RIACS

Research Institute for Advanced Computer Science
NASA Ames Research Center

G/SPLINES: A Hybrid of Friedman's Multivariate Adaptive Regression Splines (MARS) Algorithm with Holland's Genetic Algorithm

DAVID ROGERS

RIACS Technical Report No. 91.10

May 1991

G/SPLINES: A Hybrid of Friedman's Multivariate Adaptive Regression Splines (MARS) Algorithm with Holland's Genetic Algorithm[1]

David Rogers
Research Institute for Advanced Computer Science
MS Ellis, NASA Ames Research Center
Moffett Field, CA 94035
(415) 604-6363

Abstract

G/SPLINES are a hybrid of Friedman's Multivariable Adaptive Regression Splines (MARS) algorithm with Holland's Genetic Algorithm. In this hybrid, the incremental search is replaced by a genetic search. The G/SPLINE algorithm exhibits performance comparable to that of the MARS algorithm, requires fewer least-squares computations, and allows significantly larger problems to be considered.

1 INTRODUCTION

Many problems in diverse fields of study can be formulated into the problem of approximating a function from a set of sample points. For functions of few variables a large body of statistical methodology exists; these methods offer robust and effective approximations. For functions of many variables, relatively fewer techniques are available, and these techniques may not perform adequately in the desired high-dimensional setting. The interest in so-called neural-network models is due in part to their performance in these high-dimensional multivariate environments.

One class of algorithms proposed for high-dimensional environments rely on local variable selection to reduce the number of input dimensions during model construction. These methods approximate the desired function locally using only a subset of the large number of possible input dimensions. Some of the members of this class of algorithms are k-d Trees [1], CART [2], and Basis Function Trees [10]. These algorithms build an approximation model starting with the constant model, and refine the model incrementally by adding new basis functions.

Recently Friedman proposed another algorithm in this class, the Multivariate Adaptive Regression Splines (MARS) algorithm [5]. This statistical approach performs quite favorably with respect to many neural-network models. Unfortunately, the algorithm is too computationally intensive for use in problems that involve large (>1000) sample sizes or extremely high (>20) dimensions. This behavior is caused by the structure of the MARS algorithm, which builds models incrementally by testing a large class of possible extensions to a partially-constructed spline regression model, then adding the best extension.

G/SPLINES are a hybrid of Friedman's Multivariable Adaptive Regression Splines (MARS) algorithm with Holland's Genetic Algorithm [8]. In this hybrid, the incremental search is replaced by a genetic search. The G/SPLINE algorithm exhibits performance comparable to that of the MARS algorithm, requires less computation, and allows significantly larger problems to be considered.

In this paper I begin with a discussion of the problem of functional approximation models, and the use of splines in these models. I then describe the MARS algorithm and estimate the number of least-squares regressions it requires. I follow with a description of the G/SPLINE algorithm. I conclude with experiments to illustrate its performance relative to the MARS algorithm and to study properties unique to G/SPLINES.

2 THE PROBLEM

We are given a set of N data samples $\{X_i\}$, with each data sample X_i being a n-dimensional vector of predictor variables $\langle x_{i1}, x_{i2}, ..., x_{in}\rangle$. We are also given a set of N responses $\{y_i\}$. We assume that these samples are derived from an underlying system of the form:

$$y_i = f(X_i) + \text{error} = f(x_{i1}, ..., x_{in}) + \text{error}$$

The goal is to develop a model G(X) which minimizes some error criterion, such as the least-squares error:

$$LSE(G) = \frac{1}{N}\sum_{i=1}^{N} (y_i - G(X_i))^2$$

1. To appear in the proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, July 1991.

The model G is commonly constructed as a linear combination using some set of basis functions:

$$G(X) = a_0 + \sum_{k=1}^{M} a_k \phi_k(X)$$

Given an appropriate set of basis functions, standard least-squares regression techniques can be used to find a set of coefficients $\{a_k\}$ which minimizes the least-squared error [9]. This process suffers from two major weaknesses. First, if the basis functions for G do not reflect the underlying global structure of the function F, the accuracy of G is likely to be poor. Second, if too many basis functions are used in the approximation, the model may suffer from *overfitting*; while it generates reasonable approximations for F when given a data sample in $\{X_i\}$, previously unseen data samples may generate large errors. See Figure 1.

