ELSEVIER

Microelectronics Journal 33 (2002) 61–68

Microelectronics Journal

www.elsevier.com/locate/mejo

An extensible TCAD optimization framework combining gradient based and genetic optimizers

Clemens Heitzinger*, Siegfried Selberherr

Institute for Microelectronics, E360 TU Wien, Gußhausstraße 27-29, A-1040 Vienna, Austria

Received 2 February 2001; accepted 21 August 2001

Abstract

The SIESTA framework is an extensible tool for optimization and inverse modeling of semiconductor devices including dynamic load balancing, for taking advantage of several, loosely connected workstations. Two gradient-based and two evolutionary computation optimizers are currently available through a uniform interface and can be combined at will. At a real world inverse modeling example, we demonstrate that evolutionary computation optimizers provide several advantages over gradient-based optimizers, due to the specific properties of the objective functions in TCAD applications. Furthermore, we shortly discuss some issues arising in inverse modeling and conclude with a comparison of gradient-based and evolutionary computation optimizers from a TCAD point of view.

Keywords: TCAD; Optimization; Evolutionary computation; Genetic optimization; Gradient based optimization; Inverse modeling

1. Introduction

Technology CAD (TCAD) tools like SIESTA [1,2] have been successfully used for optimizing semiconductor devices [3] and for inverse modeling [4]. Although SIESTA proved to be a valuable tool and several interesting results [5,6] were achieved using it, it did not, at that time, provide any global optimizer, but two gradient-based optimizers [7]. The most recent advances include new, global optimizers, combine these two approaches to optimization, and improve on the flexibility and extensibility.

Over the years, it has been recognized that a successful TCAD optimization framework has to meet with the following criteria.

1. The ability to execute simulation tools on a number of computers in the local network, and to schedule the execution of these simulation tools in reaction to changes in this network. For example, in a heterogeneous network that is not solely dedicated to executing simulation tools the question how the tasks have to be scheduled so that the overall execution time is minimized is not trivial. Furthermore, software and hardware failures have to be taken into account.
2. *Stability*. This property is crucial for a program that usually runs for several days and has to deal with all kinds of software failure.
3. Extensibility. A TCAD framework has to deal with various programs [8], data formats [9] and combinations thereof. Evaluating the goal function (i.e. the function to be optimized) often entails several calls to simulation tools. Because of the abundance of possible goal functions, a framework has to provide a flexible extension language, which enables the user to succinctly describe the desired goal function.
4. *Specialized optimizers*. The evaluation of the goal function is usually very expensive: times range from about a minute to one hour or more for process simulations on current hardware. Strategies for finding global extrema of computationally very expensive functions are needed. The respective advantages and disadvantages of gradient based optimizers and evolutionary computation will be discussed in Section 5.
5. Finding a suitable starting value is often the most difficult and time consuming task when using a gradient based optimizer. Hence, global optimizers which do not need a starting value sufficiently close to the global extremum are called for.

It should be noted that the goal function of an optimization in TCAD analysis may not even be a function in the

* Corresponding author. Tel.: +43-1-58801-36035; fax: +43-1-58801-36099.

E-mail address: heitzinger@iue.tuwien.ac.at (C. Heitzinger).

0026-2692/02/$ - see front matter
PII: S0026-2692(01)00105-7

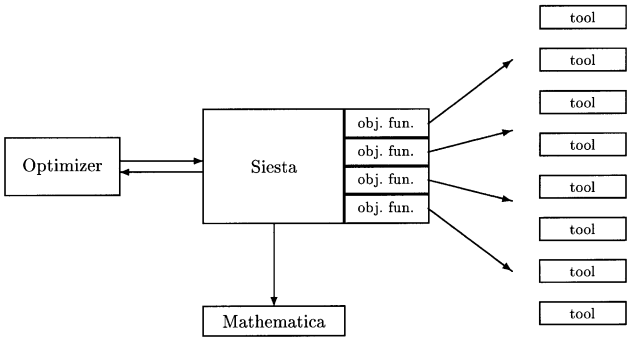

Fig. 1. Overview of SIESTA.

mathematical sense. Simulation tools like MINIMOS [10–12] provide modes of operation which are not deterministic, i.e. the same input may lead to slightly different results. In the case of MINIMOS, this is due to preconditioning, which depends on the elapsed simulation time. Furthermore, it happens in practice that simulation tools do not converge for certain inputs, or yield results only after consumption of exceptionally long computation time. These problems are addressed by the SIESTA framework and are especially handled well by stochastic global optimizers.

