Analog Circuit Optimization using Evolutionary Algorithms and Convex Optimization

by

Varun Aggarwal

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2007
[June 2007]

Author .....
Department of Electrical Engineering and Computer Science
May 24, 2007

Certified by .....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Analog Circuit Optimization using Evolutionary Algorithms and Convex Optimization

by

Varun Aggarwal

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Science in Electrical Engineering and Computer Science

Abstract

In this thesis, we analyze state-of-art techniques for analog circuit sizing and compare them on various metrics. We ascertain that a methodology which improves the accuracy of sizing without increasing the run time or the designer effort is a contribution. We argue that the accuracy of geometric programming can be improved without adversely influencing the run time or increasing the designer's effort. This is facilitated by decomposition of geometric programming modeling into two steps, which decouples accuracy of models and run-time of geometric programming. We design a new algorithm for producing accurate posynomial models for MOS transistor parameters, which is the first step of the decomposition. The new algorithm can generate posynomial models with variable number of terms and real-valued exponents. The algorithm is a hybrid of a genetic algorithm and a convex optimization technique. We study the performance of the algorithm on artificially created benchmark problems. We show that the accuracy of posynomial models of MOS parameters is improved by a considerable amount by using the new algorithm. The new posynomial modeling algorithm can be used in any application of geometric programming and is not limited to MOS parameter modeling. In the last chapter, we discuss various ideas to improve the state-of-art in circuit sizing.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Acknowledgments

My first and foremost acknowledgement is to my advisor, Dr. Una-May O'Reilly, without whose faith and encouragement this work wouldn't have happened. I really appreciate her love for scientific enquiry, academic honesty, desire to share knowledge and teach through example, her expression of excitement in gaining new scientific insights and her passion to discuss new ideas in and beyond our field. I also thank her for posing faith in me and allowing me to explore multiple ideas in a field which was partially new to both of us. Yet another thanks for being a wonderful co-author on the many papers we wrote together. The MIT experience wouldn't have been as enriching and useful without the multiple conversations I had with her. I finally thank her for financially supporting me throughout my term at MIT, arranging visits to many conferences and providing all resources needed for my research.

I will like to thank Prof. Rodney Brooks for being extremely kind, providing me office space in his lab and allowing me to attend his very interesting group meetings. A special thanks to Prof. Russ Tedrake. Thanks to Trent McConaghy, Mar Hershenson, Parikshit Shah, Prof. Stojanovic, Prof. Dawson, Ranko Sredojevic and Tania Khanna for helpful discussions.

I thank my lab-mates for their inspiring and intellectually-stimulating company, notably, Charlie Kemp, Myunghee Kim and Juan Velasquez. A very special thanks to Lynne Salameh for being a very able colleague and putting up with my digressions into history and religion! A special thanks to friends who made life at MIT a beautiful experience, notably, Anna, Mayank, Sumeet, Suman, Ambika, Abhinav, Manas, Rohit, Srujan, Michael and many-many more.

Thanks to a few figures and concepts in human history who (which) have shaped my philosophy in life and enable me to do all I have been able to do till now: Vivekananda, Subhas Chandra Bose and the theory of natural evolution. Finally, my greatest thanks to my mother, my father, my brother and sister-in-law; without their love, much-needed guidance and unflinching faith in me, I would not have come this far.

Contents

List of Figures

List of Tables

Chapter 1

Circuit Sizing: An Introduction

1.1 Approaches to Circuit Sizing

Automatic sizing of analog circuits continues to be a research focus for the EDA industry. An analog circuit has 10 to 200 real-valued parameters that must be set in order to meet its specifications (between 2 to 20 in number). The process of setting these parameters is called *circuit sizing.* For instance, a simple two-stage opamp has around 12 parameters, which includes the width and length of all transistors and passive component values which have to be set to achieve around 10 specifications such as gain, bandwidth, power, area, noise, CMMR (common-mode-rejection-ratio), offset, settling time, slew rate and power supply rejection ratio.

For SOC (System-On-Chip) design, digital synthesis is automated to a large extent. However, the manual sizing of analog blocks is a bottleneck and governs the time-to-market. The technology evolution is guided by digital circuits (lower area and power) and the behavior of the transistor with respect to efficiency in analog design is somewhat ignored as technology is scaled. The analog designer has to live with and learn to design circuits with new unrelenting transistor models. Automatic sizing frees the designer to work on new architectures and study system-level tradeoffs. It aims for better designs and shorter time to market.

Several techniques for sizing have been proposed and implemented. In the late 80's, knowledge based approaches [1,2] were proposed. These techniques captured

the expert knowledge of a designer and translated it into a set of rules which then automatically sized a circuit for a given set of specifications. These approaches were not very useful, since for every new circuit topology and technology, a new set of rules had to be created by manual labor.

More recently, circuit sizing has been cast as an optimization problem. As it is well-known, casting any design problem into an optimization problem has two aspects: Modeling the design problem as an optimization problem and solving the modeled problem. These steps are not independent and influence each other, for instance, the model of the problem will decide the optimization method that can be used. One could also look at it in another way, where the problem is molded is fit into a template in a way so that it could be easily solved.

In the context of circuits, the accurate performance of circuits is that which is measured when the circuit is fabricated on silicon. Since the designer does not have access to this during the design process, the designer relies on a simulator which models the characteristics of the silicon elements and runs numerical algorithms to calculate circuit performance. The widely accepted simulator is SPICE [37], which is the standard in industry and academia. Therefore, with respect to sizing, the final check-point is *SPICE correctness.* The most accurate model that can be used for optimization is using SPICE as a blackbox evaluator, where one feeds in the circuit parameters and gets the circuit specification values.

Apart from SPICE, the circuit designer also has access to *circuit equations.* These equations can be derived by symbolically parsing the circuit with some assumptions with regard to the transistor behavior. Humans derive these equations to understand the circuit better [24], while there are programs which automatically derive these expressions as well [16]. These expressions can be used to derive circuit performances much faster than SPICE. However they aren't as accurate as SPICE, since the assumed transistor behavior is not accurate and also, approximations are made in circuit analysis.

These two models are available for evaluating circuit performance specification. However, modeling the circuit optimization problem does not only include how the

performance specifications are measured, but also how the optimization problem is set up. There could be many variations to this such as the optimization problem can have multiple objectives and multiple constraints; one objective and multiple constraints; a series of optimization problems with one objective and multiple constraints; transformations in problem to make it convex.

Recently three approaches have been popular for circuit optimization. Table 1.1 summarizes the modeling approach and optimization algorithm related to each of this approach.

Name of Approach	Optimization Model	Optimization Algorithm
Simulation based Approach	SPICE Evaluation Single objective, multiple constraints [34, 36] Multiple objectives, multiple constraints [40]	Blackbox Optimization Algorithm, e.g., Simulated Annealing, Stochastic Pattern search Multi-objective Genetic Algorithms
Equation based Approach	Equation-based Evaluation Single objective, multiple constraints (Chapter 9 in [16]) Multiple objectives, multiple constraints	Blackbox Optimization Algorithm, e.g., Simulated Annealing Multi-objective Genetic Algorithms
Geometric Programming	Posynomial equations with log-log transformation: Single objective, multiple constraints [23, 31] Series of single objectives, multiple constraint problems [13]	Convex Optimization: Geometric Programming Reverse geometric programming

Table 1.1: Different Approaches for circuit optimization

As shown in Table 1.1, the Simulation based Approach and Equation-based Approach use black-box optimization algorithms, while they differ in the way circuit performance is evaluated, the former using SPICE simulations and the latter circuit equations. On the other hand, Geometric Programming uses circuit equations in posynomial form [7], transforms them in a certain way (log-log transform) to derive a convex problem, which can then be solved efficiently as a geometric program. For the

equation-based approach and Geometric Programming approach the way of evaluating circuit performance is same, however the optimization problem formulation and way of optimization is different. In practice, the distinctions made in the table have weak boundaries and there have been approaches which draw from more than one of the approaches described, e.g., ASTRX/OBLX uses equations for high-simulation time specifications and simulators for measure the others [34]; Some geometric programming approaches [10] use simulation data to find equations for some specifications and hand-written equations for others.

It may be noted that the popular names of these approaches are incomplete and confusing. While the first two approaches (in the order of mention in Table 1.1) derive their name according to the way circuit performance is evaluated, the latter derives its name from the approach to problem formulation and optimization. The reasons for this are historical. When circuit sizing was first cast as an optimization problem, all popular methods used black-box optimization methods and thus the names just distinguished them in terms of how performance was evaluated. Later, when geometric programming was applied to circuit sizing [23, 31], for distinction, it was named according to the name of the optimization method. In principle, the names of the approach should include both the modeling approach and the optimization method. The approaches can thus be called, 'Simulation-based Black-box Optimization Approach', 'Equation-based Black-box Optimization Approach' and 'Equation-based Convex Optimization Approach' in order.

Currently, the simulation based approach and geometric programming are most popular within the academia and industry. They both have pros and cons which are discussed in the next section.

1.2 Comparison of Sizing Approaches

Geometric Programming and simulation-based approaches have both found acceptance in academia and industry, however the purpose and methodology to use them have been different. In this section, we first compare them based on various metrics.

We then explain how they are useful in different scenarios based on these comparisons.

1. Accuracy: In geometric programming, though the optimization method finds the global optima, inaccuracy creeps in due to the inaccuracy of the equations. There are two reasons for this inaccuracy. First, the derived equations use approximate circuit analysis. Second, the equations need to be in posynomial form and not all circuit equations can be modeled as posynomials (for e.g., saturation constraints [23]). On the other hand, the optimization approach, i.e. geometric programming tranforms the problem into a global optimization problem and guarantees global optima.

   In the case of simulation-based approaches, the modeling is accurate, since SPICE is used for performance evaluation. However the optimization method provides no guarantee of finding the global optima. The popularly used techniques of simulated annealing, evolutionary algorithms, etc. provide no mathematical guarantee of finding the actual optima and the optima within an error bound. There are empirical results of convergence [12, 45], but there haven't been any studies to test these convergence results for circuit optimization problems or test problems of similar size. For instance, NSGA-II used in [9, 41] has been only benchmarked for 10 dimensional problems and two objectives [12], where it is shown to be able to find the global optima. However, a simple folded-cascode opamp has more than 20 parameters to be optimized and several objectives. Possibly, sub-optimal optimization results create inaccuracy in the optima found by simulation-based approach.

2. Effort: The effort spent by the designer to use a tool based on the simulation-based approach is moderate. The designer has to set up SPICE files for measuring different performance measures, select parameters to be optimized and set their ranges. The most time-consuming part here is that of setting the SPICE files. However, this effort can be reused across circuits with the same functionality. On the other hand, the effort spent in setting up a geometric program is much higher. In the tools released by commercial entities such as

Barcelona Design, the designer had to write equations for objectives and constraints themselves, which was cumbersome. Furthermore, the equations need to have posynomial form (in some recent commercial tools, this condition has been relaxed). This has to be done for each circuit topology. Though there has been some work in automatically deriving these equations, it hasn't yielded good results due to the inaccuracy and poor scalability of these approaches, the resultant expressions are not well-suited for optimization. No commercially available tool offers automatic modeling of equations to our best knowledge. In personal conversations, the author hasn't found designers forthcoming to write equations. They aren't sure whether they know accurate equations for each specification and claim to use what they call 'design intuition' to size circuits. Given a choice between writing equations and sizing the circuit themselves, the designer chooses the latter.

3. Time: The time component for the circuit-sizing comprises of two durations. First, the time to setup the optimization problem. Second, the time taken by the optimization algorithm to size the circuit. In geometric programming, as discussed before, the time spent in setting the circuit optimization problem is high. The designer may take any amount of time to write the equations and usually must iterate because getting equations first time correct is not easy. Thus the time may include a debug cycle. Also, the time increases with the size and complexity of the circuit. The second component of time, i.e. optimization time is low for geometric programming. A GP with 1000 variables and 10000 constraints is solved in less than a minute on a small desktop computer [7].

   In case of Simulation-based approaches, the time to setup the optimization problem is low and there is high potential of reuse of scripts. However, the time of optimization is much higher due to two reasons. Firstly, SPICE is used for evaluation in optimization and thus the time of optimization is limited by time of simulation, which is very high for transient specifications. Secondly, blackbox optimization methods are *weak methods* and not specific to properties

(or structure) of the optimization problem. Their advantage is that they work moderately well on a high number of problems, but in trade-off, their time of run is high. The most popular algorithms used for circuit optimization are population-based (evolutionary algorithms, stochastic pattern search), which gives the advantage of natural parallelization on clusters for fast execution. This has been successfully implemented in industry [22]. For instance, a 20 parameter Power Amplifier took 10 hours to optimize when run on 16 300MHz Sun-Ultra 10's and 4 300MHz dual-processor Ultra 2's [36].

4. Suitability for system Level design: Automated system-level design remains a cherished dream for the analog CAD community. In general, when we talk about circuit sizing we refer to cells. These cells together compose a system. For instance, an Analog-to-Digital converter is a system which is composed of multipliers, filters, comparators, summers, etc. which are cells. A system-level design problem is much harder given that it has a much larger size (number of parameters to be set), is more complex (leading to complex equations) and has much higher simulation time. It is not feasible to solve the system-level design problem all at once. In general, the approaches toward it first break the problem into smaller pieces (such as design of cells), solve them in individually and then combine the solutions to solve the system level problem. Though there are approaches for automated system-level design, the CAD community is often accused of over-simplification and neglecting some of the 'real issues', when addressing system-level design!

   Geometric programming has been used for system level design [10, 13]. The advantage of geometric programming is its speed and accuracy in solving large problems [7]. Given this advantage, the system-level problem if modeled correctly could be solved all at once by geometric programming.[1] This has been shown for a PLL [10]. Also, GP can be used to solve cell-level problems, which can then be combined to solve the system-level problem. In [13], a set of sev-

[1]It has to be kept in mind that formulating a system level problem correctly by using equations is not easy and will be, in general, inaccurate

eral geometric programming problems are used to enumerate the design space (or trade-off curve) for a cell, which can then be used for efficient system-level design. The low optimization time of geometric programming and the guarantee of an optimum with even large problems (provided the model is accurate) makes it extremely attractive for system-level design. However, the modeling component is of concern.

Simulation-based approaches have been used in more than one way for system-level design. They cannot and have never been used to solve the system-level problem all at once. This is not only due to the high simulation time of the system, but also due to the unknown scaling properties of black-box optimization algorithms with the size of problem. General experience says that these algorithms scale badly with the size of problem. A simulation-based approach for system-level design uses a hierarchical approach. It breaks the problem into cell-level problems, solves these individually and combines the solutions to solve the system-level problem. The approach may also require iteration between solving the system-level and cell-level problem. A survey of some of these approaches is given in [19]. Some recent approaches produce trade-off curves for cells, which are combined to find the optimal solution [40, 15]. These approaches are fundamentally limited by the time needed to generate the trade-off curve for each cell of the system, which is high. There have been some claims of reusability, however reusability is under question given that the constraints change for cells in different designs. There has been reasonable success in system-level design using stochastic optimization, but there are still a lot of open questions.

5. Suitability for robust design: Robust design refers to design of circuits which guarantee to function well in case of environmental variations, inter-die and intra-die variations in process. Earlier, corner-analysis was used to ascertain robustness of the circuit, however more recently, statistical measurement of variation has been emphasized [44]. With regard to simulation-based approaches, ascertaining robustness of a circuit requires multiple simulations (monte-carlo

analysis) and adversely effects the run-time of the algorithm (possibly 2 to 3 orders of magnitude). Researchers have focused on designing techniques which require a small number of samples to ascertain robustness as opposed to a complete monte-carlo simulation ([39], Solido Design Automation). Also, there has been design of heuristics which do robustness estimation for only a few circuit visited in the optimization [41], however the accuracy of these approaches is questionable.

With regard to geometric programming, the challenge is to model the distribution of variation parameters in a form which can be efficiently optimized. Secondly, the MOS transistor has to be expressed as a posynomial in terms of the variation parameters (Discussed in Chapter 2). There is no physical basis to suggest that the distributional properties of the variation parameters shall yield to convex optimization. In [44], the authors have shown that geometric programming can be used for robust optimization by making certain assumptions about the distribution of the random variables. The formulation of a robust GP doesn't guarantee global optima, however works well in practice. The advantage of geometric programming is that it optimizes lightning fast and is not limited by simulation time as in Simulation-based approaches. If the model for robustness is correct, geometric programming scales very well to handle a very high number of variation parameters. Evidence for this has already been shown in [44]. In summary, with regard to robustness, simulation-based approaches are prohibited by their high simulation time, while geometric programming faces the challenge of accurate modeling.

