Nonlinear MISO Modeling Using Genetic Programming

Reid S. Maust **Ronald L. Klein**

Department of Computer Science & Electrical Engineering
P.O. Box 6109
West Virginia University
Morgantown, WV 26506-6109
rmaust@wvu.edu, klein@cemr.wvu.edu

I. ABSTRACT

An algebraic model for an unknown, nonlinear, MISO (multiple input, single output) system is derived from a table of the system's input and output values. Genetic programming is used to find a model that is optimal (or nearly optimal) with respect to a nonlinear performance index. In order to apply genetic programming to this task, an encoding strategy to represent the model is devised. Then, specialized genetic operators are defined to refine the solution. The technique is shown to produce a good model for a simple nonlinear example having two inputs and one output.

II. INTRODUCTION

Model identification for unknown systems ("black boxes") has been the focus of much investigation [1]. This paper uses genetic programming [2,3,4,5] to model an algebraic, nonlinear, time-invariant system based on its input-output data. To illustrate the technique, a model is found for the widely published wind chill table [6], which is a static system that defines one output (apparent temperature) in terms of two inputs (actual temperature and wind speed). This simple example has been chosen to demonstrate some of the design issues involved in using genetic programming for this purpose.

III. PROBLEM STATEMENT

In order to derive an algebraic model for the system, the system's output to some set of input values (a "training sequence") must be known. For a physical system, this information can be measured directly. In choosing the amount of input-output data to use for this procedure, it is desirable to represent as wide a range of possibilities for the input variables as possible. A model based on too little data may not accurately represent the system's response.

For notational purposes, let the input-output pairs be numbered sequentially, such that u_k is the k^{th} input vector of the unknown system, and y_k is the corresponding output. Given u_k and y_k, it is desired to find a nonlinear function h, such that

$$\mathrm{y}_k = \mathrm{h}(\mathrm{u}_k) \tag{1}$$

for all k. In general, it is impossible to find a function h that satisfies (1) exactly. Instead, the model having the best possible fit is sought. To measure the quality of fit for a particular model, the *fitness function* is chosen to be based on the L_1 norm [7,8] of the error:

$$f = \frac{1}{1+\|f_{DZ}(e)\|_1} = \frac{1}{1+\sum_k |f_{DZ}(e_k)|} \tag{2}$$

where the output error e_k is defined as

$$e_k = y_k - \hat{y}_k \tag{3}$$

and where $\hat{y}_k$ is the model's estimate of y_k. The dead-zone function is defined as

$$f_{DZ}(x) = \begin{cases} x - UB, & x > UB \\ 0, & LB \le x \le UB \\ x - LB, & x < LB \end{cases} \tag{4}$$

where UB and LB are respectively the upper and lower bounds of the dead zone. By setting UB and LB respectively equal to the upper and lower bounds of the additive measurement error, the dead-zone function is used to penalize the error only when it exceeds the measurement noise in the data. Any error on the same order of magnitude as measurement noise is ignored.

The fitness function in (2) is structured to transform an error $\in [0,\infty)$ into a fitness value $\in (0,1]$. From (2), an error of zero corresponds to a fitness of 1, and an infinite error maps to a fitness of zero.

IV. PROBLEM SOLUTION

This section provides a brief introduction to genetic programming, lists the strengths and weaknesses of this technique, discusses the choice of encoding strategy, and defines the genetic operators used. The selection of the encoding strategy determines exactly how the candidate solutions (in this case, the function h) are represented in the population. The genetic operators are a predetermined set of rules designed to combine information from individual candidates. An intelligent choice of data structures and genetic operators will speed the algorithm's convergence. Thus, the main contribution of this work is the choice of genetic operators and encoding strategy that

0-7803-4547-9/98/$10.00 © 1998 IEEE

allow the determination of an analytic model for the wind chill table.

