Remembrance of Circuits Past: Macromodeling by Data Mining in Large Analog Design Spaces

Hongzhou Liu, Amit Singhee, Rob A. Rutenbar, L. Richard Carley
Dept. of ECE, Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213 USA
{hongzhou, asinghee, rutenbar, carley}@ece.cmu.edu

ABSTRACT

The introduction of simulation-based analog synthesis tools creates a new challenge for analog modeling. These tools routinely visit 10^3 to 10^5 fully simulated circuit solution candidates. What might we do with all this circuit data? We show how to adapt recent ideas from large-scale data mining to build models that capture significant regions of this visited performance space, parameterized by variables manipulated by synthesis, trained by the data points visited during synthesis. Experimental results show that we can automatically build useful nonlinear regression models for large analog design spaces.

CATEGORIES AND SUBJECT DESCRIPTORS

B.7.2 [Integrated Circuits]: Design aids—verification

GENERAL TERMS

Algorithms

I. INTRODUCTION

The use of simplified macromodels for analog circuits, both to accelerate simulation and to enhance early design exploration, has a long history in mixed-signal design. The earliest techniques for macromodel construction relied on design expertise to create a suitably simplified circuit model, and the analytical equations needed to map the performance of the full circuit into parameters for the macromodel. More recent techniques mix design expertise about model structure with nonlinear regression (i.e., curve fitting) to fit macromodel parameters from samples of the full circuit's performance obtained from simulation. Two recent developments suggest a need to revisit this area: the recent standardization of analog/mixed-signal hardware description languages, and the recent introduction of commercial analog synthesis tools.

The introduction of standardized behavioral simulation languages for mixed-signal systems (*e.g.*, Verilog-AMS and VHDL-AMS [1],[2]) offers designers the ability to mix device-level models, analog behavioral models, and digital blocks, all in the same simulation environment. These languages are widely expected to be significant enablers for a more top-down mixed-signal design style. But to really exploit these AMS languages, we need a more rigorous approach to building macromodels for arbitrary analog blocks. For linear systems, the last decade has seen enormous progress in the construction of reduced-order models with mathematically guaranteed accuracy, [3]. For general nonlinear systems, we have no such mature techniques. Theoretical work on nonlinear reduced-order modeling is still quite new (e.g., [4]), and as yet limited in its applicability.

A different challenge is posed by the recent development of practical analog synthesis tools [5]. These tools take a fixed circuit topology and solve for the sizing/biasing parameters needed to meet a set of performance specifications. Today's synthesis techniques come in two broad styles. *Equation-based* synthesis techniques require that each circuit topology we seek to size be represented as a set of analytically tractable performance equations. The most successful of these use ideas from geometric programming [6]. The disadvantages of this approach are the need to build custom equations for each new topology (and the underlying device models as well), and the inability to capture all performance requirements in an accurate form. In contrast, *simulation-based synthesis* ([7]-[11]) uses simulator-in-the-loop numerical search techniques that fully simulate each visited solution candidate. They are capable of sizing any circuit that can be simulated, need no partial sizing starting solution, and reuse the same qualified verification environment used for manual design sign-off. Distributed workstation parallelism renders the overall approach tractable.

Simulation-based synthesis creates a wholly new opportunity for modeling: each synthesis run visits between 10^3 and 10^5 fully simulated samples of the design space for a given circuit topology. If we synthesize just one uniquely sized circuit per day for a year, we might easily visit a few million samples of a design space. *What can we do with all these circuit data points?*

Today's *ad hoc* model simplifications and low-dimensional curve fitting strategies seem ill-equipped to exploit this opportunity. Of course, building regression models from simulations is not new: response surface methods for design centering have been in widespread use for years [12], [13]. What is new here is the *scale* of the problem. In a typical centering scenario, we try to avoid simulating more than a few hundred different circuits to build a low-dimensional local response surface. In contrast, a single synthesis run may generate 10-100X more circuit samples, and range widely over the entire design space.

The question we address in this paper is the how to build models that can capture significant regions of this performance space, parameterized by variables manipulated by synthesis, trained by the data points visited during synthesis. We build such models by adapting recent ideas from large-scale *data mining* [14], which focuses on techniques for extracting patterns, predictive formulas, or classifiers from large amounts of high-dimensional data.

The remainder of the paper is organized as follows. Sec. II reviews relevant background work. Sec. III introduces our data mining formulation. Sec. IV presents a small illustrative example. Sec. V presents experimental results from real synthesis data. Finally, Sec. VI offers concluding remarks.