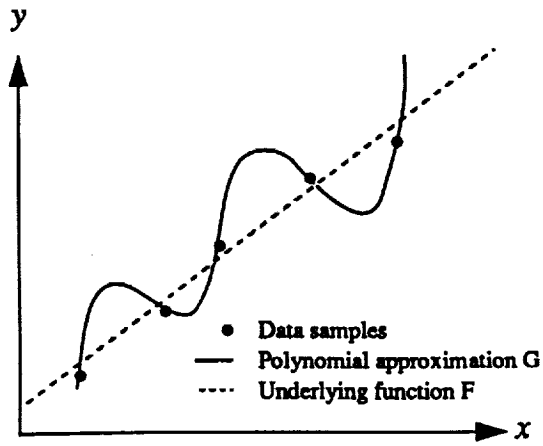

Figure 1: Overfitting. Using polynomials as the basis functions in constructing G, we create an approximation which exactly fits the data sample points but does not approximate the underlying function F well in other regions of the domain.

3 SPLINE APPROXIMATIONS

Spline functions have been used to address some of the difficulties mentioned in the previous section. The basic idea is that if global models are difficult to construct and often poorly behaved, it may be preferable to build a model piecewise using linear or low-order polynomials, each defined locally over some subregion of the domain. Because they are nonzero only in a part of the domain, they can represent local structure of functions that may not have easily-modeled global structure [4].

Such a set of spline basis functions in one dimension is given by:

$$1, x, (x - t_1)_+, (x - t_2)_+, \ldots, (x - t_K)_+$$

which leads to models of the form:

$$G(x) = a_0 + a_1 x + \sum_{k=1}^{K} a_{k+1} (x - t_k)_+$$

(In this notation, the subscript "+" means that the expression is assigned a value of zero if the argument is negative.) This type of spline is called a *truncated power spline*. The variables t_k are called "knots"; they are the locations where the spline functions subdivide the domain. The full basis set has a size (K + 2). A graph of one of these basis functions is shown in Figure 2.

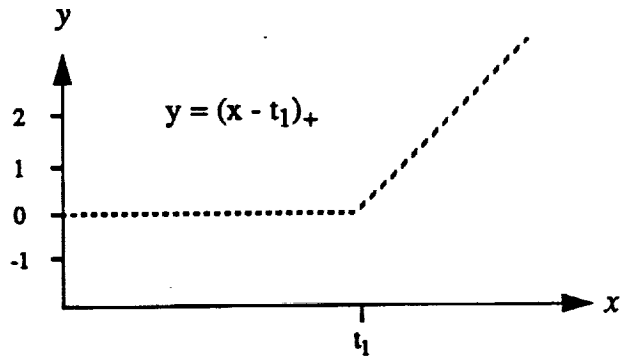

Figure 2: Spline Function. A spline function is zero over part of a domain, and a low-order polynomial over the remainder of the domain. This 1-power spline is continuous but has a discontinuous derivative. A q-power spline is continuous and has (q - 1) continuous derivatives.

Splines perform quite successfully in building low-dimensional models, but the extension to higher dimensions has proven, in the understated words of Friedman, "straightforward in principle but difficult in practice." Specifically, the standard extension of splines to higher dimensions requires $(K + q + 1)^n$ basis functions and the calculation of a corresponding number of coefficients; here, n is the number of input dimensions, K is the number of knots per dimension, and q is the order of the splines. Even for a relatively small number of dimensions, the computational costs of calculating the coefficients and the large number of data samples needed makes the procedure prohibitive.

4 THE MARS ALGORITHM

The MARS algorithm was developed to allow spline approximations in high-dimensional settings. The basic idea is to build the model using only a small subset of the $(K + q + 1)^n$ proposed basis functions. This is done by extending a partial model using an incremental search for the best new partition of the domain. This partitioning is repeated until a model with the desired number of terms is developed.

