

Behavioral Model Optimization via Combined Genetic Algorithm and Steepest Descent Method*

C.-J. Richard Shi, Yan Ye and X. Sheldon Tan
University of Iowa, ECE Department
Iowa City, Iowa 52242

Abstract

In this paper we present a new method which combines Genetic Algorithm with Steepest Descent searching locally to efficiently optimize parameters in behavioural models. The Steepest Descent Search makes use of the sensitivity information obtained from circuit simulator, SPICE, which is so time-consuming that we'd like to extract as much information as possible from it and do less of it. At the same time, to prevent the process from being trapped into any local optimum, Genetic Algorithm is utilized. Such combination makes this method to be capable of both exploitation (local search) and exploration (broad search) of the search space. It overcomes the shortcomings of the individual methods while maintaining their desirable features. The combined method is more efficient and more accurate than either of the two individual schemes.

This general-purpose algorithm is able to find the optimal solution efficiently with desired accuracy. It can be used to optimize any behavioural model as long as we have the behavioural model with undetermined but bounded parameters and the desired frequency response on hand.

The experiments we tried showed the efficiency of the proposed method with greatly reduced generation number in GA and simulation time as well as much better accuracy.

*This work is sponsored by U.S. Defense Advanced Research Projects Agency (DARPA) under grant number F33615-96-1-5601 from the United States Air Force, Wright Laboratory, Manufacturing Technology Directorate.

Parameter Optimization for Behavioural Models Using Genetic Algorithm combined with Steepest Descent Search

1. Introduction

Given a behavioural model with undetermined but bounded parameters and the desired response output, Parameter optimization has been a long-standing problem for circuit designers due to the lack of a sufficiently accurate and efficient tool. The simplest but also the most time-consuming approach is Monte-Carlo method which randomly search in the hyperplane of the parameter space, along with the simulation of the selected parameter, trying to get the optimal parameter set after fully exploration of the parameter space.

The motivation to develop an effective parameter optimization scheme arises out of the necessity to reduce the large number of circuit analysis required in the procedure.

There are two approach parties concerning this problem. Some researchers try some local minimum search algorithms [7] to get the optimal result when given a template circuit with expert-assigned parameter values, while other researchers try some global optimization scheme such like Genetic Algorithm [4], Simulated Annealing scheme [1] [3], aiming to get the global optimal solution only given a set of parameter range of a behavioural model.

Obviously, the second group's approach is of more practical significance with no need of careful initial estimates of the model parameters. However, consequently it requires much more CPU time rather than more expert knowledge.

But, as optimized stochastic search engines, GA and SA both require much CPU time even when they are closing to the optimal point. At this point, local search algorithm does nothing hurt if being used.

It would thus be more attractive to come out with an optimization tool combining these two approaches to be able to provide a good match between the desired and optimized device frequency response within a reasonable time.

Before we come up with a new method, let's explore the characteristics of the parameter optimization problem for behavioural model we're facing:

1. The circuit analysis which is required to compute the objective function in GA or SA is time con-

suming and expensive especially for a large circuit. Therefore, it's necessary to extract the maximum information from every analysis and try to save the simulation time as much as possible[6].

2. Unlike some Artificial Intelligence problem solved by GA or SA, such as the traveling salesman problem, which is difficult to get other evaluation of the attempted solution except the objective function, our time-consuming simulation is always easy to supply the sensitivity information for an attempted parameter set. Making use of these information will save much time especially when you're close to the optimal solution, it'll definitely better than do more iterations around the optimal solution using GA or SA. because it's guided.
3. There may exist many local minima in the constrained parameter space, and the global minimum must be one of them. Besides, the local minimum with certain small error is also acceptable for circuit engineers.

Based on these observations, we developed a new algorithm which combines Genetic Algorithm with Steepest Descend Search. The steepest-descend method based on the sensitivity information obtained from the SPICE, will rapidly lead to the local minimum using fibonacci line search. At the same time, we take the advantage of the global search ability of Genetic Algorithm to overcome the local minimum sticking. Basically, the Genetic algorithm explores the parameter space parallelly, and the promising solution is given special care by the Steepest Descend Search achieving the local minimum from it rapidly to see if it satisfies the engineer's accuracy requirement.

This paper is organized in the following manner. We describe genetic algorithm with local search in Section 2, and concentrate on steepest descend local search in Section 3. Examples are presented in section 4 to demonstrate the efficiency of the proposed approach. Conclusions are finally given in Section 5.