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

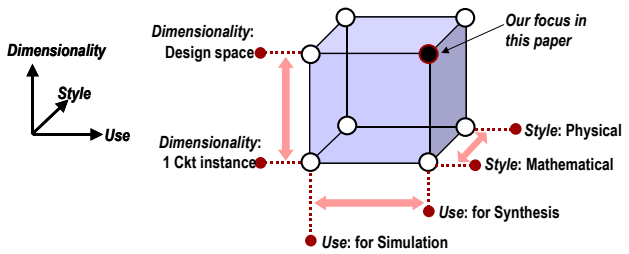

Fig. 1. A Simple Taxonomy of Analog Macromodels

II. Background: Macromodeling

The term "macromodel" has historically been used to refer to a variety of different analog modeling problems, with different associated model-building techniques, and intended end applications. Fig. 1 offers a simple taxonomy to categorize these different approaches, and clarify our own goals. We identify three separate axes in this taxonomy:

- *Use:* distinguishes between models built primarily to accelerate simulations, and those built to enable wider exploration of the design space in synthesis. Of course, these are not always mutually exclusive; the distinction usually appears on the other two axes.
- *Style*: distinguishes between models that are built explicitly as physical circuits, from those that are only intended to "fit" some of the original circuit's "true" responses, without any underlying physical model. Physical macromodels usually *are* circuits--albeit simpler circuits. Mathematical macromodels are usually obtained from some form of nonlinear regression, and offer no guarantees that all of their internal behaviors can be physically realized.
- *Dimensionality*: distinguishes between models that represent a single circuit (called an *instance*) from models intended to represent a parameterized range of circuits (called a *design space*). Instance models are built for simulation; design space models for optimization.

We can categorize relevant prior work using this simple taxonomy. Analog macromodeling initially focused on accelerating simulation. The earliest work built simplified physical macromodels for single circuit instances, e.g., the classical Boyle opamp model [15]. To handle more complex circuits and devices, later efforts turned to curve-fitting to obtain the parameters for these physical models from samples of the original instance's simulated performance [16]. Unfortunately, we often lack a good physical model to parameterize. Automated extraction of a reduced-order model is now easily done for linear systems [3]. This not the case for nonlinear systems, though recent efforts have appeared [4],[17]. The common strategy is to apply nonlinear regression techniques [18] to fit either a conjectured functional form, or use so-called *black box* modeling in which the regression itself chooses the functional form.

Synthesis-directed macromodeling usually focuses on design space models, since the goal is to explore or optimize over some space of circuit parameters. In the context of circuit tuning, response surface methods fall into this category [12], [13]. Similar ideas have been used for microwave synthesis [19]. Harjani [20] appears to be the first to explicitly build carefully constructed nonlinear regression models of analog design spaces, for use in hierarchical analog synthesis [21]. This work minimized the number of circuit simulations via careful design of experiments (DOE [22]), and proposed a regression formulation based on radial basis functions. More recently, Daems [23] developed a similar strategy for design space modeling, using a different DOE model, and regression via posynomials, motivated by [6].

In this paper, we focus on the problem of building models of analog design spaces, emphasizing synthesis rather than simulation applications. As a result, we are willing to trade some modeling accuracy for the ability to model a larger region of design space. Prior efforts here [19],[20],[23] relied on two critical assumptions: (1) simulations to obtain samples of the design space are expensive and must be minimized; and (2) model construction can control directly where to sample in this design space. In contrast, we start with many more fully simulated samples of the design space (e.g., 10,000 versus 100). However, we do not control where these samples are located, since our data is the by-product of circuit synthesis. This wealth of synthesis-produced data creates both opportunities and problems.

III. Macromodeling By Data Mining

In classical nonlinear regression, we have data points of the form $(\boldsymbol{x}, y)$, where $\boldsymbol{x} \subseteq \Re^n$ and $y \in \Re$, which represent samples of some unknown, high-dimensional function $y = F(\boldsymbol{x})$. We seek to construct an approximation to this function $R(\boldsymbol{x})$, called a *regressor*, which minimizes some *error function* that penalizes the difference between the true function and our approximation (*e.g.*, sum of squared errors), measured over a population of sample data points $P = \{(\boldsymbol{x}, y)_i\}$:

$$\sum_{(\boldsymbol{x}, y) \in P} \text{error}[y, R(\boldsymbol{x})] \qquad (1)$$

To construct a suitable regressor we must solve three problems:

- *Model selection*: select an appropriate functional form for $R(\boldsymbol{x})$
- *Model fitting*: determine parameters for this model, so as to minimize overall fitting error (1), using the training data from P.
- *Model validation*: using a different set of testing data, verify how well the model fits data points outside of the training data points.

The essential problem is that for a large population of high-dimensional data points, it becomes extraordinarily difficult to find a suitable functional form for the regressor that can adequately fit the data, while remaining simple enough to allow us to solve for the fitting parameters. This is the principal motivation behind the radial basis functions in [20] and posynomial-approximated signomials in [23]; these are both forms with relatively tractable fitting procedures.