The algorithm begins with the linear model:

$$G_0(X) = a_0$$

At each partitioning step, the current model is extended by selecting: a basis function currently in the model; a dimension not currently partitioned in that basis function; and a knot location, assigned by selecting in turn the value for that dimension in each data sample. This triple (b, v, t) defines a possible extension to the current model:

$$G_{m+2}(X) = G_m(X) + a_{m+1} BF_b(X)(x_v - t)_+ + a_{m+2} BF_b(X)(t - x_v)_+$$

The coefficients of the newly generated model are computed using least-squares regression. All possible triples of (b, v, t) are tried; the model $G_{m+2}(X)$ which best fits the data samples is selected, and becomes the current model for further partitioning. A more detailed "C" description of the core MARS algorithm is given in Figure 3.[1]

The most computationally-intensive part of the MARS algorithm is the calculation of the least-squares coefficients for the newly proposed model. Thus, one estimate of the cost of building the final model is the number of least-squares regressions that must be performed. The upper limit on the number of models the MARS algorithm must generate and test at a given step is (N*m*n), where N is the number of data samples, m is the current number of basis functions in the model, and n is the number of input dimensions. If the number of basis functions in the final model is M_{max}, the maximum number of models generated is:

$$\text{max models} = (N \times n) \times \sum_{m=1}^{\frac{M_{max}}{2}} (2m+1) = (N \times n)\left(\frac{M_{max}^2}{4} + M_{max}\right)$$

1. Figure 3 contains what Friedman calls the *forward stepwise* portion of the algorithm; I do not describe the *backwards stepwise* portion in this paper. That procedure is a pruning process conducted on the model discovered by the forward stepwise algorithm; it gives some minor additional optimization. (In the experimental, however, both forward and backward stepwise sections were used before comparison with G/SPLINES.)

```
1   Model = constant_model ();
2   for (size = 1; size <= number_of_BFs; size += 2) {
3     lowest_score = INF;
4     for (m = 1; m <= size; m++) {
5       var = NONE;
6       while (var = next_unused_var(BF (m), var)) {
7         for (s = 1; s <= number_of_samples; s++) {
8           if (basis_function_nonzero(BF(m), data[s])) {
9             knot = data[s][var];
10            new = partition (Model, BF(m), var, knot);
11            least_squares_fit (new);
12            new_score = LOF(new, data);
13            if (new_score < lowest_score) {
14              best_model = new;
15              lowest_score = new_score;
16            }
17          }
18        }
19      }
20    }
21    Model = best_model;
22  }
```

Figure 3: The MARS Algorithm. The MARS algorithm is an incremental search to find the best new partition to the current spline model, starting with the constant model. The outer loop is over the number of desired partitions; the inner three loops choose a basis function, variable, and knot. The new model is tested after least-squares regression using a lack of fit (LOF) function, and the best new model is used for the next partitioning.

The good news is that the number of intermediate models is linear in both the number of samples N and the number of input dimensions n. The bad news is that this can still be a very large number of intermediate models, so the MARS algorithm can only be used with a relatively small number of data samples and input dimensions. Friedman claims the algorithm is effective for up to 1,000 data samples and 20 input dimensions; this places many interesting problems out of reach of the procedure.

Further, while the MARS algorithm is effective at creating well-performing models for many problems, models which cannot be reached through an incremental search are not discovered. Thus, functions which have a large number of non-linear interactions between the input variables may not be well-suited to modeling using the MARS algorithm.

5 G/SPLINES

The idea behind G/SPLINES is to use a genetic algorithm to do a search using full-size models rather than using the incremental search. The G/SPLINE algorithm starts with a collection of functional models, generated randomly. The coefficients for each model are determined using least-squares regression, and the lack of fit (LOF) over the data set is measured. The inverse of that lack of fit score is used as the fitness criterion. The main cycle begins by probabilistically choosing two parent models

based on their inverse LOF score. Crossover is used to generate a new model. Mutation operators may be used to add additional terms to the model. The worst scoring function in the collection is then replaced by this new model. A more detailed "C" description of the G/SPLINE algorithm is given in Figure 4.

```
1   for (i = 0; i < number_of_functions; i++) {
2     fcn[i] = random_function();
3     least_squares_fit(fcn[i]);
4     function_score[i] = 1.0 / LOF(fcn[i], data);
5   }
6   for (i = 1; i < number_of_cycles; i++) {
7     select_parents(function_score, &par1, &par2);
8     child = worst_function(function_score);
9     crossover(fcn[par1], fcn[par2], fcn[child]);
10    if (random_new_mutation())
11      add_new_BF(fcn[child]);
12    if.(random_merged_mutation())
13      add_merged_BF(fcn[par1], fcn[par2], fcn[child]);
14    least_squares_fit(fcn[i]);
15    function_score[i] = 1.0 / LOF(fcn[child], data);
16  }
17  model = best_function(function_score);
```

Figure 4: The G/SPLINE Algorithm. The G/SPLINE algorithm is a genetic search over a set of models, replacing the worst model with a crossover of two highly-rated models, using an initial setup of random functions. The models are tested after least-squares regression using the inverse of a lack of fit (LOF) function as the fitness score.