A. Introduction to Genetic Programming

Genetic programming (GP) and genetic algorithms (GAs) [2,3,4,5] are iterative optimization techniques based on artificial intelligence. These methods are based on Darwin's survival of the fittest hypothesis. Genetic algorithms are a special case of genetic programming. While a genetic algorithm manipulates lists of numbers, genetic programming uses more complicated data structures. In some applications, GP is used to choose among several modules of executable computer code [4]. For the wind chill problem, the GP is used to manipulate algebraic functions. The data structures and genetic operators used in solving the wind chill problem will be explained in greater detail later.

With both GP and GA, candidate solutions to a problem are represented as individual members of a population. For the wind chill problem, each member of the population represents a possibility for the input-output function h. Each candidate model is simulated, and the model's simulated output is compared to the actual system's output. As the generations progress, the candidate solutions are refined until convergence is reached.

Although the initial population can be a random collection of individuals, the individuals interact and breed to form future generations. Stronger individuals reproduce more often than do weaker individuals. Presumably, the population will get collectively stronger as generations pass and weaker individuals die out. The quantitative application of these basic ideas to an actual algorithm is a combination of science and art.

In a genetic optimization problem, the objective is to maximize a *fitness function*. The fitness is calculated for each member of the population, and some individuals are selected to survive into the next generation. Under *roulette-wheel* selection [4,5], a function's probability of survival is directly proportional to its fitness value. The selection operation forms the next generation of solutions by copying randomly chosen survivors from the previous generation. It is possible that some very fit functions might be copied into the next generation more than once (cloning), while some unfit functions might not be copied at all (death). Because of the probabilistic nature of this selection mechanism, it is also possible for the best solution to be passed over and not be chosen for survival. This work uses *elitism* (the automatic copying of the best solutions to the next generation) [4,5] to guarantee that the best solution will always survive.

Once the new generation of candidate solutions is formed, genetic operators are used to combine information from various candidate solutions. The operators used here are discussed in a later section, but most operators are a variation on the themes of *crossover* or *mutation*. Genetic *crossover* operators simulate breeding by selecting "parents" and combining them using specified rules to form "children" which include some information from each parent. The choice of crossover operator depends on the problem being optimized and the structure of the candidate solutions. *Mutation* (randomly changing some members of the population) [5] is used to introduce diversity into the population and to help avoid convergence to a local (rather than global) maximum of the fitness function. As another way to introduce diversity, a few completely random individuals are placed in the population during each generation.

B. Strengths & Limitations of Genetic Programming

Genetic Programming has the following advantages over conventional optimization techniques:

1. Because of its iterative, evolutionary nature, GP can optimize with respect to a nonlinear, analytically intractable performance index.
2. GP does not require a differentiable performance index. Thus, this research is not restricted to using the least-square error criterion.
3. GP can readily enforce constraints on the solutions. In contrast, enforcing constraints using conventional techniques can result in an intractable set of partial differential equations to be solved.
4. If some information is already known about the problem (such as a heuristic way of solving the problem), GP can use that information by including in the initial population candidate solutions reflecting that information.
5. Because GP maintains a population of potential solutions, it is less susceptible than other optimization methods to becoming trapped in a local optimum.
6. The structure of the optimization technique can become more or less complicated to match the complexity of the problem. If a problem consists only of finding the best set of parameters, the full tree structure of GP is unnecessary. In this case, genetic programming reduces to a genetic algorithm.

These advantages give GP great flexibility in modeling. However, like any computation technique, GP has its limitations. Two important limitations of GP (and how to lessen their impact) are:

1. *Computational effort required.* GP can require the evaluation of thousands of candidate solutions before converging on the best solution. The wind chill problem is simple enough that this is not really a factor. Complicated modeling problems would have to be solved off-line, rather than in real-time.
2. *Difficulty in Choosing a Stopping Criterion.* It is sometimes difficult to determine whether the GP algorithm has found the optimal solution. Although GP is less susceptible to getting trapped in a local (rather than global) optimum than other techniques such as hill climbing or simulated annealing [4,5], converging to a suboptimal solution is still possible. Increasing the population size, evolving the population

for more generations, or increasing the amount of mutation in the population can counteract suboptimal convergence. In other words, it is difficult to determine whether or not the algorithm should continue searching for a better solution. Sometimes, an arbitrary choice of stopping criterion is made (such as iterating for a set number of generations) and then the answer is subjectively judged as to whether it is sufficiently accurate.