Table 1.2 summarizes how Simulation-based Approaches and Geometric Programming compare on various metrics of concern. In a broad-sense, geometric programming is useful to solve large problems quickly, needs high effort and time in optimization problem modeling and is promising for robust and system level design. On the other hand, simulation-based approaches are not well suited for large problems (when not decomposed), need low effort and time in setup and have shown some success in

Metric	Geometric Programming	Simulation based Approaches
Accuracy	Inaccuracy in process and circuit modeling Accurate global optimization on given models	Accurate circuit models since SPICE is invoked in-loop Inaccuracy or no guarantee in stochastic optimization.
Effort	High designer effort in writing accurate equations	Little effort by designer needed for setting SPICE scripts for specification measurement and choosing variables to be optimized.
Time	High time in optimization problem formulation: Time spent in analyzing circuit and writing equations. Very fast optimization by interior-point methods	Little time needed in problem formulation High time of optimization because i. SPICE is invoked in loop of optimization, ii. Weak algorithm and not circuit specific
Suitability for system level Design	By generating fast trade-off curves for cells and their use for system-level optimization By combining cell-level equations to form system-level equations and optimizing the whole system.	Multi-objective approaches allow trade-off generation, which can be used for hierarchical bottom-up synthesis [9].
Suitability for robust optimization	Challenges in inclusion of robustness in a form optimizable by convex techniques Fast optimization on modeling	Challenges in decreasing number of SPICE simulations needed for robustness measurement.

Table 1.2: Comparison of Simulation-based Approaches and Geometric Programming on various metrics

system level and robust design.

These differences have interestingly resulted in different scenarios and ideology for use of these two approaches. Given the low time and effort of setup and general usability for any circuit, simulation-based approaches are used in *sizing tools*. These sizing tools are used by designers for doing cell-level optimization. Given the high run time, a popular model is to set up the optimization in evening and let the computer do the work, when humans sleep! The designer gets the results in the morning. The tools may give approximately good results, which can be used as starting points by the designer. Another popular feature of these tools is to show trade-off curves between important performance measures (such as power and area), which are not accessible by manual approaches. In this scenario, the tools don't directly address system-level design(however, may do so in consultancy models). Analog Design Automation and Neolinear Inc. commercialized such tools and were subsequently bought by Synopsys and Cadence.

On the other hand, geometric programming needs high effort and time in modeling the circuit and setting up the optimization problem, but gives quick and accurate optimization given the model is good. To suit this trade-off, geometric programming is used in a library-based model and an IP-based model.[2] In the first model, the CAD company provides libraries containing already setup optimization problems for some of the most commonly used and important analog circuit topologies. The designer can then size these blocks lightning fast and accurately. However the designer is limited to the topologies provided by the CAD vendor and cannot size a newly designed topology. This becomes more complicated given that the new topologies are generally proprietary and cannot be disclosed to the CAD vendor. In the IP model, the IP is sold and not the CAD tool. Generally these IPs are at the system-level. The vendor studies and models some of the most important IPs (such as PLLs) as a geometric program, which could then be synthesized very fast. Given the customer specifications, the vendor can very quickly size the IP using geometric programming.

[2]IP is abbreviation for Intellectual Property here. In the semiconductor industry, IP refers to synthesized block, which can be a circuit, a digital block or a processor.

This decreases the time to market and ensures the design is well-optimized. The IP approach is ideal to address the system-level design problem, but is again limited to the few topologies mastered by the vendor. Both these models can be extended to consultancy models, but they could run into issues of confidentiality. Sabio Labs and Barcelona Design has experimented with both the library and IP model (and also the not-so-good tools model!).

Given the presented scenario, we discuss our approach to research in the field of analog CAD.

1.3 Research Approach and Problem Statement

We are interested in enhancing the state-of-art in circuit optimization by improving the current methodologies and designing new ones. Given the trade-offs enumerated between the different approaches above, it will be a contribution if we can improve the accuracy of optimization or decrease its time without increasing (or minimally increasing) the effort spent by the designer. We want our new techniques to be suitable for extension to robust circuit optimization and system-level design.

In this thesis, we will improve the accuracy of the geometric programming flow without increasing the effort spent. Our approach will accommodate system-level design and robust optimization. As will be explained in detail later, the geometric programming flow proposed by Hershenson and Mandal (independently) has a step which encapsulates process model into the circuit design equations to get the final model for optimization. The inaccuracy in geometric programming is due to two reasons: i. Inaccurate Process (transistor) Models, ii. Inaccurate design equations. We are interested in exploring how we can address the former, i.e., design more accurate posynomial process models. We use a hybrid of genetic algorithm and convex optimization technique (linear programming or quadratic programming) for this purpose. Since these models are reusable with any circuit, this technique improves geometric programming accuracy without compromising the time or effort required. The technique will also be useful to build accurate process models with multiple statistical

parameters for robust optimization.

The proposed technique is a general technique to build posynomial models with real-valued coefficients, exponents and variable number of terms. It is not just limited to circuits. It can thus be used for modeling of various applications for which geometric programming may then be used for optimization. We will also discuss in future work, how the technique can be used to lower the effort in geometric programming approaches and make it more accurate by addressing the second modeling step, i.e. of design equations.

In Chapter 2, we will discuss and compare the approaches used to cast circuit sizing as a convex optimization problem. We will motivate the idea of reuse in circuit optimization. In chapter 3, we will discuss the state-of-art posynomial modeling techniques, their deficiencies and describe in detail our algorithm for posynomial modeling. In Chapter 4, we will quantitatively show that our technique outperforms the state-of-art techniques. Chapter 5 will discuss future work.

Chapter 2

Circuit Sizing as a convex Optimization Problem

In this Chapter, we will discuss the different approaches to express an analog circuit sizing problem as a convex optimization problem, primarily a geometric program. There are two basic approaches to do this. Based on the terminology of naming circuit sizing approaches introduced in Chapter 1, we name these two approaches henceforth: Equation-based Convex Optimization and Simulation-based Convex Optimization.

In Section 2.1, we will discuss the form of a geometric program and how it can be converted to a convex optimization problem. In Section 2.2 and 2.3, we will discuss two methodologies for analog circuit sizing using geometric programming. In Section 2.4, we will provide a perspective on these two approaches and in Section 2.5, we will discuss how we improve the accuracy of one of these approaches without sacrificing run-time or increasing effort of designer.

2.1 Geometric Programming

A geometric program is a non-linear optimization problem, which can be transformed into a convex form and solved efficiently [7]. It should be noted that a geometric program is not convex, i.e. its objectives and constraints may or may not be convex. However, by using a log-log transformation it can be converted into a convex form.

To represent a geometric program, we first define a posynomial function. Let $\overline{x}$ be a vector of n real positive variables. A function f is called a posynomial function of $\overline{x}$ if it has the following form:

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{t} c_k x_1^{\alpha_{1,k}} x_2^{\alpha_{2,k}} \ldots x_n^{\alpha_{n,k}}, \quad c_j > 0, \quad \alpha_{i,j} \in \Re$$

Posynomials are like polynomials, but differ in two ways. First they can have fractional exponents and second, they have only positive coefficients for all terms. When $t = 1$, i.e. a single term posynomial is called a monomial. Geometric programming solves an optimization problem of the following form:

$$\begin{array}{lll} \text{minimize} & f_0(\overline{x}) & \\ \text{subject to} & f_i(\overline{x}) \leq 1, & i = 1, \ldots, m, \\ & g_i(\overline{x}) = 1, & i = 1, \ldots, p, \\ & x_i > 0, & i = 1, \ldots, n \end{array}$$

Here f_i and f_0 are posynomials while g_i are monomials. A geometric program can be converted to a convex optimization problem with the following transformations. The original variables, x_i are replaced with their logarithms, $y_i = \log x_i$ ($x_i = e^{y_i}$). The logarithm of the objective f_o is minimized instead of f_o. The constraint $f_i(\overline{x}) \leq 1$ is replaced with $log(f_i(\overline{x})) \leq 0$ and the constraint $g_i(\overline{x}) = 1$ is replaced by $log(g_i(\overline{x})) = 0$. Since we take log of both the input variables and each posynomial function (constraints and objectives), this transformation is called log-log transformation. The constraints $x_i > 0$ are implicit in the log-transformation. The log-transformations are valid since logarithm is a monotonic function.

On doing a log-log transformation, posynomial functions become convex, while monomials become linear. This makes the objective and inequality constraints convex and the equality constraints linear. This is the classical form of a convex program and can be solved efficiently for the global optimum. A geometric program with 1000 variables and 10000 constraints is solved in less than a minute on a small desktop

computer [7].

2.2 Equation-based Convex Optimization

The first approach to use geometric programming for circuit optimization was invented by Hershenson and Mandal [31, 23]. It was observed that hand-written circuit equations with the MOS transistor abstracted by the square-law yielded posynomial expressions for circuit specifications. There were some exceptions to this, such as expression for saturation constraints. However work-arounds and approximations were designed to express these as posynomial to formulate the circuit as a geometric program. This yielded very quick optimization for circuits.

However, it is well-known that the square-law is inadequate to model the transistor, more so with shrinking technology. Currently, the very complicated BSIM model is used to model transistors for SPICE with parameters learnt from actual fabrication data. In fact, the transistor behavior has become so complicated, that fab delivered models for sub-micron technology nodes use several different BSIM models for different operating range of the transistor, since one model is inadequate to capture the behavior for the whole range accurately.[1]

Given this scenario, the square-law approximation is inappropriate for optimization formulation. To address this, a hierarchical decomposition of the circuit equations was identified.[2] This decomposition is shown in Figure 2-1.

The optimization formulation goal is to express the circuit constraints and specifications as a function of circuit parameters, such as the width and length of the transistors and value of passive components or various other choice of design variables [28]. This formulation is decomposed in two-steps as follows:

1. The MOS transistor parameters are expressed as function of the transistor design variables. One choice for design variables is width, length, gate-source

[1]Specific details are omitted due to intellectual property issues!

[2]Though this decomposition is apparent in the works done by Hershenson, et.al., it hasn't been explicitly stated and discussed as a general principle with its implications in the CAD community! We will talk about these implications in a Section 2.4.

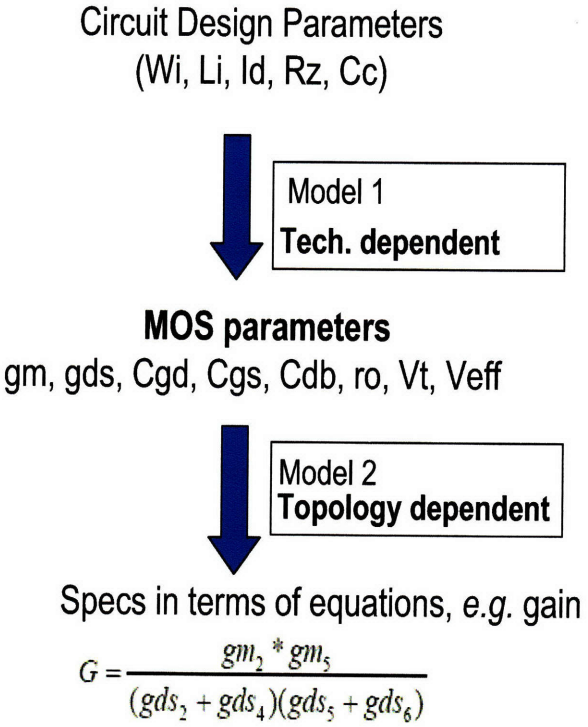


Figure 2-1: **Circuit specifications as a function of circuit parameters: A two-step decomposition**

voltage and drain-source voltage. Alternatively, gate-source voltage could be replaced by drain current. The modeled transistor parameters include small-signal parameters and large signal parameters. A simplified small-signal model and parameters are shown in Figure 2-2. For simplicity, it is assumed that the body and source are connected. The small signal parameters comprise of the various transconductance like gm, gds, etc., resistances, r_0 and various capacitances. The large signal parameters are various voltages like threshold voltage (V_t), saturation voltage (V_{sat}), effective voltage (V_{eff}) and the drain current (I_d). Depending on whether the drain current or gate-source voltage has been considered as a design variable, the other could be modeled as a parameter.

2. The circuit specifications are expressed as a function of MOS transistor parameters by hand-written equations. As an example, in Figure 2-1, gain of a simple two-stage opamp is expressed as a function of the transconductance of various transistors which compose the circuit.

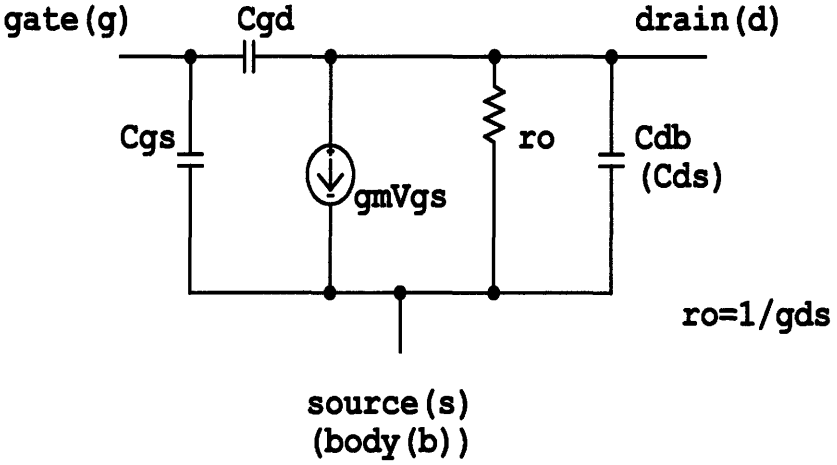


Figure 2-2: **A simplified small-signal model of the MOS transistor**

This decomposition liberates the optimization formulation from dependence on the square law. The transistor parameters are learnt from actual simulation data

for a given technology and are not based on the square law. In [23, 10, 13, 8], monomial models were learnt for transistor parameters as a function of MOS design variables. Then, these models were used to replace the transistor parameters in hand-written equations for all specifications. Since monomials yield posynomials on addition, multiplication and division, this results in posynomial objectives and constraints. This encapsulation allows the expressions of specifications as posynomials of design variables. All details of how exactly the optimization problem is formulated for a simple opamp is given in [23].

To summarize, the Equation-based Geometric Programming approach uses simulation data to derive models for transistor parameters in terms of design variables and uses hand-written equations for expressing specifications as functions of transistor parameters. The learnt models for transistor parameters are plugged in to the specifications to form a geometric program.

2.3 Simulation-based Convex Optimization

The second approach first used by Daems, et.al. [11] and later used in [29] follows a different strategy to cast a circuit as a geometric program. This approach has the following flow:

1. All design variables for the circuit are enumerated and ranges are set for all these variables.

2. Many sets of values of design variable are chosen which represent the complete design space well statistically. In [11], this was done by using Design of Experiments (DOE) [26]. The circuit is simulated using SPICE at all these sets of design variables to find value for all specifications.

3. Given this data, a posynomial expression is derived for each specification in terms of the design variables. This derivation is done using black-box regression techniques.

4. This results in a geometric program, which is then solved.

This approach uses simulation of the whole circuit for deriving posynomial expressions for circuit specifications. It doesn't use hand-written equations or any encapsulation of models to derive the final expressions. It should be noted that this approach doesn't use any assumption (Square law or otherwise) about the MOS transistor behavior and doesn't suffer from inaccuracy in that regard.

2.4 A perspective on the Approaches

In the last two sections, we have discussed two approaches to geometric programming. Though both these approaches use convex optimization, the methodology to setup the optimization problem makes a big difference in how they compare on various metrics of concern. In this section, we will compare these approaches and provide a perspective on how the former could be improved. We will show how the two step decomposition done in Equation-based Convex Optimization approach has far-reaching consequences on the run-time of the methodology, which in turn influences the scalability and suitability for robustness.