2. Genetic Algorithm with Local Search

Genetic Algorithm(GA) is essentially a robust search algorithm which is suitable for problems having comparatively larger solution spaces. The algorithm use randomized information exchange between the solutions to obtain optimal solutions. It starts with a finite set of potential solutions called the initial

population and then apply the operators: reproduction, crossover and mutation repeatedly thereby guiding the search towards better and better solutions. Apart from numerous applications of GA in various areas, the use of GA in small-signal modeling can be found in [4].

A genetic algorithm for a particular problem must have the following five components:

1. A genetic representation for potential solutions to the problem, every individual potential solution is called genome.
2. A way to generate an initial population of potential solutions;
3. An objective function which decide the "fitness" of the potential solution;
4. Genetic operators that alter the composition of individuals;
5. Adjustable various parameters that the genetic algorithm uses, e.g. population size, probabilities of applying genetic operators, etc.

Shown as Figure 1, next we succinctly give the outline of our extended Genetic Algorithm with featured local search, followed by explanatory discussions about its components.

1. Generate randomly the initial population P_{init} of n individuals based on (1), and evaluate their fitness based on (2),
2. Iteratively *fitness – biased* select parents from the old population P , apply *crossover* operator on them to generate at most n new children and put them into the new population P_{new} ,
3. Apply mutation operator to the new population P_{new} ,
4. Evaluate every individual of the new population based on cost function 2,
5. If the best individual of the previous population P is better than the best individual of the new population P_{new} , include it in P_{new} .
6. Apply steepest descend search to m' individuals which is selected from m ($m > m'$) best individuals of P_{new} ,
7. If the best solution doesn't meet the accuracy criteria, and doesn't exceed maximum CPU time, go to 2,

8. Report the best solution, and exit.

A. The Genome representation

Each *genome* instance represents a single solution to the problem, in our problem, it's one bounded parameter set. A set of potential solutions comprise a *population*.

How to encode the solution is the first question we encountered. Classical bit string representation of solutions has dominated GA research for years. But, as [9] studied, it has some drawbacks when applied to multidimensional, high-precision, especially, continuous, numerical problems. Either increasing the precision or extending the representing domain requires binary coding to expand the binary length rapidly, which would considerably slows down the algorithm. Using floating point representation is not only straight forward but also scalable especially in parameter optimization problem which with quite large high-precision domains. Nevertheless, corresponding genetic operators have to be developed for them.

Suppose we have k parameters needed to optimized in the behavioural model, each parameter x_i , called gene, is bounded by $[x_i^{min}, x_i^{max}]$, respectively. We represent x_i by a float variable directly, and we denote a float-point genome, which is a combination of genes, as a parameter set vector $X = \langle x_1, \dots, x_i, \dots, x_k \rangle$.

B. The Initialization

A good initialization strategy is important and meaningful for GA if it can fairly cover the solution set. But, The simple random initialization is not good enough for our problem. Suppose one parameter x_i has a range of $[10\Omega, 10K\Omega]$, given a random sampling, about 90% random sample will site in the interval of $[1K\Omega, 10K\Omega]$, little chance is give to the interval of $[10\Omega, 1K\Omega]$. Such sampling method is not desirable for circuit parameters, especially when the range is pretty large, it's almost misleading. Therefore, we decide to change the initialize strategy to fit this application better.

Given a random value $r \in [0, 1.0]$, we initialize x_i as:

$$x_i = \exp(\log x_i^{min} + r * (\log x_i^{max} - \log x_i^{min})); i = 1, \dots, k. \quad (1)$$

to make sure it scatter well in every decade(??).

C. Objective Function

The fitness of genome, an individual of the population, is calculated by the objective function defined

in (2). Every parameter set is fed into the circuit simulator and the output is used to calculate the error norm compared to the desired output. In this setting, we try to minimize the error, so the less value the objective function give, the better the genome is.

$$C(X) = \frac{1}{N} \times \sum_{j=0}^N \frac{\sqrt{\text{Re}(f_k(X) - h_k)^2 + \text{Im}(f_k(X) - h_k)^2}}{\max \|h_k\|} \quad (2)$$

where N is the number of frequency point we sampled, h_k indicates the desired value of every frequency point, $\|h_k\|$ is the magnitude of h_k . and $f_k(X)$ is the SPICE output value of each frequency point using the parameter set X ,

C. Genetic Operators

Since the role of GA in this application is tend to be a global search engine more than a local search engine. We select a Elitist Expected Value GA as a template, which has robust global search behavior as studied by De Jong[8], and extended by float genetic operators.