5.1 CROSSOVER

The core process in the G/SPLINE algorithm is a crossover algorithm. In G/SPLINE, two well-performing models are chosen, and the worst-performing model in the system is chosen for replacement. (While there are a number of possible procedures that could be used, basis functions seemed natural as the atomic unit in the crossover algorithm.)

The fitness function used was the inverse of the lack-of-fit function developed by Friedman in his MARS research. The lack of fit function is based on the least-squared error, with an additional penalty term related to the size of the model and the number of data samples. Without this penalty, the size of the models grows without bound, resulting in increased computational costs and increased risk of overfitting. Since both MARS and G/SPLINES use the same lack-of-fit function as their error measure, comparisons between MARS and G/SPLINES are more informative.

The process begins by probabilistically selecting two parents based upon the inverse lack-of-fit score. The two model parents are in this form:

$$\text{Model-1}(X) = a_0 + \sum_{k=1}^{M_1} a_k \phi_k(X)$$

$$\text{Model-2}(X) = b_0 + \sum_{k=1}^{M_2} b_k \beta_k(X)$$

In each parent model, we randomly choose a cut point, and select one of the generated two segments for inclusion into the child. We denote the selected segments of each models with the inclusive sets $[\text{Start}_1, \text{End}_1]$ and $[\text{Start}_2, \text{End}_2]$. We then construct the new child model as:

$$\text{Child-Model}(X) = c + \sum_{k=\text{Start}_1}^{\text{End}_1} d_k \phi_k(X) + \sum_{k=\text{Start}_2}^{\text{End}_2} e_k \beta_k(X)$$

The child model is a linear combination of basis functions derived from each parent. (Some genetic algorithms use the unselected segments to create an additional child; in this initial work, I found that creating both children did not appreciably improve the performance of the G/SPLINES algorithm.) Once the basis functions for the child are determined, the coefficients are derived using a standard least-squares regression.

5.2 MUTATION OPERATORS

As the crossover process proceeds, two effects are seen. First, the number of different basis functions is reduced as combinations of better-approximating basis functions propagate through the models. Second, the models often contain basis functions which contribute no benefit to the quality of the model and increases the cost of the least-squares computation. To counteract these effects I used three mutation operators: NEW, MERGE, and DELETION. After the standard crossover is performed, there is a probability that one or more of these mutation operators may be applied, resulting in the addition or removal of basis functions.

The NEW operator creates a new basis function by randomly choosing an input variable v, a sign s (+1 or -1), and a data sample $<t_1, ..., t_n>$. These parameters are used to create a basis function of the form:

$$BF_{new}(X) = (s \cdot (x_v - t_v))_+$$

The MERGE operator takes a random basis function from each parent, and creates a new basis function by multiplying them together, that is:

$$BF_{new}(X) = \text{randBF}(X, \text{par1}) \cdot \text{randBF}(X, \text{par2})$$

It is through the MERGE operator that basis functions containing multiplicative terms are introduced.

For both ADD and MERGE, the newly generated basis function is added to the new model generated by the crossover process. This has a cost for functions, which find the crossover search slowed by the additional variance caused by this additional factor. However, in the longer term, this keeps the pool of basis functions from becoming dangerously small, and aids the process in finding high-quality approximations.

The DELETION operator ranks the basis functions in order of minimum maximum contribution to the generated model. Unlike the other two mutation operators, DELETION requires an additional least-squares operation to calculate the coefficients of the generated model. However, while it doubles the number of least-squared operations, it also speeds convergence and encourages compact models.

5.3 ALPHABET CARDINALITY

Considerable study has gone into developing effective codings for genetic algorithms, but coding design remains an art. In this work, an alphabet of high cardinality seemed the most appropriate: the basis functions are the atomic unit in the crossover process, and there is an extremely large number of possible basis functions. However, this choice places the work in the middle of an ongoing debate regarding the size of the alphabet best suited for genetic algorithms.

