

where r(h,t) is the raw fitness for individual h at time t. Unlike raw fitness, the adjusted fitness is larger for better individuals in the population. Moreover, the adjusted fitness lies between 0 and 1.

Each such adjusted fitness value a(h,t) is then normalized. The "normalized fitness" value n(h,t) is (5).

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)} \quad (5)$$

The normalized fitness not only ranges between 0 and 1 and is larger for better individuals in the population, but the sum of the normalized fitness values is 1. Thus, normalized fitness is a probability value.

The Genetic Operations

The two primary genetic operations for modifying the structures undergoing adaptation are Darwinian fitness proportionate reproduction and crossover (recombination). They are described below.

The Fitness Proportionate Reproduction Operation. The operation of fitness proportionate reproduction for the genetic programming paradigm is the basic engine of Darwinian reproduction and survival of the fittest. It is an asexual operation in that it operates on only one parental S-expression. The result of this operation is one offspring S-expression. In this operation, if $s_l(t)$ is an individual in the population at generation t with fitness value $f(s_l(t))$, it will be copied into the next generation with probability (6).

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))} \qquad (6)$$

Note that the operation of fitness proportionate reproduction does not create anything new in the population. It increases or decreases the number of occurrences of individuals already in the population. To the extent that it increases the number of occurrences of more fit individuals and decreases the number of occurrences of less fit individuals, it improves the average fitness of the population (at the expense of the genetic diversity of the population).

The Crossover (Recombination) Operation. The crossover (recombination) operation for the genetic programming paradigm is a sexual operation that starts with two parental S-expressions. Typically the first parent is chosen from the population with a probability equal to its normalized fitness and the second parent is chosen from the population using a random probability distribution. The result of the crossover operation is two offspring S-expressions. Unlike fitness proportionate reproduction, the crossover operation creates new individuals in the populations.

Every LISP S-expression can be depicted graphically as a rooted point-labeled tree in a plane whose internal points are labeled with functions, whose external points (leaves) are labeled with atoms, and whose root is labeled with the function appearing just inside the outermost left parenthesis. The operation begins by randomly and independently selecting one point in each parent using a specified probability distribution (discussed below). Note that the number of points in the two parents typically are not equal. As will be seen, the crossover operation is well-defined for any two S-expressions. That is, for any two S-expressions and any two crossover points, the resulting offspring are always valid LISP S-expressions. Offspring contain some traits from each parent.

The "crossover fragment" for a particular parent is the rooted sub-tree whose root is the crossover point for that parent and where the sub-tree consists of the entire sub-tree lying below the crossover point (i.e. more distant from the root of the original tree). Viewed in terms of lists in LISP, the crossover fragment is the sub-list starting at the crossover point.

The first offspring is produced by deleting the crossover fragment of the first parent from the first parent and then impregnating the crossover fragment of the second parent at the crossover point of the first parent. In producing this first offspring the first parent acts as the base parent (the female parent) and the second parent acts as the impregnating parent (the male parent). The second offspring is produced in a symmetric manner.

Because entire sub-trees 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 parental LISP S-expression:

```
(% (+ 0.85 GNP82) GNP82)
```

The "%" function above is the division function defined so that division by zero delivers zero as its result. Now, consider the second parental S-expression below:

```
(- FM2 (* FM2 1.65))
```

These two LISP S-expressions can be depicted graphically as rooted, point-labeled trees with ordered branches.

The two parental LISP S-expressions are shown in Fig. 2.

Assume that the points of both trees are numbered in a depth-first way starting at the left. Suppose that the second point (out of 6 points of the first parent) is randomly selected as the crossover point for the first parent and that the third point (out of 6 points of the second parent) is randomly selected as the crossover point of the second parent. The crossover points are therefore the "+" in the first parent and the "*" in the second parent.

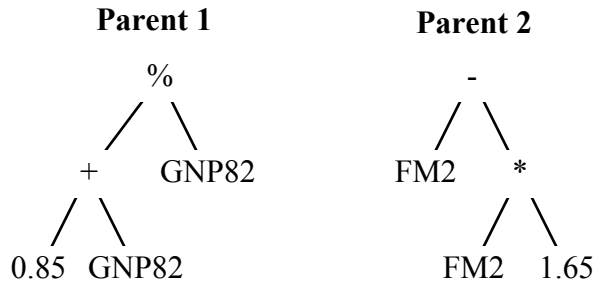

Fig. 2. Two parental LISP S-expressions for the crossover operation.

