A Hierarchical Genetic Algorithm for System Identification and Curve Fitting with a Supercomputer Implementation

Mehmet Gulsen and Alice E. Smith[1]
Department of Industrial Engineering
1031 Benedum Hall
University of Pittsburgh
Pittsburgh, PA 15261 USA
412-624-5045
412-624-9831 (fax)
aesmith@engrng.pitt.edu

To Appear in *IMA Volumes in Mathematics and its Applications - Issue on Evolutionary Computation* (Springer-Verlag)

Editors: Lawrence Davis, Kenneth DeJong, Michael Vose and Darrell Whitley

1998

---

[1] Corresponding author.

A Hierarchical Genetic Algorithm for System Identification and Curve Fitting with a Supercomputer Implementation

Abstract

This paper describes a hierarchical genetic algorithm (GA) framework for identifying closed form functions for multi-variate data sets. The hierarchy begins with an upper GA that searches for appropriate functional forms given a user defined set of primitives and the candidate independent variables. Each functional form is encoded as a tree structure, where variables, coefficients and functional primitives are linked. The functional forms are sent to the second part of the hierarchy, the lower GA, that optimizes the coefficients of the function according to the data set and the chosen error metric. To avoid undue complication of the functional form identified by the upper GA, a penalty function is used in the calculation of fitness. Because of the computational effort required for this sequential optimization of each candidate function, the system has been implemented on a Cray supercomputer. The GA code was vectorized for parallel processing of 128 array elements, which greatly speeded the calculation of fitness. The system is demonstrated on five data sets from the literature. It is shown that this hierarchical GA framework identifies functions which file the data extremely well, are reasonable in functional form, and interpolate and extrapolate well.

1. INTRODUCTION

In the broadest definition, system identification and curve fitting refer to the process of selecting an analytical expression that represents the relationship between input and output variables of a data set consisting of iid observations of independent variables (x_1, x_2, ... , x_n) and one dependent variable (y). The process involves three main decision steps: (1) selecting a pool of independent variables which are potentially related to the output, (2) selecting an analytical expression, usually a closed form function, and (3) optimizing the parameters of the selected analytical expression according to an error metric (e.g., minimization of sum of squared errors).

In most approaches, closed form functions are used to express the relationship among the variables, although the newer method of neural networks substitutes an empirical "black box" for a function. Fitting closed form equations to data is useful for analysis and interpretation of the observed quantities. It helps in judging the strength of the relationship between the independent (predictor) variables and the dependent (response) variables, and enables prediction of dependent variables for new values of independent variables. Although curve fitting problems were first introduced almost three centuries ago [25], there is still no single methodology that can be applied universally. This is partially due to the diversity of the problem areas, and particularly due to the computational limitations of the various approaches that deal with only subsets of this broad scope. Linear regression, spline fitting and autoregressive analysis are all solution methodologies to system identification and curve fitting problems.

2. HIERARCHICAL GENETIC APPROACH

In the approach of this paper, the primary objective is to identify an appropriate functional form and then estimate the values of the coefficients associated with the functional terms. In fact, the entire task is a continuous optimization process in which a selected error metric is minimized. A hierarchical approach was used to accomplish this sequentially as shown in Figure 1. The upper genetic algorithm (GA) selects candidate functional forms using a pool of the independent variables and user defined functional primitives. The functions are sent to the lower (subordinate) GA which selects the coefficients of the terms of each candidate function. This selection is an optimization task using the data set and difference between the actual and fitted *y* values.

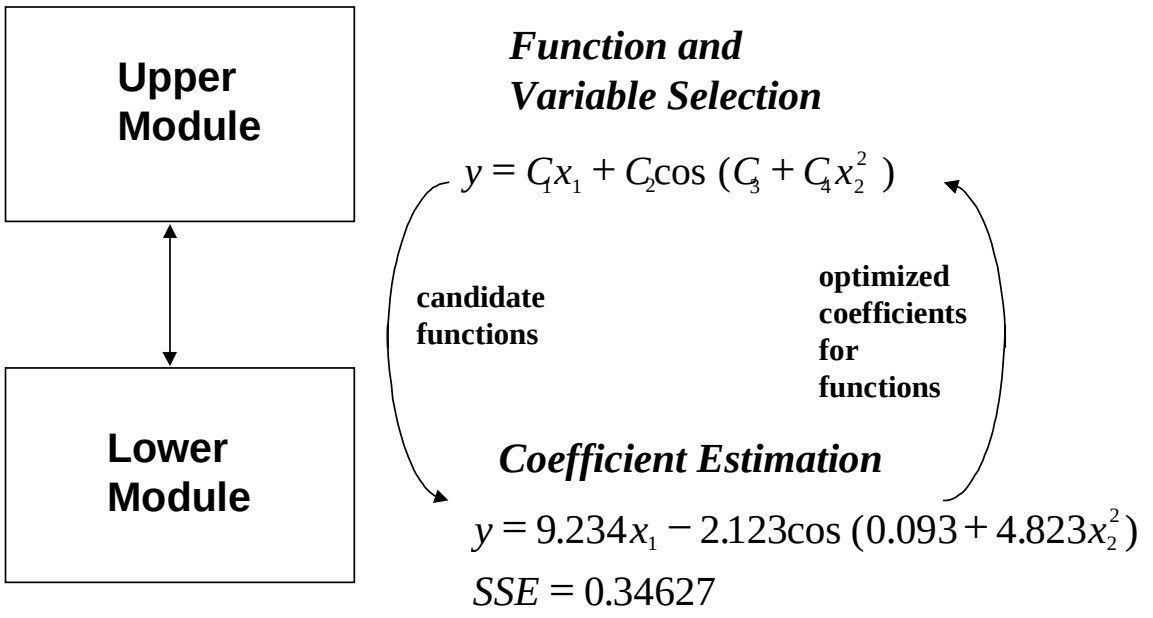

Figure 1 Hierarchical GA Framework

GA has been applied to system identification and curve fitting by several researchers. The relevant work on these can be categorized into two groups. The first category includes direct adaptation of the classical GA approach to various curve fitting techniques [17, 23]. In these works GA replaces traditional techniques as a function optimization tool. The second category includes more comprehensive approaches where not only parameters of the model, but the model itself, are search dimensions. One pioneering work on this area is Koza's adaptation of Genetic Programming (GP) to symbolic regression [18]. The idea of genetically evolving programs was first implemented by [8, 12]. However, GP has been largely developed by Koza who has done the most extensive study in this field [18, 19]. More recently, Angeline used the basic GP approach enhanced with adaptive crossover operators to select functions for a time series problem [1].

3. COEFFICIENT ESTIMATION (LOWER GA)

The approach consists of two independently developed modules which work in a single framework. The lower GA works as a function call from the upper one to measure how well the selected model fits the actual data. In the lower GA, the objective is to select functional term coefficients which minimize the total error over the set of data points being considered. A real value encoding is used. The next sections describe population generation, evolution, evaluation, selection, culling and parameter selection.

3.1 The Initial Population

Using the example of the family of curves given by equation 1, there are two variables, x_1 and x_2, and four coefficients. Therefore each solution (population member) has four real numbers, C_1, C_2, C_3, C_4.

$$y = C_1 + C_2 x_1^3 + C_3 \cos x_2 + C_4 x_1 x_2 \qquad (1)$$

The initial estimates for each coefficient are generated randomly over a pre-specified range defined for each coefficient. The specified range need not include the true value of the coefficient (though this is beneficial). The uniform distribution is used to generate initial coefficient values for all test problems studied in this paper, although this is not a requirement either (see [15] for studies on diverse ranges and other distributions).

3.2 Evolution Mechanism

A fitness (objective function) value which measures how well the proposed curve fits the actual data is calculated for each member of the population. Squared error is used here (see [15] for

studies on absolute error and maximum error). Minimum squared error over all data points results in maximum fitness.

Parents are selected uniformly from the best half of the current population. This is a stochastic form of truncation selection originally used deterministically in evolution strategies for elitist selection (see [2] for a good discussion of selection in evolution strategies). Mühlenbein and Schlierkamp-Voosen later used the stochastic version of truncation selection in their Breeder Genetic Algorithm [20]. The values of the offspring's coefficients are determined by calculating the arithmetic mean of the corresponding coefficients of two parents. This type of crossover is known as extended intermediate recombination with $\alpha = 0.5$ [20].