Both the approaches suffer from the fundamental limitations of geometric programming, i.e. the need for posynomial models for specifications. Some of the circuit specifications and transistor parameters are not posynomials, which leads to inaccuracies. The second level of inaccuracy creeps in depending on the question as to how accurately can we identify the posynomial representing the specification, assuming that it indeed has a posynomial expression. This depends on the technique to fit posynomial models and also has repercussions on the run time of the methodology. Thus it isn't easy to discuss the run time and accuracy of the approaches separately and we will discuss them here together. We will finally argue that run time and accuracy can indeed be decoupled for Equation-based Geometric Programming.

The Simulation-based Geometric Programming approach builds posynomial models for the circuit specifications directly in terms of the circuit parameters. This approach seemingly gets rid of the inaccuracies of the hand-written equations. However there are significant problems with this approach. Firstly, the posynomial models

comprise all design variables of the circuits, which results in a very high dimensional fitting problem (circuit design variable count goes anywhere between 10 to 100). The higher dimensionality of the problem makes it harder to get accurate models that will generalize well throughout the space. To address this, in [11], second order posynomial models with integer-valued exponents were used. There isn't an intuition why second order models will generalize well. In fact, the first order expression for gm involves square-roots.

Secondly, to get good models for a high-dimensionality problem, a high number of design points need to be sampled. As the size of the circuit increases, the dimensionality of the problem increases leading to a need of exponentially more number of modeling points for accurate models due to the 'curse of dimensionality' [21]. An exponentially increasing sample set implies an exponentially scaling time of simulation. Thus the use of simulations for modeling the optimization problem renders bad scaling properties. With bigger problems, it will also be more difficult to derive accurate models. Again, for modeling variation for robust optimization, similar scaling issues will be encountered. This makes the Simulation-based Convex Optimization approach similar to Simulation-based black-box optimization approaches with regard to accuracy, run-time and scaling.

The Equation-based Geometric Programming approach uses simulation data to derive transistor models and relies on hand-written equations for specifications. The current approaches [23] express transistor models as monomials, whereas more expressive posynomial models may be used. This may lead to inaccuracies. The second source of inaccuracy lies in the hand-written equations, which are generally approximate. This inaccuracy becomes more of an issue with expressions for transient specifications like settling time, which are highly non-linear and hand-written expressions are not sufficient. The modeling time comprises the time to build models for the transistor and designer time in writing equations.

The decomposition used in this approach leads to decoupling between run-time and accuracy and also has very nice scaling properties. This will be now discussed. It should be noted that the first step, i.e. of building transistor models is independent of

the topology of the circuit that is optimized. It just depends on the technology node. Therefore, once this model is derived, it can be used for any circuit designed in the given technology. This facilitates *reuse* of the same model over several circuits. This in turn implies that one can amortize a lot of one time effort in deriving very accurate models for the transistor without adversely affecting the run-time of optimizing any circuit. This decouples accuracy of models and run-time of optimization. This is also attractive from the point of view of robust optimization, since the effort will be only spent on modeling the transistor as a function of variation parameters.

Secondly, the models for circuit specifications in terms of transistor parameters are independent of the technology node and only depend on the topology.[3] Again, the time and effort spent on deriving accurate expressions for a topology can be re-used for the circuit several times on various technology nodes. This works well for large circuits, since once their model is derived, it can be reused leading to very good scaling. It should be noted, that this modeling is not fundamentally limited by simulation time unlike all Simulation-based Approaches. This fits well with the library based model and IP model for geometric programming discussed in the Chapter 1.

In summary, for Simulation-based Convex Optimization the accuracy and run-time of optimization are coupled, whereas for Equation-based Convex Optimization accuracy and run-time of optimization are decoupled due to the potential of reuse of models.

2.5 Our Approach

In the last section, we motivated that improving the accuracy of Equation-based Convex Optimization Approach does not sacrifice the run-time or scaling of the approach. This improvement can be accomplished by improving the accuracy of either or both of the two decomposed steps of the optimization modeling. In this thesis, we will develop new techniques to derive accurate posynomial models for MOS transistor. This

[3]This holds true for the small signal and DC specifications, but not necessarily transient specifications. We shall come back to this in Chapter 5: Future Work.

approach remains loyal to our goals set in Chapter 1, i.e. improve the time or accuracy of the circuit sizing flow without adversely influencing the effort spent. In the next Chapter, we will discuss various state-of-art posynomial modeling approaches and describe our algorithm for posynomial modeling.

Chapter 3

Algorithm to model posynomials

In this Chapter, we describe our algorithm for deriving posynomial models for a given set of data. In Section 3.1, we will present the problem of posynomial modeling. In Section 3.2, we will discuss the various state-of-art methods for posynomial modeling. In section 3.3, we will discuss our approach to solve the posynomial modeling problem. In Section 3.4 and consequent sections, we will discuss our genetic algorithm to solve posynomial modeling problem.

3.1 Problem Statement

Assume a functional space, where $\overline{x}$ is the input vector and y is the corresponding output values. A set of data which samples this space at various input vector values and corresponding output values is given. The problem is to find a posynomial mapping between $\overline{x}$ and y. The posynomial expression generated should be representative of the actual underlying mapping between the input. The exponents of the terms of the posynomial expression and coefficients belong to the real and positive-real domain respectively. This problem can be expressed as minimization of the error between the output values predicted by the generated posynomial and the actual outputs. A function such as the mean-square error, mean-absolute error, max absolute error, etc can be used as the error metric. The formulation of these different error metrics has been tabulated in Table 3.1.

Abbr.	Name	Expression	Solver
RMSE	Root Mean Square Error	$(\frac{\sum_i (f(\overline{x_i})-y_i)^2}{N})^{0.5}$	Quadratic Program
MeAE	Mean Absolute Error	$\frac{\sum_i \lvert f(\overline{x_i})-y_i \rvert}{N}$	Linear Program
MaAE	Max Absolute Error	$\max_i \lvert f(\overline{x_i}) - y_i \rvert$	Linear Program
RMRSE	Root Mean Relative Square Error	$(\sum_i \frac{(f(\overline{x_i})-y_i)^2}{y_i^2}/N)^{0.5}$	Quadratic Program
MeRAE	Mean Relative Absolute Error	$\sum_i \lvert \frac{f(\overline{x_i})-y_i}{y_i} \rvert /N$	Linear Program
MaRAE	Max Relative Absolute Error	$\max_i \lvert \frac{f(\overline{x_i})-y_i)}{y_i} \rvert$	Linear Program

Table 3.1: Different error-metrics and the technique to optimize them

Specifically in the circuit context, each MOS transistor parameter modeled will constitute a separate problem. In each of these problems, MOS transistor design variables will be $\overline{x}$. These will be chosen as width, length, current and drain-source voltage. Also, gate-source voltage may be used instead of drain current. The output variable y in each of these problems will be one of the following parameters of the MOS transistor: gm, gds, ro, Veff, Vt, Vdsat, Cgs, Cgd, Cgs.

3.2 Current posynomial modeling Approaches

In this section we enumerate previously published methods used for designing posynomials. We also include a method to generate a max-monomial which is convex in log-log space and can be used in optimization formulation:

1. Monomial Fitting algorithm: In [7], an algorithm for generating a monomial fit is presented. A monomial becomes a hyperplane on log-log transformation. This observation is used to convert the monomial fitting problem into a linear regression problem. This is expressed mathematically as follows:

   $y = cx_1^{\alpha_1}...x_n^{\alpha_n}$

   $log(y) = log(c) + \alpha_1 log(x_1) + ... + \alpha_n log(x_n)$

   $log(y)$ is a linear function of $log(x_i)$. A linear regression can be done to find c and α_i to minimize RMSE (and other metrics too) in the log-log domain. Note

that optimal RMSE in log-log space does not translate to optimal RMSE in real space. Mean square error in log-log space approximately corresponds to RMRSE in real space.

In [7], the author also recommends to use the monomial generated by log-log regression to generate a posynomial heuristically. Monomial terms for the posynomial may be generated by tweaking the exponents of the initial monomial. A gradient descent algorithm may then be used to find the optimal coefficients for the terms of the posynomial. It should be noticed that this method is local and searches in the space around the best-fit monomial. It gives no guarantees of optimality.

2. Quadratic Posynomial fitting: In [11], a method to find quadratic posynomial models is presented. The author presents three methods which are now described. The first method called, 'Indirect Fitting Method', fits a quadratic polynomial to the function by linear regression. It then approximates the terms with negative coefficients with terms with positive coefficients and negative exponents (for exact transformation, see [11]). The second method called, 'Direct Fitting Method without Template Estimation', fits a second order posynomial (with negative exponents as well) to the data. They invent a new algorithm to do this, however, the same problem can be solved with quadratic programming since it has a convex objective function and linear constraints. The third method, 'Direct Fitting Method with Template Estimation' does an initial fit with a polynomial with few terms. It estimates new monomial terms from terms with negative coefficients. Once, it has estimated all terms, it re-fits the coefficients by the constrained optimization approach to be all positive. The authors recommends to use this approach due to the explosive computational complexity of the 'Direct Fitting Method without Template Estimation' with the number of input variables. In [29], a new approach for fitting of quadratic posynomials is presented. It uses a projection based method to reduce the computational complexity of the 'Direct Fitting Method'.

All these methods fit only quadratic posynomials and also, cannot have real-valued exponents. In [14], a method for fitting posynomials with real-valued exponent is published, however it simply does a gradient descent to re-tune exponents of the quadratic exponent. The method is essentially local.

3. Max-Monomial (MaxMon): In [30], a method to fit max-monomials is presented. Max-monomials are piecewise linear (and convex) in log-log space (Note: Posynomials are convex in log-log space, not necessarily piece-wise linear). They can also be used in convex optimization [7], however may have issues with sharp transitions. The method in [30], starts with an initial set of partitions of the space and fits monomials to each partition using log-regression. It uses a heuristic to expand and contract partitions depending upon the errors of the fit in each partition. The method is a heuristic and doesn't claim global optimality.

We know that first order MOS models have fractional exponents for design variables (consider gm). However, none of the published methods address the problem of fitting posynomials with real-valued exponents. The two approaches for the same [7, 14] are local in nature: the first one doing a local search around the best fit monomial; the second one being local to integer-valued exponents in the range $[-2, 2]$. There is nothing that suggests that the posynomial underlying the data is quadratic or local to a quadratic posynomial. Neither are there suggestions that the two terms constituting a posynomial have to be local to each other. The Max-Monomial method is useful and global in nature. It fits max-monomials instead of posynomials, which could be used for optimization as well (though with limitations). We shall compare our technique with this method. We will also compare our technique to a degenerate two term posynomial (POSY-2) learnt from the monomial generated by the log-log regression (first in order in above list). We fix the exponents of the monomial model, however re-learn the coefficient of the monomial and an extra constant by a QP formulation (constrained linear regression). This generates a two term posynomial (POSY-2), a monomial term and a constant term.

3.3 Solution Approach

The problem of posynomial modeling may seem to be a typical regression problem. This indeed is the case if the model is constrained to have integer-valued exponents for all design variables and the exponents are limited in a range. In this case given the exponent ranges all possible terms can be enumerated combinatorially. Then the problem can be solved for minimizing RMSE using a quadratic program or a lagrangian approach with the constraint that all coefficients should be more than zero [21].

Simple as this may seem, it becomes tricky due to the combinatorial explosion in the number of terms even for a modestly large range of exponents and number of design variables. This creates two problems: a. The problem becomes computationally very expensive to solve (intractable in some cases), b. It leads to the problem of overfitting. These can be solved using techniques like Support Vector Machines, subset selection, etc. and other regularization techniques [21]. Variations of these techniques were used in [11] to design quadratic posynomial models.

However, given the fact that posynomials for MOS parameters will have fractional exponents, we want to develop a modeling approach where the exponents can be real-valued. This largely changes the scenario. For any given range of exponents, the number of possible terms are now infinite and not countable. They cannot be explicitly enumerated in a combinatorial way as done before.

We can still bound the number of terms by explicitly setting the maximum number of terms in the posynomial. This reduces the problem to search for exponents and coefficients from the real-domain to get the best-fit posynomial. This can be formulated as an optimization problem in the following way:

$$\overline{\alpha_{opt}}, \overline{c_{opt}} = \arg \min_{\alpha_{jk}, c_m \forall j,k,m} \sum_i (f(\overline{x_i}, \overline{\alpha}, \overline{c}) - y_i)^2 \tag{3.1}$$

$$f(\overline{x_i}, \overline{\alpha}, \overline{c}) = \sum_{k=1}^{t} c_k x_{i1}^{\alpha_{1,k}} x_{i,2}^{\alpha_{2,k}} \ldots x_{in}^{\alpha_{n,k}} \tag{3.2}$$

Unfortunately this problem isn't convex and there are no efficient ways to solve it deterministically. We therefore decided to solve it heuristically. We wanted to design an approach which could exploit in some way the structure of the problem than doing a straight black-box optimization treating all optimization variables the same way. Such an approach will be inefficient.

Given the setup of the problem, one could observe that given the exponents of all terms, the problem reduces to a regression problem (with constraint, $c_i > 0$) which can be efficiently solved using a quadratic program. This fact could be exploited to decompose the problem in two parts: a. A heuristic which searches for the exponent values of all posynomial terms, b. A quadratic program which finds the coefficients given the exponent values found by the search. This is depicted in Figure 3-1.

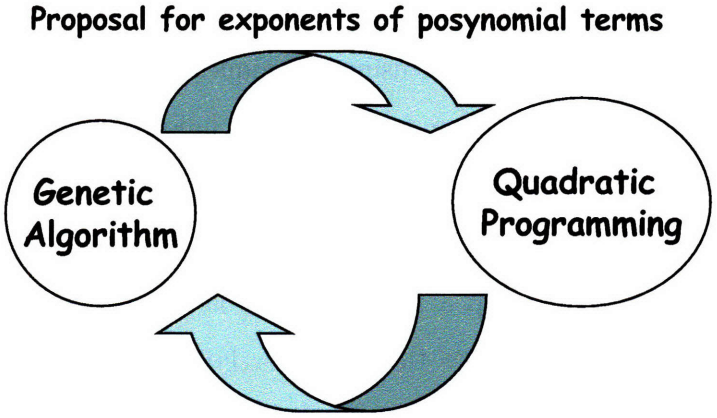


Figure 3-1: **Decomposition of posynomial fitting problem into two parts, one that searches for exponents and the other that searches for coefficients given exponents.**

One still needs to design the heuristic for the first part and piece these two parts into an algorithm. We decided to use a genetic algorithm which serves the purpose of searching for the exponents and simultaneously pieces the two parts together. We shall describe the algorithm in detail in later sections. Combining genetic algorithm

with another optimization algorithm has been generally termed as Hybrid Genetic Algorithms or Memetic Algorithms. [42, 33]

Instead of the genetic algorithm, simulated annealing, tabu search or other adaptive-search methods could also be used. This step remains replaceable and the operators developed in this thesis for genetic algorithms are transferable to other approaches. Our choice of algorithm was motivated by the better convergence properties of genetic algorithms compared to these techniques. Genetic programming has also been used for evolving terms of the kind required here using tree structures (known as symbolic regression in the Genetic Programming community) [25]. Genetic Programming is useful when the requirement is to find *any* function which fits the data well, such as functions containing logarithms, exponents of input variables, multiple fractions, etc. These cases cannot be handled by conventional techniques. In these cases too, it is not clear whether the tree structure of GP and corresponding tree operators actually do meaningful operations or are just random. We didn't see any rationale for using a tree-structure or usefulness of its operators for evolving terms for the posynomials. Posynomial terms have a well-constrained form and Genetic Programming would be an overkill for it. We instead designed modular operators (to be discussed later) for our genetic algorithm that are well suited to the posynomial modeling problem.

To summarize, our approach is to design a genetic algorithm which searches for exponents of posynomial terms and works in tandem with an efficient solver for coefficients given the sought after exponent values. Additionally, we design GA operators specially suited for the posynomial modeling problem. These operators are transferable to other heuristic approaches.

3.4 Genetic Algorithms

Genetic algorithms [32] are a class of algorithms inspired by natural evolution. They have been widely used in analog CAD for real-valued optimization [40, 9, 41]. However, they are much broader than this and have been used for solving traveling salesman problem, combinatorial optimization and structural synthesis of circuits [6].