The two crossover fragments are two sub-trees shown in Fig. 3.

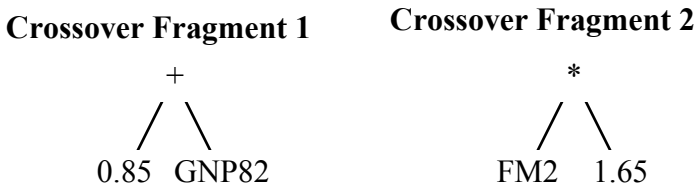

Fig. 3. The two crossover fragments.

The remainders are shown in Fig. 4.

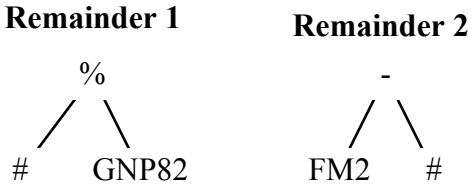

Fig. 4. The remaining LISP S-expressions. Each of the places from which the crossover fragments shown in Fig. 3 were removed are identified with a "#".

These two crossover fragments correspond to the bold, underlined sub-expressions (sub-lists) in the two parental LISP S-expressions shown above. The two offspring resulting from crossover are shown in Fig. 5.

Note that the first offspring above is a perfect solution for the exchange equation, namely

```
(% (* FM2 1.65) GNP82).
```

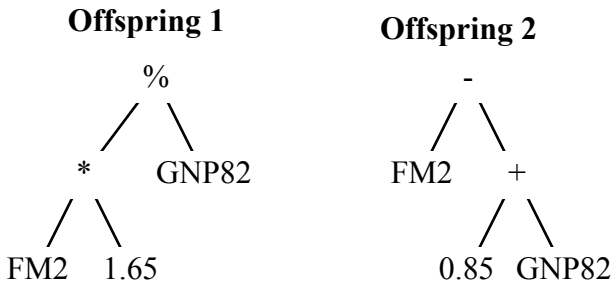

Fig. 5. The two offspring from the crossover operation.

The Method for Selecting Operations

In this paper, the population size was 500. Crossover was performed on 90% of the population. That is, if the population size is 500, then 175 pairs of individuals from each generation were selected (with reselection allowed) from the population with a probability equal to their normalized adjusted fitness. In addition, fitness proportionate reproduction was performed on 10% of the population on each generation. That is, 50 individuals from each generation were selected (with reselection allowed) from the population with a probability equal to their normalized adjusted fitness. Note that the parents remain in the population and can often repeatedly participate in other operations during the current generation. Several minor parameters are used to control the computer implementation of the algorithm. In this paper, a maximum depth of 15 was established for S-expressions. This limit prevented large amounts of computer time being expended on a few extremely large (and usually highly unfit) individual S-expressions. Of course, if we could execute all the individual LISP S-expressions in parallel (as nature does) in a manner such that the infeasibility of one individual in the population does not disproportionately jeopardize the resources needed by the population as a whole, we would not need this kind of limit.

REDISCOVERING THE "EXCHANGE EQUATION" FROM EMPIRICAL TIME SERIES DATA

An important problem area in virtually every area of science is finding the empirical relationship underlying observed values of the variables measuring a system (Langley et. al. 1987). In practice, the observed data may be noisy and there may be no known way to express the relationships involved in a precise way. The problem of discovering such empirical relationships from actual observed data is illustrated by the well-known econometric "exchange equation" $P=\frac{MV}{Q}$, which relates the price level P, money supply M, the velocity of money V, and the gross national product Q of an economy. Suppose that our goal is to find the relationship between quarterly values of the price level P and the three other elements of the equation. That is, our goal is to rediscover the multiplicative (non-linear) relationship (7),

$$GD = \frac{(M2 * 1.6527)}{GNP82} \qquad (7)$$

from the actual observed time series data given the 120 quarterly values (from 1959:1 to 1988:4) of the four econometric time series GNP82, GD, FYGM3, and M2.

The four time series were obtained from the CITIBASE data base of machine-readable econometric time series (Citibank 1989). The CITIBASE™ data was accessed by an Apple Macintosh II™ computer using software provided by VAR Econometrics Inc. (Doan 1989).