However, recent work in the data mining community suggests a different direction for coping with the problem of large, high-dimensional data sets. Rather than struggling to build a single regressor capable of fitting well across a very large sample space, we build instead a *committee of regressors*, each of which fits very well in some portions of the design space, and less well in others [14]. We predict the output value for any new data point by *combining* the individual regressors, in a procedure called *voting*. The mechanics of committee construction and voting are attractively simple. To build the committee of regressors, we use a powerful, relatively new statistical technique called *boosting* [14],[24]; to vote the regressors, we adapt ideas from *instance-based methods* [25]. We describe these ideas next.

A. Building a Committee of Regressors via Boosting.

The original idea was first developed by Freund and Schapire [24] and applied to so-called *categorical* classification problems, in

which the model to be fit has a discrete two-element output range, e.g., {0, 1}. The idea is illustrated in Fig. 2. We begin with some data set S with N weighted training samples. Each sample $(\boldsymbol{x}, y)_i$ has weight w_i, all samples equally weighted to start. We *sequentially* fit a set of regressors $R_t(\boldsymbol{x})$, each more heavily weighting those data points that were poorly fit in the previous iteration. Thus, after each iteration, we up-weight those points that were poorly fit, and down-weight points that were well fit. This has the interesting side effect of allowing well-fit samples to *vanish* from the population as boosting progresses: these points are so well fit by prior regressors that we no longer want to focus fitting efforts on them. The number of iterations t varies by application, but it is not uncommon to run hundreds of boosting cycles [14]. At the end, we have a committee of regressors, which we combine via weighted voting to construct the final output. The boosting method is easy to implement, and independent of the form of the individual regressors, $R_t(\boldsymbol{x})$. With a carefully constructed weighting update, boosting also comes with some surprisingly good theoretical bounds on its performance [24].

For building macromodels for large analog design spaces, a boosted committee of regressors is immediately attractive. However, boosting has to date been used primarily for *classification*--fitting regressors that map real-valued inputs to a small set of discrete category labels. The assumption of a discrete output set is critical to the construction of theoretically strong voting methods for boosting [24]. On the other hand, our interest is to apply boosting to quantitative nonlinear regression. Prior work here is more sparse, with Drucker [26] perhaps the most notable.

We use the boosting iteration from [26], illustrated in Fig. 3. However, we suggest a different mechanism for voting the individual regressors, discussed in the following subsection.

As a practical matter, note that the choice of form for individual regressors $R_t(\boldsymbol{x})$ will determine whether one can weight the training data for suitable effect. If not, the alternative is to *resample* the data. With suitable normalization, we can interpret weight w_i on each point as a probability. Given the data set S_t used to build regressor $R_t(\boldsymbol{x})$, construct S_{t+1} via sampling from S_t with w_i as the probability of selection for any point. Well-fit points, having small probabilities, may not be sampled. Poorly fit points, with large probabilities, may be *duplicated* in the population. When we cannot upweight an individual data point, we can replicate it in the population to increase its importance during fitting. To enhance this effect, we can choose a population size for each S_t larger than $N=|S|$, the size of the original data set.

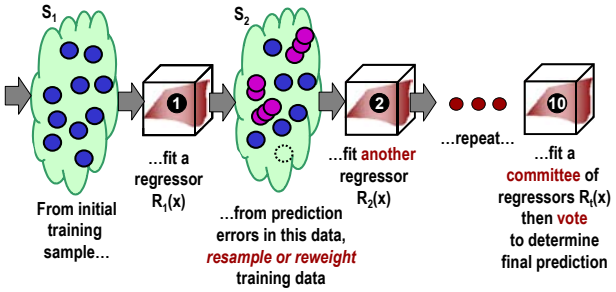

Fig. 2. Boosting for Regression

1. Given data set S with N training points of the form $(\boldsymbol{x}, y)_i$, with $\boldsymbol{x}=\langle x_1, x_2, ..., x_n\rangle$. Set weight $w_i=1/N$ for each point.
2. Set $t=0$. Set $S_0 = S$.
3. Set $t=t+1$. Construct set S_t by sampling AN points $(A>1)$from S_{t-1} with the probability that we select point $(\mathbf{x}, y)_i = w_i$.
4. Fit a new nonlinear regressor $R_t(\boldsymbol{x})$ to data in S_t
5. Calculate (normalized) training error for each point $(\boldsymbol{x}, y)_i$ as
$$L_i = \text{error}(y_i, R_t(\boldsymbol{x}_i)) = \frac{|y_i - R_t(\boldsymbol{x}_i)|}{\max_{(\boldsymbol{x}, y)_j \in S_t} |y_j - R_t(\boldsymbol{x}_j)|}$$
6. Calculate average weighted training error for entire data set S_t
as $L = \sum_{(\boldsymbol{x}, y)_j \in S_t} w_j L_j$
7. Calculate *confidence parameter* for $R_t(\boldsymbol{x})$ as $\beta = L/(1-L)$
Note: *smaller* β implies *higher* confidence in the fit for $R_t(\boldsymbol{x})$
8. Update weight for each data point $(\boldsymbol{x}, y)_i$ as $w_i \Leftarrow w_i \beta^{(1-L_i)}$
9. If we want to boost another regressor, go to step **3**.