Genetic algorithms have been considered *weak methods* and the argument has been "backed-up" by the No-Free-Lunch-Theorem (NFLT) [43]. The author doesn't agree to this. This is so because in the author's view a genetic algorithm is not a single algorithm, but a paradigm to design algorithms to solve different problems. A genetic algorithms can be abstracted to 4 steps shown in Figure 3-2 and described as follows: The algorithm builds an initial population of possible solutions, i.e. genotypes. It evaluates the performance of each and assigns it a corresponding value referred as fitness. It then selects the better solutions from the population, applies variation operators to them to create a new population for the next generation and iterates.

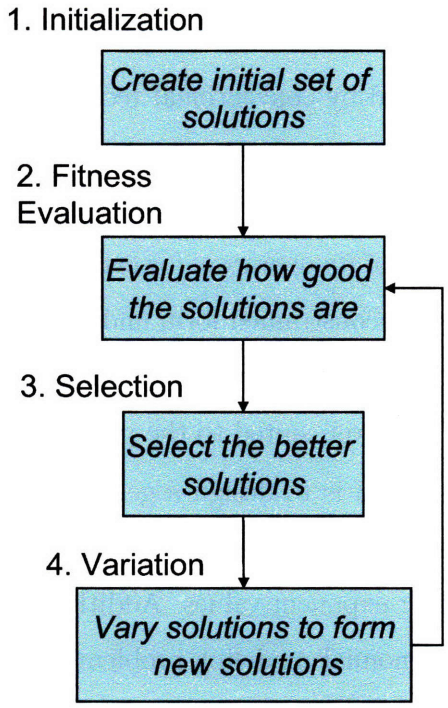


Figure 3-2: **The typical Genetic Algorithm flow**

This abstraction is not a complete algorithm in itself, since the variation operators are not concretely defined.[1] Algorithms which use some universal operators to go with this abstraction to solve any problem are indeed weak, for instance, mechanical

[1]The selection method is also not defined, but in the author's view the choice of the selection method doesn't strongly correlate with the problem being solved (in general).

use of one-point, two-point or uniform crossover, random mutation [20], operators of evolutionary strategy [5], etc. Instead, these algorithms can be tuned to the problem which is being solved by designing operators well suited to the structure of the problem (construed by some as providing an evolutionary path to the solution [1]). This independence makes genetic algorithms more of a paradigm to generate algorithms well-suited to different problems than an algorithm in itself. The resultant algorithms need not be weak and instead are well-suited to the problem at hand. This also liberates genetic algorithms from the argument of NFLT, since there isn't a claim of a universal algorithm which solve all problems efficiently.

In the next sections, we will discuss how the genetic algorithm paradigm was used to design an algorithm for evolving posynomials. The algorithm is depicted in Figure 3-3. In the following sections, we will describe in detail the representation of solution, the variation operators [2] and method of fitness evaluation. Fitness evaluation encapsulates the quadratic programming step previously discussed.

3.5 Posynomial Representation: Genotype to phenotype mapping

Genotype refers to the representation used for the solution in the genetic algorithm, while phenotype is the solution itself. Our phenotype is the posynomial expression. The genotype is a matrix of real numbered values as shown in Figure 3-4 (the figure depicts the situation when posynomial models of the MOS transistor parameters are being designed with the input variables, W, L and I). Figure 3-4 also shows the mapping of genotype to phenotype. Each row represents a term of the posynomial. The number of rows is fixed. A choice parameter associated with each row decides whether the row is actually used or not (1:used, 0:don't care). This allows us to have posynomials with varying number of terms in the population. The number of rows

[2]Variation operators and solution representation don't have an independent meaning as far as the algorithm dynamics are concerned. This decomposition and separate description is used for human-understandability and ease of programming.

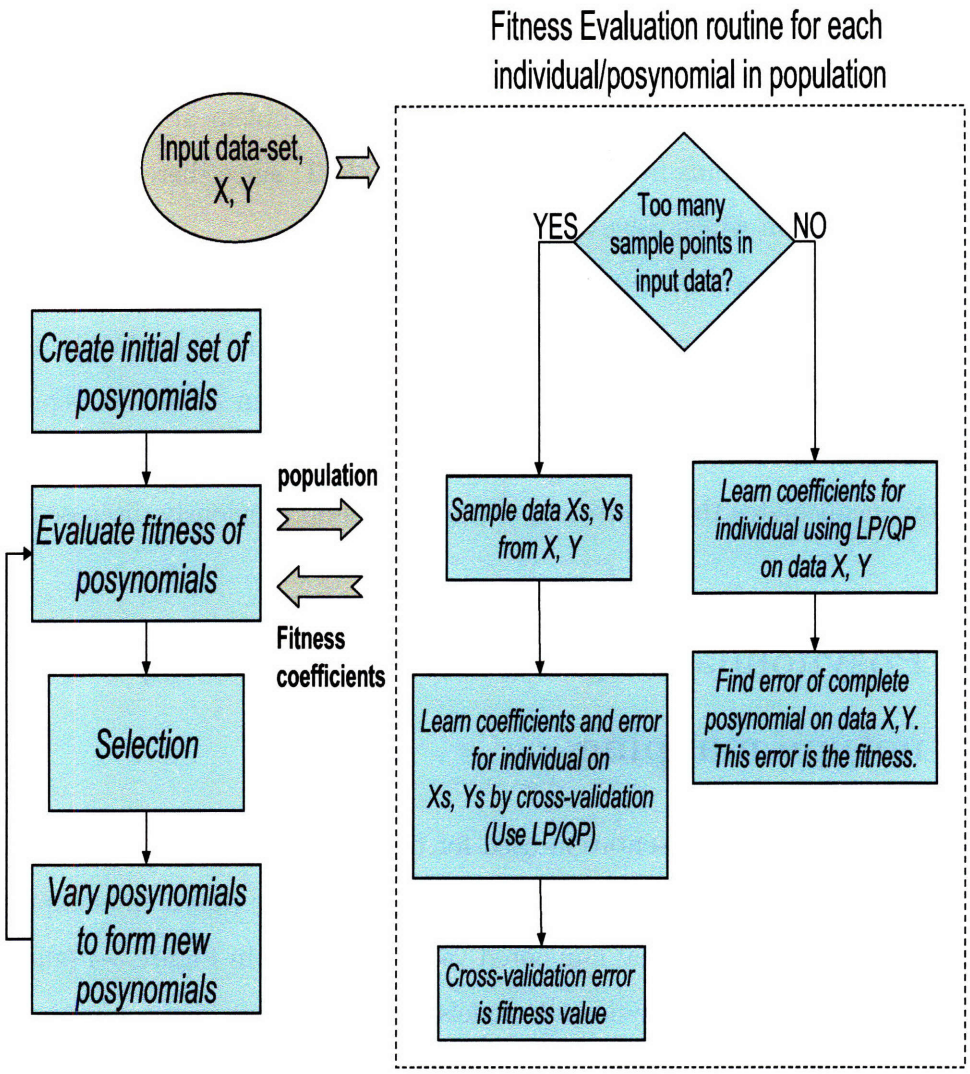


Figure 3-3: **The flow of the posynomial modeling algorithm**

is equivalent to the maximum number of possible terms in the posynomial. Each column is associated with one of the input variables. The value in a cell encodes the exponent of the variable (represented by the column) for the term (represented by the row). All cell values are in a specified range $[minVal, maxVal]$. The coefficient of each term is not a part of the genotype.

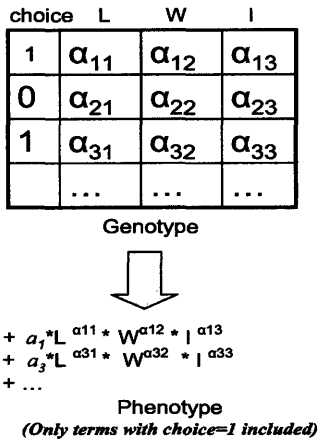


Figure 3-4: **GA genotype to phenotype mapping**

This genotype might be interpreted to state that the value of the choice parameter helps exploration of posynomials with different number of terms. This is not sufficient due to high possibility of *bloat*. Bloat has been discussed in context of genetic programming and variable-length genomes [27]. Bloat is the phenomena of evolution of larger and larger genomes as the evolutionary algorithm progresses. The primary reasons for this is a fitness bias observed for the higher-length genomes for many problems. The genome for our GA could be considered as a variable row genome with a bound on the maximum number of rows. It also has a fitness advantage if it has more rows with choice parameter 1 (equivalent to more number of terms), since it provides more degrees of freedom to fit the function. Thus, even if we use the choice parameter, in absence of a second regulatory process, one would observe that in very early generations itself, the evolution selects and propagate solutions with all choice parameters as 1.

Our regulatory process to balance the likeliness of the system to bloat emerges implicitly from the way we do fitness evaluation. Because of how we determine the coefficients (that are not in the genotype), a coefficient may become zero valued. This incorporates automatic *feature selection* (term selection) [21] in the algorithm. The determination of coefficient values and how the number of terms of the posynomial varies will be discussed in terms of fitness evaluation in Section 3.6.

In the current representation, we haven't included a bias to set any exponents to explicit zero. This could be included and can be potentially useful when the number of input variables are large. Also, the same representation can be modified to use only integer-valued exponents.

Note: We had previously claimed that representation and variation operators do not have independent meaning. This can be illustrated in the present context. We have described a matrix representation for the posynomial. But given the fact that it is stored as a matrix or a vector (all rows could be put side to side in a single row) doesn't imply any difference to the performance of the algorithm. The matrix is probably more intuitive for representing the posynomial, but it could be stored as a matrix or a vector without making much difference. The representation will become meaningful when we define the variation operators on it. Again, the same variation operators can be implemented with a vector representation. However, the variation operators will be easier to implement and make cognitively more sense with the representation shown. In summary we imply that representation doesn't mean anything whatsoever, it is representation and variation operators together that have a meaning [3] Thus, one should beware of being hoodwinked by stylish representations if the variation operations don't match the style!

[3]Since representation has to be defined for describing the variation operators, once can claim that representation is inherent in the variation operators and there is no need for a separate section on representation. We lend support to such a view!

3.6 Fitness Evaluation

Fitness evaluation implies providing a number to each solution in the population a measure of how good it is. For posynomial modeling, one of the error-metrics mentioned in Section 3.1 could be used. The lower the value of the error, the better is the individual. The better individuals are then propagated with a higher probability by the selection step. For the discussion ahead, we will assume the RMSE metric. The discussion remains valid for other error metrics described in Section 3.1 as well.

The GA evolves the exponents of all variables for each term of the posynomial. To determine the complete posynomial form of the candidate solution, the coefficient of each term must be determined. The optimal coefficients to minimize the mean-squared error is found by formulating a Quadratic program. The objective of the problem is to find the coefficients to minimize the mean square error. This is quadratic and convex in the coefficients. The constraint is that all coefficients have to be greater or equal to zero. These are linear constraints. The minimum value of the error (minimum MSE) is the fitness of the individual. All the error metrics mentioned in Section 1 can be solved globally and efficiently using a linear or a quadratic program. The corresponding solving technique for each error-metric is tabulated in Table 3.1.

As mentioned before, the fitness calculation yields some coefficients as absolute zero. This happens because of two reasons: a. The structure of the constraints, i.e. all coefficients have to be more than zero leads to optimal coefficients having many zeros; b. Since the problem is convex, we can efficiently solve it and find the optima. If it wasn't efficiently solvable, even though the optimal solution had some dimensions as zero, we may not have found them. [4] These effects is now discussed.

The quadratic programming problem can be visualized as an optimization problem with feasible space constrained by hyperplanes which bound the a single quadrant of the space. Each hyperplane has value of one of the coefficients as zero everywhere on it. The intersection of the hyperplanes have more than one coefficient equal to zero.

[4]This would have happened if we did not decompose the problem into two separate searches for exponents and coefficients as we have done now and instead used a black-box optimization algorithm to solve the whole problem.

The solution of the QP problem (and LP in case of other error metrics) in many cases lie on one of the constraining hyperplanes or their intersection. This yields some of the coefficients to exact zero. This implicitly performs *feature selection* on the evolved terms by setting the coefficients of *useless terms* to zero. This balances the tendency of bloat, since even though the choice parameter is 1, the coefficients of terms being zero eliminates them from the expression. The exploration of variable length genomes or posynomials with variable number of terms is primarily achieved through the way fitness is evaluated and not the choice parameters.

Managing large data sets: The use of QP (or LP) for each individual may become computationally very expensive if the data set is large. This was the case with transistor parameter modeling, where in some instances, we were modeling 70,000 points. To address this, we use a small uniformly sampled fraction of the data-set.[5] Using this smaller fraction requires that the evolved model does not overfit the sampled points. To ensure this, we use 2-fold cross validation on the sampled data set and use the cross validation error as the fitness of the individual [21]. This biases the search to propagate models which when trained on one set of data generalize well on a different set.

At each generation, we fit the coefficients of only the best-of-generation solution on the entire data-set and calculate its error. This allows us to choose the solution with the best error at end of the run. This error value doesn't play any role in determining the fitness of the individual.

Availability of data and generalization: It is worth noting here, that the problem of building models for MOS parameters and models for other applications of geometric programming has some differences from a typical Machine Learning (ML) model building problem. In ML problems, a limited amount of training samples are available and one has to deliver a model which best generalizes to the phenomena underlying the data. In our scenario, we have a blackbox for the underlying function already available and we are modeling it as a posynomial. We have the independence to use as large an input data set as we need (provided the computational cost to

[5]There are other ways to do this such as dynamic subset-selection [18], etc.

sample the blackbox is not high, as in case of MOS parameters). We use a fraction of the data and cross-validation, not because the data isn't available but due to the computational concerns of the algorithm. However we have much more independence than a ML problem scenario, since we can use the complete data set should the need arise, for instance, to implement things like dynamic subset selection or fitting the best-of-generation individual to the entire data set. Interestingly, our algorithm can be used for posynomial model-building also in the ML scenario, since it implements cross-validation, which takes care of generalization. A second validation set can be used to track when the algorithm starts overfitting [17].

3.7 Variation Operators

Variation is the step which derives new solutions from the current set of selected solutions, some of which will be fitter and hence drive the search ahead. The operators which modify the current set of solutions to form the new solutions are called variation operators. The purpose of the variation operator is to modify the current solution in a way which preserves locality in the fitness space. Given the locality, some individuals will have worse fitness and some (hopefully) better, driving the search ahead. The variation operator is thus a modifier in the genotype space, which provides locality in the fitness space.[6] The second requirement of the variation operator is the need to be global in the genotype space (as opposed to local in fitness space!), such that it samples the solution space well. These two orthogonal requirements are balanced by some magic numbers decided on the basis of available computational resources.

The two customary operators in genetic algorithms are crossover and mutation. In the most abstract sense, crossover is an operator which combines two individuals

[6]The Estimation-of-Distribution algorithms [35] suggest the need of a more restricted property in variation. They look for structure which is present in the selected or better solutions and propagate that. According to our perspective, they propagate structure possessed by individuals with similar fitness value (better in their case), thus they indeed preserve fitness locality. Since selection finds the better individuals, operators preserving fitness locality are enough. If there is a different structure in higher fitness and lower fitness individuals, then it can be beneficial to use the structure in fitter solutions and thus the EDA algorithms use it. For our purposes, we have found a general structure valid throughout the space.

in some way to generate a new individual. Mutation modifies a single individual in some way to create a new individual. Crossover and mutation are applied to an individual by a probability, p_{cross} and p_{mut} respectively. We now discuss how these operators were devised.

Crossover Operator: Given two individuals i.e. real-valued matrices, they can be combined in a number of different ways, only limited by the imagination of the designer. They can be sliced horizontally and the sliced parts can be exchanged; multiple vertically sliced parts can be exchanged; diagonally sliced parts can be exchanged, so on and so forth. Routine crossover operators such as one-point, two-point, uniform-crossover could be somehow be modified for a matrix and used.

How does one decide? The two ideas to bring together are firstly preservation of fitness locality and secondly exploitation of the problem structure to hypothesize a transformation in the genotype space to facilitate this. We know that the solution has a sum-of-product form. Each product term contributes (is correlated) to the output variable and additively compose the solution. An operator which exchanges some of these terms to create new solutions shall preserve fitness locality to a good extent due to the additive nature of the terms. The effect is amplified due to automatic tuning of term coefficients by the fitness evaluation step. The second feature of this operator is that it is fairly global in nature in the genotype space, which is also a requirement. The operator makes much more sense than slicing and exchanging parts of the matrix in a different way or just randomly, which would indeed be global in the genotype space, but not local in the fitness space.