The sum of the squared errors over the entire 30-year period involving 120 quarters (1959:1 to 1988:4) between the actual gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4 was 0.077193. The R^2 value was 0.993320. A plot of the corresponding residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown in Fig. 6.

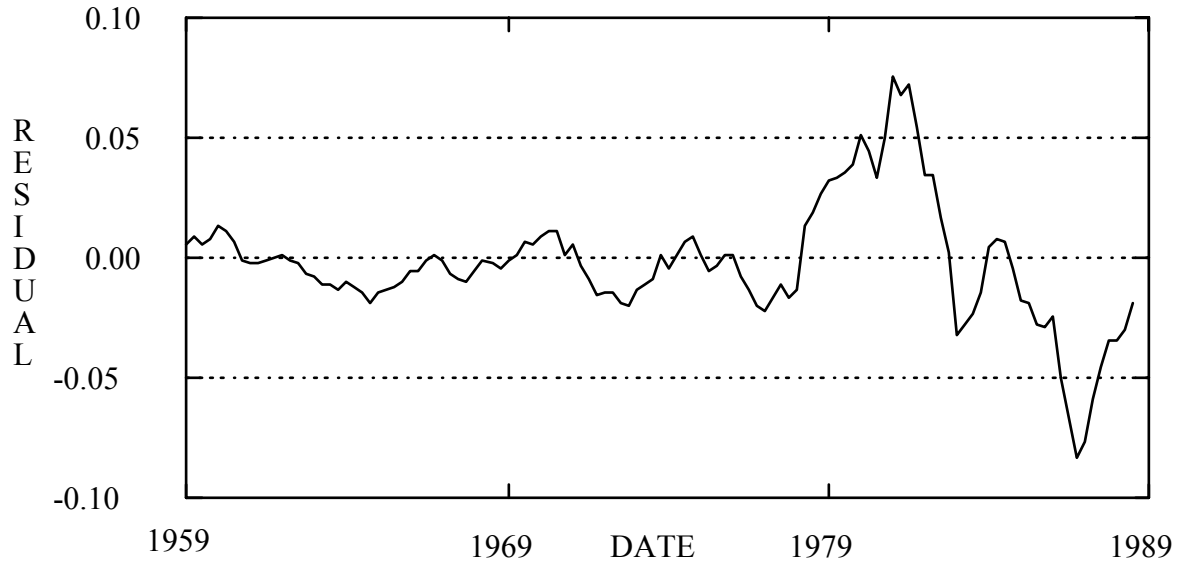

Fig. 6. The corresponding residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4.

Model Derived from First Two-Thirds of Data

We first divide the 30-year, 120-quarter period into a 20-year, 80-quarter "in-sample" period running from 1959:1 to 1978:4 and a 10-year, 40-quarter "out-of-sample" period running from 1979:1 to 1988:4.

The set of functions available for this problem is F = {+, -, *, %, EXP, RLOG}. The set of terminals available for this problem is T = {GNP82, FM2, FYGM3, R}, where "R" is the ephemeral random constant atom allowing various random floating point constants to be inserted at random as constant atoms amongst the initial random LISP S-expressions (See Koza 1990a for details). The terminals GNP82, FM2, and FYGM3 provide access to the values of the time series. In effect, these "terminals" are functions of the unstated, implicit time variable. A population size of 500 individuals was used.

Notice that we are not told *a priori* whether the functional relationship between the given observed data (the three independent variables) and the target function (the dependent variable GD) is linear,

multiplicative, polynomial, exponential, logarithmic, or otherwise. Notice also that we are not told that the addition, subtraction, exponential, and logarithmic functions as well as the time series for the 3-month Treasury bill rates (FYGM3) are irrelevant to the problem.

The initial random population (generation 0) was, predictably, highly unfit. Examples of randomly generated individuals that appeared in the initial generation (generation 0) for the "exchange equation" problem are (RLOG GNP82), (+ FYGM3 (EXP -0.92)), (RLOG (+ 0.27 (EXP 0.65))), etc. In one run of the genetic programming paradigm, the sum of squared errors between the single best S-expression in the population and the actual GD time series was 1.55. The value of R^2 was 0.49.

After the initial random population is created, each successive new generation in the population is created by applying the operations of fitness proportionate reproduction and genetic recombination (crossover).