Fig. 3. Algorithm for Sequential Boosting Iteration [26]

B. Voting a Committee of Regressors via an Instance-Based Method

Prior efforts in both classification and regression have assumed that *all* the regressors in a boosted committee should be combined to create the final voted prediction. This fits well with categorical data, where we literally "outvote" the poor predictors. We argue that this is inappropriate when we know that our input data is likely to be sparse and of high dimension, and for circuits, is sampling very nonlinear behaviors. We want an alternative that selects the "best" set of regressors to combine.

Another technique from data mining can be applied here. In classical regression, we use our training data to build a suitable nonlinear regressor, but then we only use the analytical form of the regressor to make future predictions--the training data itself is abandoned. Like the scaffolding used to construct a building, we expect the scaffolding to be removed when the building is completed. This need not be the case: we might instead use the training data itself as an integral part of the final regressor. These are called *instance-based* methods; our strategy was influenced by the instance-based locally weighted polynomial regression ideas from Moore [25].

Fig. 4 shows our instance-based heuristic for combining regressors. For any new data point $\boldsymbol{x}$**'**, we look up the K nearest neighbors of $\boldsymbol{x}$**'** in the training data, using ordinary Euclidean point-to-point distance. With each of these training samples $(\boldsymbol{x}, y)_j$ we associate the index $t(j)$ of the best boosted regressor from among our committee of boosted regressors $R_t(\boldsymbol{x})$. Thus, the K nearest neighbor data points

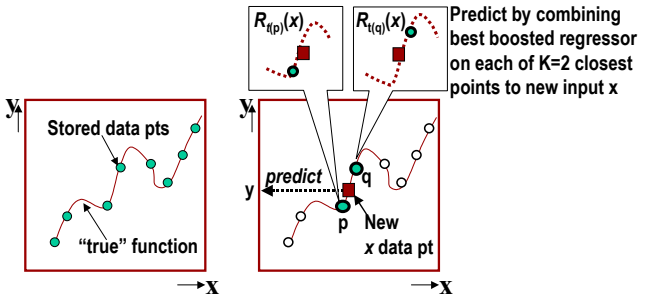

Fig. 4. Instance-Based Voting with K=2 Nearest Neighbors for 2-D Nonlinear Function

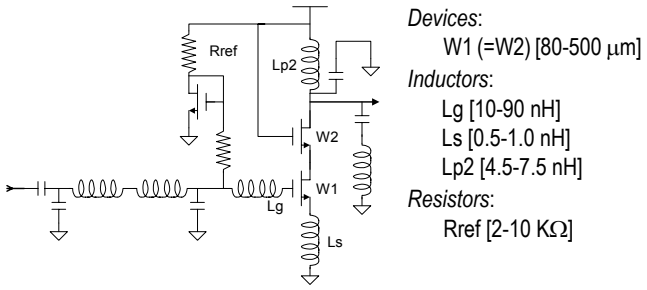

Fig. 5. Simple Example Circuit: RF LNA

also specify (at most) K different regressors that we know fit *well* in this local region. This has two very useful advantages:

- *Efficient scaling to large data sets:* Although the size of the training data set itself may be quite large, we only need to use a few data points for any given prediction. Nearest-neighbor lookup in high dimensions is a well studied area, and there are several data structures that make look-up efficient, e.g., [27]. In our work, we use simple linear scans through all the data points.
- *Many regressors are fit, few are evaluated*: we are free to fit a large number of boosted regressors if this is necessary to "cover" our design space. But for any given prediction, we only need the regressors on the nearest K neighbors to compute our new data point. Of course, we also hope--but cannot guarantee--that we need fewer than K regressors in many well-fit regions of our design space.