a) Select Operator

Selection is to choose the parents from the old population randomly but with a fitness bias. Individuals with better fitness will have better chance to be chosen, which is based on survival-of-the-fittest philosophy. Typically, we employ the widely used stochastic remainder selection without replacement [8] as the Selector function. To avoid population clustering and premature convergence the Linear Scaling is used to scale objective scores.

b) Crossover Operator

We adopt Two-Point-Crossover operator for float-point genome, by which the parameter sets will change fewer parameters than One-Point-Crossover, illustrated by Figure ???. It can preserve the consistency of parameter sets better in the algorithm. The crossover operator happens with probability p_c .

c) Non-Uniform Mutation Operator

This is a special mutation operator applied to float-point genome, with probability p_m . If $X_t = \langle x_1, \dots, x_i, \dots, x_k \rangle$ is a genome and element x_i with the domain of $[x_i^{min}, x_i^{max}]$ is selected for mutation, the result is a vector $X_{t+1} = \langle x_1, \dots, x'_i, \dots, x_k \rangle$,

$$x'_i = \begin{cases} x_i + \Delta(t, x_i^{max} - x_i) & \text{if a random flip is 0} \\ x_i - \Delta(t, x_i - x_i^{min}) & \text{if a random flip is 1} \end{cases} \quad i = 1, \dots, k.$$

and

$$\Delta(t, y) = y * (1 - r^{(1-\frac{t}{T})^b}),$$

where the function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as t increases. This property causes mutation happens uniformly initially, and very locally at later stages[9].

D. The Stopping Criterion

The Algorithm stops when the satisfying optimal parameter set is found, or the computation time exceeds. Genetic Algorithm always comes to a convergence when the difference between the best fitness and the average fitness of P is very small after many generations. But our approach is usually able to get a optimal solution before it comes to the convergence.

3. Steepest Descend Local Search

As well known, genetic algorithms display inherent difficulties in performing local search for numerical applications. We take GA as a preprocessor to perform the global search, and once the high performance regions of the search space are identified by it, we invoked the steepest descend search routine to employ the domain knowledge, the sensitivity information from the circuit simulator, to do fine local tuning. Both of the accuracy and efficiency are greatly improved by this way.

Whenever we feel necessary to do local tuning, we employ SPICE to calculate the sensitivity of parameter set X at each frequency point k , generating $\frac{\partial f_k(X)}{\partial X}$ for real and imaginary part respectively. But, considering the variety range of different parameter, we use normarized sensitivity $g_k(X)$ as defined in (3). With them, we can get the differential value for objective function (2) by (4).

$$g_k(X) = \frac{\partial f_k(X)/f_k(X)}{\partial X/X} \quad (3)$$

$$\frac{\partial C}{\partial X} = \frac{1}{N \times \|h_k\|} \sum_{j=0}^{j=N} \frac{Re((f_k(X) - h_k) \times g_k(X)) + Im((f_k(X) - h_k) \times g_k(X))}{\sqrt{Re(f_k(X) - h_k)^2 + Im(f_k(X) - h_k)^2}} \quad (4)$$

The steepest descend is defined by the iteration algorithm:

$$X_{t+1} = X_t - \alpha_t \times c \frac{\partial C}{\partial X} i \times (X^{max} - X^{min})$$

where α_t is a nonnegative scalar selected minimizing $C(X_{t+1})$. In words, from the point X_t we search along the direction of the negative gradient to a minimum point along this line, the minimum point in objective function is taken to be X_{t+1} .

The critical issue now is how to find the appropriate α_t in a efficient way. In our approach, Fibonacci line search method [10] is employed since it requires relative smaller number of calculations during line search. Let d_1 is the initial width of uncertainty and d_k is the width of uncertainty after k calculations. Then

$$d_k = \left(\frac{F(N - k - 1)}{F(N)}\right)d_1$$

where $F(n)$ are Fibonacci number and N is the specified calculation times in one line search. Due to the exponential increase in Fibonacci sequence, if N is appropriately selected (usually about 6-9), the uncertainty d_k will become small enough such that every point in it can be viewed as the minimum point with very small errors.