In generation 1, the sum of the squared errors for the new best single individual in the population improved to 0.50.

In generation 3, the sum of the squared errors for the new best single individual in the population improved to 0.05. This is approximately a 31-to-1 improvement over the initial random generation. R^2 improved to 0.98. In addition, by generation 3, the best single individual in the population came within 1% of the actual GD time series for 44 of the 80 in-sample points.

In generation 6, the sum of the squared errors for the new best single individual in the population improved to 0.027. This is approximately a 2-to-1 improvement over generation 3. R^2 improved to 0.99.

In generation 7, the sum of the squared errors for the new best single individual in the population improved to 0.013. This is approximately 2-to-1 improvement over generation 6.

In generation 15, the sum of the squared errors for the new best single individual in the population improved to 0.011. This is an additional improvement over generation 7 and represents approximately a 141-to-1 improvement over generation 0. R^2 was 0.99.

A typical best single individual from a late generation of this process had a sum of squared errors of 0.009272 over the in-sample period and is shown below:

```
(% (+ (* (+ (* -0.402 -0.583)
      (% FM2(- GNP82 (- 0.126
      (+ (+ -0.83 0.832)
      (% (% GNP82 (* (- 0.005 GNP82)
      (% GNP82 GNP82)))
      0.47)))))) FM2) FM2) GNP82).
```

This individual is equivalent to (8).

$$GD = \frac{(M2 * 1.634)}{GNP82} \tag{8}$$

This individual can be graphically depicted as a rooted, point-labeled tree with ordered branches as shown in Fig. 7.

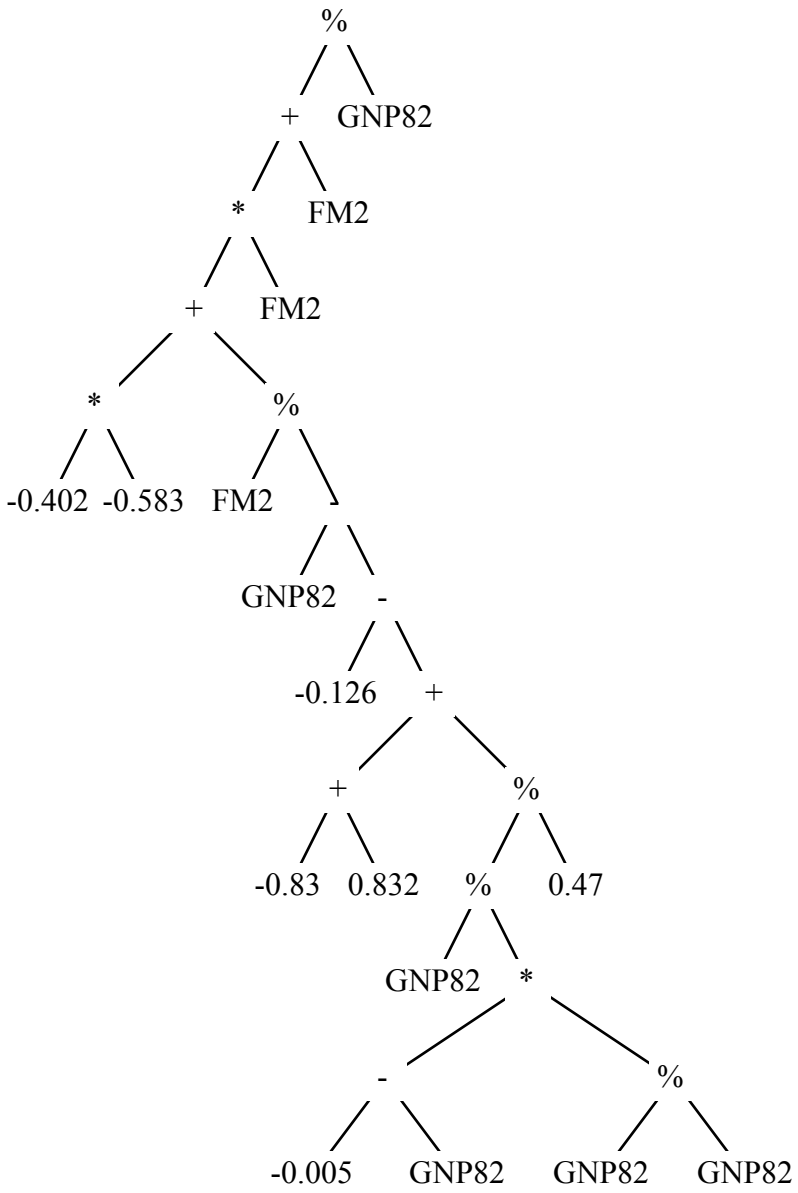

Fig. 7. A typical best individual depicted as a tree.