These K nearest-neighbor points $(\boldsymbol{x}, y)_{n1}$, $(\boldsymbol{x}, y)_{n2}$, ..., $(\boldsymbol{x}, y)_{nK}$, specify K regressors $R_{t(n1)}(\boldsymbol{x})$ through $R_{t(nK)}(\boldsymbol{x})$. How might we combine these into a single numerical prediction? Simple averaging proves to be a workable scheme. However, when the data is of high dimension and sparse, even a few "nearest" neighbors may not be all that near. As well as an appropriate, empirically chosen value for K, it is also useful to weight each local regressor inversely with its distance from the point to be predicted. If the set of distances to the K nearest neighbors is $\{d_{n1}, d_{n2}, ..., d_{nK}\}$, then we vote our K local regressors as

$$y_{predicted} = \sum_{i=1,K} \left[\frac{(d_{ni})^{-1}}{D} \right] R_{t(ni)}(\boldsymbol{x}') \qquad (2)$$

where $D = \sum_{i=1,K} (d_{ni})^{-1}$. Note that if each neighbor has a unique best local regressor, then this simply favors the regressors on the closer points. But in the opposite case, where each neighbor chooses the same regressor, this weighting does not distort the prediction.

C. Fitting Each Local Regressor

The final component of our regression strategy is the choice of the individual regressors $R_t(\boldsymbol{x})$ for each boosting iteration. It is again worth emphasizing that both boosting and our instance-based voting heuristic can be applied to any choice for the regressor.

We use a standard feed-forward neural network for each regressor [28]. Neural nets have a reputation for successful fitting for highly nonlinear functions, and are reasonably close to "black box" models in that the structural choices needed to select a neural network model are relative few. They have also been successfully used in other circuits-oriented modeling and optimization applications [19]. We use two hidden layers (10 sigmoid neurons per layer) and one output layer (1 linear unit) as the network architecture. Our implementation is based on the Matlab Neural Network toolbox [30].

IV. A Simple Macromodeling Example

We first illustrate our macromodeling formulation with a simple RF LNA circuit shown in Fig. 5. The circuit has 5 independent sizing parameters, variable over the ranges given in the figure The circuit is realized in 0.35μm TSMC CMOS.

Since the problem is rather small, we simply generate uniform random samples over the 5-dimensional hypercube defined by the input ranges. We generated 2000 total samples of the performance of this circuit (each of which required several SPICE runs), and used a randomly selected half to train our regressors, and reserved the other half to test (validate) our regressors. As does [23], we omit circuits whose random sizing renders them improperly biased (*e.g.*, transistors turned off). Fig. 6 shows the results using 10 boosting cycles to vote up to 10 boosted neural net regressors. We show both mean and maximum errors. (Note that in contrast to [23], this is a rather harsher set of metrics: there are *always* some very poorly predicted points for any regressor, and we evaluate on testing data *not* seen during model construction.) The data are interpreted as follows:

- **train1:** the leftmost column on each plot shows the error from fit-

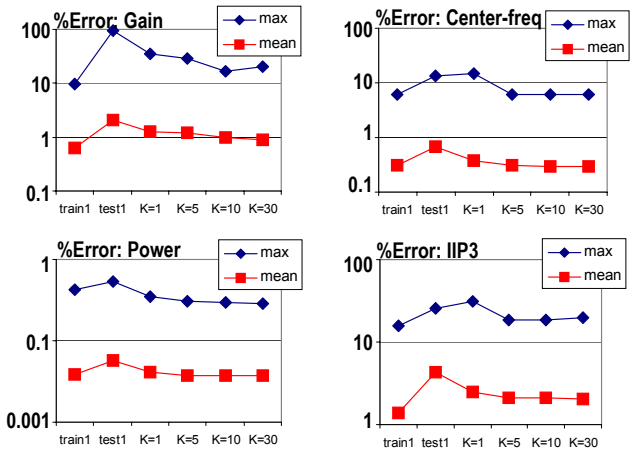

Fig. 6. Regression Results for Simple RF LNA Circuit

ting one single neural net regressor to the training data itself. We expect this to provide the best fit, since we evaluate only training data.

- **test1**: the second plot column to the right show the error from using only this single neural net regressor to predict the points in the testing data set. We expect this to be inferior, since this is data *unseen* during fitting. These are the points against which we need to compare and validate our new fitting techniques.
- **K=1,...K=30:** the rightmost 4 plot columns show the errors from predicting outputs over the testing data, using instance-based voting of the *K* nearest-neighbors from the training data, each labeled with the best from among 10 boosted neural net regressors. We evaluate K as 1, 5, 10, and 30.

The results are encouraging. For gain, center-frequency, power, and IIP3, boosting improves both the maximum and mean errors over the use of a single conventional neural net predictor. Usually, *K*=1 or *K*=5 nearest neighbors are sufficient: we select the local 1 or 5 best regressors and combine them, via inverse-distance weighting. Setting *K* too large usually has the effect of increasing the error: we are overfitting, trying to use "too much" model for the data. These extra neighbors are too far away to be of use in improving the local prediction accucuracy.