Usually, after the initial measurement of three points (the first point is X_t), the other points are made symmetrically at a distance of $(F(N - 1)/F(N))d_1$ from the ends of the initial intervals. The next uncertainty interval with length $(F(N - 1)/F(N))d_1$ is determined in such a way that lesser value point will be a one of successive measurement point and the other point become a one end of the uncertainty interval. So, each uncertainty determination only requires one measurement (SPICE simulation). Such saving is essential when it's so expensive to do the circuit simulation. The fibonacci line search with $N=5$ is illustrated in Fig. ???. If there exist more than two minimum along a line, our line search at least can find one of them within certain precision.

Another issue concerning find local minimum is to determine the maximum α_t to make sure every element of X_{t+1} still be bounded by its range.

Here, we firstly map the range of every parameter $[x_i^{min}, x_i^{max}]$ to $[0.0, 1.0]$, and get the maximum α_t to satisfy:

$$0.0 < \frac{x_i^t - x_i^{min}}{x_i^{max} - x_i^{min}} - \alpha_t \times \frac{\partial C}{\partial X} < 1.0$$

4. Experiments and Results

We compared our algorithm with genetic algorithm alone without local search. The experiment shows that the new algorithm greatly reduced the CPU time to optimize the parameter set of behavioral models. The new algorithm only need about as $\frac{1}{10}$ CPU time as the corresponding genetic algorithm needs, with even higher precision.

Table 1 summarize the result of our experiment. We took miller.cir and single.cir for examples respectively.

The parameters used in the extended GA is:

Size of one population $n = 50$

The probability of crossover operation $p_c = 0.9$

The probability of mutation operation $p_m = 0.002$

Select $m' = 3$ individuals from the best $m=15$ individuals for local search.

The parameters for the compared genetic algorithm alone are almost the same as the above except:

Size of one population $n = 100$

Thereby, the CPU time required by two algorithm are almost the same. because,

the simulation times of GA alone = $n = 100$

the simulation time of our method = $n + m' \times \text{simulation-time-per-local-search}$

And the average simulation-time-per-local-search is about 16 when using fibonacci search. Thus, the above two value are almost equal. At the same time, the simulation time is the most significant part of CPU time consumed. So, Comparison of the number of generation means the same as comparison of CPU time.

On the other hand, assign bigger population size the mere GA is reasonable, because without special local search engine, it needs more sampling to explore the solution space more thoroughly.

Compared to other approaches by other researchers, our method is still of its special significance, although no standard benchmark available right now. For example, Menozzi [4]

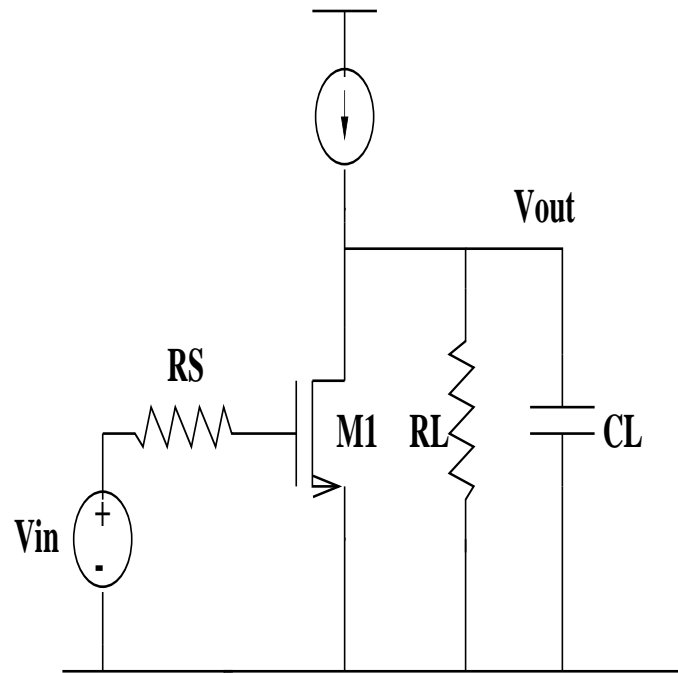


Figure 1: A single.cir example.