After an overview of the framework, a real world example and the advantages and disadvantages of the optimizers currently available are discussed. Local, gradient-based optimizers require a starting point and repeatedly use the gradient of the target function at the working point to find better points. They generally find local extrema. Global optimizers try to not only find local extrema but global ones and use different techniques. Stochastic optimizers like genetic algorithms use random numbers and the results generally differ from run to run.

2. Overview of the SIESTA framework

This section is devoted to an overview of the SIESTA framework. Fig. 1 depicts the control and data flow in a typical optimization run. After setting up an experiment, one of four optimizers is started by SIESTA and its evaluation requests are fulfilled in parallel, where each evaluation entails an arbitrary number of calls of simulation tools on remote machines. In each run, the definition of the experiment and the progress of the optimization are saved to files and can be examined from within MATHEMATICA.

2.1. The extension language

SIESTA was originally implemented in a dialect of Lisp [13,14] called xlisp [15]. Since xlisp was written, because of the new developments in language design and implementation, the choice of a suitable base language for SIESTA was thought over. In order to meet with the requirements described in Section 1, we posed the following demands on a suitable base language.

1. It has to provide an interface to the underlying operating system and network.
2. There has to be one (preferably more) stable and well supported implementation.
3. It has to support multiprocessing (or multithreading).
4. An extension language is necessary in order to provide the required flexibility.
5. The implementation must be able to load additional code at run time and to enable the user to execute commands interactively.
6. It should be well designed and preferably standardized.

After evaluating several languages, namely C, C++, Common Lisp, Java, Perl, Python, Scheme, and Tcl, we decided to use Common Lisp [16] with multiprocessing support. In addition to fulfilling all of our requirements, it provides the following features, which helped reducing the implementation time.

1. Common Lisp supports the paradigms of functional programming and object oriented programming.
2. All language constructs are available at run time.
3. Several implementations on all major platforms are available and all of these provide compilers and interactive listeners.
4. A powerful macro system makes Common Lisp a very extensible programming language.
5. Its condition system and operators like `unwind-protect` and `ignore-errors` contribute to stability and robustness.
6. Common Lisp is an ANSI (American National Standards Institute) standard.

SIESTA runs on UNIX platforms, since UNIX provides a good support for executing commands on remote computers and distributing files in a cluster of computers. Apart from these requirements, SIESTA is platform independent.

The requirements on the software infrastructure installed on the cluster of computers to be used in an experiment have been reduced to a minimum. Early versions of SIESTA required the user's home directory to be visible on all computers of the cluster and relied on network file system (NFS). However, NFS is a source of problems since it does not provide sufficient synchronization between the states of files on different computers. Files are synchronized only after a pause, which leads to problems when the result file of a simulation tool exists on a client, but is not seen on the computer where SIESTA runs. Furthermore, it is not possible to request synchronization between two computers, manually.

Several solutions for that problem were tried, yet none worked satisfactorily. One of the attempts was to wait for a certain amount of time (up to 30 s), after a simulation tool had finished. Because of this limitation of NFS, we decided to dispense with it and use `rcp` or `scp` instead.

In order to run SIESTA, the following programs have to be installed on a cluster.

- One of these possiblities for communication has to be chosen:
  `rsh`, `rcp` and `rshd`. These are standard on UNIX, but not secure.
  `ssh`, `scp` and `sshd`. These are not found on every UNIX system by default, but are secure.

Of course, these programs have to be set up such that no password entry is needed for each individual login, but only once per session.

- Standard UNIX commands like `kill`, `mkdir`, `rm`, `top` and `uptime`.
- Finally, it goes without saying that the simulation tools to be used have to be installed. Licenses are managed as described in Section 2.5.

2.2. Parallelization

Because of the need for parallelizing several evalutions of the goal function, and thus the simulation tools, we chose to extend the language with a macro called `parallel`. It takes an arbitrary number of expressions as input and returns a list of results after the evalution of all the input forms (which is done in several processes), has finished.

The `parallel` macro and its sister function `p-apply` are an important building block of SIESTA and may of course be called by the user.

2.3. Setting up experiments

In this section, we show how SEISTA is typically used. When starting up, it reads its initialization file in which the hosts to be used are defined via `define-all-hosts`. The information about the hosts will be used later by the task manager, which schedules the execution of the various simulation tools. Hosts may later be enabled and disabled interactively by the user. Furthermore, for every host, a function can be provided which decides if a host is usable right now; this is useful when certain hosts must not be used at certain times of a day.