Note that GP is not generally used for problems easily optimized using conventional techniques. For difficult optimization problems such as nonlinear modeling, the power and flexibility of GP outweigh the limitations.

C. Encoding Strategy for the Wind Chill Problem

As stated earlier, the GP solutions represent the input-output function *h(u)*. However, the candidate functions that estimate *h* are not stored directly in the population. If the candidates were stored in the population without regard to mathematical structure, a great deal of computational effort would be used to discard solutions that fit the data poorly. Instead, an interpolation technique inspired by the finite-element method [9] of solving differential equations is used. The basic premise is that a function of *n* independent variables can be rewritten as a systematic, nonlinear combination of *n* scalar functions, one for each variable. Since the interpolation technique constrains the estimate to pass exactly through four of the data points, a separate step is used to introduce some error at these data points, in order to reduce the error at the other points. The wind chill problem demonstrates the technique.

The wind chill problem's input-output function *h* has two independent variables, actual temperature and wind speed. Graphically, let the inputs be the basis for the xy-plane, and let the output (wind chill temperature) be represented by *z*, the elevation above the plane. Let the x and y axes in the plane be scaled and translated so that each independent variable is mapped onto the interval [0,1]. The mapping used here is

$$x = 0.5 - \frac{t}{70} \qquad y = \frac{w-5}{40} \tag{5}$$

where $t \in [-35,35]$ is the temperature in °F and $w \in [5,45]$ is the wind speed in miles per hour. Other mappings are also acceptable, provided that the mapping is one-to-one between the original data interval and the interval [0,1].

Let *f(x)* and *g(y)* be scalar functions of one variable that pass through (0,0) and (1,1). That is, f(0)=0, f(1)=1, g(0)=0, and g(1)=1. Then the following relation between *f*, *g*, and *h* ensures that the function *h* passes through the four data points corresponding to the corners of the unit square in the xy-plane:

$$\begin{aligned} h(x,y) \;=\; & f(x)g(y)z_{(1,1)} + f(x)[1-g(y)]z_{(1,0)} \\ & + [1-f(x)]g(y)z_{(0,1)} + [1-f(x)][1-g(y)]z_{(0,0)} \end{aligned} \tag{6}$$

where $z_{(x,y)}$ is the measured output corresponding to the input pair (x,y). Note the behavior of these four terms at the corners of the unit square, where (x,y) ∈ {(0,0), (0,1), (1,0), (1,1)}. For any (x_1,y_1) in this set (that is, at a corner), the term corresponding to (x_1,y_1) is multiplied by 1, while the other three terms are multiplied by 0. For example, suppose that $(x_1,y_1)=(0,1)$. Thus, $f(x_1)=0$ and $g(y_1)=1$, because these functions are constructed to pass through (0,0) and (1,1). Therefore, any term multiplied by *f(x)* is multiplied by 0 when evaluated at $x=x_1=0$, and any term multiplied by *[1-g(y)]* is multiplied by 0 when evaluated at $y=y_1=1$. The only remaining term is $z_{(0,1)}$, which is multiplied by 1.

Thus, the form of (6) constrains the function *h* to pass exactly through the data corresponding to the four corners of the unit square. For input pairs in the interior of the unit square, the effect of this equation is to interpolate the data. The shape of the interpolation is determined by the shapes of f(x) and g(y). As an illustration, assume that *f(x)=x* and *g(y)=y*. The coefficient of $z_{(0,1)}$ is then *(1-x)y*, which is graphed in Figure 1.