For mutation, first the range for each coefficient is calculated by determining the maximum and minimum values for that particular coefficient in the current population. Then the perturbation value is obtained by multiplying the range with a factor. This factor is a random variable which is uniformly distributed between $\pm k$ coefficient range. The value of k is set to 2 in all test problems in this paper. However, the range among the population generally becomes smaller and smaller during evolution. In order to prevent this value from becoming too small, and thus insignificant, a perturbation limit constant, Δ, is set. This type of mutation scheme is related to that used in Mühlenbein and Schlierkamp-Voosen's Breeder Genetic Algorithm [20]. However their range was static where the range in this approach adapts according to the extreme values found in the current population with a lower bound of Δ. Mutation is applied to M members of the population, which are selected uniformly. After mutation and crossover, the newly generated mutants and offspring are added to the population and the lowest ranked members are culled, maintaining the original population size, ps.

Cycle length, *cl*, is used as a basis for terminating the search process. It refers to the number of generations without any improvement in the fitness value prior to termination. If the algorithm fails to create offspring and mutants with better fitness values during this fixed span of generations, the search process is automatically terminated.

3.3 Parameter Selection

The lower GA was examined at various population parameters, namely population size *ps*, number of mutants *M*, number of offspring *O*, perturbation limit constant Δ and cycle length *cl*, as shown in Table 1. The experimental design covered a wide range of values and a full factorial experiment was performed.

Table 1 Evolution Parameter Levels for the Lower GA

Level	***ps***	***O***	***M***	Δ	***cl***
Low	30	10	5	10^{-6}	10
				10^{-5}	50
					100
Medium	50	20	10	10^{-4}	500
				10^{-3}	1000
					2000
High	100	50	20	10^{-2}	5000

Based on experimental test results, $ps = 30$, $O = 10$, $M = 10$ $\Delta = 10^{-5}$ and $cl = 100$ were selected for the lower GA. The strategy in selecting Δ and cycle length is based on selecting the maximum Δ value and minimum cycle length provided that the algorithm satisfies convergence criteria for all cases. Increasing Δ or decreasing *cl* reduces computational effort, but it may also deteriorate the accuracy of the final results. An opposite strategy, decreasing Δ or increasing *cl* improves accuracy, but it will take the algorithm more effort to converge to the final results.

3.4 Test Problems

Since the lower GA module was initially developed independently from the framework, it was tested on three test problems where the correct functional form is provided. Two of these three test problems were taken from previous GA studies on curve fitting.

The first test problem is to estimate the value of two coefficients (C_1 and C_2) of the straight line given in equation 2. This was studied by [17] with their GA approach.

$$y = C_1 x + C_2 \tag{2}$$

The second test problem, which is given in equation 3, has 10 independent variables and 9 coefficients to be estimated. This was studied for spline fitting using a GA by [23] and, before that, for conventional spline fitting by [11]. A set of 200 randomly selected data points was used, and the problem was designed so that the response depends only on the first five variables. The remaining variables, x_6 through x_{10}, have no effect on the response function, and they are included in the data set as misleading noise. Therefore the optimal values of C_5 through C_9 should be zero.

$$y = C_1 \sin(\pi x_1 x_2) + C_2 (x_3 - 0.5)^2 + C_3 x_4 + C_4 x_5 + \sum_{n=6}^{10} C_{n-1} x_n \tag{3}$$

The third test problem is a general second level equation of three variables. The function, which is given in equation 4, has 10 coefficients to be estimated, and it is larger and more complex than any that can be found in the previous literature on using genetic algorithms for coefficient estimation.

$$y = C_1 + C_2 x_1 + C_3 x_2 + C_4 x_3 + C_5 x_1^2 + C_6 x_2^2 + C_7 x_3^2 + C_8 x_1 x_2 + C_9 x_1 x_3 + C_{10} x_2 x_3 \tag{4}$$

The performance of the lower GA was tested with respect to data set size, population distribution and interval for sampling, error metric, initial coefficient range and random seeds. Robust performance was observed for all of the above parameters and performance exceeded previous approaches in terms of minimization of error (for details see [15]). The lower GA is highly flexible, and it can be adopted to different problem instances and parameters with very minor modifications in the algorithm. The coefficient optimization module is used as a subroutine to the function identification module described in the next section.

4. FUNCTIONAL FORM SELECTION (UPPER GA)

Although the desire to develop a heuristic method to select the functional form was expressed in the literature as early as 1950s, it was suggested as a possible extension of variable selection in stepwise regression [7, 9, 28]. In the approach of this paper, a very liberal selection procedure is used so that the search domain is open to all possible functional forms, but can also be restricted by the user.

4.1 Encoding

In terms of GA parlance, each member of the population set is a closed form function. Since the elements of the population are not simple numeric values, a binary array representation is not applicable. Instead, a tree structure is used to represent each closed form function. As an example, the tree structure for the candidate solution given below (equation 5) is illustrated in figure 2.

$$y = 3.2 + 7.5x_1^3 + 1.6\cos(-4.1x_2) + 4.6x_1x_2 \qquad (5)$$

The nodes in a tree structure can be categorized into two groups according to the number of outgoing branches: the "terminal node" is the last node in each branch without any outgoing branches. The nodes between the first and terminal nodes are called "intermediate nodes", and they have at least one outgoing branch. Each intermediate node has an operand (e.g., sin, cos, exp, ln, +, ×) and the terminal node carries a variable or a constant. The numbers beside nodes in figure 2 are real valued coefficients (obtained from the lower GA).

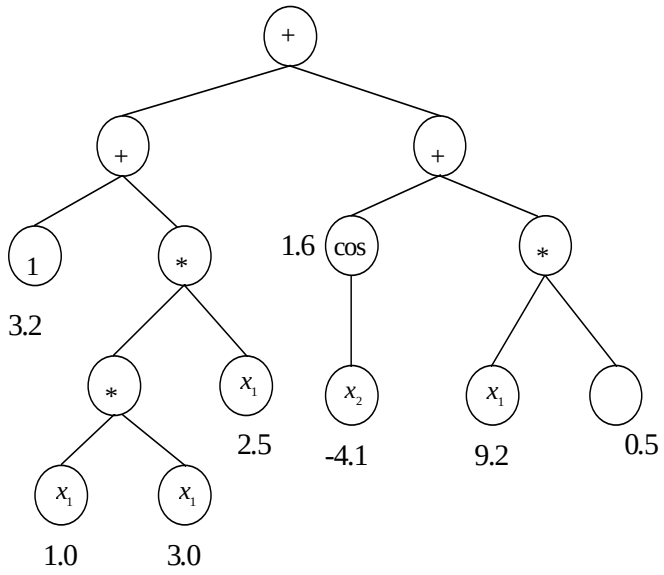

Figure 2 Tree Structure Representation of Equation 5

Each tree structure has a starting node where branching is initiated. The number of branches originating from a node is determined probabilistically, restricted to 0, 1 or 2. The only exception is the first node where there must be at least one branch in order to initiate the generation of the tree structure. The branches connecting the nodes also determine the "parent-offspring relation" between the nodes. For a branch coming out of node *a* and going into node *b*, node *a* is named the parent of node *b*. In a similar manner, node *b* is called a child of node *a*. Each parent node can have one or two child nodes. It is also possible that the parent may not

have a child. If a node has two children, they are respectively called the left-child and the right-child. However, if a node has only one child, it is still called the left-child.

For computational reasons, such as the fitness calculation, the nodes are indexed. The indexing process starts from the first node, which is indexed as 1. At this point the number of children are also calculated for the node. Then the process moves, first to the left-child of the current node (now the child becomes a parent), and indexes it as 2. This process continues until a node is reached with no left-child. In such a situation the process backups to the parent node and looks for a right-child. If parent has a right-child which is not indexed yet, then it is indexed. Otherwise, the procedure climbs up to next parent (or grandparent). This recursive process, which is analogous to a "depth first search" algorithm, stops when all the nodes are indexed. Figure 3 illustrates the tree structure of figure 2 which is indexed according to this procedure.