The log error scale of Fig. 6 obscures the impact that boosting has on the actual distribution of the errors; we illustrate this more directly in Fig. 7 by showing histograms of the prediction error at each testing point for the IIP3 fitting experiment of Fig. 6. Note how even *K*=1 (select the best boosted regressor on the *single* training point nearest the testing point) improves the average error--though not the worst-case error in this case. *K*=5 trims the worst errors from the right-side tail of the distribution. Subsequent boosting with *K*=10 has minimal effect, and boosting with *K*=30 neighbors starts to degrade the fitting, as shown in the increased spiking in the distribution.

V. Macromodeling A Synthesis Design Space

We turn now to a much larger experiment. Fig. 8 shows a circuit automatically sized in a commercial simulation-based analog synthesis tool [29]. The schematic has roughly 50 devices, and has been sized to achieve roughly 450MHz with power under 15mW. We have approximately 37,000 samples of the design space for this circuit, as a result of synthesis. Each data point has 27 independent design variables, and 12 circuit performance outputs. In contrast to prior fitting experiments of which we are aware, this is a much larger, much higher-dimensional case study.

It is also different for a reason fundamental to synthesis: synthesis tools are designed to *converge* to a good solution [7]-[11]. No particular invocation of synthesis need necessarily explore any region of the space beyond that needed to find a good solution. To emphasize this, we fit two portions of the this data set: the first 20% of the data points (time-ordered by synthesis) and the last 20%. This creates two populations of roughly 7500 samples of the design space. One salient difference from the experiments of the last section is that these two data sets are very different in character. Samples from the first 20% of the run range more widely over the design space (as expected, since search ranges widely at the start). Samples from the last 20% of the run cluster more closely about the final solution. As a consequence, we see much larger worst-case prediction errors for the first 20% because the of existence of more "outlier" points.

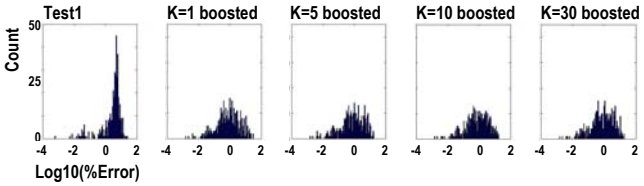

Fig. 7. Error Histograms for IIP3 Data from Fig. 6

We fit using the same techniques as in the previous section. We randomly select 3/4 of each data set to use for training, and reserve the remaining 1/4 for validation of our regression fit. We boost 10 cycles of regressors. Fitting the next neural network in each boosting cycle requires roughly 1 hour of CPU time, in our current Matlab implementation. We evaluate using instance-based nearest-neighbor voting for *K*=1,5,10,30. Results appear in Fig. 9. Let us briefly consider each experiment:

- **Gain**: boosting at *K*=1 or *K*=5 clearly helps for the first data set, but interestingly, does little to help the last data set. Mean error declines very slightly (but the fit is quite good in the first place), while maximum error increases.
- **Power**: boosting is more clearly helpful on the first data set. On the the later data, boosting helps the mean slightly, but at the cost of an increased maximum error. Bias/variance trade-offs are a common by-product of regression. A single regressor probably suffices.
- **Slew rate:** this is clearly a very challenging fit. Boosting improves the mean, but at the cost of some deterioration in maximum error.
- **Unity Gain Frequency (UGF):** as with slew, the trade-off is an improved mean, but a worsened maximum error.
- **Phase Margin:** also a very difficult fit like the slew, but in this case, boosting shows a clear advantage in both data sets.

This analysis should simply remind us of the need to regard these data mining techniques with an eye toward proper model selection and validation. A single trained regressor sometimes fits well, a committee of boosted regressors is sometimes "too much" fitting. On the other hand, a boosted committee is sometimes able to achieve significant fitting improvements. We regard this as a very satisfactory set of initial results for this very difficult new problem.

VI. Conclusions

Simulation-based synthesis tools routinely visit many fully simulated circuit solution candidates. We showed how to adapt ideas from large-scale data mining to build models that capture significant regions of these large, nonlinear high-dimensional spaces. Results are encouraging: there are spaces where our boosted regressors are clearly superior. However, as with all regression problems, no one technique fits best in all circumstances; in some cases a single regressor suffices, in others, our boosting techniques offer useful trade-offs between mean and worst-case error. Our regression architecture is a first attempt to apply these ideas to analog design space modeling; other constructions using these techniques also appear promising.

Acknowledgment: This work was funded by the Semiconductor Research Corp., the National Science Foundation under contract 9901164, the DARPA NeoCAD program managed by the Sensors Directorate of the Air Force Laboratory, USAF, Wright-Patterson AFB, and Texas Instruments. We thank Rodney Phelps and Akshat Shah for valuable discussions about this work.