Commands are entered in any number of interactive listeners. After loading a file containing the definition of an experiment, the run command starts an optimization run. The value returned by run can be stored in a file, although an automatically generated file contains all the results and information about the experiment and the progress of the optimization. The result can also be used as a starting value for the next run.

Experiments are defined using `define-experiment`. The definition of an experiment consists of an optional description, the list of the free variables, their interval and their default values, the list of user variables which enable sophisticated customized setups, the goal function, the default value of the goal function to be used when no attempt was successful, the constraint function, and the configuration of one or more optimizers.

States correspond to points in the search space or to individuals of the population in the language of evolutionary computation. A state consists of a list of all (free) variables and their respective values, the experiment it belongs to and — after evaluation — the value of the goal function. States can be manipulated with the `make-state`, `copy-state`, `with-state-vars`, `setq-in-copied-state` and `show-state operators`.

Setting up the evaluation function is usually the most difficult part in defining an experiment. `test-run` can be used to evaluate the goal function on the default values interactively and to see if it works satisfactorily.

2.4. Calling simulation tools

Since several types of software (and hardware) failures may occur when running simulation tools, especially in a networked environment, we extended the base language with a macro called `with-retries`. Its calling signature is `(number-of-tries & body default-forms) & body body`. `with-retries` executes its `body` `until` no error was raised, but at most `number-of-tries` times. Upon success, the result values are those of the last form in `body`, otherwise, the values returned by `default-forms`.

Among the inputs to the call of a simulation tool, there is always one state. Calling simulation tools through the preferred interface entails the construction of an instance of class `task`, or a subclass thereof. Tasks contain all the information about calling a simulation tool and returning the required data. While subclasses and methods specialized to certain simulation tools are provided, the class hierarchy starting at `task` can also be specialized by the user, as well as the methods acting on tasks, e.g. the `execute` method.

The task manager schedules the execution of the tasks. In order to execute a task, it looks for reachable hosts (i.e. hosts whose load average can be retrieved), which are not disabled and not too busy. If such a host is available, the task is run on the host, which currently provides most of the computational resources. Otherwise, it waits until a host becomes available.

2.5. License management

When using commercial simulation tools, the number of available licenses for a certain program is often limited. Thus, we have to provide a way to ensure that at any point of time only a certain user prescribed amount of licenses of such programs are in use. Users can call simulation tools not only by using the predefined functions of the framework, but also from self written programs like shell scripts which are in turn called from within the

framework. A suitable license management scheme has to consider this.

In order to make our license management scheme meet these needs, it works independently from the predefined functions of the framework. The following steps are necessary to use it. Whenever the number of requested licenses exceeds the number of available ones, certain processes have to wait until the required number becomes available. In order to show how many licenses are currently in use, the command `show-licenses` can be used.

1. Define the names of the licenses and how many of each may be used simultaneously. This is accomplished with `define-licenses` and is usually done in your SIESTA configuration file.
2. When using a license, wrap the code into `with-locked-licenses`.

Users expressed interest in varying the number of available licenses, during optimization. This need frequently arises in a setting where people want to reserve one or two licenses for interactive work at certain times, but want all of them to be used, e.g. at night time. For simply changing the number of totally available licenses the function `set-number-of-licenses` can be used.

2.6. Input deck handling

Nearly all simulation tools use a text file for configuration. The configuration files of MINIMOS are called input deck [12] files, and we will use this term for all simulation tools. SIESTA generates these files by substituting the values of the variables of a state in template files. If a template file contains a string ⟨(`foo`)⟩ and the value of `foo` in the current state is, for e.g. `1.23`, the string will be replaced with `1.23`. This applies to free and user variables of a state.

In case the use of ⟨(and)⟩ leads to collisions in the input deck file of some simulation tool, the begin and end marker can be changed.

2.7. Available optimizers

The following brief overview lists all optimizers currently available in SIESTA, namely, two gradient-based [17] and two stochastic global ones.

2.7.1. Genopt

The interface to GAlib [18], a C++ library for genetic optimization, is called genopt. It provides standard selection, crossover, mutation, scaling and termination methods [19].