One school, of which Goldberg [6] is representative, is "almost obsessed with the idea of binary codings." This is not simply a preference for binary representations, but rather a rejection of other representations: "... in general the use of high-cardinality alphabets so severely reduces implicit parallelism that it is inappropriate to call these schemes genetic algorithms in the sense of Holland."

In reality, this argument is a poorly-disguised version of the holism/reductionism debate; what is the "appropriate" level of description? [7] Goldberg may be semantically correct in stating that high-cardinality alphabets do not result in "genetic algorithms in the sense of Holland," but he is mistaken to assume that the use of a high-cardinality alphabet "severely reduces implicit parallelism"; by using a higher-level description, the search (and resulting implicit parallelism) is simply being conducted on a different level of representation, not eliminated. Arguments can certainly be given for and against the use of different alphabets in different situations, but across-the-board claims of superiority for one side or another should be viewed with great suspicion.

6 EXPERIMENTAL

A training and a test data set were created for the experiments. Each data set contained 200 data samples. The function modeled was from Friedman [5]:

$$f(X) = 10 \cdot \sin(\pi x_1 x_2) + 20 \cdot (x_3 - 1/2)^2 + 10 \cdot x_4 + 5 \cdot x_5 + \sum_{n=6}^{10} 0 \cdot x_n$$

The data contained 10 predictor variables; the response is dependent on the first five variables, and independent of the next five variables. Noise was added to the response so that the signal/noise ratio was 4.8/1.0.

The domain for G/SPLINE was a population of 200 functions. Each function was initialized with 10 basis functions generated using the NEW mutation operator. After each crossover, there was some probability that one or more of the mutation operators would be applied to the child. The model with the lowest error on the training data set was chosen for testing against the testing data set.

The model generated by the MARS algorithm was reduced using his backward-stepwise algorithm, then applied to the testing data set.

6.1 Experiment 1

The first experiment was designed to see if the G/SPLINE algorithm could compete with the MARS algorithm in being able to generate models with comparable quality in an environment that favored the MARS algorithm.

The least-squared error versus the number of least-squared regression operations was graphed. The results of this experiment are shown in Figure 5.

This experiment illustrates the rapid learning capability of the G/SPLINE algorithm relative to the MARS algorithm. After 5000 least-squares operations, the MARS model has placed its first knot; the G/SPLINE algorithm is already nearing its asymptote. The final MARS model slightly outperformed the G/SPLINES model, but only after doing an order-of-magnitude more least-squares operations.

Did the model found by the genetic search use relationships in the data set reflective of the underlying function, or only discover easily-modeled patterns in random data? Figure 6 addresses this issue; it demonstrates how the variable use in the set of discovered models indeed reflects the underlying structure of the generating function.

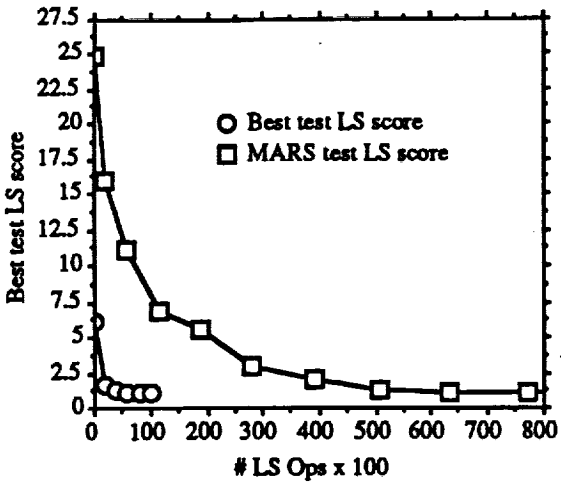

Figure 5: MARS vs. G/SPLINE. Measured in the number of least squares regression operations that must be performed, the G/SPLINE algorithm performs significantly better than the MARS algorithm. The G/SPLINES algorithm was close to convergence after 4,000 least-squared operations (LS ops), and showed no further improvement after 10,000 LS ops. The MARS algorithm was close to convergence after 50,000 LS ops, and showed no further improvement after 80,000 least-squares operations. The final least-squared error of the best G/SPLINES model was 1.17; the final least-squared error of the MARS model was 1.12. The optimal model would have a least-squared error of 1.08 on the test set.