Table 1 shows the sum of the squared errors and R^2 for the entire 120-quarter period, the 80-quarter in-sample period, and the 40-quarter out-of-sample period.

Table 1. The sum of the squared errors and R^2 for the 120-quarter period, the 80-quarter in-sample period, and the 40-quarter out-of-sample period.

Data Range	*1- 120*	*1 - 80*	*81 - 120*
R^2	0.993480	0.997949	0.990614
Sum of Squared Error	0.075388	0.009272	0.066116

Figure 8 shows both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted

GD series calculated from the above genetically produced model for 1959:1 to 1988:4. The actual GD series is shown as line with dotted points. The fitted GD series calculated from the above model is simple line.

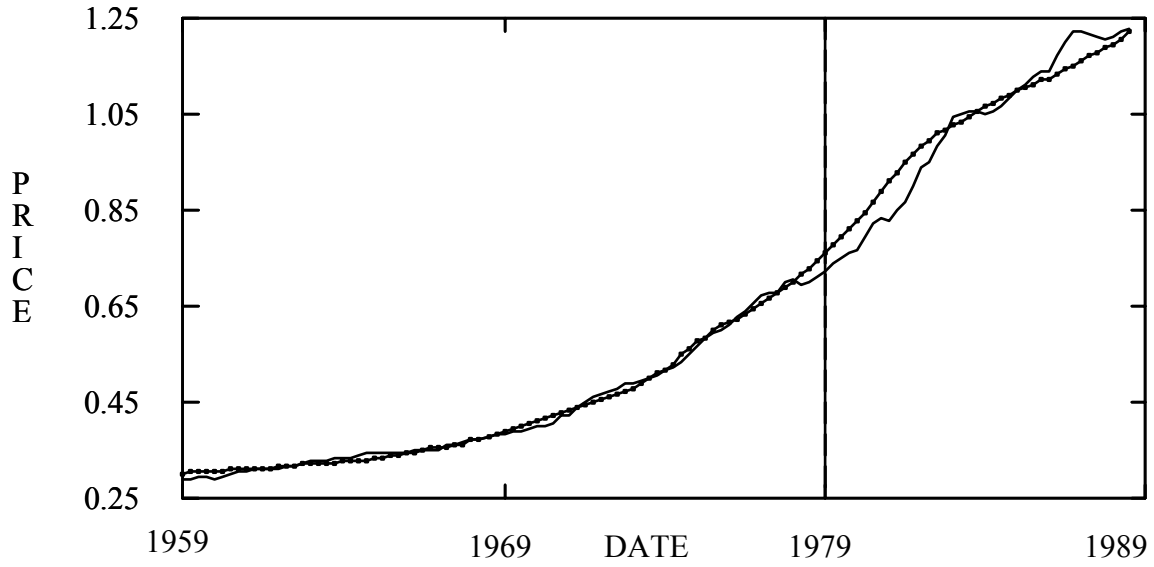

Fig. 8. The gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above genetically produced model for 1959:1 to 1988:4. The actual GD series is shown as line with dotted points. The fitted GD series calculated from the above model is simple line.