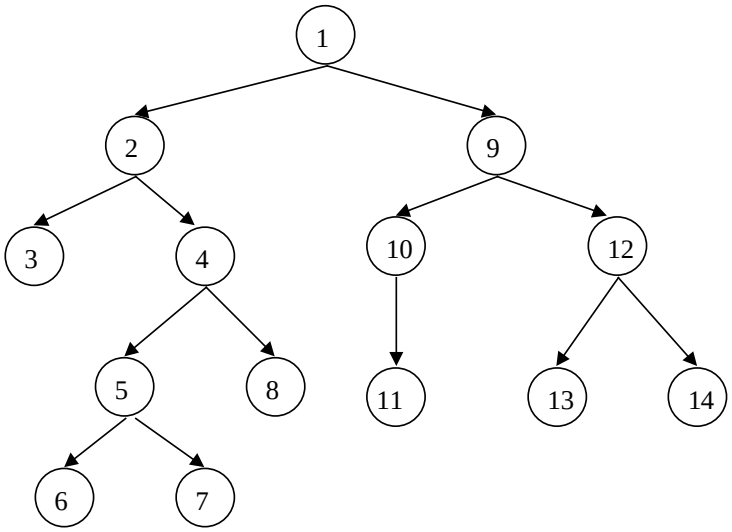

Figure 3 Indexing of a Tree Structure

The number of branches from a node will be Pr(B_1, B_2, B_3) where B_1, B_2 and B_3 represent the probability of having 0, 1 or 2 offspring nodes respectively. Special attention should be paid when assigning values for B_1, B_2 and B_3. For example if B_1, B_2 and B_3 are selected as 0.2, 0.4 and

0.2, each node, on the average, will have $0.2 \times 0 + 0.4 \times 1 + 0.2 \times 2 = 1.2$ offspring nodes. With such a number for the mean number of offspring nodes, it is possible to have tree structures with less than 10 or 15 nodes. However, it is also possible that tree structures which have tens or hundreds of nodes will be created. To remedy this, B_1, B_2 could be increased and B_3 decreased. However, this will create deep structures which will reduce the efficiency of representing the functional forms. As a remedy, the values of B_1, B_2 and B_3 change during the course of branching. At the early stages, there is a higher probability for double branching which diminishes gradually as the number of nodes increase. This is done by choosing a factor, δ, that is equal to 0.04 × *index number*. The δ value is added to B_1, and it is subtracted from B_3 at each node. As the number of nodes becomes larger, the probability of double branching decreases with respect to the probability of no branching.

One dimensional arrays were used to store the tree structures. The following information is stored for each node: (1) index number, (2) number of offspring nodes, (3) index number of left child node, (4) index number of right child node, (5) index number of parent node, (6) type of functional primitive assigned to the node, and (7) whether the node carries a coefficient or not.

4.2 Functional Primitives

The functional primitives are categorized into three hierarchical groups. The lowest level includes the independent variables of the data set and the constant 1. These are assigned only to terminal nodes and all carry a coefficient. If the variable *x* is assigned to a node and has coefficient value of 4.3, then it corresponds to 4.3*x*. Similarly, a node with constant 1 and coefficient -2.3 corresponds to -2.3×1 = -2.3. The second and third groups include unary and binary operators respectively. While operators such as sin, cos, ln, exp, are categorized as second group operators,

multiplication "×", division "÷", and addition "+" operators form the third group (table 2). Functional primitives can be customized by the user according to estimated characteristics of closed form representation and the desired restrictiveness of the search.

The assignment of primitives to the nodes is carried out in the following manner: the first group elements x_n and the constant 1 are assigned to terminal nodes where there is no outgoing branch. The unary operators are assigned to nodes with a single outgoing branch and the binary operators are assigned to nodes with two outgoing branches. Each of the first and second group operators is proceeded by a coefficient. The third group operators do not have a coefficient preceding them, because they simply combine two sub-functions which already have coefficients.

Table 2 Node Primitives

Levels	**Primitives***
First Group	$x_1,...,x_n$, 1
Second Group (unary)	sin, cos, exp, ln
Third Group (binary)	+, ×, ÷

**n* is the number of independent variables in the data set

4.3 Initial Population and Fitness

The initial population is randomly created in three stages. In the first stage, the tree structure is developed for each member of the population. At the second stage, appropriate primitives are assigned to the nodes in a random manner. After the functional form representation is obtained for the initial population, the fitness value of each member is calculated. The coefficient estimation process is done by the subordinate GA program, which is used as a function call in the main program. Two metrics are used for fitness: the first one is the raw fitness, which is the sum of squared errors. The second fitness value is developed to penalize complex functional

representations in order to prevent overfitting. This secondary fitness value, called "penalized fitness", is calculated by worsening the raw fitness in proportion to the complexity of the closed form function.

The complexity of a closed form function is determined by the number of nodes it has. However complexity is not a absolute concept and the user needs to judge the level of complexity for each data set. The objective is to balance the bias and variance in a closed form representation, where bias results from smoothing out the relationship and variance indicates dependence of the functional form on the exact data sample used [14]. Ideally, both bias and variance will be small. A penalty function with user tunable parameters is used to balance bias and variance of the functional form in the GA approach.

The penalty is determined by dividing the number of nodes in the tree structure by a threshold, T, and then taking the $(P)^{th}$ power of the quotient (equation 6), where $0<P<1$. T corresponds to the minimum number of nodes in an unpenalized tree structure. In this paper T = 5 was used, thus every tree structure was subject to some level of penalty if there were more than 5 nodes. Conservatively, a small value of T can be used safely while letting P impose the severity level of the penalty. A sample calculation for penalized fitness is presented in table 3.

$$\text{Penalized Fitness} = \text{Raw Fitness} \times \left(\frac{\text{Number of Nodes}}{T} \right)^{P} \qquad (6)$$

Table 3 Penalized Fitness

Raw Fitness	9.78
Number of Nodes	12
T	5
P	0.4
Complexity Coefficient	$\left(\frac{12}{5}\right)^{0.4} = 1.4193$
Penalized Fitness	$1.4193 \times 9.78 = 13.88$

4.4 Evolution Mechanism

The evolution mechanism is carried out through crossover and mutation. In each generation, O offspring and M mutants are generated. Selection for crossover and mutation, the culling process and algorithm termination are done as in the lower GA.

Crossover is carried out by swapping sub-branches of two members at randomly selected nodes (figures 4 and 5). Each crossover process produces two offspring. Mutation is performed by replacing a sub-branch (selected at any node randomly) with a new randomly generated one. One solution could produce more than one mutant provided that branching is performed at different nodes. The new branch used for replacement is a full function (like a member of the initial population) but smaller in size (figures 6 and 7).

Before: **Parent 1** **Parent 2**

$y = C_1 + C_2 x_1^3 + C_3 \cos(C_4 x_2) + C_5 x_1 x_2$ $\quad y = C_1 \sin(\frac{C_2 \ln(C_3 x_1)}{C_4 x_2} + C_5)$

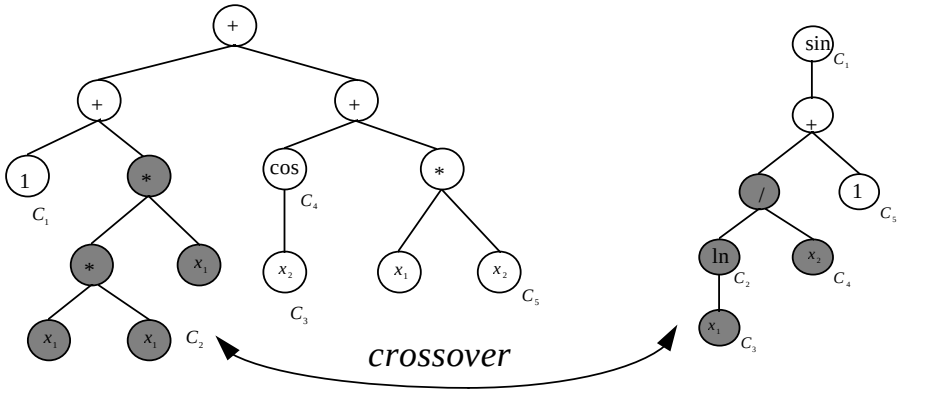

Figure 4 Two Functions Before Crossover

After: **Offspring 1** **Offspring 2**

$y = C_1 + \frac{C_2 \ln(C_3 x_1)}{C_4 x_2} + C_5 \cos(C_6 x_2) + C_7 x_1 x_2$ $\quad y = C_1 \sin(C_2 x_1^3 + C_3)$