For our experiments, we mainly use the following setup, because it provides good results in an acceptable amount of computation time. Since all parameters are reals chosen from intervals, we represent them as floating point numbers, and not as binary vectors as favored in early genetic optimization. We use a mutation operator which adds a random number from a normal distribution, more precisely, $x \in [a, b]$ is changed to $\min(\max(N(x, \sigma), a), b)$, where σ depends on the length of the interval.

As crossover operators, we use two point and uniform crossover. Most populations consist of about 40–50 individuals. Some optimization tasks allow us to evaluate about 20 generations per hour, which amounts to roughly 500 generations per day. Typical runs last for two or three days.

Constraints handling is done using the popular penalty method, i.e. the scores of states which do not fulfill given constraints, which are defined as an arbitrary function, are increased by prescribed amounts.

2.7.2. Siman

Simulated annealing [20,21] was invented by Kirkpatrick in 1982 and is a modified version of hill climbing. Starting from a random point in the search space, a random move is made. If this move yields a better point, it is accepted. If it yields a worse point, it is accepted only with a certain probability $p(t)$, which depends on the time t. The function $p(t)$ initially is close to 1, but gradually reduces towards 0 in analogy to the cooling of a solid. Hence, initially any moves are accepted, but as the temperature reduces, the probability of accepting a negative move is lowered. Negative moves are essential sometimes if local maxima are to be escaped, but obviously, too many negative moves will simply lead away from an extremum. Versions like fast re-annealing, adaptive annealing and parallel annealing have been developed. In our framework, we provide an interface to an implementation [22] by Lester Ingber.

2.7.3. Donopt

This gradient-based optimizer [7,1] minimizes a scalar value and supports equality and inequality constraints. It is based on donlp2 [23,24] by Peter Spellucci.

2.7.4. Lmmin

The Levenberg–Marquardt algorithm [25] is an efficient method to solve nonlinear least squares problems, and is therefore well suited for inverse modeling tasks. SIESTA provides an interface to the implementation found in the MINPACK [26,27] project.

The parameter values are chosen from prescribed intervals. However, arbitrary constraints are not supported by this optimizer. The step size used for the gradient computation and a tolerance value acting as termination criterion, can be adjusted.

2.8. Interface to MATHEMATICA

The progress and the results of the various optimization runs are saved to text files and can be examined via an interface to MATHEMATICA [28], which enables the user to take advantage of its graphics and statistics capabilities. All the score and parameter values, and some auxiliary values,

are visible in MATHEMATICA and can be inspected using built-in and custom functions (`reportRun`, `best`, `plotProgress`, `plotInverseModeling`, `plotRunOneVar`, `plotRunTwoVars` etc.).

3. Inverse modeling (parameter extraction)

Many models in TCAD applications contain free parameters which depend on properties of the device material and have to be calibrated using measurements. Usually vectors of measured values are fit to characteristic curves of the device in question.

It is not obvious which goal function should be used in an inverse modeling experiment, where the distance between two vectors (where one is constant) is to be minimized, and several functions have been used (e.g. [29,30]).

In the following $m \in \mathbb{R}^n$, is the measured vector and $s \in \mathbb{R}^n$ is a vector of simulation results. We define the relative error of the two vectors as the vector resulting when applying the relative error function component wise, i.e. $r_k := (s_k - m_k)/m_k$ for $r = \mathrm{RE}(s, m)$ being the vector of relative errors. The quadratic mean M_2 of a vector $x \in \mathbb{R}^n$ is defined as:

$$M_2(x) := \sqrt{\frac{\sum_{k=1}^{n} x_k^2}{n}},$$

and the weighted quadratic mean $M_{2,w}$ with weights w_k is defined as:

$$M_{2,w}(x) := \sqrt{\frac{\sum_{k=1}^{n} w_k x_k^2}{\sum_{k=1}^{n} w_k}},$$

where $w_k > 0$ for all k.

In this regard, the reader is referred to [31] for properties of mean values. Of course, $\|\cdot\|_2$ and M_2 are equivalent norms. However, the quadratic mean is easier to interpret since the number of comparison points does not influence the value.

For p being a vector of parameters to be fit, we believe minimizing

$$f(p) := M_2(\mathrm{RE}(s(p), m))$$

to be a natural and advantageous formulation of the problem of parameter extraction. Variations of $s(p)$ or m over many magnitudes do not have an ill effect as compared to other functions working with absolute errors, and by using the weight factors f can be adjusted to individual needs.

We provide the `log10`, `p-metric`, `p-norm`, `p-mean` and `relative-error` (for vectors) functions in our framework.