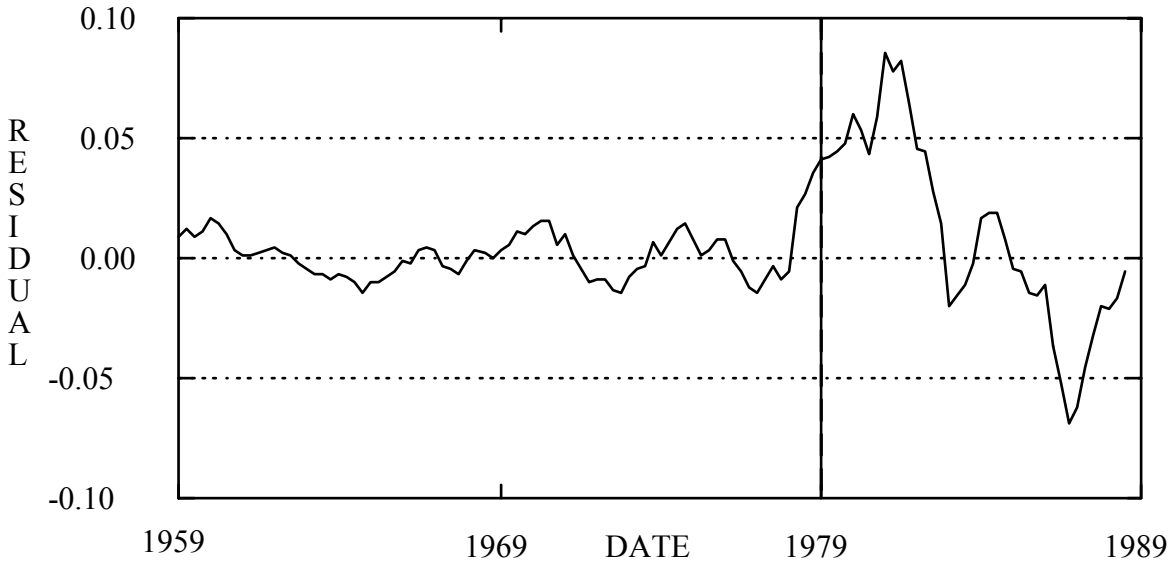

Fig. 9. A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4

A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown in Fig. 9.

Model Derived from Last Two-Thirds of Data

We now divide the 30-year, 120-quarter period into a 10-year, 40-quarter "out-of-sample" period running from 1959:1 to 1958:4 and a 20-year, 80-quarter "in-sample" period running from 1969:1 to 1988:4.

A typical best single individual from a late generation of this process had a sum of squared errors of 0.076247 over the in-sample period and is shown below:

```
(* 0.885 (* 0.885 (% (- FM2
          (- (- (* 0.885 FM2) FM2)
             FM2)) GNP82)))
```

This individual can be graphically depicted as a rooted, point-labeled tree with ordered branches as is shown in Fig. 10

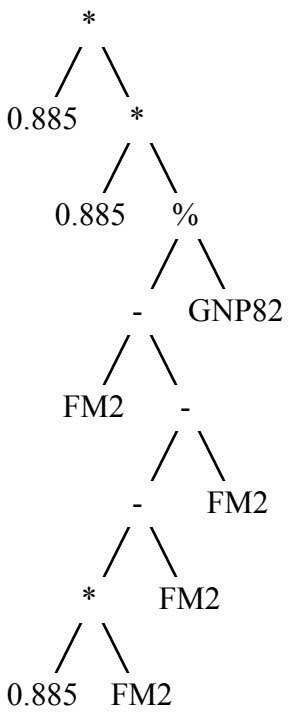

Fig. 10. The individual graphically depicted as a rooted, point-labeled tree with ordered branches

This individual is equivalent to (9).

$$GD = \frac{(M2 * 1.6565)}{GNP82} \tag{9}$$

Table 2 shows the sum of the squared errors and R^2 for the entire 120-quarter period, the 40-quarter out-of-sample period, and the 80-quarter in-sample period.

Table 2. The sum of the squared errors and R^2 for the 120-quarter period, the 40-quarter out-of-sample period, and the 80-quarter in-sample period.

Data Range	*1- 120*	*1 - 40*	*41 - 120*
R^2	0.993130	0.999136	0.990262
Sum of Squared Error	0.079473	0.003225	0.076247

Figure 11 shows both the actual gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above genetically produced model for 1959:1 to 1988:4. The actual GD series is shown as a line with dotted points. The fitted GD series calculated from the above model is simple line.