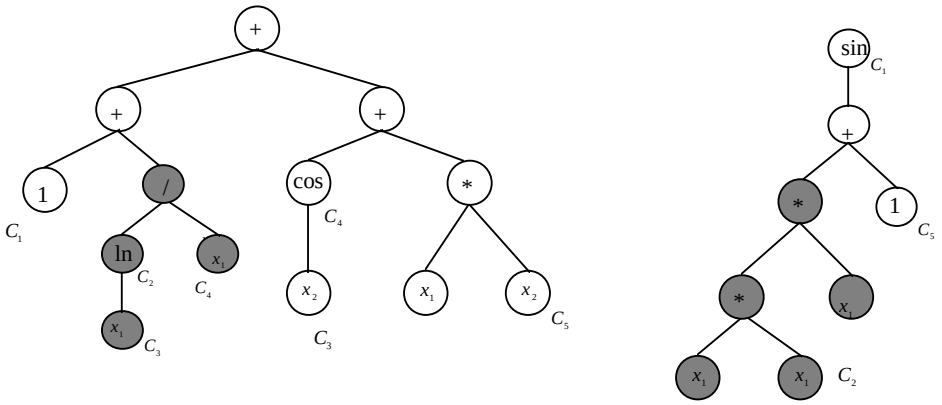

Figure 5 Two Functions After Crossover

Before: **Parent 1**

$$y = C_1 + C_2 x_1^3 + C_3 \cos(C_4 x_2) + C_5 x_1 x_2$$

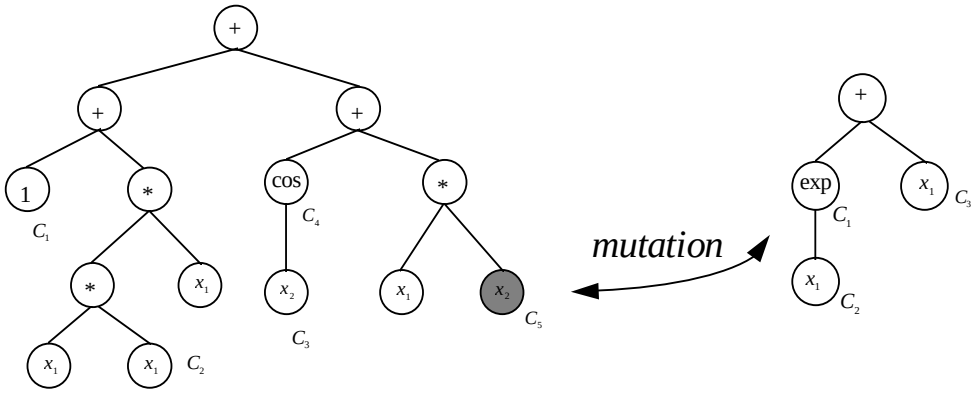

Figure 6 Function Before Mutation

After: **Mutant**

$$y = C_1 + C_2 x_1^3 + C_3 \cos(C_4 x_2) + C_5 x_1 \exp(C_6 x_1) + C_7 x_1^2$$

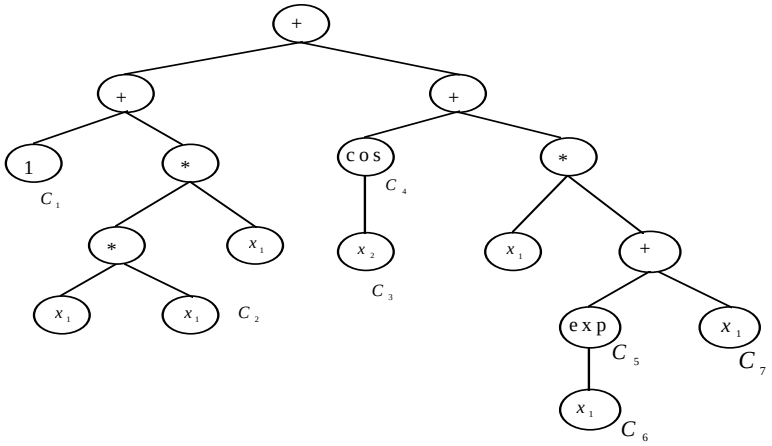

Figure 7 Function After Mutation

4.5 Parameter Setting

The experiments on the upper GA were performed using the selected parameters of the lower GA held constant and a cycle length of 100 generations for the upper GA. Since the algorithm converged to sufficiently accurate results for all population parameter combinations tested (Table

4), the population parameters were selected only on the basis of computational efficiency (total number of fitness calculations performed during the search process). Based on this criterion, *ps* = 30, *O* = 10, and *M* = 20 were selected for the upper GA. The penalty severity factor *P* was harder to choose. A low value enables better fit as measured by the sum of the squared errors (SSE) but encourages more complex functions. A high value has the opposite effect. In the test problems, small data sets were run with *P* = 0.40 and larger data sets were run at both *P* = 0.05 and *P* = 0.40. However, the user will probably want to run the GA at several values of *P* and compare the resulting error and functional forms.

Table 4 Evolution Parameter Levels for the Lower GA

Level	***ps***	***O***	***M***	***P***
Low	30	10	20	0.05
				0.10
Medium	50	20	40	0.20
				0.40
High	100	50	60	0.80

5. COMPUTATIONAL CONSIDERATIONS

The framework includes two sequential search algorithms (the upper and lower GAs). For each closed form representation, the coefficients are determined by running a secondary search routine. In a similar manner, for each set of coefficient values, an error metric is calculated over the data points. The computational requirement expands exponentially from top to the bottom (figure 8).

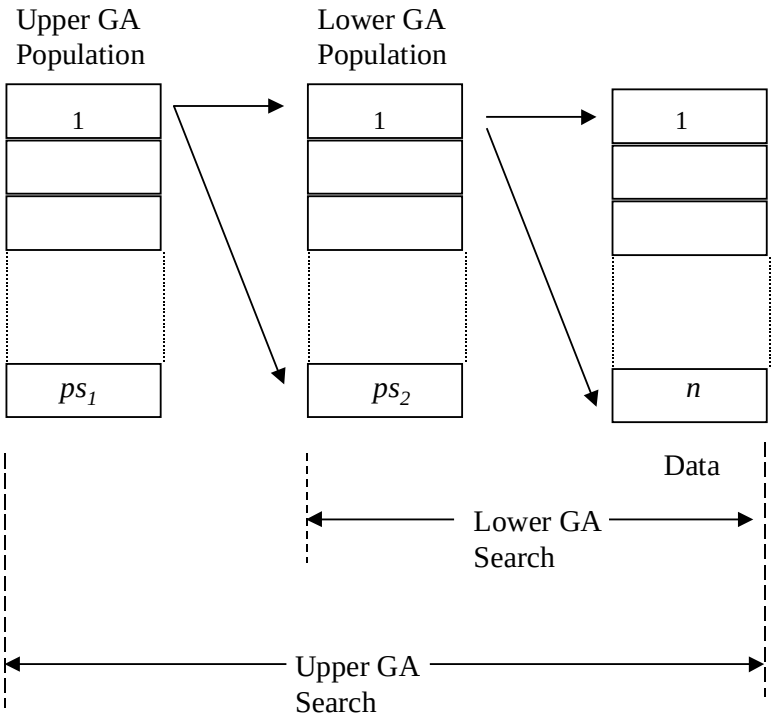

Figure 8 Problem Layout

Because of the exploding computational characteristic of the problem, maximum attention was paid to increase the efficiency of the code. First, early detection of pathological behavior at the higher levels was done. For example, anticipated floating point errors, ln, division or exp terms in sequence with themselves, and three or more unary terms in sequence all terminate the GA for that solution.