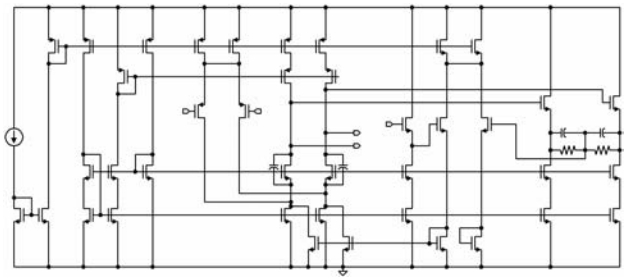

Fig. 8. Amplifier Circuit for Synthesis Design Space Macromodel Experiment.

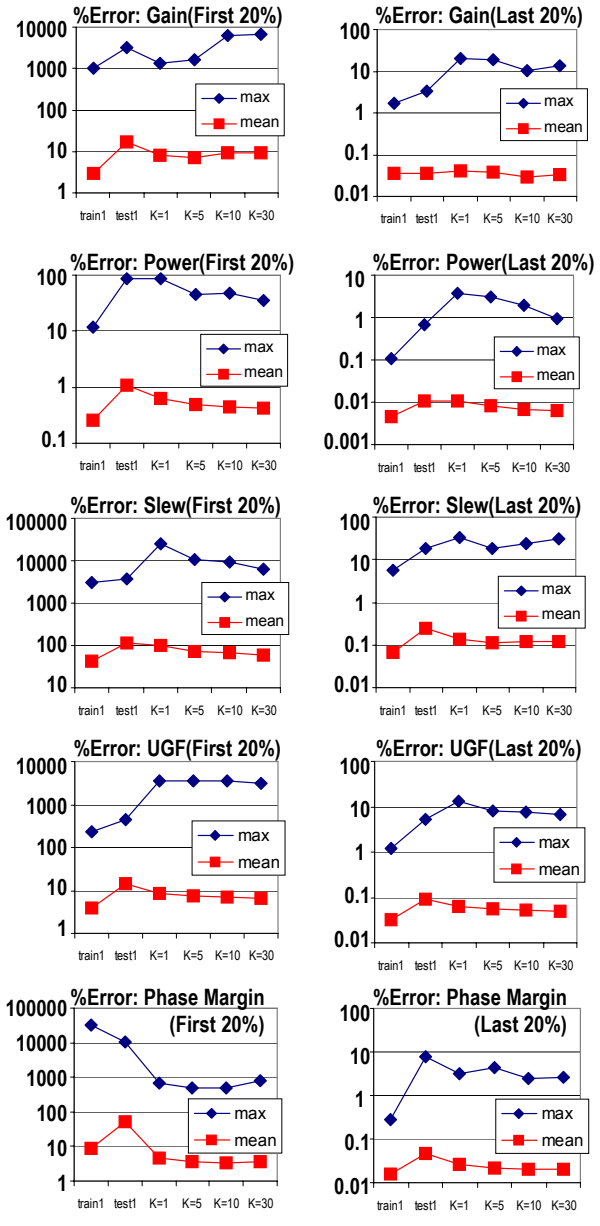

Fig. 9. Regression Results for Design Space of Circuit from Fig. 8.

REFERENCES

[1] P. Frey, D. O'Riordan, "Verilog-AMS: Mixed-Signal Simulation and Cross Domain Connection Modules," *Proc IEEE Behavioral Modeling and Simulation Conference (BMAS)*, 2000.

[2] E. Christen, K. Bakalar, "VHDL-AMS--A Hardware Description Language for Analog and Mixed-Signal Applications", *IEEE Trans. Circuits and Sys II: Analog and Digital Sig. Proc.*", vol. 46, no. 10, Oct. 1999

[3] A. Odabasioglu, M. Celik, and L. T. Pileggi, ``PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm," *IEEE Trans. CAD*, vol. 17, no. 8, pp. 645-654, 1998.

[4] Joel Phillips, "Projection Frameworks for Model Reduction of Weakly Nonlinear Systems," *Proc. ACM/IEEE DAC*, June 2000.

[5] G.G.E. Gielen and R.A. Rutenbar, "Computer-Aided Design of Analog and Mixed-Signal Integrated Circuits, *Proc IEEE*, vol. 88, no. 12, Dec. 2000.

[6] M. Hershenson, S. Boyd, T. Lee, "GPCAD: a Tool for CMOS Op-Amp Synthesis", *Proc. ACM/IEEE ICCAD*, pp. 296-303, 1998

[7] M. Krasnicki, R. Phelps, R.A. Rutenbar, L.R. Carley, "MAELSTROM: Efficient Simulation-Based Synthesis for Analog Cells," *Proc. ACM/IEEE DAC*, June 1999.