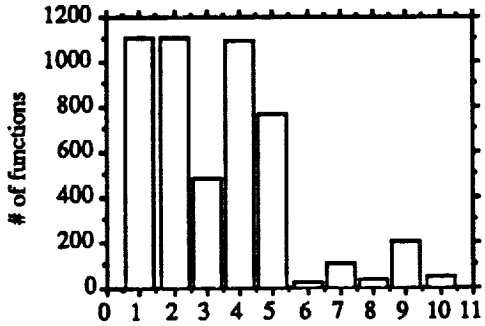

Figure 6: Variable use. The graph is the index versus the number of times the variable is used in some basis function in a discovered model. The variables were counted after 5,000 genetic crossover operations. The underlying function depends on the first five variables, and does not depend on the next five variables. In this case, the use of the variables reflects the underlying function.

The performance of G/SPLINE is reminiscent of the performance curves comparing Genetic Algorithm-based systems with Backpropagation-based neural networks [3]. In these systems, the genetic search is rapid at the beginning of the process, far outperforming the neural network. It is only after the problem is well-developed that the backpropagation algorithm begins to compete with the genetic search; eventually, the Backpropagation-based neural network slightly outperforms an algorithm based solely on genetic search. It is possible that the MARS algorithm and the G/SPLINE algorithm have a similar relationship with respect to their performance and speed.

6.2 Experiment 2

This experiment is identical to the first experiment, with the exception that the function is changed to have 5 dependent and 95 dependent variables, for a total of 100 predictor variables. The size of the data sets was unchanged, with each containing 200 samples. (Note that this change increased the problem size beyond Friedman's stated capabilities of the MARS algorithm.)

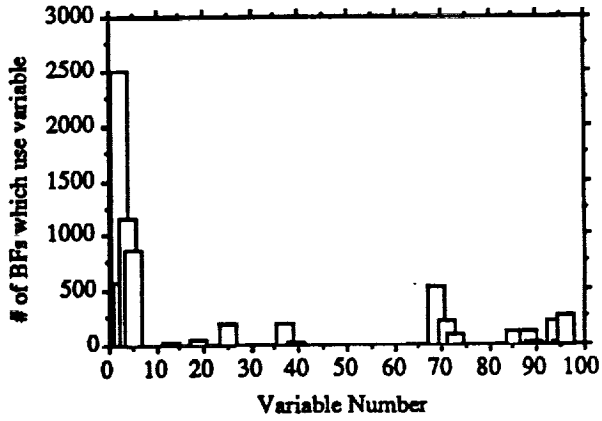

Figure 7: Variable elimination. The graph is the index versus the number of times the variable is used in some basis function. The variables were counted after 10,000 genetic crossover operations. The underlying function depends on the first five variables, and does not depend on the other 95 variables. The five dependent variables were the top five variables in terms of actual use in basis functions.

As Figure 6 shows, even with only 200 samples of data, the G/SPLINES algorithm still discovers the relative importance of the 5 dependent variables amidst the 95 independent variables.

6.3 Experiment 3

In this experiment I wanted to study the effect of sample set size on the rate of elimination of the independent variables.

This experiment is identical to the first experiment, except that two different data set sample sizes were used. The first set was the standard 200-sample set, and the second a 50-sample set.