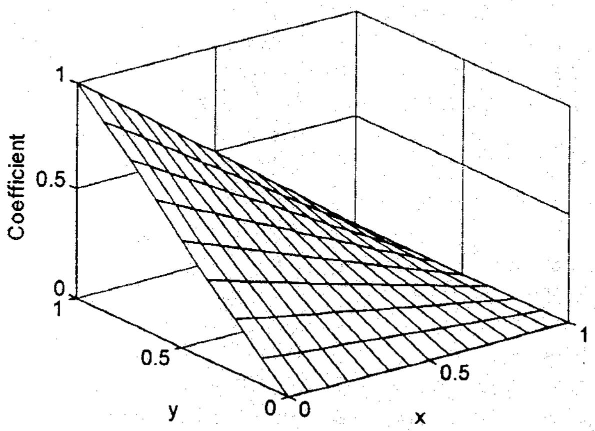

Figure 1. Coefficient for $z_{(0,1)}$

The coefficient exhibits a smooth transition from (x,y)=(0,1), where it equals 1, to the other three corners, where it equals 0. The portion of the graph that is hidden from view in Fig. 1 is symmetrical to the portion shown. An analogous transition occurs for the other three coefficients. From (6), the final estimate is a linear combination of four graphs like that in Fig. 1.

However, for the wind-chill modeling problem, it is not desired to constrain the model to be exact at the four corners. Instead, it is desired to allow some error in the four corners, if allowing this error reduces the error elsewhere in the square and improves the overall fitness of the solution. Thus, a perturbation is added to each data value in (6), giving

$$\begin{aligned} h(x,y) \;=\; & f(x)g(y)[z_{(1,1)}+c_1] + f(x)[1-g(y)][z_{(1,0)}+c_2] \\ & + [1-f(x)]g(y)[z_{(0,1)}+c_3] + [1-f(x)][1-g(y)][z_{(0,0)}+c_4] \end{aligned} \tag{7}$$

where the perturbations c_i are parameters found by the GP algorithm to optimize the fitness of the solution.

To use this technique, the GP maintains three separate populations, one each for *f*, *g*, and c_i. The corresponding elements of these populations are then combined according to (7) to form *h*. The initial population is chosen randomly. For each individual in the population, the function *f(x)* is randomly chosen from the set $\{x, 2x-x^2, [1-\exp(ax)]/[1-\exp(a)]\}$, and the function *g(y)* is randomly chosen from a similar set of functions in y. These functions were heuristically chosen because of their simplicity and their resemblance to the data. Of course, more functions could have been added to the set if warranted by the problem structure. The parameter *a* in the exponential form is randomly chosen to be a positive number. Note that this function has been scaled to pass through (1,1). The perturbation values, c_i in (7), are all initially set to zero.

D. Genetic Operators for the Wind Chill Problem

The wind chill model is optimized through the use of four kinds of crossover operator and two kinds of mutation. The crossover operators are binary operators designed to exchange information between two candidate solutions ("parents"). Each operator defines a way to form two "children" by combining pieces of each parent. The children replace their parents in the population. In contrast, the mutation operators are unary operators designed to provide a mechanism for changing the structure of a candidate solution. All crossover and mutation operators are constructed so that the new candidate solutions produced have component functions (*f* and *g*) that continue to pass through (0,0) and (1,1). Furthermore, to prevent these operators from destroying good solutions, elitism guarantees that a predetermined number of the best solutions are guaranteed to survive into the next generation.

Let $\{f_1(x),g_1(y)\}$ and $\{f_2(x),g_2(y)\}$ represent two "parents" whose input-output relation is given in (7). *Simple crossover* breaks these pairs and re-distributes the halves. Thus, it defines the children as $\{f_1(x),g_2(y)\}$ and $\{f_2(x),g_1(y)\}$. *Arithmetic crossover* [4] randomly selects an independent variable and performs a weighted average of the corresponding parts of the parents. If *x* were chosen for this operation, the children would become $\{[af_1(x)+(1-a)f_2(x)],g_1(y)\}$ and $\{[(1-a)f_1(x)+af_2(x)],g_2(y)\}$. The parameter *a* is a random scalar between 0 and 1.