Concretely, the crossover operator takes two individuals (parents) and creates two new individuals by choosing each row of the new individuals from one of the two parents with a given probability. The operator is depicted in Figure 3-5. The fraction of rows which are exchanged between the parents is called the mixing ratio. The operator can be thought of as row-wise uniform-crossover operator [20]. The crossover operator thus designed is a *coarse operator* in the sense that is coarse in its search in the genotype space, while reasonably local in the fitness space.

Mutation Operator:: The mutation operator modifies some cells of the geno-

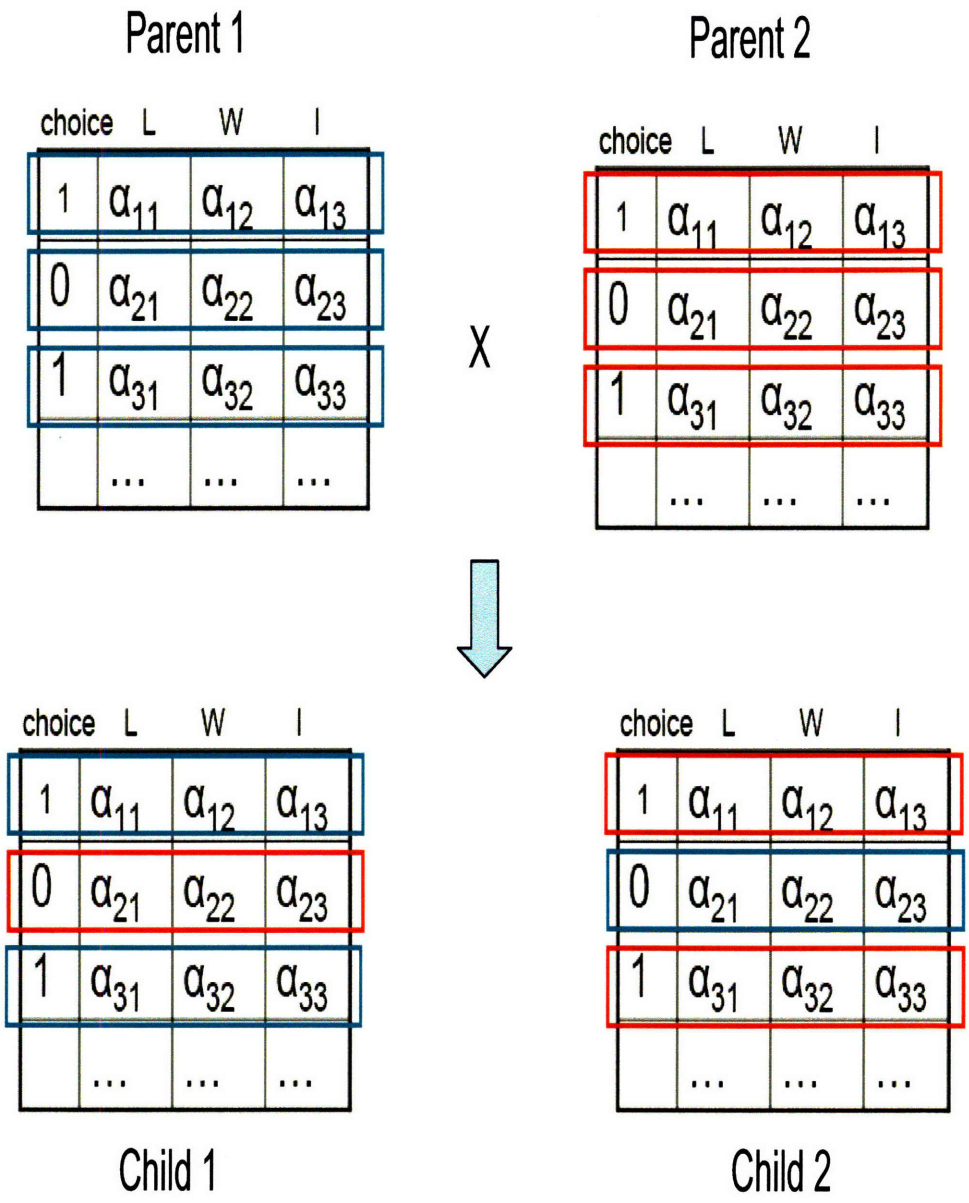


Figure 3-5: **A depiction of the crossover operator**

type matrix. We ask the question: Which cells of the matrix should be modified to preserve fitness locality?

To answer this question, we use two pieces of information. First, the idea that each row additively contributes to the solution and preservation of a row leads to preservation of locality in fitness (as used in crossover). Secondly, a row with a zero coefficient (as learnt by LP or QP) does not add towards the solution and thus its modification doesn't adversely effect fitness locality.[7] It should be however noted that even though a row may have a zero coefficient in one individual, it can learn a non-zero value in a different individual. This may happen because the coefficient of a row (monomial term) is not independent of the other rows (monomial terms) in the individual. Thus, it may not be a plausible idea to always modify a row which has a zero coefficient in an individual, since it can be useful to another individual when inherited due to crossover.

We use these two pieces of information in devising a row-wise mutation operator in the following way. A row with a zero coefficient is mutated by a probability (p_{zero}), while a term with a non-zero coefficient is mutated by a different probability (p_{non_zero}). By doing a row-wise mutation, the mutation operator preserves some rows as it is, leading to fitness locality. Secondly, p_{zero} is higher than p_{non_zero}. This allows for higher exploratory power (global search) in genotype space without sacrificing locality in fitness space. The operator is depicted in Figure 3-6.

To further support the global search element, whenever a row with zero coefficient is chosen for mutation, all cells (exponents) are randomly re-initialized. In case a row with non-zero element is chosen for mutation, all cell values are not modified. A cell in the given row is chosen for mutation by a probability, p_{cell}. The average number of cells chosen for variation in a row is $p_{cell} * (numberInputVariables)$. The value of p_{cell} can be used to control locality in genotype space. Once a cell is chosen for mutation, it is randomly reinitialized by a probability of p_{cell_reinit}. By a probability of $p_{cell\_perturb}$ (equivalent to $1 - p_{cell\_reinit}$), a normal distribution centered at zero with

[7] In [2], we devise an operator which doesn't use this second piece of information and show that it performs worse than the operator reported in this thesis.

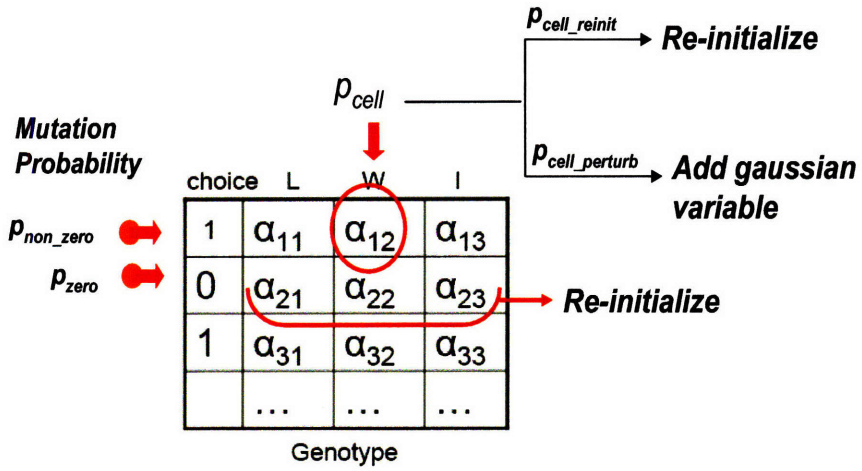


Figure 3-6: **A depiction of the mutation operator**

a given variance (λ) is added to the value in the cell. The variance of the gaussian can be used to control genotype locality (correlated to fitness locality). We adaptively decrease the variance (λ) with generations to let the algorithm be global in genotypic search in initial generations and local in it in latter generations.

This formulation of the mutation operator gives it nice properties of global search in genotype space, while preserving fitness locality. The different probability values provide knobs to trade-off one for the other. Our philosophy of a genetic algorithm design is to allow for more global search in genotype space in earlier generations, while bias towards fitness locality (leading to genotypic locality in some cases) in the later generations.

This concludes the discussion of our algorithm for posynomial modeling. We propose a hybrid of a genetic algorithm and convex optimization technique (linear programming or quadratic programming) to generate posynomials with real-valued exponents and variable number of terms. We use properties of posynomials in devising specialized variation operators for our algorithm. In the next chapter, we compare our technique with other state-of-art posynomial modeling techniques.

Chapter 4

Results and Discussion

To test the developed algorithms, we did two sets of experiments. The first set of experiments tests the efficiency of the algorithm to derive posynomial models for data actually generated from posynomial functions. We hand-write posynomials with different characteristic features and sample data from these models. We then learn posynomial models for this data set using various algorithms and compare. We show that our algorithm does fairly well in discovering the underlying posynomial structure and outperforms other algorithms. In the second set of tests, we use the algorithm to derive models for MOS transistor data.

4.1 Algorithms

We built models using two error-metrics: Root Mean Square Error and Root Mean Relative Square Error (Refer Chapter 3, Table 3.1). The first metric is the conventional metric used in machine learning problems. The second error metric will be motivated in discussion of results of experiments with RMSE.

For comparing the posynomials generated by our algorithm, we used two previous algorithms for generating models. In the first method (Posy-2), a monomial is generated using log-log regression [7]. We fix the exponents of the model, however re-learn the coefficient of the monomial and an extra constant by a QP formulation (constrained linear regression). The QP is formulated for RMSE or RMRSE, as re-

quired. The second method [30] (MaxMon) generates a max-monomial that fits the data. A max-monomial is piecewise linear in log-log space. Max-monomials can also be used in convex optimization [7], however may have issues with sharp transitions. We set the parameters of the algorithm as recommended in [30]. We use an initial partition size range from 1 to 20 and give 10 trials for each of these setting. This is as recommended in [30]. We report the model with the least error (RMSE or RMRSE) as the resultant model from the algorithm.

The parameters for our genetic algorithm (GA-Posy) are given in Table 4.1. The genetic algorithm was run 4 times for each parameter and the posynomial with least error was reported. Each genotype of generation 0 is initialized using a uniform random distribution bounded by $[-3, 8]$ for each cell element. The number of rows in the genome is 6. The choice parameter is randomly initialized to 1 or 0 such that the average number of terms per individual in the initial generation is 4. We use a generation based GA with tournament selection [32] . The population size is 100, number of generations is 1000 and tournament size is 6. The genetic algorithm parameters are given in Table 4.1.

Parameter	Value
Initial λ	5
λ rate	Halved every 75 generations
$p_{crossover}$	0.3
Mixing Ratio	0.7
$p_{mutation}$	1
p_{zero_term}	0.7
$p_{non\_zero\_term}$	0.3
p_{cell}	0.5
p_{cell_reinit}	0.65
$p_{cell\_perturb}$	0.35

Table 4.1: GA-Posy Parameters

4.2 Design of artificial posynomials

We needed to generate posynomials with different characteristics to test whether our algorithm can fit each of these families of posynomials. To design these posynomials, a few insights into their structure are useful. These are as follows:

1. Posynomials are always convex in log-log space (equivalent to log-log transformation, Refer Chapter 2). They can be non-monotonic, but with only one sign change from positive to negative. In real space, they can be convex or concave; non-monotonic with one sign change.

2. Monomials are linear (linear functions are both concave and convex) in log-log space. They are always monotonic, globally increasing or decreasing, in both real and log-log space.

Name	Property	Expression
MON	Linear in log-log space	$5x^{1.6}$
POSY-NL	Convex, Non-linear in log-log space	$10^{-18}(10^{6}(10^{8}x^{0.82}+5x^{7.23})+x^{9.95})$
POSY-NM	Convex, Non-monotonic in log-log space	$10^{-4}*(x^{2.1}+10000*(x^{-1.4}))$
FUNC-CAVE	Concave monotonic in log-log space	$1-e^{\frac{-x}{10}}$

Table 4.2: Expressions and characteristics of designed posynomials in one variable

Given these structural properties, we define four categories of functions in one-variable. The expressions are tabulated in Table 4.2 and their graphs are shown in Figure 4-1 and Figure 4-2. They are described as follows:

1. MON: A one-term posynomial which is linear in log-log space.

2. POSY-NL: Posynomial which is non-linear in log-log space. A monomial cannot fit this accurately. The modeling approach should be capable of evolving a two-term posynomial to be able to fit this. We illustrate this family by a three term posynomial in Table 4.2.

3. POSY-NM: A posynomial which is non-monotonic in log-log space (non-monotonicity in real space is implied). Again, non-monotonic characteristic cannot be expressed by a monomial. A non-monotonic posynomial requires at least two terms with at least one variable exponentiated to a positive exponent in one term and a negative in the other.

4. FUNC-CAVE: This is a function which is concave in log-log space, however it is monotonic. This cannot be fit by a posynomial. We use this function to investigate, how well a learned posynomial can fit this function. The particular choice of concave-monotonic function is inspired by characteristics of some MOS parameters. It is argued in [4], that all MOS parameters have monotonic characteristics. It was also shown that some of them, for instance *gm* and *gds* are concave with drain current [38].

The first three functions represent the different distinguishing features of posynomials, where the fourth function cannot be represented by a posynomial. Though useful for understanding, these functions are over-simplistic since they are single-dimensional. All of these were effortlessly found by the algorithm. To test the algorithms, we designed more complicated posynomials by combining the single-dimensional posynomials. These set of posynomials are two dimensional and combine the alluded characteristics in different ways. These are tabulated in Table 4.3 and shown in Figure 4-3

The first posynomial here has a single term (monomial), the next two have 4 terms each while the last one is not a posynomial. The chosen range for the two input variables is $[1, 100]$. It can be observed in the Table 4.3 that the range of each posynomial sweeps several orders of magnitude. The algorithms were tested on these posynomials. Each input variable was sampled in the range $[1, 100]$ with a step size of 3. This results in total 1156 points.