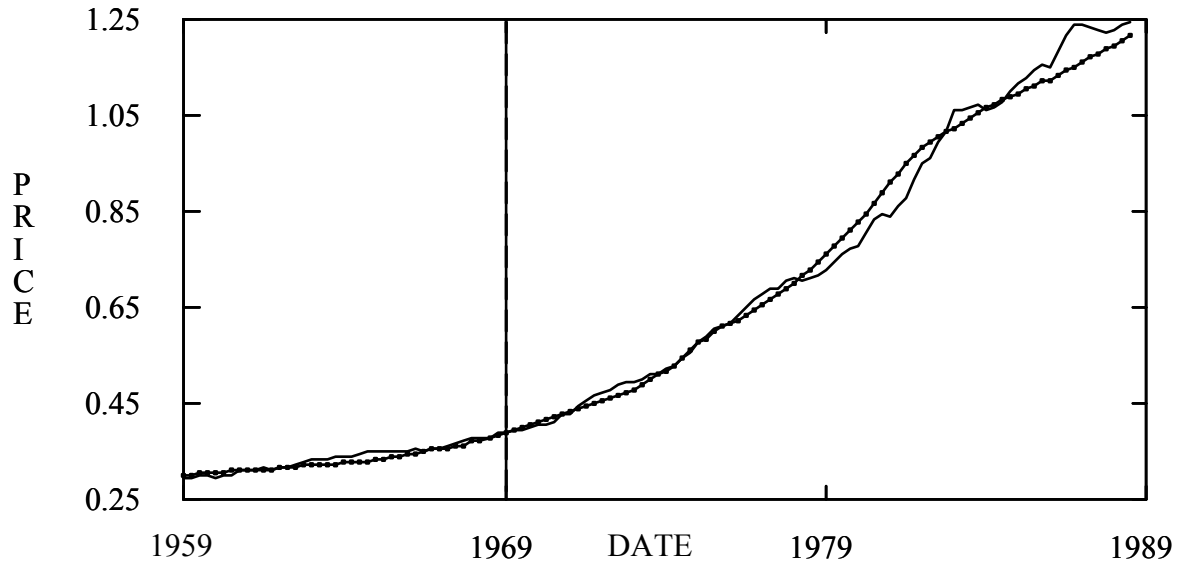

Fig. 11. Both the gross national product deflator GD from 1959:1 to 1988:4 and the fitted GD series calculated from the above model for 1959:1 to 1988:4. The actual GD series is shown as a line with dotted points. The fitted GD series calculated from the above model is simple line.

A plot of the residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4 is shown in Fig. 12.

CONCLUSION

We have shown how the newly developed genetic programming paradigm can be used to create an econometric model by rediscovering the well-known non-linear econometric "exchange equation" relating the price level, gross national product, money supply, and velocity of money in an economy.

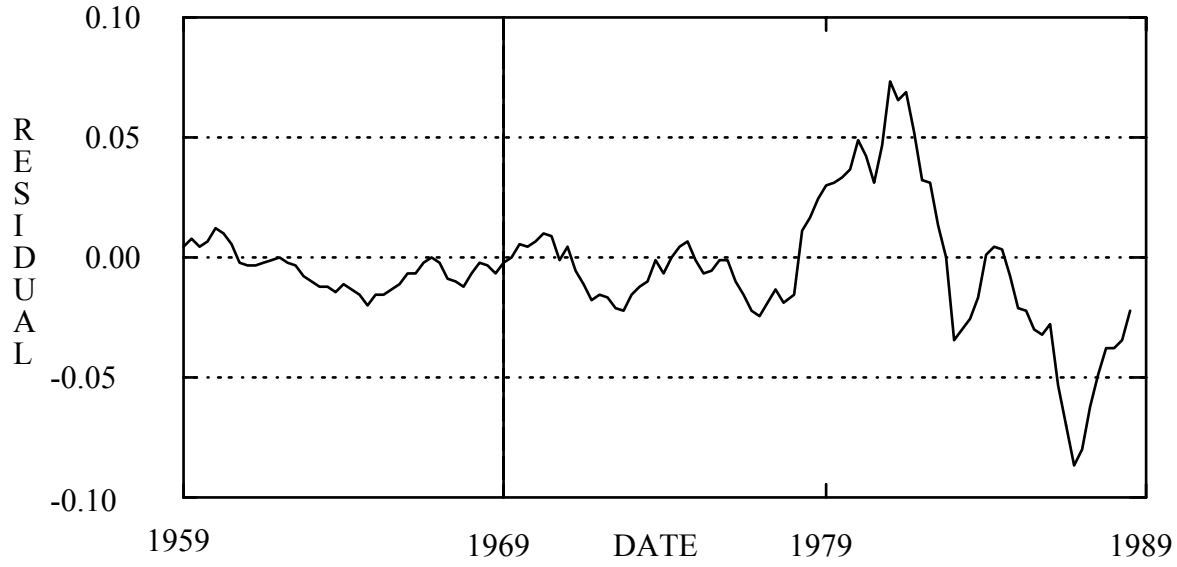

Fig. 12. The residuals from the fitted GD series calculated from the above model for 1959:1 to 1988:4.