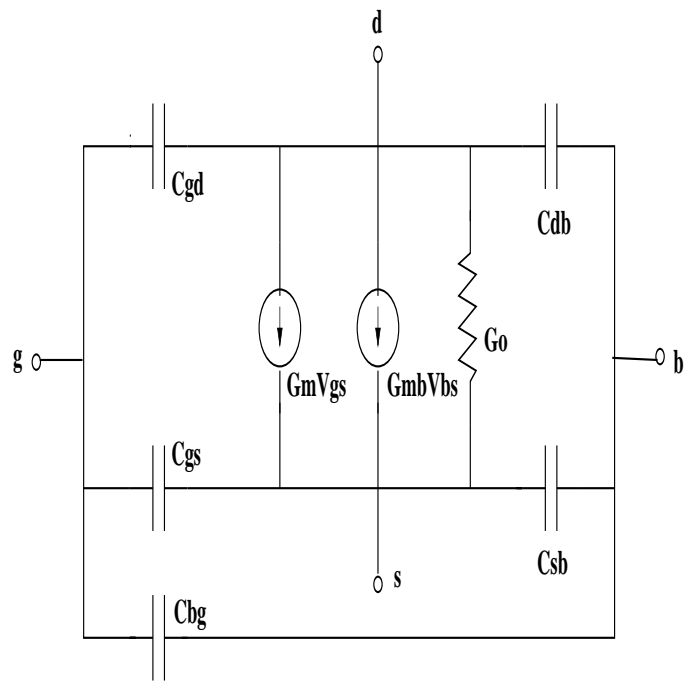


Figure 2: msmodel

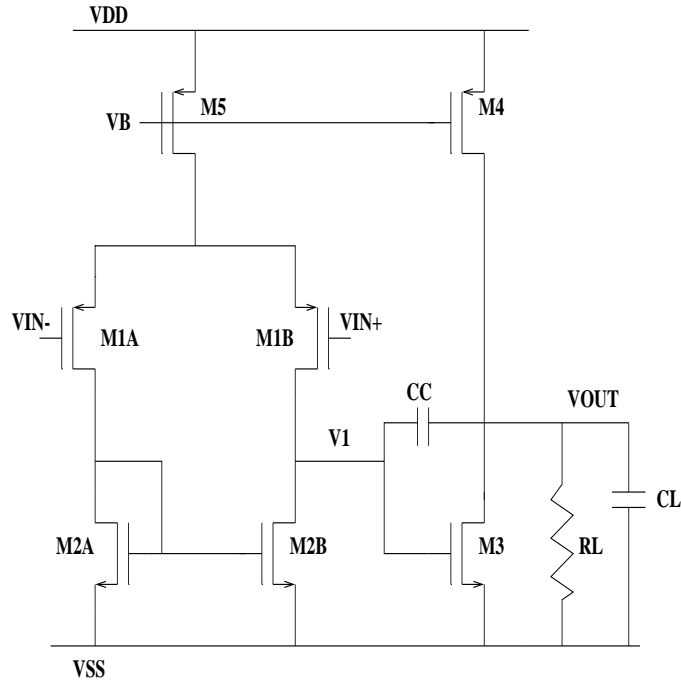


Figure 3: A miller.cir example.

Table 1:

5. Conclusion

References

- [1] Y. C. Ju, V. B. Rao, R. A. Saleh, “Consistency Checking and Optimization of Macromodels”, *IEEE Trans. Computer-Aided Design*, vol. 10, No. 8, pp. 957–967, Aug. 1991.
- [2] J. P. Courat, G. Raynaud, I. Mrad, P. Siarry, “Electronic Component Model Minimisation Based on Log Simulated Annealing”, *IEEE Trans. Circuits and Systems*, vol. 41, No. 12, pp. 790–795, Dec. 1994.
- [3] S. G. Skaggs, J. Gerber, G. Bilbro, and M. B. Steer, “Parameter Extraction of Microwave Transistors Using a Hybrid Gradient Descent and Tree Annealing approach”, *IEEE Trans. Microwave Theory Tech*, vol. 41, pp. 726-729, Apr. 1993.
- [4] R. Menozzi, A. Piazzzi, and F. Contini, “Small-Signal Modeling for Microwave FET Linear Circuits Based on a Genetic Algorithm”, *IEEE Trans. Circuits and Systems*, vol. 43, No. 10, Oct. 1996.

- [5] L. Getreu, and D. Teegarden, "An Introduction to Behavioural Modelling", *Microelectronics Journal*, vol. 24, No.7, 1993.
- [6] M. A. Styblinski, and S. Aftab, "Combination of Interpolation and Self-Organizing Approximation Techniques –A New Approach to Circuit Performance Modeling", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.12, No.11, Nov. 1993.
- [7] G. Casinovi, and A. Sangiovanni-Vincentelli, "A Macromodeling Algorithm for Analog Circuits", *IEEE Trans. Computer-Aided Design*, vol. 10, No. 2, Feb. 1991.
- [8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, New york: Addison-Wesley, 1989.
- [9] Z.Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin: springer-Verlag, 1994.
- [10] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 1984.