The search algorithm was coded on a C/Unix platform. The program includes five imbedded loops as shown in figure 9: a "do-while and for" loop combination for each search routine and an innermost "for" loop for fitness calculations. In both search routines, the "do-while" loops control the search process. The next level "for" loops control execution of the code for a set of candidate solutions generated within each "do-while" loop. The initial four imbedded

loops are highly "fragile" with several control mechanisms which may skip loop execution for some cases or even break the loop.

```
do {
    create O crossovers and M mutants
        for(i=0; i<O+M)
                do {
                        create O1 crossovers and M1 mutants
                            for(i=0; i<O2+M2)
                                    for(i=0; i<ndata)
                                        fitness calculation
                                } while (lower GA search criteria is not satisfied);
        } while (upper GA search criteria is not satisfied);
```

Figure 9 Pseudo-code for Search Algorithm.

Test problems 1, 2 and 5 were done on Sun/SPARC™ workstations. For test problems 3 and 4, the program was run on a Cray C90™ supercomputer. In order to use the "vectorization" utility, the innermost loop, where the error metric (fitness) is calculated over the data set, was reorganized to eliminate all data dependencies. The vector size was 128, which means 128 operations when calculating fitness can be done in parallel. With a fully vectorized innermost loop the program's flop rate ranged between 170 Mflops to 350 Mflops, depending on the data set size.

6. EMPIRICAL DATA SETS

In order to evaluate the effectiveness of the GA framework, five benchmark problems from the literature were studied. The size of the test problems ranges from single variable/50 observations to 13 variables/1000 observations. Each of the five test problems addresses a specific issue in the curve fitting field. The first two problems are classic examples from the nonlinear regression literature. The third test problem is sunspot data, which is one of the benchmark problems in time series analysis. The fourth test problem includes 13 variables and approximately 1000

observations. This data set was originally used for neural network modeling of a ceramic slip casting process. The last problem with 5 variables was studied earlier for qualitative fuzzy modeling and system identification using neural networks.

The GA was run five times for each penalty factor with different random number seeds. For test problems 1 and 2 only one penalty factor, 0.4, was used. For test problems 3 through 5, two penalty factors, 0.4 and 0.05, were used. As expected with the higher penalty factor, the algorithm converged to more compact functions. When the penalty was relaxed, the final closed form representations became more complex, but with better fitness values.

6.1 Test Problem 1

This example is taken from Nash [21], and it is one of the classic data sets of nonlinear regression that has been studied by many researchers. The data gives the growth of White Imperial Spanish Onions at Purnong Landing. It includes only one independent variable, density of plants (plants/meter squared), and a dependent variable, yield per plant. For the data set, Nash cites three potential models for the relationship between the independent variable, x (density), and the dependent variable, y (yield):

Bleasdale and Nelder [4]: $y = (b_1 + b_2 x)^{-1/b_3}$ (7)

Holliday [16]: $y = (b_1 + b_2 + b_3 x^2)^{-1}$ (8)

Farazdaghi and Harris [10]: $y = (b_1 + b_2 x^{b_3})^{-1}$ (9)

Since density figures are in the order of hundreds and power terms are present in the models, the parameters are logarithmically scaled. Determination of model parameters of the above equations was based on minimizing a natural logarithm error metric.

The GA used the data directly, not in the logarithmic form. Using a penalty factor = 0.4, four runs were selected for further examination. As can be seen from the listing in table 5, the

simplest closed form function has two imbedded logarithmic terms divided by the independent variable, *x*. The second representation is more complex than the first one, and it has an exponential term with a compound power argument. The third and fourth functions are complex versions of the second and first functions respectively. Apart from the first closed form function, all equations have sinusoidal components. Since the actual data is not cyclic in nature, presence of such terms implies overfitting the data. The plot of the first function is presented in figure 10. The comparison of SSE values of the GA models with respect to those cited by Nash is given in table 6. In order to establish a common base for comparison, SSE values were recalculated for the four GA models after a logarithmic transformation. Although the error metric for fitness calculation was different than those used in the models cited by Nash [21], all of the GA models gave superior results in terms of logarithmic SSE values.

Table 5 Selected Closed Form Functions for Test Problem 1

Model	**Equation**	**SSE**
A	$2190.3680\ln(33.7043(0.0597x))/x$	5065.244
B	$17.6377\exp(-2.8443\sin(-0.3734\ln(-35.8300\cos(0.9751x)+8.7108x)))$	4021.240
C	$12.6422\exp(-3.0512\sin(-0.3957\ln(-28.811\cos(-5.2830\sin(12.3316x))+4.7735x)))$	3651.378
D	$176.2871\ln(116.0904\cos(-1.0273\sin(182.5961x)))/(35.6118/x+0.0833x)$	3388.789

Table 6 Comparison of Results for Test Problem 1

Model	**SSE**
Bleasdale-Nelder	0. 2833
Holliday	0. 2842
Farazdaghi-Harris	0. 2829
Model A	0. 276
Model B	0. 238
Model C	0. 216
Model D	0. 214

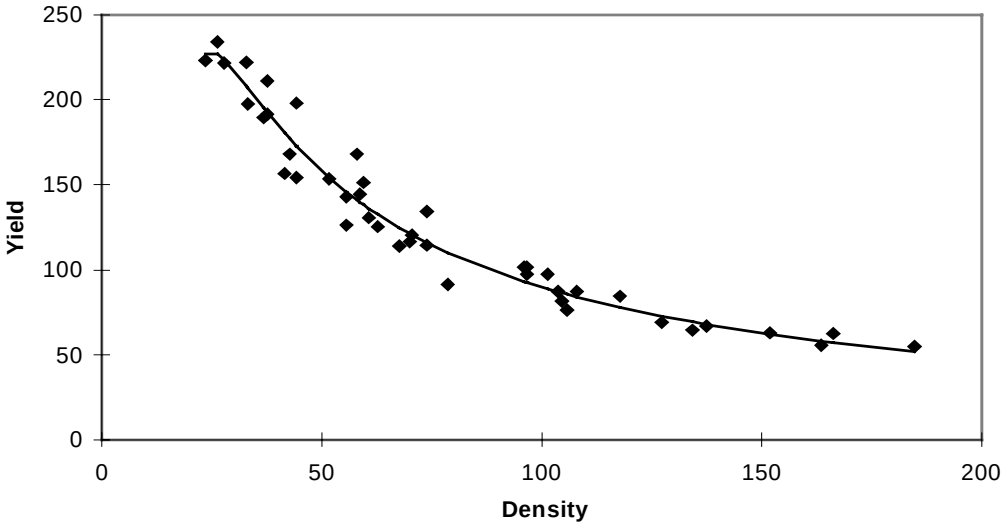

Figure 10 Fitted Function and Data Set for Test Problem 1 (Model A)

6.2 Test Problem 2

This example is taken from Galant, and it is one of the classic data sets of the nonlinear regression literature [13, 24]. The data set has 70 observations for two variables; age of preschool boys in months (independent variable) and weight/height ratio (dependent variable) (Figure 11). In terms of SSE value, the best fit using nonlinear regression for the data set is obtained by a piece-wise polynomial regression where the data set is divided into separate regions. The SSE values for a two region quadratic-linear and a three region quadratic-quadratic-linear models are 0.03790 and 0.03754 respectively [13, 24].

The closed form representations for four selected solutions are given in table 7. The first two functions include linear combination of two components; $C_1 x$ and $C_2 e^{C_3 f(x)}$. Since the power term of the exponential component has a negative coefficient (C_3), its contribution to the output diminishes as *x* becomes larger, and the closed form function behaves as a simple linear function. The third and fourth functions are more complex and they both have the following representation:

$C_1 \ln(C_2 f(x) + C_3 x)$. Since $f(x)$ consists only of sinusoidal terms, the function is bounded by $c_3 x$ for the large values of x, and the effects of $f(x)$ becomes marginal as x becomes larger.