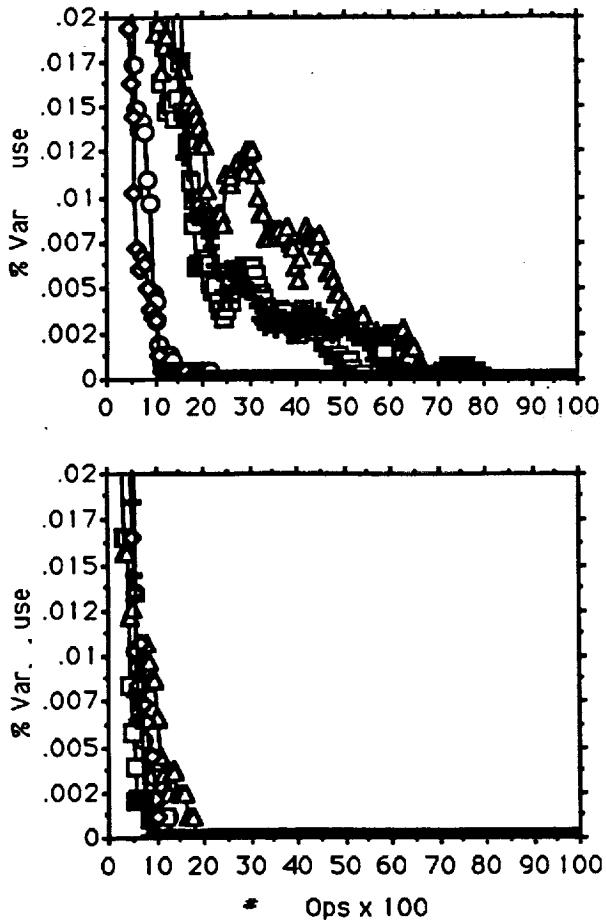

Figure 8: Variable elimination and sample size. The graphs are the number of genetic operations versus the percentage use of the five independent variables. The top graph is for a sample size of 200; the bottom graph for a sample size of 50. In the top graph, the variables are gradually eliminated. For the smaller sample size, the variables are eliminated rapidly.

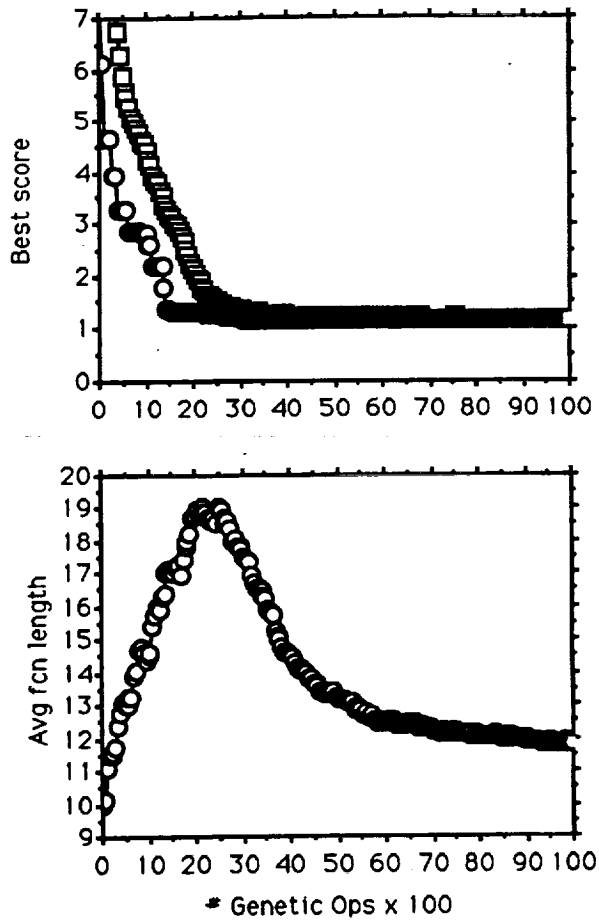

Figure 9: Function length. The top graph is the number of genetic operations versus the least-squared error (squares: test set, circles: training set). The bottom graph is the number of genetic operations versus the average model length (in basis functions). There is a rapid increase in average function length, followed by a slower decrease after the score is minimized.

Figure 6 shows the surprising result that the independent variables are eliminated from consideration *faster* when we have only 50 samples of data rather than 200. This is because G/SPLINES does not eliminate variables through positive identification of them as independent of the input, but rather, these variables get left behind as variables with more predictive power are preferentially selected for crossover. With smaller data sets, there is more pressure for smaller models, which causes a fiercer competition and earlier removal of basis functions which do not substantially contribute to a model's performance.

6.4 Experiment 4

In this experiment I wanted to study the effect of the genetic algorithm on the size of the generated models. The experimental conditions were identical to those of the first experiment.

Figure 6 shows that there is a rapid increase in model size until the score is nearly minimal; after that, there is a slower but consistent decrease in model size. This is likely due to pressure from the genetic algorithm; a compact representation is more likely to survive the crossover operation without loss. Thus, the genetic algorithm encourages compactness of representation in addition to the advantages such compactness affords during scoring. Since we are often interested in compact models (everything else being equal), this is a beneficial effect.