[8] R. Phelps, M. Krasnicki, R. A. Rutenbar, L. R. Carley, J. R. Hellums, "Anaconda: Simulation-Based Synthesis of Analog Circuits via Stochastic Pattern Search," *IEEE Trans. CAD*, vol. 19, no. 6, June 2000.

[9] R Phelps, M. Krasnicki, R.A. Rutenbar, L.R. Carley, J. Hellums, "A case Study of Synthesis for Industrial-Scale Analog IP: Redesign of the Equalizer/ Filter Frontend for an ADSL CODEC," *Proc. ACM/IEEE DAC*, June 2000.

[10] M.J., Krasnicki, R. Phelps, J.R. Hellums, R.A. Rutenbar, L.R. Carley, "ASF: A Practical Simulation-Based Methodology for the Synthesis of Custom Analog Circuits," *Proc. ACM.IEEE ICCAD*, Nov. 2001.

[11] R. Schwenker, J. Eckmueller, H. Graeb, K. Antriech, "Automating the Sizing of Analog CMOS Circuits by Consideration of Structural Constraints,", *Proc DATE99*, March 1999.

[12] A. R. Alvarez, B. L. Abdi, D. L. Young, H. D. Weed, J. Teplik, and E. R. Herald, "Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design", *IEEE Trans. CAD,* vol. 7, no. 2, pp. 272-288, Feb. 1988.

[13] K.K. Low and S.W. Director, "A New Methodology for the Design Centering of the IC Fabrication Process," *IEEE Trans. CAD*, vol. 10, no. 7, pp. 895-903, July 1991.

[14] E. Bauer, R. Kohavi, "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants". *Machine Learning* Vol 36, Nos. 1/2, pp. 105-139, July/August 1999.

[15] G.R.Boyle, B.M.Cohn, D.O.Pederson, and J.E.Solomon, "Macromodeling of Integrated Circuit Operational Amplifier," *IEEE J.Solid-State Circuits*, vol. SC-9, pp. 353-363, Dec.1974.

[16] Y-C Ju, V.B. Rao and R. Saleh, "Consistency Checking and Optimization of Macromodels", *IEEE Transactions on CAD*, vol. 10, no. 8, Aug. 1991.

[17] J. Phillips, "Automated Extraction of Nonlinear Circuit Macromodels," *Proc. IEEE CICC*, May 2000.

[18] T. Hastie and R. Tibshirani, *Modern Regression and Classification Short Course*, Washington DC, June 2000.

[19] M. H. Bakr, J. W. Bandler, K. Madsen, and J. Sondergaard, "Review of the Space Mapping Approach to Engineering Optimization and Modeling," *Optimization and Engineering*, vol. 2, 2001.

[20] R.Harjani and J. Shao, "Feasibility and Performance Region Modeling of Analog and Digital Circuits", *Analog Integrated Circuits and Signal Processing*, Kluwer Academic Publishers, 1996.

[21] R. Harjani, R.A. Rutenbar and L.R. Carley, "OASYS: a framework for analog circuit synthesis," *IEEE Trans. CAD*, vol. 8, no. 12, Dec. 1989.

[22] G. Box, W. Hunter and J. Hunter, *Statistics for Experimenters: and Introduction to Design Data Analysis and Model Building*, John Wiley, 1978.

[23] W. Daems, G.G.E. Gielen, W. Sansen, Gielen, "Simulation-Based Automatic Generation of Signomial and Posynomial Performance Models for Analog Integrated Circuits," *Proc. ACM/IEEE ICCAD*, Nov. 2001.

[24] Y. Freund and R.E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, 1997.

[25] A. W. Moore, J. Schneider, and K. Deng, "Efficient Locally Weighted Polynomial Regression Predictions", *Proc. 14th Int'l Conf on Machine Learning*, Morgan Kaufmann, pp 236-244, 1997.

[26] H. Drucker, "Improving Regressors Using Boosting Techniques," *Proc. 14th International Conference on Machine Learning,* ed. Douglas H. Fisher, Jr., Morgan -Kaufmann, pp. 107-215, 1997.

[27] Hjaltason, G.R. and Samet H., "Ranking in Spatial Databases," in M.J. Egenhofer and J.R.Herring, Eds., Advances in Spatial Databases - Fourth International Symposium, SSD'95, pp. 83-95, 1995.

[28] B. Ripley, *Pattern Recognition and Neural Networks,* Cambridge University Press, 1996.

[29] E. Hennig, R. Sommer, L. Charlack, "An Automated Approach for Sizing Complex Analog Circuits in a Simulation-Based Flow," *Proc. DATE02*, March 2002.

[30] Mathworks, Inc. http://www.mathworks.com.