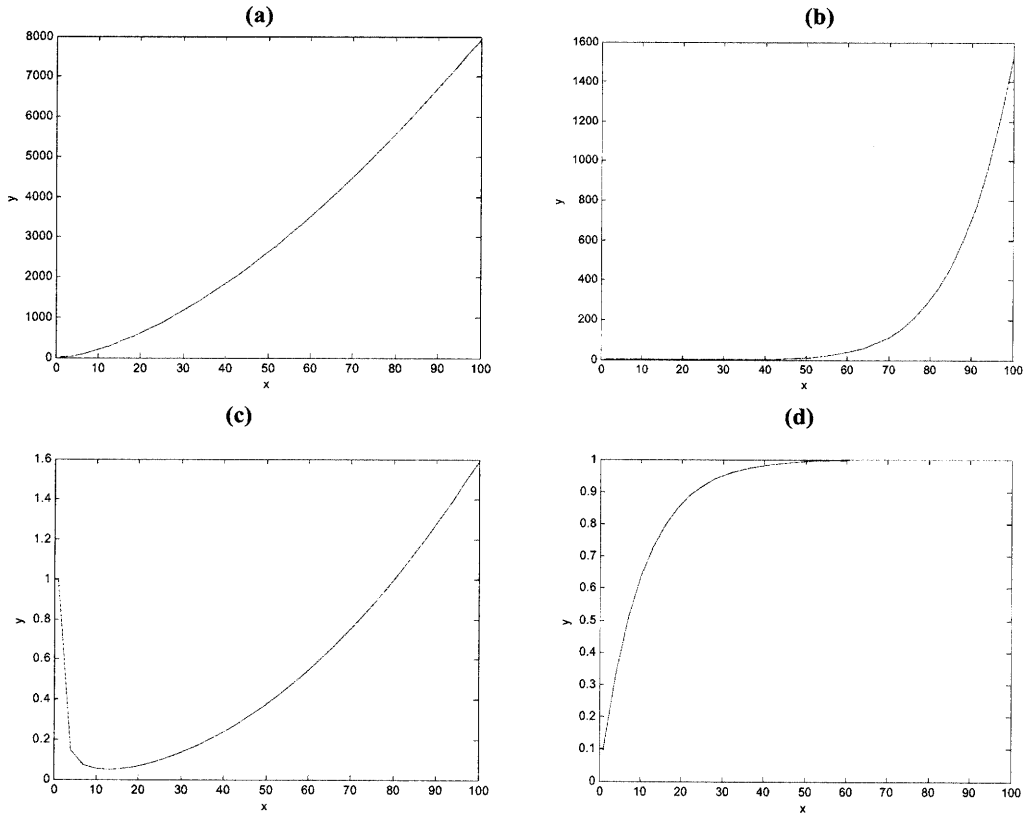


Figure 4-1: **Posynomials in one variable, plots in real space, a. MON, b. POSY-NL, c. POSY-NM, d. FUNC-CAVE**

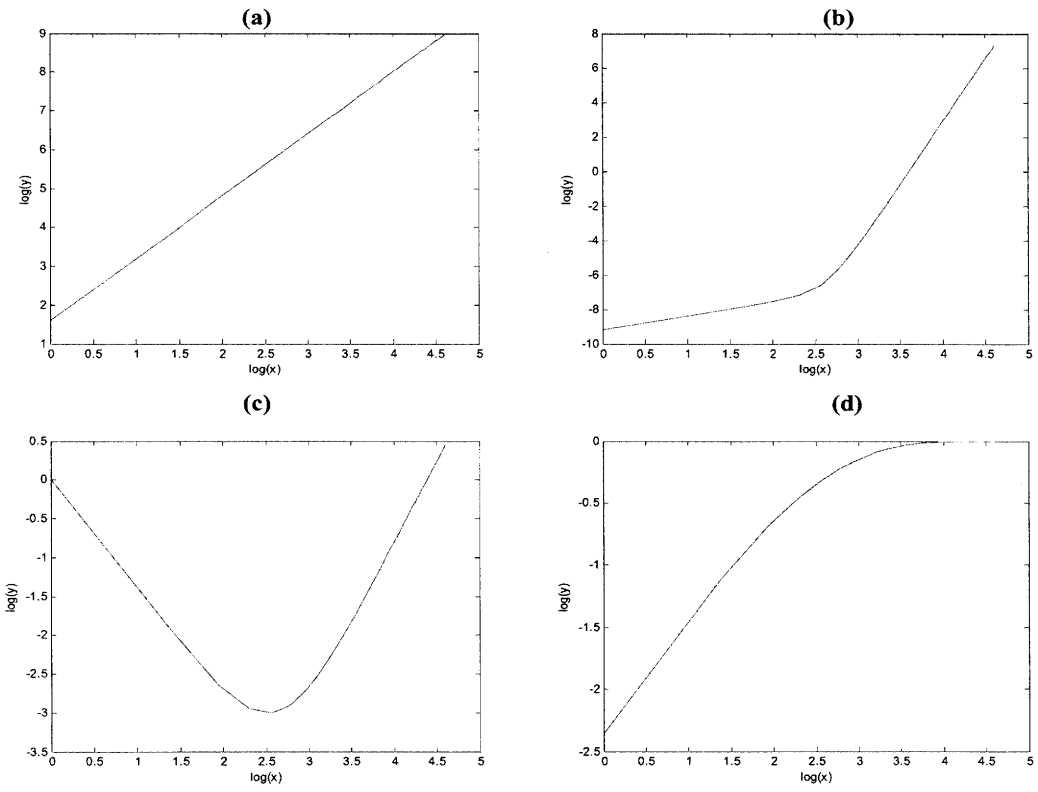


Figure 4-2: **Posynomials in one variable, plots in log-log space, a. MON, b. POSY-NL, c. POSY-NM, d. FUNC-CAVE**

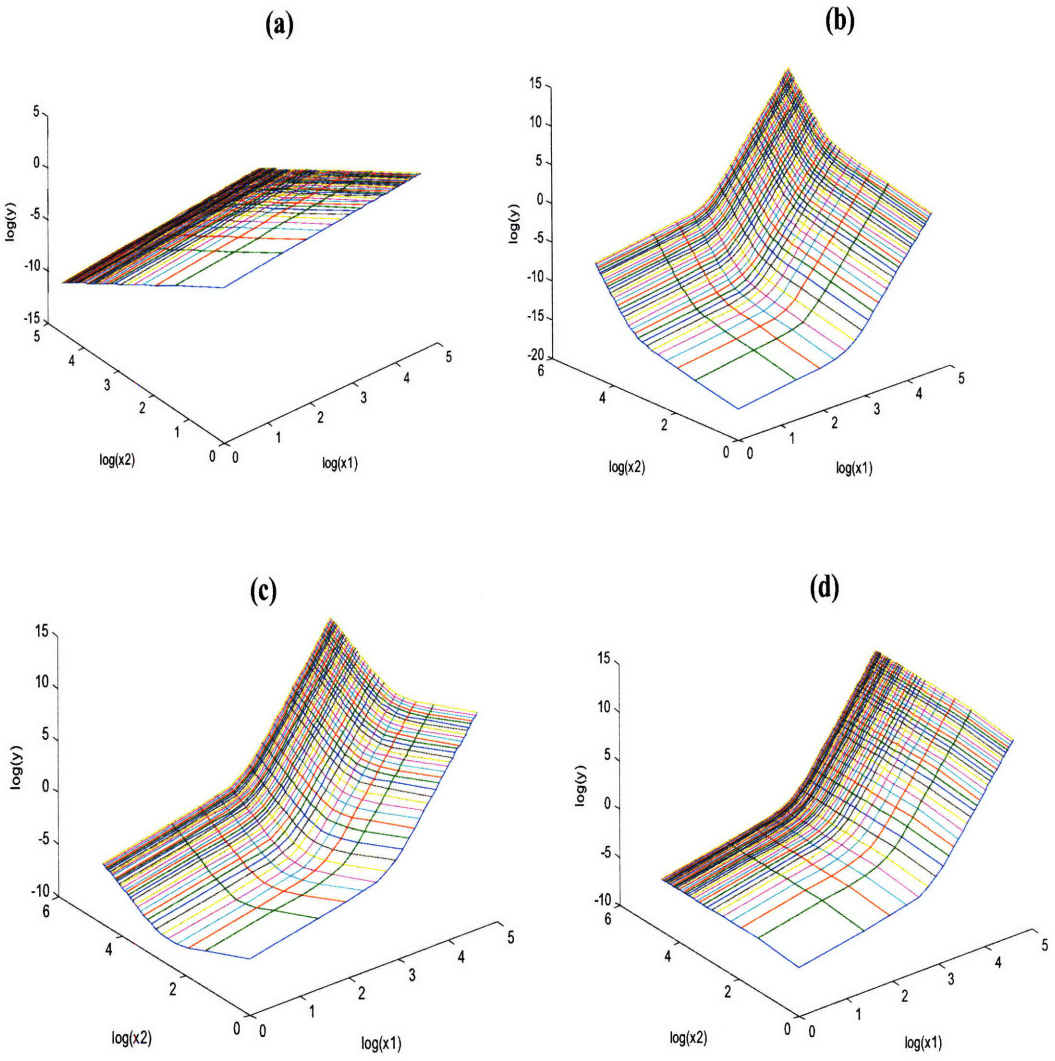


Figure 4-3: **Posynomials in two variables, plots in log-log space. (a) MON, (b) NL-NL, (c) NL-NM, (d) NL-CAVE**

Name	Property	Expression	Range
MON	Linear in log-log space	$y = x_1^{0.45} x_2^{-2.2}$	$[3.98 * 10^{-5}, 7.94]$
NL-NL	Convex, Non-linear in both dimensions in log-log space	$10^{-7} * (x_1^{0.82} + 5 * 10^{-8} x_1^{7.23})(x_2 + 5 * 10^{-8} x_2^6)$	$[10^{-7}, 7224]$
NL-NM	Convex, Non-linear in x1 and non-monotonic in x2 in log-log space	$10^{-6}(x_1 + 5 * 10^{-8} x_1^7)(x_2^{2.1} + 10000 x_2^{-1.4})$	$[4.94 * 10^{-4}, 79325]$
NL-CAVE	Non-linear in x1 and concave in x2 in log-log space	$y = 10^{-2}(x_1 + 5 * 10^{-8} x_1^7)(1 - e^{-x_2})$	$[6.3e^{-3}$ 5001]

Table 4.3: Characteristics, expression and range for artificial posynomials in two variables

4.3 Artificial Posynomials: Results and Discussion

4.3.1 Root Mean Square Error

The Root Mean Square Error of all generated functions is tabulated in Table 4.4. The evolved posynomial expressions for these functions are in Appendix A.1. It is clear that the GA outperforms the other two algorithms with respect to the MSE of the generated expressions for all functions except the monomial. The error of the GA is several magnitudes higher than the other two approaches for the monomial (MON). However the error of the expression evolved by GA is three orders of magnitude lower the minimum value of MON ($3.98 * 10^{-5}$). For all practical purposes, the expressions generated by the three algorithms are equivalent. The first two algorithms have very low error value because they do a deterministic fit in the log-log space. Given that the generated data is a monomial, this results in finding the exact exponents and coefficient. In fact, the error of their expressions is zero and the observed error is due to numerical errors. The current situation will arise only when the data is generated from an exact monomial. Since the GA does a numerical search for exponents and coefficients, it cannot find the exact coefficients and exponents and thus have a finite

error. One can use smaller mutation step size for getting an even lower error, but as noted before, for all practical purposes the error is 0.

Function	Posy-2	MaxMon	GA-Posy	Imp %
MON	$2.13e^{-15}$	$2.34e^{-16}$	$3.12e^{-8}$	-
NL-NL	$2.31e^{3}$	1.05	$5.50e^{-3}$	99.5
NL-NM	$5.00e^{3}$	21.11	$6.37e^{-2}$	99.7
NL-CAVE	$3.23e^{3}$	705	691	1.98

Table 4.4: Comparison of RMSE for different models. Posy-2: Two-term posynomial, MaxMon: Max-monomial, GA-Posy: Evolved Posynomial, Imp %: Percentage improvement of GA-Posy with respect to the better of Posy-2 and MaxMon

We can get more insight into the results by visualizing how the fits look. We show the POSY-2 plots in both real and log-log space for NL-NL and NL-NM functions in Figure 4-4. In all figures, green represents the actual posynomial function and blue represents the fitted posynomial function. The algorithm fits hyperplanes in the log-log space. It cannot express the non-linearity or the non-monotonicity of the functions. This skews the results in the real-space as well, which is perceptible in Figure 4-4.

With regard to MaxMon and GA-Posy, the plots for monomial fit are not interesting, since they are exact fits. For NL-NL and NL-NM, there is no perceptible difference between the function and fit in the real-space. The fits for NL-NL and NL-NM functions in log-log space are shown for both MaxMon and GA-Posy in Figure 4-5.

It is interesting to note that though the GA posynomial has a lower RMSE, they look worse in the log-log space as compared to MaxMon. Why does this happen? Before we explain this, note that neither of the two algorithms can find the exact posynomial. For MaxMon, this is so because the functions (NL-NL and NL-NM) are not actually piecewise monomials, but they are posynomials. On the other hand, in the case of GA, though the evolved expression are posynomials, since the exponents are real and the GA does a numerical search, it cannot find the exact exponents.

The good fits of MaxMon expressions in log-log space is obvious. This is so because MaxMon actually transforms all data in to log-log space and then fits it.

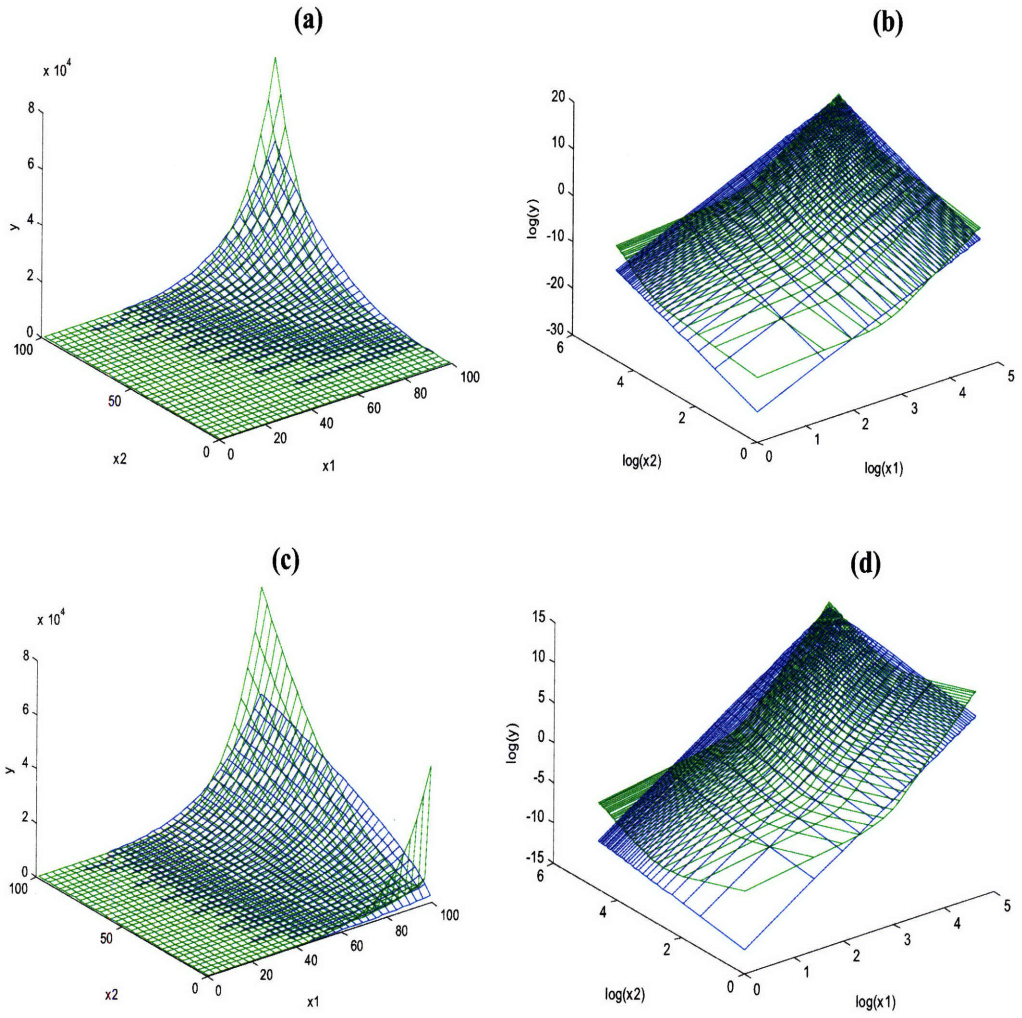


Figure 4-4: **POSY-2 fits for functions, (a) NL-NL, real space, (b) NL-NL, log-log space, (c) NL-NM, real space, (d) NL-NM, log-log space**

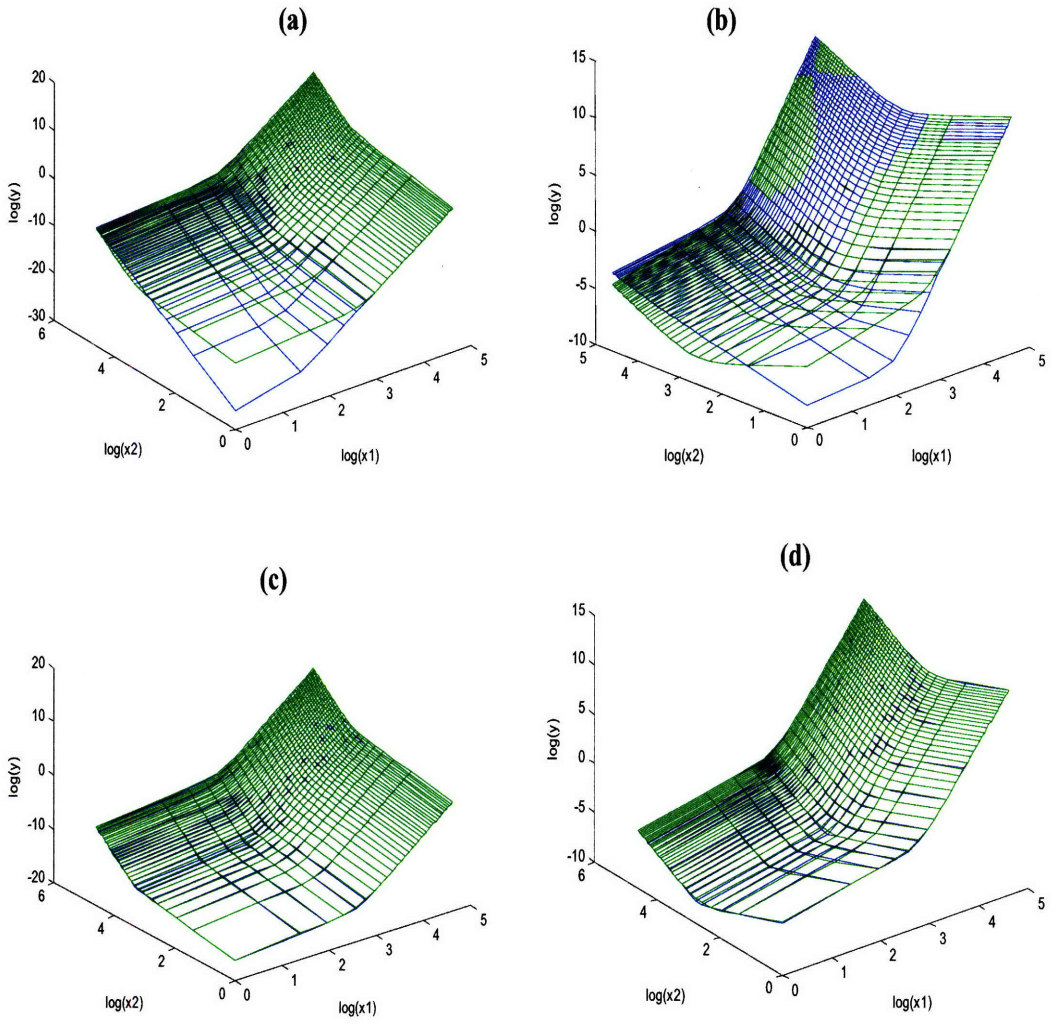


Figure 4-5: **Posynomial fits in log-log space, a. NL-NL (GA-Posy), b. NL-NM (GA-Posy), c. NL-NL (MaxMon), d. NL-NM (MaxMon)**

This gives it good fit in log-log space at the expense of the real-space error. Also, note that MaxMon cannot find the exact fit in log-log space because it approximates a posynomial with a max-monomial.