Table 1
Parameters

Name	Interval	Default value
`et`	[0,2]	0
`ew`	$[-0.6, -0.4]$	-0.425
`nt`	$[10^{12}, 10^{16}]$	10^{13}
`sr`	[100, 500]	200
`srve`	[10, 10000]	5000
`ste`	$[10^{-20}, 10^{-17}]$	10^{-19}

4. A real world example

The goal of this example is to show that a typical class of optimization problems, namely inverse modeling or parameter extraction problems, can be automatically solved using evolutionary computation optimizers. In this real world problem, we extract six parameters from the drain currents of the select transistor of a storage cell and try to fit two transfer characteristics (two bulk voltages, two times 27 points), in the process. This presentation is not intended as a complete treatment of this particular problem.

For the purpose of this optimization, we treat the measurements and simulated vectors as one vector with 54 components and try to minimize $f(p) = M_2(\mathrm{RE}(s(p), m))$, as proposed in Section 3. MINIMOS NT [12] was used as the device simulator and Table 1 lists the six parameters, their intervals, and their default values. The default values were used as starting values for the gradient-based optimizer. The variable `ew` is the work function of the gate material, and the variable `sr` is the source resistance. The other variables pertain to the Shockley–Read–Hall model [12, page 71]. Table 2 lists the three optimizers used (cf. Section 2.7) and their configuration, which is the default configuration in all three cases.

Fig. 2 shows the progress of the three optimizers. For all optimization runs, we used a cluster of fifteen workstations with twenty CPUs and dynamic load balancing. We note that the gradient-based optimizer does not yield a good result (cf. also Fig. 3), although its initial progress is fast. Furthermore, the evaluations of the genetic algorithm,

Table 2
Configuration of the three optimizers

Optimizer	Name of parameter	Value of parameter
Genopt	Algorithm type	Steady state
	Population size	50
	Probability of replacement	0.7
	Probability of crossover	0.8
	Crossover type	Two point crossover
	Probability of mutation	0.2
	Number of best genomes	50
Siman	Block	15
	Block max	30
	Moving average	3
Donopt	del0	0.5

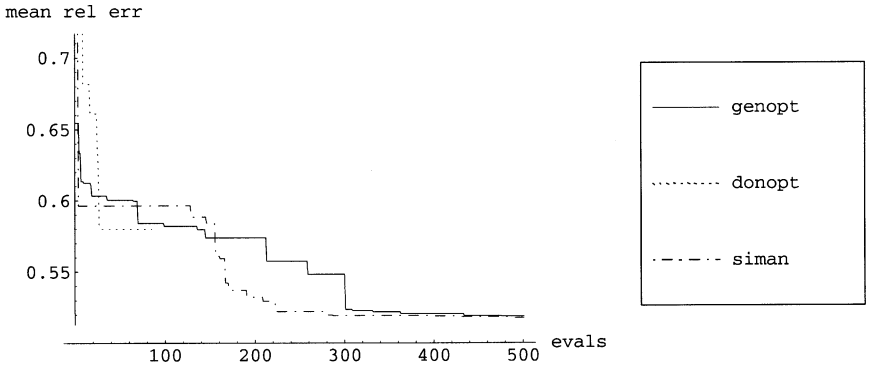

Fig. 2. Optimization progress.

genopt, are better parallelized on the twenty CPUs than those of simulated annealing, siman, and thus in terms of wall clock time elapsed, the genetic algorithm is the fastest optimizer.

Fig. 3 shows the best fitting simulated transfer characteristics (for two bulk voltages, left and right half) found by the gradient based optimizer, donopt, yielding a value of the objective function of 0.58. The agreement in the range of the points numbered 1–10 and 28–38 is mediocre. Fig. 4, shows the best fitting vector found after at most 500 evaluations with each optimizer. Siman yields good agreement and a value of 0.52. Both plots are logarithmic and the comparison points are numbered from 1 to 54.

In Section 5, we will discuss the advantages and disadvantages of evolutionary computation optimizers and gradient-based optimizers, based on the experience of this and similar examples. Most notably, evolutionary computation optimizers provide automatic global optimization.

5. Comparing gradient based and evolutionary computation optimizers

During the last three or four decades, there has been increasing interest in optimization algorithms which work similar to processes found in nature. The methods of genetic algorithms [32,33,19] and evolutionary strategies [34], although having different roots, have converged and are now commonly known under the name of evolutionary computation. A journal of the same name [35] has been published since 1993.