In order to allow the algorithm to build complicated functions from simple pieces, *composition* and *multiplication* (two additional crossover operators) are defined. Each of these operators randomly chooses an independent variable and combines the corresponding parts of the candidates. If *x* were chosen for composition, the children would be $\{f_1(f_2(x)),g_1(y)\}$ and $\{f_2(f_1(x)),g_2(y)\}$. If *x* were chosen for multiplication, the children would be $\{f_1(x)f_2(x)),g_1(y)\}$ and $\{f_1(x)f_2(x)),g_2(y)\}$.

Two mutation operators are used to provide a way for the algorithm to escape from a local (rather than global) optimum. The *scaling* operator multiplies one of the candidate solution's independent variables by a constant, *a*. The value of *a* is determined by successively setting *a* equal to values in the set {.5, .75, .9, .95, .995, 1, 1.005, 1.05, 1.1, 1.5, 2} and keeping the one producing the solution with the highest fitness. If *y* were chosen for scaling, $\{f_1(x),g_1(y)\}$ would be replaced by $\{f_1(x),g_1(ay)/g_1(a)\}$. This operation is especially useful for optimizing curvature of nonlinear functions such as parabolas or exponential functions. This operator has no effect on a linear function, because the parameter *a* cancels out of the quotient. Because of this observation and the fact that the algorithm was observed to set f(x) equal to x, the scaling operator is only applied to y for this wind chill problem. The other mutation operator, *reflection*, reflects of one of the candidate's component functions with respect to a straight line. If *x* were chosen for reflection, $\{f_1(x),g_1(y)\}$ would be replaced by $\{[2x-f_1(x)],g_1(y)\}$.

Although the perturbation values, c_i in (7), are all initially set to zero, the algorithm searches for the best values whenever a new solution is generated. The algorithm searches the possibilities of adding each member of the set {–0.5, –0.25, –0.1, 0, 0.1, 0.25, 0.5} to the current value of each c_i. Each c_i is examined separately. Thus, each search examines $4\times7 = 28$ possibilities.

Since the GP algorithm randomly selects when to use these crossover and mutation operators, a probability of choosing each operator must be defined. For the wind chill problem, these probabilities are chosen heuristically.

V. RESULTS

A population of 24 functional forms was evolved for 200 generations. Since the wind chill table is rounded to the nearest 1°F, the dead zone's upper and lower bounds in (4) are set to UB=0.5 and LB= –0.5 to reflect the measurement error. The probability of each genetic operator is given in Table 1.

Table 1. Probability of Each Genetic Operator

Operator	Probability
Simple Crossover	0.80
Arithmetic Crossover	0.15
Composition	0.15
Multiplication	0.10
Scaling	0.20
Reflection	0.15

The best input-output relation found was

$$\begin{gathered} f(x) = x \\ g(x) = 1.059[1-\exp(-2.889y)] \\ [c_1 \quad c_2 \quad c_3 \quad c_4] = [0.80 \quad -0.05 \quad 0.10 \quad 0.20] \end{gathered} \tag{8}$$

The fitness of this final solution is 0.4971. From (7), the final estimate of the input-output relation simplifies to

$$\begin{aligned} h(x,y) = {} & 0.1142 - 113.3\,x \\ & + [32.09 + 39.13\,x]\exp(-2.889y) \end{aligned} \tag{9}$$

Using (5) gives the input-output relation in terms of t and w, which are temperature (°F) and wind speed (mph), respectively:

$$h(t,w) = -56.52 + 1.618\,t + [\,74.11 - 0.8021\,t\,]\exp(-0.07223\,w) \tag{10}$$

The estimated and actual input-output relations are graphed in Fig. 2.