7 CONCLUSIONS

It is difficult to properly compare the utility of two algorithms, even when they share many similar features or are applied to the same data sets. Such comparisons are too often uninformative and unneccessarily harsh to the losing algorithm (which is usually *not* the algorithm developed by the author). My goal in this work was not to

supplant the MARS algorithm, which I find elegant in concept and has a proven record of success in practice. Rather, my goal was to extend the reach of the algorithm by proposing a variant that may not suffer some of the limitations of the procedure as proposed by Friedman, yet may retain most of its advantages. I believe I was successful, but will leave the final judgement to the reader.

In this paper, the problems we selected were relatively small, and generated relatively small models, containing perhaps a dozen basis functions. As the complexity of the problems grow, the necessary size of the model will also grow; a genetic approach such as G/SPLINE may be the most effective technique for deriving such models. Similarly, the MARS model is most effective when most of the predictor variables have additive effects on the response; the G/SPLINE model, since it is not based on the incremental search, may be better suited to discover appropriate models in this case.

The number of least-squares regressions performed was proposed as a measure of the inherent efficiency of the algorithms. The structure of the MARS algorithm is such that the cost of the least-squares regression can be greatly reduced, so direct comparisons of the number of least-squares regressions is not truly a measure of the amount of computation involved in executing the algorithms. Still, even with these improvements, the MARS algorithm cannot effectively handle large (>1000) data samples or large (>20) numbers of input variables. It is appropriate to look for algorithms which keep the many advantages of the MARS approach while overcoming its limitations.

Finally, while the G/SPLINE algorithm uses linear splines as its basis functions, there is no reason the algorithm could not use non-linear splines and non-spline basis functions. While splines have attractive properties that make them generate useful models in many circumstances, there is no reason that other functions, which may perform well in circumstances where splines fail, should not be included in the set of possible basis functions. Work on this extension to G/SPLINE is ongoing.

Program Availability

The WOLF program implements the G/SPLINE algorithm. This program currently runs under UNIX on Sun and Silicon Graphics minicomputers, and under THINK/C 4.0 on the Apple Macintosh II microcomputer. The UNIX version of the software and data is available by FTP on the INTERNET by sending mail to drogers@riacs.edu. The Macintosh version is available on floppy disk for a $20 copying fee.

It is my goal that the timely sharing of both the software and the data sets will encourage comparison of this work to other work, speed the dissemination of the algorithm, and encourage others to similarly share their algorithms and data. My rapid progress in developing this research program was due in part to Dr. Friedman's policy of openly releasing his software (admittedly comment-free...) for distribution; I encourage others to join me in following that excellent precedent.

Acknowledgments

This work was supported in part by Cooperative Agreements NCC 2-387 and NCC 2-408 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA). Special thanks to Doug Brockman (who shared my enthusiasm even though he didn't know what the hell I was up to), and to my father Philip Rogers, who made me want to become a scientist.

References

[1] Bentley, J., "Multidimensional Binary Search Trees used for Associative Searching," *Communications ACM*, 18, pp. 509-517, 1975.

[2] Breiman, L., Friedman., J., Olshen., R., and Stone, C., *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[3] Davis, L., *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann, Los Altos, CA, 1987.

[4] deBoor, C., *A Practical Guide to Splines*, Springer-Verlag, New York, NY, 1979.

[5] Friedman, J., "Multivariate Adaptive Regression Splines," Technical Report No. 102, Laboratory for Computational Statistics, Department of Statistics, Stanford University, November 1988 (revised August 1990).

[6] Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[7] Hofstadter, D., *Gödel, Escher, Bach: an Eternal Golden Braid*, Basic Books, New York, NY, 1979. [Chapter X has a formal discussion of levels of description, but I recommend the section titled "...Ant Fugue" for a more informal and inspired discussion on reductionism.]

[8] Holland, J., *Adaptation in Artificial and Natural Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[9] Ralston, A., and Rabinowitz, P., *A First Course in Numerical Analysis*, McGraw-Hill, New York, NY, 1978.

[10] Sanger, T., "Basis-Function Trees as a Generalization of Location Variable Selection Methods for Function Approximation," Proceedings of the 1990 Neural Information Processing System Conference, Denver, CO, 1991.