Table 7 Selected Closed Form Functions for Test Problem 2

Model	**Equation**	**SSE**
A	$0.7357\exp(-0.6072\exp(-0.2837x)) + 0.0038x$	0.0383
B	$0.7324\exp(-0.4853\sin(1.7409\exp(-0.3532x))) + 0.0039x$	0.0373
C	$0.1545\ln(-44.8315\sin(6.4001x) + 12.3580\sin(-3.4739\sin(-5.4145x)) + 8.5875x)$	0.0355
D	$0.1648\ln(-9.7605\sin(-14.6836 + 3.3565x) + 27.4178\sin(6.1689x) + 9.0167 + 5.7035x)$	0.0320

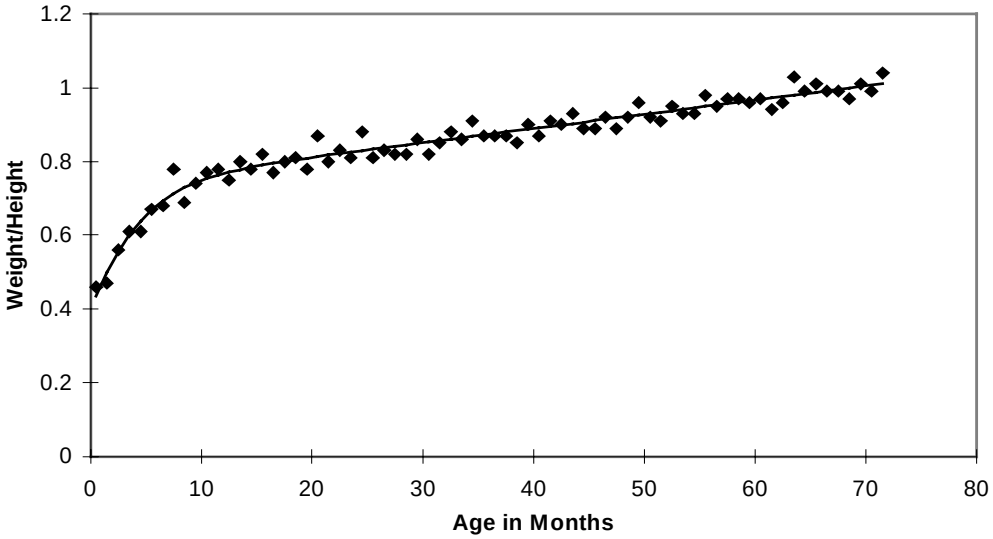

Figure 11 Fitted Function for Test Problem 2 (Model A)

6.3 Test Problem 3

For time series analysis sunspot data from 1700 to 1995 was used. The sunspot number is computed by counting each individual spot and cluster of spots on the face of the sun. According to a technique developed by Johann Rudolph Wolf (1816-1893) of the Zurich Observatory, each cluster of spots is counted as 10 spots. Currently this number is referred to as the Zurich Sunspot

Number, and it is the most commonly used measure for sunspot counting. Since results vary according to interpretation of the observer and the atmospheric conditions above the observation site, an international number is computed by averaging the measurements of 25 cooperating observatories throughout the world.[2] The data is highly cyclic with peak and bottom values approximately in every 11.1 years. However, the cycle is not symmetric, and the number of counts reaches a maximum value faster than it drops to a minimum.

The original data which had only one input and the output value (time (t) and sunspot number Y(t)) was reorganized to include 15 additional input variables, sunspot numbers Y(t-15) to Y(t-1) (equation 10). The observations start from 1715 and the first observation includes the following variables: time (t=15) and 15 sunspots numbers from 1700 to 1714. The data was divided into two groups. The first 265 observations (until 1979) were used to develop the models. In order to evaluate the model performance beyond the training data, sunspot data from 1980 to 1995 was used.

$$Y(t) = f(t, Y(t-1), Y(t-2), \ldots\ldots, Y(t-15)) \qquad (10)$$

Table 8 Selected Closed Form Functions for Test Problem 3.

Model	**Equation**	**SSE**
A	$1.1965(t-1) - 0.4585(t-2) + 0.2471(t-9)$	61964
B	$0.8337(t-1) - 1.1989\exp(-0.3512(t-9)) + 15.7476\exp(-2.7260\exp(-0.3263(t-1)) - 0.6271(t-2))$	45533
C	$1.2410\exp(-0.6099\cos(1.4097(t-4) + 0.4282(t-1)) - 0.8446(t-2))(t-1) + 0.8064(t-1) - 0.1316(t-4) + 0.1148(t-9)$	40341

[2] For more information about sunspots and computation techniques see the homepage of the National Oceanic and Atmospheric Administration at http://www.ngdc.noaa.gov.stp.SOLAR.SSN.ssn.html.

D	1.6258exp(-0.5564cos(-1.4893(t - 4) + 0.6979exp(-3.3442cos(0.2561(t - 4)) + 2.8807(t - 2)) - 3.1756(t - 2)) - 0.7485(t - 2)) - 0.9362(t - 2)(t -1) + 0.8253(t -1) - 0.1413(t - 4) + 0.1046(t - 9)	38715

The closed form representations for the selected solutions are given in table 8. The simplest model (model A) has only linear components of, (t-1), (t-2), (t-9), and is similar to a Box Jenkins model [5] which is one of the early models of sunspot data. However, the Box Jenkins model consists only of (t-1) and (t-2), whereas the GA model adds one extra term (t-9). The second model includes the same components but its nonlinear representation is more complex than the first. The third and fourth functions have one additional term (t-4) and they are both more complex than first two models (A and B). The plot of the last model (D) is presented in figure 12. The extrapolation of this model on a forecasting range of 1980 to 1996 is given in figure 13 when forecasting one period ahead.

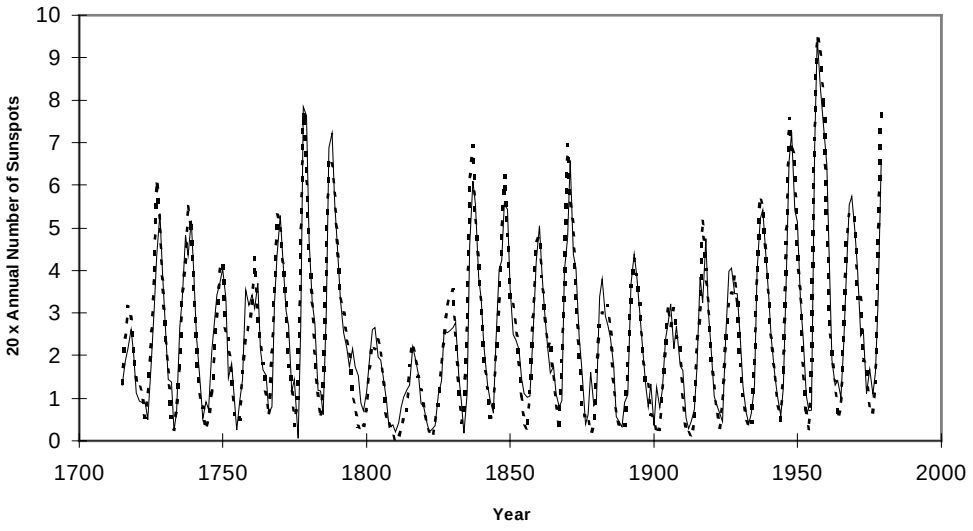

Figure 12 Fitted Function for Test Problem 3 (Model D)

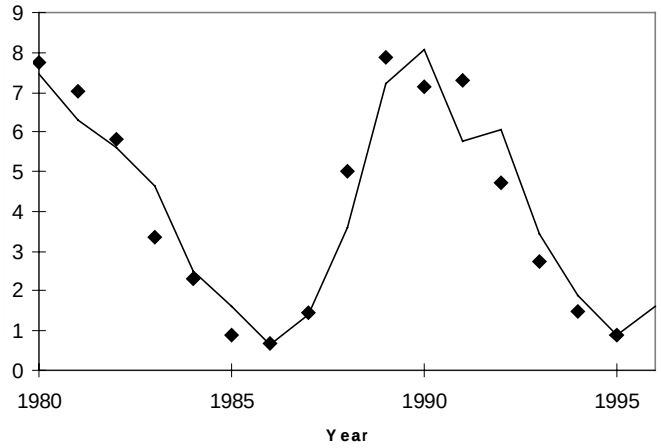

Figure 13 Extrapolation of Models Beyond Training Data Range

Sunspot data has been analyzed by many researchers [1, 22, 27]. In a recent study by Park [22], different neural network structures were employed to model sunspot data from 1770 to 1869. His study reports mean squared error values (MSE) ranging from 233.8 to 151.9. The MSE values in the GA models ranges from 234.7 (model A) to 146.7 (model D). For forecasting data MSE ranges from 291.1 (model A) to 145.5 (model C). Angeline [1] also studied this data set with a GP approach. He used values of (t-1), (t-2), (t-4) and (t-8) only, and reports SSE over the period 1708 to 1993 of 80000 and average percent error of 8%. This compares with the GA SSE of about 62000 to 39000, equating to roughly 5% to 6% error.