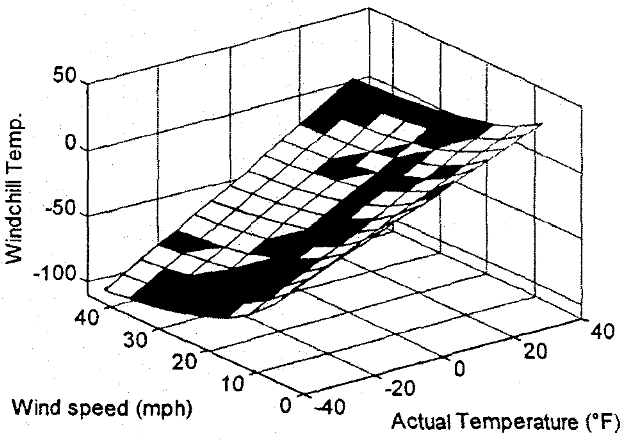

Figure 2. Comparison of Actual (Shaded) and Estimated (Unshaded) Wind Chill Curves

In the shaded regions of Fig. 2, the actual curve is visible because it is higher (more positive) than the estimate. In the unshaded regions, the estimate is more positive curve and therefore visible. Although the two surfaces are not exactly coincident, the difference between them is small. In fact, the surfaces never differ by more than 1.5°F.

In an attempt to ascertain whether the solution in (10) is optimal, the improvement in fitness as the generations progress is shown in Figure 3.

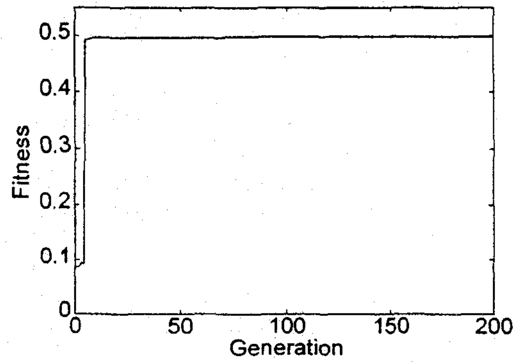

Figure 3. Fitness of Each Generation's Best Solution

Most of the improvement in the fitness of the solution was achieved in the first 10 generations, and no improvement occurred after 78 generations. The fact that the final 122 generations could not improve this solution suggests, but does not prove, that this solution is optimal. This suggestion, together with the fact that the estimate is never more than 1.5°F in error, provides motivation to accept (10) as an analytic model for the wind chill table.

VI. CONCLUSION

The wind chill example presented here demonstrates the ability of GP to find a nonlinear model of an algebraic system. Although the wind chill example is a simple problem, it demonstrates the need for an intelligent choice of encoding strategy and genetic operators to improve convergence of the algorithm. Thus, the techniques outlined here are applicable to nonlinear system modeling.

VII. REFERENCES

[1] P. Davis, *Interpolation and Approximation* (Waltham, Mass.: Blaisdell Pub. Co., 1963)

[2] J. J. Greffenstette, "Genetic Algorithms," *IEEE Expert*, Vol. 8, No. 5 (October 1993), pp. 5-8

[3] P. Wayer, "Genetic Algorithms," *Byte*, Vol. 16, No. 1 (January 1991), pp. 361-368

[4] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition (New York: Springer-Verlag, 1996)

[5] L. Davis, editor, *Handbook of Genetic Algorithms*, (New York: International Thompson Computer Press, 1996)

[6] National Oceanic and Atmospheric Administration, *Wind Chill Table*

[7] G. Strang, *Linear Algebra and its Applications*, 3rd edition (San Diego: Harcourt Brace Jovanovich 1988)

[8] E. Kreyzig, *Advanced Engineering Mathematics*, 7th edition (New York: John Wiley & Sons, 1993)

[9] G. Strang and G. Fix, *An Analysis of the Finite Element Method* (Englewood Cliffs, NJ: Prentice Hall, 1973)