A brief outline of evolutionary algorithms is as follows. They work with sets (or populations) of potential solutions. Starting from a random population, each individual is assigned a score via a goal function. In the selection step, certain individuals are chosen to proliferate and form a new population. Operators (e.g. mutation) are applied to the individuals of the new population with prescribed probability, and the population is evaluated again. New generations are formed in this way until a termination condition is fulfilled. Obviously, many alternatives for every step in this algorithm exist and have been described and discussed in many publications [19]. Although the Schema Theorem [19, page 53] and similar theorems explain the way in which evolutionary algorithms work in a quantifiable way, and many special algorithms have been intensively studied, no consistent theory of evolutionary computation exists to date

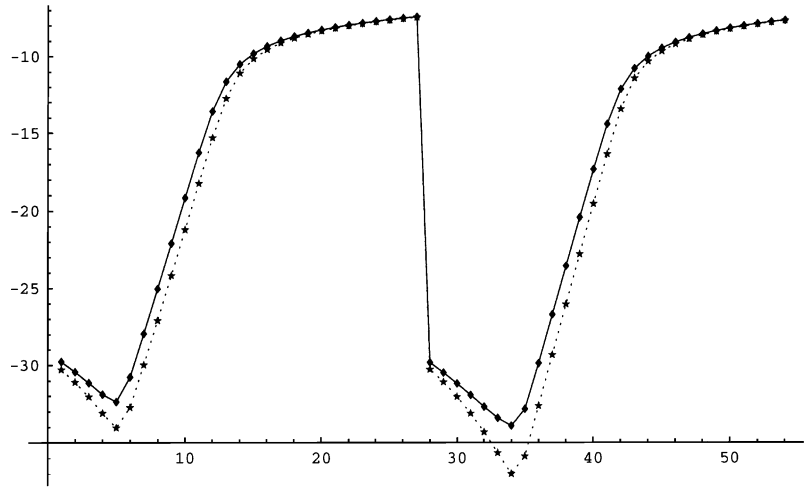

Fig. 3. Logarithmic plot of best result by optimizer donopt, score 0.579942. Solid line: measurement; dashed line: simulation.

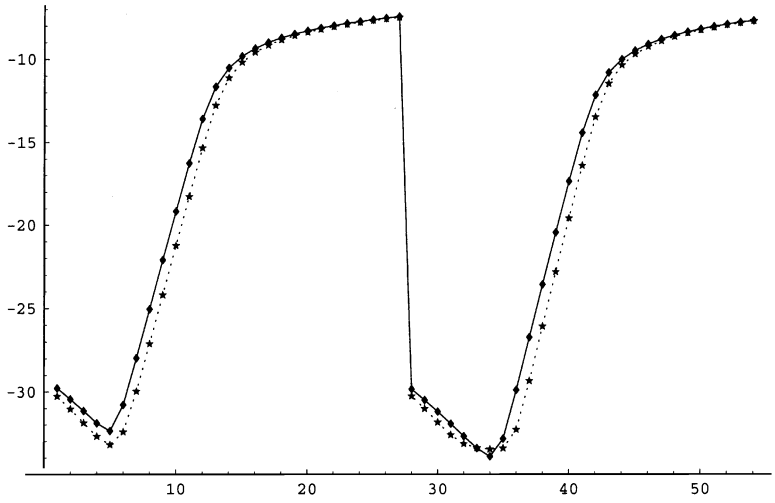

Fig. 4. Logarithmic plot of overall best result after 500 evalutions, optimizer siman, score 0.517803. Solid line: measurement; dashed line: simulation.

and the question of which evolutionary algorithm to choose for a given problem can be answered only by experimentation and experience. Nevertheless, evolutionary algorithms proved to be very general and valuable tools. While domain specific optimizers typically perform better than their general evolutionary algorithm counterpart, evolutionary algorithms can easily be adapted to the problem at hand and are usable whenever the lack of detailed knowledge about the goal function prohibits developing a domain specific optimization algorithm.

Although evolutionary computation is a well established optimization technique today, its application to TCAD analysis has been limited. Reasons are certainly the need for lots of computational resources and the requirements outlined in Section 1. While most research in evolutionary computation has been done relatively cheap to evaluate goal functions, the optimization of semiconductor devices has to cope with a relatively limited number of evaluations.