6.4 Test Problem 4

This example with 13 independent variables and approximately 1000 observations was the largest test problem used in this research. The data comes from a ceramic casting process where the relationship between the casting rate and 13 operational input parameters was investigated [6]. In figure 14, the output is plotted for each observation. As can be seen from this figure, because of high dimensionality of the data set, it is impossible to visually detect any pattern in the data.

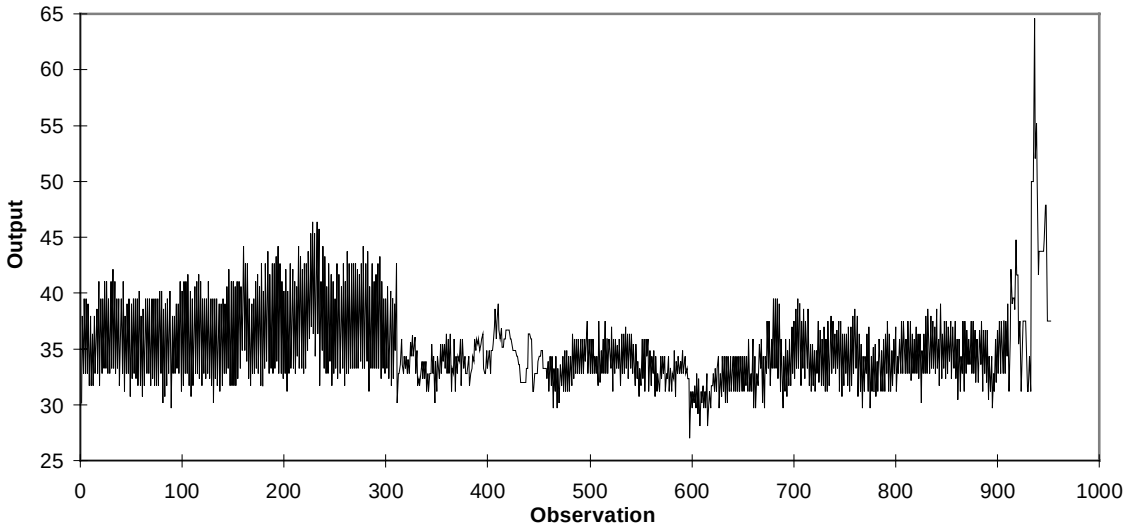

Figure 14 Multi-variable Casting Data.

Selected closed form representations are given in table 9. Three of the first four functions include linear combination of two variables, x_3 and x_{10}, which seem to be the most important variables in the data set. Interestingly, the coefficients associated with these variables are very close to each other in all three functions. This strongly suggests that the additional components in the third and fourth equations have a lesser effect on the output. The plot of the difference between the fitted model and the actual data points (error) is illustrated in figure 15. The results of model E roughly correspond to the same error rate obtained by the neural network of [6], however the GA approach has the advantage of a discernible form.

Table 9 Selected Closed Form Functions for Test Problem 4

Model	Equation	SSE
A	$3.2142x_{10} + 0.2184x_3$	6861.575
B	$-46.8821\sin(22.5976x_3) + 10.4534\cos(6.3422x_5)$	6818.134
C	$3.2526x_{10} + 0.1438x_4 + 0.2221x_3$	6070.081
D	$0.2192x_3 + 3.1044x_{10} + 0.1705\exp(-4.4522\cos(6.1689x_5))$	5948.197
E	$3.0454\ln(-2.7639x_{11} + 19.0614x_{12}x_9) + 0.5194(((x_3(7.8126\cos(-4.8424x_3) + 2.5578x_{10}))/x_5)/(-6.9509\cos(6.0312\ln(5.6902x_4)) + 1.5221x_{12}))((x_1x_{10})/x_5)$	3994.057

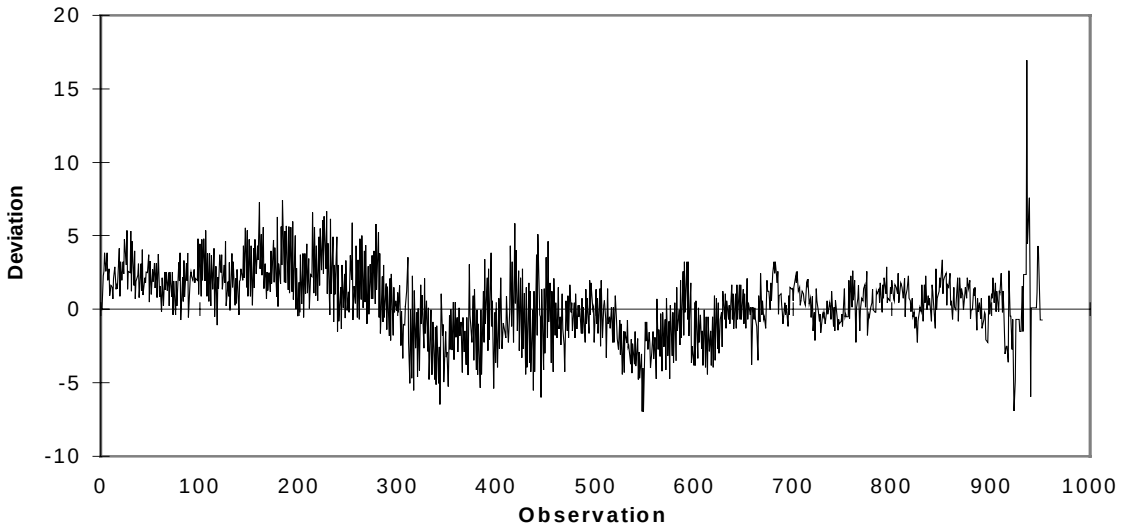

Figure 15 Difference Between the Data and the Fitted Function (Model D)

6.5 Test Problem 5

This example is taken from [26] where a qualitative fuzzy model for an operator's control of a chemical plant is built. The chemical process has five candidate input variables which may be manually adjusted to control the output, set value for monomer flow rate. The candidate

variables are (1) monomer concentration (x_1), (2) change of concentration (x_2), (3) actual monomer flow rate (x_3), (4) temperature 1 inside the plant (x_4) and (5) temperature 2 inside the plant (x_5). The data set includes 70 observations of the above six variables (figure 16). The authors first generated six data clusters by using fuzzy clustering and identified x_1, x_2 and x_3 as the most relevant parameters.

The same data set was also studied by [3]. Their neural network model identified x_3 as the single most significant input parameter followed by x_1. Their model also includes x_5, which has a very marginal effect on the accuracy of the model.

Table 10 Selected Closed Form Functions for Test Problem 5

Model	**Equation**	**SSE**
A	$0.1877\exp(-0.9109\sin(0.8913x_3)) + 0.9694x_3$	0.6057
B	$-0.2033\exp(-0.8714\cos(3.2885\exp(-3.8097\sin(1.0549x_3)))) + 0.9903x_3$	0.3864
C	$1.8257\cos(2.0012x_5 + 0.2486x_2) - 2.3442x_1 + 0.2013\sin(-0.8969x_3) + 0.8915x_3$	0.3553
D	$0.0505\sin(7.9847x_5\cos(-5.8208x_3 - 2.7400\sin(-3.0262x_3)) + 1.5718x_3)x_3 + 1.046464x_3$	0.3321
E	$0.1318\exp(-1.4083\sin(3.0478\cos(1.6935 - 2.7229x_3) - 1.8731\sin(3.3575 + 3.8175x_5 + 3.0451 - 6.9550x_3) + 0.4712x_3)) + 0.9968x_3$	0.2753

There are some similarities between the GA results and the previous research. All five solutions given in table 10 contain a linear component of x_3, which was identified as the most significant parameter by both [3] and [26]. This shows that there is one to one relationship between x_3 and the output. In the actual system, the output is the set point value for monomer flow rate, and x_3 represents the actual flow rate. Thus, such a relationship between these two

parameters could be anticipated. The exponential term with a negative coefficient of three of the functions reflects that the deviation between the output and x_3 is diminishing in later observations. Although there is not any information about the data collection procedure, the nature of the closed form functions suggests that the data comes from a start up process. In figure 16, one of the closed form functions (model E) is plotted with respect to actual data points. As can be seen from this figure, the fitted model represents the actual data points very well.