In case of the GA, observe that the error in the log-log space is higher when the values of the output is lower and lower when the values of the output is higher. It is known that error in the log-log space between two points roughly corresponds to the relative error in the real-space [7]. Therefore, the error we perceive in the log-log space is actually a depiction of the relative error in the real space. Why is the relative error higher for low-valued points? The answer to this becomes clear if we express RMSE in terms of relative error at each point:

$$RMSE = (\sum_i (f(\overline{x_i}) - y_i)^2/N)^{0.5}$$
$$=> RMSE = (\sum_i y_i^2 (\frac{f(\overline{x_i}) - y_i}{y_i})^2/N)^{0.5}$$
$$=> RMSE = (\sum_i y_i^2 * RE_i^2/N)^{0.5}$$

Here, RE_i is the relative error at the i^{th} point, the perceived error in the log-log plot. Thus, RMSE can be seen as root of weighted mean of squared relative error at each point, where the weight is the square of the value at the given point. Clearly the weight for relative error (RE) of points with high value is higher than those with low value. Thus, the RMSE metric sacrifices the relative error of lower valued points for better relative error of higher valued points (since their weights are higher). This difference becomes perceptible when we plot the fits in log-log space.

It is thus interesting that different mechanisms of the MaxMon and GA-Posy lead to different error patterns. Both algorithms are incapable of finding the exact posynomial. MaxMon gets a good fit in the log-log domain at expense of the error values in real space. It does a good job in balancing relative errors of all points. On the other hand, GA-Posy optimizes in the real space and sacrifices the relative error of low valued outputs to get a better relative error for high-valued outputs providing an overall lower RMSE than MaxMon. Thus given that the requirement of the modeler is a lower RMSE, GA-Posy outperforms MaxMon.

NL-CAVE is interesting from the point of view of concavity with respect to x_2.

The plots for the expressions of MaxMon and GA-Posy both in real and log-log space are shown in Figure 4-6. It can be seen that the fits are perceptibly very different from the functions in real space. In the log-log space, MaxMon fits a convex function (with respect to x_2), while GA-Posy fits a straight line. It is proved in [38], that the best fit for a concave function by a posynomial will be a hyperplane in log-log space.

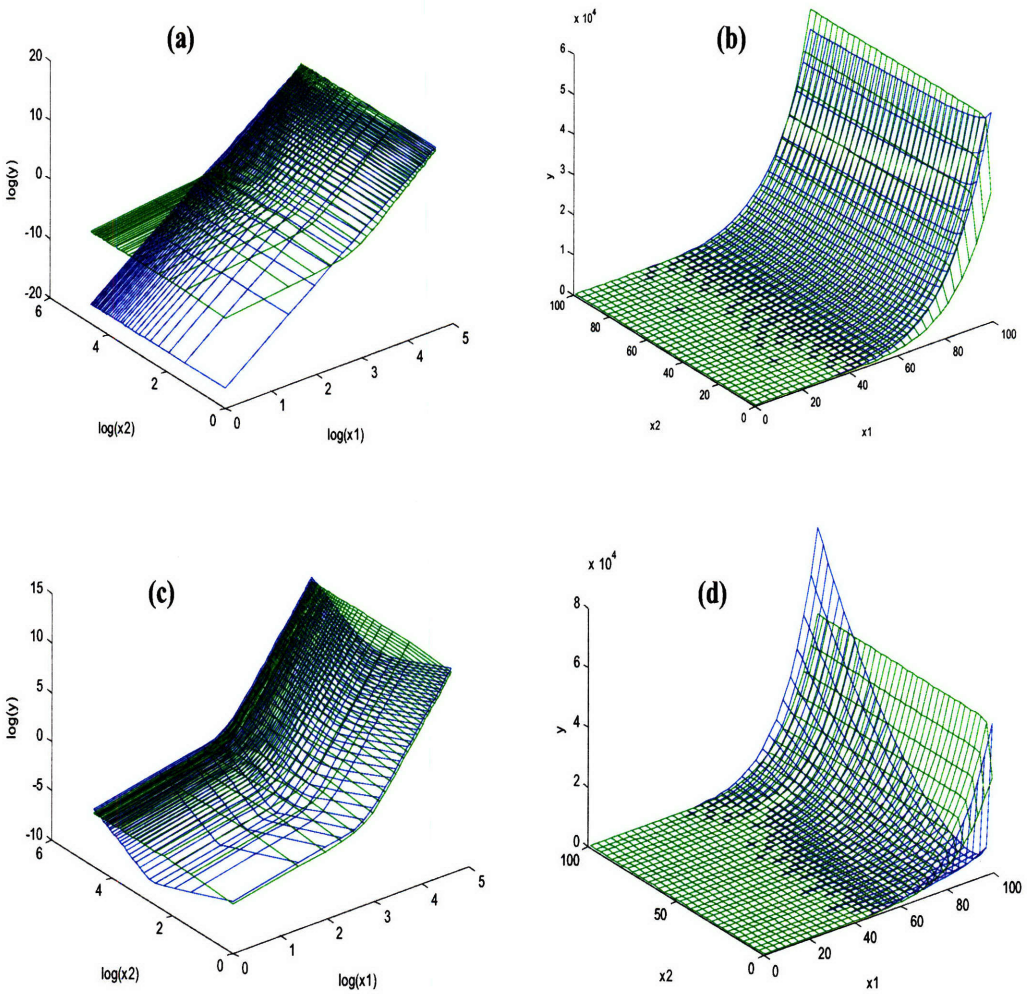


Figure 4-6: **Posynomial fits for NL-CAVE, a. GA-Posy log-log, b. GA-Posy real, c. MaxMon log-log, d. MaxMon real**

The results can be thus be summarized as follows:

1. The GA outperforms or equals the other methods on all functions for RMSE.

2. The monomial generating algorithm has the least expressive power and performs worst.

3. Neither MaxMon nor GA-Posy can find the exact posynomial. The GA sacrifices relative errror of lower valued outputs to get a lower RMSE. MaxMon does a good log-log fit at cost of the real domain fit.

4. The concavity is approximated by a line and a convex function by GA-Posy and MaxMon respectively.

Given this discussion, one realizes that the modeler might be interested in minimizing RRMSE. Given that the output ranges several orders of magnitude, minimizing the sum of percent (or relative) error makes sense, more so, from the point of view of optimization. It will also be interesting to observe how the two algorithms compare with this error metric.

4.3.2 Root Mean Relative Square Error

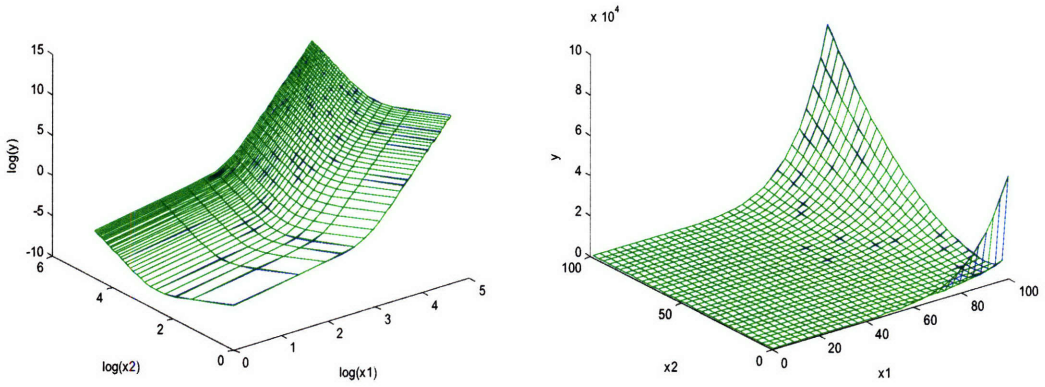
The comparison of the MaxMon and GA-Posy expressions with respect to RMRSE is given in Table 4.5. The evolved expressions are recorded in Appendix A.2. The GA outperforms MaxMon for all functions (for MON, they are equivalent, as discussed before). For NL-NL, the MaxMon error is 2.27%, while the GA Posy has an error of 0.091% (more than an order decrease in error). For NL-NM, the GA posy has half the error as MaxMon. For NL-CAVE, the GA does slightly better. It can be observed that the percentage improvement in error by GA-Posy is lesser in RMRSE as compared to that in RMSE.

The plots for these fits are not very interesting, since there isn't a perceptible difference between the function and the fits for the first three expressions. We show the fit of GA-Posy for NL-NM both in log-log space and real space (Figure 4-7). The GA now does a good job of fitting in the log-log space. At higher values of function, one can perceive some difference in value of function and fit in real-space,

Function	MaxMon	GA Posy	Imp %
MON	$7.93e^{-14}$	$1.17e^{-4}$	-
NL-NL	2.27	0.091	95.9
NL-NM	2.19	1.1	49.7
NL-CAVE	6.64	6.47	2.6

Table 4.5: Comparison of RMRSE (%) for different models. MaxMon: Max-monomial, GA-Posy: Evolved Posynomial, Imp %: % Improvement of GA-Posy error with respect to MaxMon error.

which confirms our hypothesis. Thus even with RMRSE, the GA does a better job in providing a fit for the function than MaxMon. Now, perceived fit in log-log space is also better. This confirms the superiority of our approach for both measures.



(a) **(b)**

Figure 4-7: **GA fit for NL-NM function, a. log-log Space, b. Real Space**

Note: To confirm that the posynomials evolved by minimizing RMRSE have a worse RMSE than those evolved for minimizing RMSE, we have reported RMSE for MaxMon and GA Posy in Table 4.6. It can be observed that RMSE for all expressions are worse than those reported in the last Section.

Function	MaxMon	GA Posy
MON	$1.43e^{-12}$	$4.14e^{-6}$
NL-NL	6.88	5.63
NL-NM	60.11	78.84
NL-CAVE	716.19	757

Table 4.6: Comparison of RMSE for models generated by optimizing RMRSE. Max-Mon: Max-monomial (MaxMon), GA Posy: Evolved Posynomial (GA-Posy)

4.4 MOS Modeling

We design models for 10 MOS parameters: GM, GDS, CDB, CGSR, VT, VDSATk RO, CGD, VGS, INVGM (GM^{-1}).[1] The design variables were W, L, I_d and Vds. We optimized the models for RMRSE. The data generation and results for these parameters is now discussed.[2]

4.4.1 Data Generation

We created posynomial models for parameters of MOS technology TSMC $0.18u$ and voltage $1.8V$. The value of the 10 MOS parameters for an NMOS transistor were simulated in SPICE and logged. The ranges and step size for design variables were same as in [38]. Only, in-saturation points were used. A total set of approximately 900 points was used for modeling.

4.4.2 Results and Discussion

The results for various MOS parameters are tabulated in Table 4.7 and also shown in Figure 4-8. MaxMon outperforms GA-Posy for Cdb, Vt, Cgd and $InvGm$. The difference in error for Cdb, Vt, $Cgsr$ and Cgd is very low and the two algorithms can be considered equivalent. For $InvGM$, MaxMon does considerable better than GA-Posy for $InvGM$. For the rest 5 parameters, GA-Posy outperforms MaxMon. In

[1]These parameters are described in Chapter 2.

[2]We also conducted a separate experiment where models were built as a function of W, L and I_d. We show in [3], that our algorithm outperforms Max-MON for all but one parameter with respect to RMSE.

summary MaxMon and GA-Posy gives equivalent results for 4 parameters, GA-Posy outperforms MaxMon for 5 parameter, while MaxMon is better for one parameter.

Parameter	MaxMon	GA-Posy	Alg	Imp %
gm	39.77	32.18	GA-Posy	19.08
gds	105.21	52.72	GA-Posy	50.5
Cdb	1.93	1.97	Equal	−2.03
$Cgsr$	4.79	4.70	Equal	1.87
Vt	0.64	0.65	Equal	−1.50
$Vdsat$	43.90	34.60	GA-Posy	21.18
Ro	52.56	38.36	GA-Posy	27.01
Cgd	3.14	3.16	Equal	−0.63
Vgs	22.78	20.67	GA-Posy	9.26
$InvGM$	2.57	4.15	MaxMon	−61.1

Table 4.7: RMRSE (%) for MOS parameters: MaxMon: Max-monomial, GA Posy: Evolved Posynomial, Alg: The algorithm which performs better. Incase, both perform similarly, we use the label 'Equal'. Imp %: % Improvement of one algorithm with respect to the other. Negative sign indicates MaxMon is better than GA-Posy

It should be noted that GA-Posy provides substantial improvement for gm (19.08%), gds (50.5%), $Vdsat$ (21%) and Ro (26.8%). The improvement in Vgs is also not trivial. Though, MaxMon does considerably better than GA Posy for $InvGM$, it should be noted that both methods have a considerably small error for $InvGM$. Also to note is that the error values are very high for gm, gds and Ro. The reason for this is explained in [38], where it is observed that gm is concave in I_d, while gds and Ro have both concave and convex characteristics in different parts of the space. This explains why $InvGm$ provides a good fit, since it becomes convex with I_d.

These results show that our approach is useful in deriving accurate posynomial models for MOSFET parameters. The improvement is non-trivial and it performs worse than Max-Mon only for one parameter, $InvGm$. In the geometric programming flow, one can use a max-monomial for $InvGM$ instead of our posynomial. Also, the $InvGm$ max-monomial can be used as a starting point in our algorithm to derive a better posynomial.