The most important difference from the usual practice of evolutionary algorithm optimizers is that runs are usually finished before a common termination condition like '95% of the population are identical' is fulfilled.

In the following, we discuss the advantages and disadvantages of gradient based and evolutionary algorithm optimizers and show why the combination of both is worthwhile. An optimization run with an evolutionary algorithm optimizer usually yields a set, or population, of nearly optimal solutions. The best of these is used as the starting point for a run with a gradient based optimizer, thus bringing together global and local optimization methods. Other combinations are also possible, for example: starting populations can be constructed manually; other states than the best in a population may yield better final results because they lie closer to the global optimum; the configuration of an optimizer can be changed and the computation restarted with the latest population or starting point.

5.1. Advantages and disadvantages of gradient based optimizers

Most importantly, gradient based optimizers are hill climbing algorithms and therefore local optimization techniques. Although highly sophisticated algorithms [17] have been developed, they all depend on a suitable starting point. In practice, finding this starting point has been found to be the major hurdle when trying to do unattended, automatic optimizations. Typically, finding such a point and shortening the parameter intervals so that the goal function can actually be evaluted, requires several tries and can easily take several days.

When the number of variables increase, the number of evaluations increases as well. While goal functions with few variables are feasible, optimizations with about 20 variables are usually impractical. Evolutionary algorithms do not suffer as much from this effect.

5.2. Advantages and disadvantages of evolutionary algorithm optimizers

Evolutionary algorithm optimizers are global optimization methods and scale well to higher dimensional problems. They are robust with respect to noisy evaluation functions and the handling of evalution functions, which do not yield a sensible result in a given period, is straightforward.

The algorithms can easily be adjusted to the problem at hand. Almost any aspect of the algorithm may be changed and customized. On the other hand, although lots of research has been done, on which evolutionary algorithm is best suited for a given problem, this question has not been answered satisfactorily. Although the standard values usually provide reasonably good performance, different configurations may give better results. Furthermore, premature convergence to a local extremum may result from

adverse configuration and not yield (a point near) the global extremum.

6. Conclusions

SIESTA has been used for several optimizations of real world devices on a cluster of a dozen workstations with about twenty CPUs and has proven to be very robust. The combination of gradient based optimizers and evolutionary computation allows us to take advantage of the benefits of both approaches. While the default configurations of the optimizers provide reasonably good performance, many aspects of an optimization run can be customized. The output of one or more optimization runs can be combined and used as input for the next run. This interoperability allows for interesting combinations of optimizers, comparisons of their performance and specialization to the problem at hand.

Acknowledgements

The authors acknowledge support from the 'Christian Doppler Forschungsgesellschaft', Vienna, Austria. Many fruitful discussions with Thomas Binder and Robert Klima were valuable and always appreciated.