ACKNOWLEDGMENTS

James P. Rice of the Knowledge Systems Laboratory at Stanford University made numerous contributions in connection with the computer programming of the above. Christopher Jones of Stanford University analyzed the results of the genetic algorithms and produced the graphs.

REFERENCES

Belew, R. and Booker, L. (eds) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc, San Mateo..

Citibank, N. A. (1989). CITIBASE: Citibank Economic Database (Machine Readable Magnetic Data File), 1946 - Present. Citibank N.A., New York.

Davis, L. (1987) (ed). *Genetic Algorithms and Simulated Annealing.* Pittman, London.

Davis, L. (1991). *Handbook of Genetic Algorithms* Van Nostrand Reinhold.

Doan, T. A. (1989). *User Manual for RATS - Regression Analysis of Time Series.* VAR Econometrics, Inc., Evanston.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading.

Hallman, J. J., Porter, R. D. and D. H. Small (1989). *M2 per Unit of Potential GNP as an Anchor for the Price Level.* Board of Governors of the Federal Reserve System. Staff Study 157, Washington,DC.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Holland, J. H. (1986) Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In *Machine Learning: An Artificial Intelligence Approach,* . (R. S. Michalski, J. G. Carbonell and T. M. Mitchell eds.), Vol II, pp. 593-623. Morgan Kaufman, Los Altos.

Holland, J. H. (1990). The global economy as an adaptive system. In *Santa Fe Institute Studies in the Sciences of Complexity: The Economy as an Evolving Complex System.*(P. W. Anderson, K. J. Arrow, and D. Pines eds.). Addison-Wesley, Redwood City.

Humphrey, T. M. (1989). Precursors of the P-star model. *Economic Review.* Federal Reserve Bank of Richmond. July-August 1989. 3-9.

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

Koza, J. R. (1990a). *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.

Koza, J. R. (1990b). A Genetic Approach to Econometric Modeling. Paper presented at Sixth World Congress of the Econometric Society, Barcelona, Spain.

Koza, J. R. (1990c). Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence. *Proceedings of the Second International Conference on Tools for AI.*

Koza, J. R. (1991a). Genetic Evolution and Co-Evolution of Computer Programs. In *Artificial Life II, SFI Studies in the Sciences of Complexity*. (D. Farmer, C. Langton, S. Rasmussen and C. Taylor, eds.), Vol XI. Addison-Wesley, Redwood City.

Koza, J. R. (1991b). Evolution and co-evolution of computer programs to control independent-acting agents. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.* (J-A. Meyer and S. W. Wilson, eds.), MIT Press, Cambridge.

Koza, J. R. (1991c). Concept Formation and Decision Tree Induction using the Genetic Programming Paradigm. In *Parallel Problem Solving from Nature.* (H-P. Schwefel and R. Maenner, eds.), Springer-Verlag, Berlin.

Koza, J. R. (1991d) *Genetic Programming*. (Forthcoming). MIT Press, Cambridge.

Koza, J. R. and M.A. Keane (1990a). Cart Centering and Broom Balancing by Genetically Breeding Populations of Control Strategy Programs. In *Proceedings of International Joint Conference on Neural Networks*, *Washington,* Vol I. Lawrence Erlbaum, Hillsdale.

Koza, J. R. and M. A. Keane (1990b). Genetic Breeding of Non-Linear Optimal Control Strategies for Broom Balancing. In *Proceedings of the Ninth International Conference on Analysis and Optimization of Systems*. Springer-Verlag, Berlin.

Langley, P., H. A. Simon, G. L. Bradshaw and J. M. Zytkow (1987). *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge.

Marimon, R., E. McGrattan and T. J. Sargent (1990). Money as a medium of exchange in an economy with artificially intelligent agents. *Journal of Economic Dynamics and Control.* 14 329-

373.

Schaffer , J. D. (ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers Inc, San Mateo.