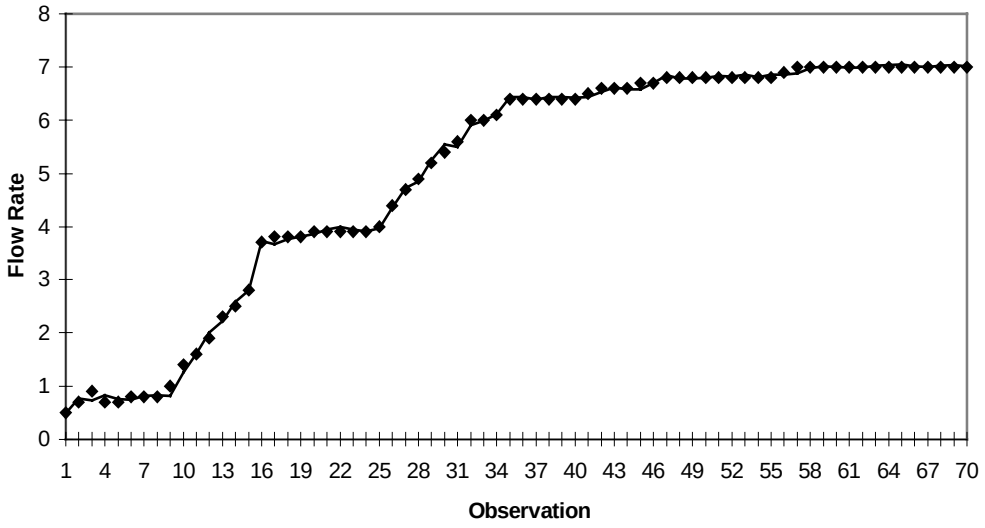

Figure 16 Fitted Function for Chemical Plant Problem (Model E)

7. CONCLUDING REMARKS

For classical nonlinear regression problems the genetic framework gave comparable or better numerical results (i. e., lower SSE values) than models cited in the literature. It should be noted that the final population always contains complex functions with better numerical results than those cited in the literature. The simpler functions in the final population have less accurate results compared to those obtained by alternative techniques such as neural networks. However, unlike neural networks, this approach provides a function which can be used directly for both intuitive analysis and mathematical computation.

One distinguishing feature of this approach is that it gives a set of solutions to the user. The user can study those solutions and choose the model that satisfies both complexity and numerical accuracy constraints. The user can also review the set of good functions for similarity in terms (independent variables and primitive functions) so that the relationship can be better understood. Furthermore, it is very easy for the user to specify different penalty severities, different sets of primitives, and different error metrics to explore the variety of functions, their form and fit with the data. This distinguishes the hierarchical GA framework as an effective global approach for system identification and curve fitting. The only significant drawback is the magnitude of computational effort required. However, since system identification and curve fitting are usually done infrequently and off-line, there is no imperative for extreme computational speed.

ACKNOWLEDGEMENT

Alice E. Smith gratefully acknowledges the support of the U.S. National Science Foundation CAREER grant DMI 95-02134.

REFERENCES

[1] Angeline, Peter J., "Two self-adaptive crossover operators for genetic programming," *Advances in Genetic Programming Volume II* (editors: Peter J. Angeline and Kenneth E. Kinnear, Jr.), MIT Press, Cambridge MA, 89-109, 1996.

[2] Bäck, T. and Schwefel, H. P., "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, 1, 1-23, 1993.

[3] Bastian, A. and Gasos, J., "A type I structure identification approach using feed forward neural networks," *IEEE International Conference on Neural Networks*, 5 (of 7), 3256-3260, 1994.

[4] Bleasdale, J. K. A. and Nelder, J. A., "Plant population and crop yield," *Nature*, 188, 342, 1960.

[5] Box, G. E. and Jenkins, G. M., *Time Series Analysis: Forecasting and Control*, Holden-Day, San Francisco, CA, 1970.

[6] Coit, D. W. and Jackson-Turner, B. and Smith, A. E., "Static neural network process models: considerations and case studies," *International Journal of Production Research*, forthcoming.

[7] Collins, J. S., "A regression analysis program incorporating heuristic term selection," *Machine Intelligence*, Donald Michie Editor, American Elsevier Company, New York NY, 1968.

[8] Cramer, N. L., "A representation for the adaptive generation of simple sequential programs," *Proceedings of an International Conference on Genetic Algorithm and Their Applications*, 183-187, 1985.

[9] Dallemand, J. E., "Stepwise regression program on the IBM 704," *Technical Report of General Motors Research Laboratories*, 1958.

[10] Farazdaghi, H. and Harris, P. M., "Plant competition and crop yield," *Nature*, 217, 289-290, 1968.

[11] Friedman, J., "Multivariate adaptive regression splines," Department of Statistics Technical Report 102, Stanford University, 1988 (revised 1990).

[12] Fujiko, C. and Dickinson, J., "Using the genetic algorithm to generate LISP source code to solve prisoner's dilemma," *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, 236-240, 1987.

[13] Gallant, A. R., *Nonlinear Statistical Models*, John Wiley and Sons, New York, 1987.

[14] Geman S. and Bienenstock, E., "Neural networks and the bias/variance dilemma," *Neural Computation*, 4, 1-58, 1992.

[15] Gulsen, M., Smith, A. E. and Tate, D. M., "Genetic algorithm approach to curve fitting," *International Journal of Production Research*, 33, 1911-1923, 1995.

[16] Holliday, R., "Plant population and crop yield," *Field Crop Abstracts*, 13, 159-167, 247-254, 1960.

[17] Karr, C. L., Stanley D. A., and Scheiner B. J., "Genetic algorithm applied to least squares curve fitting," *U.S. Bureau of Mines Report of Investigations* 9339, 1991.

[18] Koza, J. R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge MA, 162-169, 1992.

[19] Koza, J. R., *Genetic Programming: Automatic Discovery of Reusable Programs*, MIT Press, Cambridge MA, 1994.

[20] Mühlenbein, H. and Schlierkamp-Voosen, D., "Predictive models for the breeder genetic algorithm. I. Continuous parameter optimization," *Evolutionary Computation*, 1, 25-49, 1993.

[21] Nash, J. C. and Walker-Smith, M., *Nonlinear Parameter Estimation: An Integrated System in BASIC*, Marcel Dekker, New York, 1987.

[22] Park, Young R. and Murray, Thomas J. and Chen, Chung, "Predicting sun spots using a layered perceptron neural network," *IEEE Transactions on Neural Networks*, 7, 501-505, 1996.

[23] Rogers, D., "G/SPLINES: "A hybrid of Friedman's multivariate adaptive regression splines (MARS) algorithms with Holland's genetic algorithm," *Proceedings of the Fourth International Conference on Genetic Algorithms*, 384-391, 1991.

[24] Seber, G. A. F. and Wild, C. J., *Nonlinear Regression*, John Wiley and Sons, New York, 1989.

[25] Stigler, S. M. *History of Statistics: The Measurement of Uncertainty Before 1900*, Harvard University Press, Cambridge MA, 1986.

[26] Sugeno, M., and Yasukawa, T. "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, 1, 7-31, 1993.

[27] Villareal, J. and Baffes, P., "Sunspot prediction using neural networks," *University of Alabama Neural Network Workshop Report*, 1990.

[28] Westervelt, F., *Automatic System Simulation Programming*, unpublished Ph.D. Dissertation, University of Michigan, 1960.