References

[1] R. Strasser, Rigorous TCAD investigations on semiconductor + fabrication technology. Dissertation, Technische Universität Wien, 1999. http://www.iue.tuwien.ac.at/diss/strasser/diss-new/diss.html.
[2] R. Plasun, M. Stockinger, R. Strasser, S. Selberherr, Simulation based optimization environment and its application to semiconductor devices, in: International Conference on Applied Modelling and Simulation, Honolulu, Hawaii, USA, August 1998, pp. 313–316.
[3] R. Plasun, C. Pichler, T. Simlinger, S. Selberherr, Optimization tasks in technology CAD, in: W. Hahn, A. Lehmann (Eds.), Ninth European Simulation Symposium, Society for Computer Simulation International, Passau, Germany, 1997, pp. 445–449.
[4] R. Strasser, R. Plasun, S. Selberherr, Practical inverse modeling with SIESTA, Simulation of Semiconductor Processes and Devices, Kyoto, Japan, September 1999, pp. 91–94.
[5] M. Stockinger, Optimization of ultra-low-power CMOS transistors. Dissertation, Technische Universität Wien, 2000.
[6] M. Stockinger, A. Wild, S. Selberherr, Closed-loop MOSFET doping profile optimization for portable systems, in: Proceedings of the Second International Conference on Modeling and Simulation of Microsystems, San Juan, Puerto Rico, USA, April 1999, pp. 411–414.
[7] R. Plasun, Optimization of VLSI semiconductor devices, Dissertation, Technische Universität Wien, 1999. http://www.iue.tuwien.ac.at/diss/plasun/diss-new/diss.html.
[8] J. Daniell, S. Director, An object-oriented approach to CAD tool control, IEEE Trans. Comput.-Aided Des. 10 (1991) 698–713.
[9] T. Binder, S. Selberherr, Object-oriented design patterns for process flow simulations, Proceedings of the Fourth Annual IASTED International Conference on Software Engineering and Applications, Las Vegas, November 2000.
[10] S. Selberherr, W. Fichtner, H. Pötzl, MINIMOS — a program package to facilitate MOS device design and analysis, in: B. Browne, J. Miller (Eds.), Numerical Analysis of Semiconductor Devices and Integrated Circuits, vol. I, Boole Press, Dublin, 1979, pp. 275–279.
[11] T. Simlinger, H. Kosina, M. Rottinger, S. Selberherr, MINIMOS-NT: a generic simulator for complex semiconductor devices, in: H. de Graaff, H. van Kranenburg (Eds.), 25th European Solid State Device Research Conference, Editions Frontieres, Gif-sur-Yvette Cedex, France, 1995, pp. 83–86.
[12] T. Binder, K. Dragosits, T. Grasser, R. Klima, M. Knaipp, H Kosina, R. Mlekus, V. Palankovski, M. Rottinger, G. Schrom, S. Selberherr, M. Stockinger, MINIMOS-NT User's Guide, Institut für Mikroelektronik, Technical University Vienna, Austria, 1998.
[13] J. McCarthy, Recursive functions of symbolic expressions and their computation by machine (Part I), Commun. ACM 3 (1960) 184–195.
[14] J. McCarthy, Lisp 1.5 Programmer's Manual, MIT Press, Cambridge, MA, 1962.
[15] D. Betz, XLISP: An Object-Oriented Lisp, Version 2.1. Apple, Peterborough, New Hampshire, USA, 1989.
[16] P. Graham, ANSI Common Lisp, Prentice Hall, Englewood Cliffs, NJ, 1996.
[17] C. Kelley, Iterative Methods for Optimization, SIAM, Philadelphia, 1999.
[18] M. Wall, GAlib: a C++ genetic algorithm library, 1994. http://lancet.mit.edu/ga/.
[19] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1996.
[20] D. Beasley, D. Bull, R. Martin, An overview of genetic algorithms: part 2, research topics, Univ. Comput. 15 (1993) 170–181.
[21] R. Rutenbar, Simulated annealing algorithms: an overview, IEEE Circuits Devices (1989) 19–26.
[22] L. Ingber, Adaptive simulated annealing, 1993. http://www.ingber.com/#ASA-CODE.
[23] P. Spellucci, donlp2 Users Guide. Part of the netlib project, 1995.
[24] P. Spellucci, Solving general convex QP problems via an exact quadratic augmented Lagrangian with bound constraints, June 1996. http://www.mathematik.th-darmstadt.de/ags/ag8/spellucci.
[25] D. Marquardt, An algorithm for the estimation of nonlinear parameters, Soc. Ind. Appl. Maths. J. 11 (1963) 431–441.
[26] J. Moré, B. Garbow, K. Hillstrom, Users Guide for MINPACK-1, Argonne National Laboratory Report ANL-80-74, Argone, IL, 1980.
[27] J. Moré, D. Sorensen, K. Hillstrom, B. Garbow, The MINPACK project, Sources and Development of Mathematical Software, Prentice-Hall, Englewood Cliffs, NJ, 1984.
[28] S. Wolfram, Mathematica — A System for Doing Mathematics by Computer, 2nd ed., Addison-Wesley, Reading, MA, 1991.
[29] R. Cartuyvels, R. Booth, S. Kubicek, L. Dupas, K. De Meyer, A powerful TCAD system including advanced RSM techniques for various engineering optimization problems, in: S. Selberherr, H. Stippel, E. Strasser (Eds.), Simulation of Semiconductor Devices and Processes, vol. 5, Springer, Wien, 1993, pp. 29–32.
[30] R. Young, F. Morehead, S. Fischer, Calibrating a complex process simulator for predicting device characteristics, First International Workshop on Statistical Metrology, Honolulu, 1996.
[31] G. Hardy, J. Littlewood, G. Pólya, Inequalities, 2nd ed., Cambridge University Press, Cambridge, 1952.
[32] D. Goldberg, Genetic Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[33] J. Holland, Adaption in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1975.
[34] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Frommann-Holzboog, Stuttgart, 1973.
[35] K.D. Jong (ed.), Evolutionary Computation, Journal published by MIT Press, 1993.