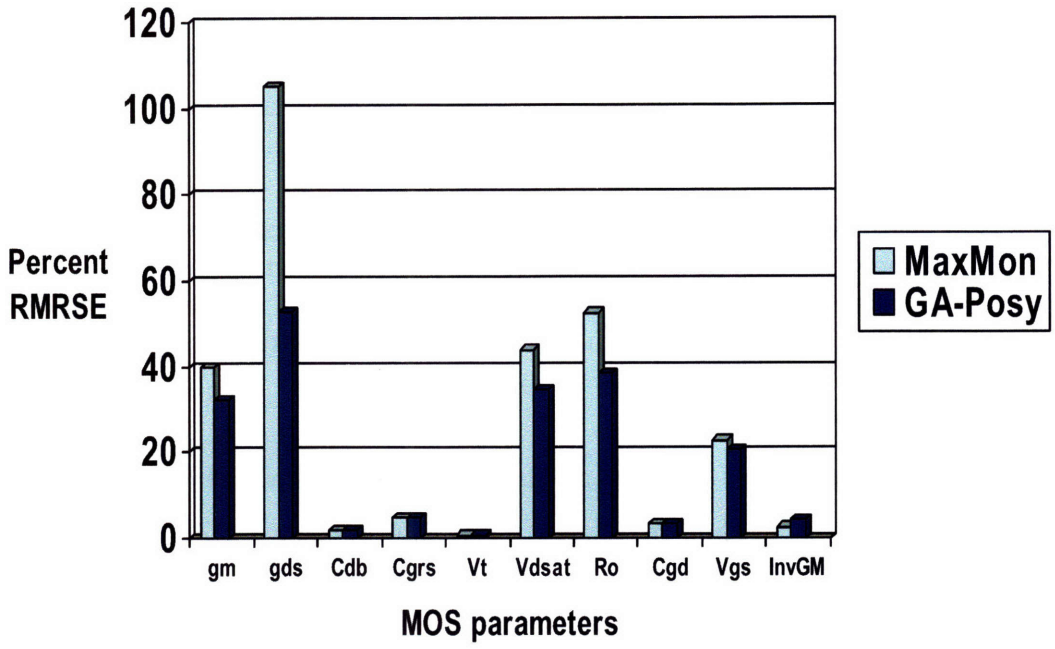


Figure 4-8: **Comparison of RMRSE (%) of MaxMon and GA-Posy for MOS parameters**

4.5 Conclusion

We have shown that our algorithm can fit posynomials with good accuracy and outperform the state-of-art approaches for monomial and max-monomial fitting. Our algorithm provides better results than MaxMon for both RMSE and RMRSE. We also show that our methods provides substantial improvement in deriving more accurate posynomial models for MOS parameters.

Chapter 5

Future Work

In this thesis, we developed a new algorithm for generating posynomial models with variable number of terms and real-valued exponents. The algorithm is a hybrid of a genetic algorithm and a convex optimization technique. This algorithm outperforms the state-of-art algorithms on benchmark problems. We also showed that the accuracy of posynomial models of MOS parameters is improved by a considerable amount by using this method. This improvement in MOS model accuracy leads to improvement in accuracy of the geometric programming flow.

We also argued that the accuracy of geometric programming can be improved without adversely influencing the run time or increasing the designer's effort. This is facilitated by decomposition of geometric programming modeling into two steps, one being technology dependent, while the other being topology dependent (Refer Chapter 2). This decomposition leads to reuse of the models generated for these two steps, which decouples accuracy of models and run-time of geometric programming.

With regard to the MOS modeling problem, this work can be extended in the following ways:

1. A qualitative study of the MOS parameter data can be done to understand why the error in modeling is very high for parameters like *gm* and *gds*. (Taken up in [38])

2. The technique can be extended to other error-metrics such as Maximum Rel-

ative Absolute Error (Refer Section 3.1). The lowest MRAE of the models provide an upper-bound on the error in the optimized solution of the geometric programming flow. (Taken up in [38])

3. The transistor models generated by the new algorithm should be instantiated in the geometric programming flow of optimization to measure the percentage improvement in optimization accuracy due to these new models.

4. The proposed technique can be used for building models for MOS transistors which account for statistical variation using modeling parameters such as t_{ox} [44].

With regard to the identified decomposition possible in circuit equation expressions, the following could be pursued in future:

1. One could pursue improving the accuracy of the second step of the decomposition, i.e., the expressions for circuit specifications in terms of MOS parameters. Till now, these expressions have been written by hand or symbolic analysis techniques [16] have been used to automatically generate them. Both these techniques are limited due to bad scaling properties and accuracies. Instead, statistical model building techniques can be used to address this step. The posynomial generating algorithm proposed in this thesis can be used to generate models for circuit specifications in terms of MOS parameters.

2. The 2 step decomposition is valid only for small-signal specifications. Theoretical work to extend this to transient specifications can be taken up. The literature of digital CAD could be helpful in this regard, given their requirement for measuring delays and other transient characteristics for large systems.

3. The 2 step decomposition's usability is not limited to geometric programming. It can be used in 'Equation-based blackbox optimization' Approaches. This shall liberate the models for either steps to be posynomials.

The proposed posynomial modeling algorithm could be used for other applications of geometric programming as well.

Appendix A

Evolved Posynomials

A.1 Models for RMSE

The posynomial models generated by running the genetic algorithm with RMSE error metric are shown here. The following information is provided: RUN and Generation from which the model was extracted; the RMSE for the model. The genotype for the model is presented. The first column is the choice parameter, the second column is the coefficients of monomial terms and the last 2 columns represent the exponents of x_1 and x_2 respectively. The last row is the constant term.

```
MON
------------------
Best Run Number: 3 Best Generation: 769
Best mean RMSE: 3.122565e-08
1.000000 1.212778e-01 4.495413e-01 -2.196054e+00
1.000000 8.778498e-01 4.500603e-01 -2.200553e+00
1.000000 0.000000e+00 0.000000e+00 0.000000e+00
1.000000 0.000000e+00 4.843562e+00 1.775476e-01
1.000000 0.000000e+00 0.000000e+00 0.000000e+00
1.000000 8.725624e-04 4.529428e-01 -2.192972e+00
0.000000e+00
```

NL-NL

Best Run Number: 2 Best Generation: 965

Best mean RMSE: 5.506506e-03

0.000000 0.000000e+00 -1.492885e+00 -1.341616e+00

1.000000 4.802766e-12 7.282495e-01 4.533393e+00

1.000000 4.900602e-15 7.234932e+00 9.991426e-01

1.000000 2.496761e-22 7.229562e+00 6.000364e+00

1.000000 1.951359e-21 6.537381e+00 3.868463e+00

1.000000 2.964506e-25 7.499710e+00 5.798311e+00

0.000000e+00

NL-NM

Best Run Number: 2 Best Generation: 975

Best mean RMSE: 6.378945e-02

1.000000 4.993011e-10 6.999807e+00 -1.399499e+00

1.000000 1.463183e-12 6.947625e+00 -1.679339e+00

1.000000 4.997644e-14 7.000052e+00 2.100047e+00

1.000000 2.988460e-04 3.421412e-01 1.011540e+00

1.000000 4.667265e-08 9.113333e-01 2.756229e+00

1.000000 1.616901e-10 3.803649e+00 1.229604e+00

0.000000e+00

NL-CAVE

Best Run Number: 4 Best Generation: 579

Best mean RMSE: 6.911850e+02

1.000000 0.000000e+00 0.000000e+00 3.210539e+00

1.000000 7.903235e-07 3.478263e+00 4.003187e-02

0.000000 0.000000e+00 4.795973e+00 3.981057e-02

0.000000 0.000000e+00 0.000000e+00 -2.450131e+00

```
1.000000 4.277035e-10 7.000675e+00 3.904043e-02
0.000000 0.000000e+00 0.000000e+00 7.761098e+00
0.000000e+00
```

A.2 Models for RRMSE

The posynomial models generated by running the genetic algorithm with RRMSE error metric are shown here. The information and format is same as last section.

```
MON
-------------------
Best Run Number: 3 Best Generation: 997
Best mean RRMSE: 1.177889e-06
1.000000 9.986456e-01 4.500392e-01 -2.200024e+00
1.000000 -8.680759e-23 -2.897331e+00 6.065083e+00
1.000000 3.409851e-23 6.352068e+00 0.000000e+00
1.000000 -7.677031e-19 -3.586161e-01 4.344559e+00
1.000000 0.000000e+00 -1.519497e+00 5.190007e-01
1.000000 1.352850e-03 4.202694e-01 -2.181548e+00
0.000000e+00
NL-NL
-------------------
Best Run Number: 4 Best Generation: 982
Best mean RRMSE: 9.142265e-04
1.000000 2.488048e-22 7.231608e+00 5.999647e+00
1.000000 4.918507e-15 8.207012e-01 6.003693e+00
1.000000 -8.040154e-12 2.661716e+00 -1.557506e+00
1.000000 5.066994e-15 7.227278e+00 9.988577e-01
1.000000 1.002136e-07 8.200427e-01 9.979625e-01
```

1.000000 3.479528e-11 4.575147e-01 2.019477e+00

-2.591414e-10

NL-NM

Best Run Number: 3 Best Generation: 977

Best mean RRMSE: 1.098479e-02

1.000000 2.391279e-14 7.572042e+00 8.478811e-01

1.000000 9.182111e-07 1.004891e+00 2.117475e+00

1.000000 5.916723e-10 6.964097e+00 -1.432680e+00

1.000000 2.976275e-03 1.027264e+00 -1.056519e+00

1.000000 3.990888e-14 6.988372e+00 2.158454e+00

1.000000 8.010427e-03 9.038068e-01 -1.699354e+00

0.000000e+00

NL-CAVE

Best Run Number: 1 Best Generation: 946

Best mean RRMSE: 6.216804e-02

1.000000 7.161940e-03 1.036713e+00 6.163797e-02

1.000000 0.000000e+00 -6.947550e-01 -2.009450e+00

0.000000 0.000000e+00 4.031503e+00 4.308821e+00

1.000000 3.924691e-10 7.000140e+00 6.129915e-02

1.000000 0.000000e+00 1.430573e+00 7.757254e-01

1.000000 -9.726498e-04 -2.573898e+00 -1.011068e+00

9.840537e-04

Bibliography

[1] Russ Abbott, Behzad Parviz, and Chengyu Sun. Genetic programming reconsidered. In Hamid R. Arabnia and Youngsong Mun, editors, *IC-AI*, pages 1113–1116. CSREA Press, 2004.

[2] V. Aggarwal and U.M. O'Reilly. Design of posynomial models for mosfets: Symbolic regression using genetic algorithms. *Genetic Programming: Theory and Practice IV*, pages 219–236, 2006.

[3] Varun Aggarwal and Una-May O'Reilly. Simulation-based reusable posynomial models for mos transistor parameters. In *Design Automation and Test in Europe, 2007.*

[4] Varun Aggarwal and Una-May O'Reilly. Monotonicity information in simulation-based approaches for efficient circuit sizing and hierarchical synthesis. In *In submission*, 2007.

[5] Thomas Baeck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1(1):1–23, 1993.

[6] Hans-Georg Beyer and Una-May O'Reilly, editors. *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005.* ACM, 2005.

[7] S. Boyd, S.-J. Kim, L. Vaudenberghe, and A. Hassibi. A tutorial on geometric programming. In *Technical report, EE Department, Stanford University*, 2004.

[8] Stephen P. Boyd and Seung Jean Kim. Geometric programming for circuit optimization. In *ISPD '05: Proceedings of the 2005 international symposium on Physical design*, pages 44–46, New York, NY, USA, 2005. ACM Press.

[9] Min Chu, David J. Allstot, Jeffrey M. Huard, and Kim Y. Wong. NSGA-based parasitic-aware optimization of a 5GHz low-noise VCO. *ASP-DAC*, 00:169–173, 2004.

[10] D. M. Colleran, C. Portmann, A. Hassibi, C. Crusius, S. S. Mohan, S. Boyd, T. H. Lee, and M. Hershenson. Optimization of phase-locked loop circuits via geometric programming. In *IEEE CICC*, pages 377–380, 2003.

[11] W. Daems and G. Gielen. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE Trans. CAD of Integrated Circuits and Systems,*, 22(5):517–534, 2003.

[12] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In *PPSN*, pages 849–858, 2000.

[13] Maria del Mar Hershenson. Efficient description of the design space of analog circuits. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 970–973, New York, NY, USA, 2003. ACM Press.

[14] Tom Eeckelaert, Walter Daems, Georges Gielen, and Willy Sansen. Generalized posynomial performance modeling. In *DATE '03*, page 10250, 2003.

[15] Tom Eeckelaert, Trent McConaghy, and Georges Gielen. Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1070–1075, Washington, DC, USA, 2005. IEEE Computer Society.

[16] J. L. Huertas F. V. Fernandez, A. R. Vazquez and G . G. E. Gielen. *Symbolic Analysis Techniques: Applications to Analog Design Automation.* IEEE Press, 1997.

[17] Christian Gagné, Marc Schoenauer, Marc Parizeau, and Marco Tomassini. Genetic programming, validation sets, and parsimony pressure. In Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Budapest, Hungary, 10 - 12 April 2006. Springer.

[18] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature III*, volume 866, pages 312–321, Jerusalem, 9-14 1994. Springer-Verlag.

[19] Georges Gielen, Trent McConaghy, and Tom Eeckelaert. Performance space modeling for hierarchical synthesis of analog integrated circuits. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 881–886, New York, NY, USA, 2005. ACM Press.

[20] David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms.* Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[21] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[22] James R. Hellums, L. Richard Carley, Michael J. Krasnicki, Rob A. Rutenbar, and Rodney Phelps. A case study of synthesis for industrial-scale analog IP: Redesign of the equalizer/filter frontend for an ADSL CODEC. *dac*, 00:1–6, 2000.

[23] M. Hershenson, S. Boyd, and T. Lee. GPCAD: A tool for CMOS op-amp synthesis. In *ICCAD, pages 296–303*, 1998.

[24] D. Johns and K. Martin. *Analog Integrated Circuit Design.* John Wiley and Sons, 1997.

[25] Maarten Keijzer, Martin Baptist, Vladan Babovic, and Javier Rodriguez Uthurburu. Determining equations for vegetation induced resistance using genetic programming. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1999–2006, New York, NY, USA, 2005. ACM Press.

[26] W. David Kelton. Design of experiments: experimental design for simulation. In *WSC '00: Proceedings of the 32nd conference on Winter simulation*, pages 32–38, San Diego, CA, USA, 2000. Society for Computer Simulation International.

[27] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.

[28] Franky Leyn, Georges G. E. Gielen, and Willy M. C. Sansen. An efficient DC root solving algorithm with guaranteed convergence for analog integrated CMOS circuits. In *ICCAD*, pages 304–307, 1998.

[29] Xin Li, P. Gopalakrishnan, Yang Xu, and T. Pileggi. Robust analog/RF circuit design with projection-based posynomial modeling. In *Proc. ICCAD '04*, pages 855–862, 2004.

[30] A. Magnani and S. Boyd. Convex piecewise-linear fitting. In *submission to J. Optimization and Engineering*, 2006.

[31] Pradip Mandal and V. Visvanathan. CMOS op-amp sizing using a geometric programming formulation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(1):22–38, 2001.

[32] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

[33] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[34] Emil S. Ochotta, Rob A. Rutenbar, and L. Richard Carley. Astrx/oblx: tools for rapid synthesis of high-performance analog circuits. In *DAC '94: Proceedings of the 31st annual conference on Design automation*, pages 24–30, New York, NY, USA, 1994. ACM Press.

[35] Martin Pelikan, Kumara Sastry, and Erick Cantu Paz. *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications.* Springer-Verlag, Berlin Heidelberg, 2006.

[36] Rodney Phelps, Michael Krasnicki, Rob A. Rutenbar, L. Richard Carley, and James R. Hellums. Anaconda: simulation-based synthesis of analog circuits viastochastic pattern search. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(6):703–717, 2000.

[37] T. Quarles, A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli. SPICE 3, version sf5 user's manual. 1994.

[38] Lynne Salameh. Posynomial modeling for mosfets. M.eng. thesis, Massachusetts Institute of Technology, May 2007.

[39] Amith Singhee, Claire F. Fang, James D. Ma, and Rob A. Rutenbar. Probabilistic interval-valued computation: toward a practical surrogate for statistics inside cad tools. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 167–172, New York, NY, USA, 2006. ACM Press.

[40] B. De Smedt and G.C.E. Gielen. WATSON: Design space boundary exploration and model generation for analog and RF IC design. *IEEE Trans. on CAD of Int. Circuits and Systems*, 22(2):213–224, 2003.

[41] Saurabh K. Tiwary, Pragati K. Tiwary, and Rob A. Rutenbar. Generation of yield-aware pareto surfaces for hierarchical circuit design space exploration. In *DAC*, pages 31–36, 2006.

[42] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[43] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.

[44] Yang Xu, Kan-Lin Hsiung, Xin Li, Ivan Nausieda, Stephen Boyd, and Lawrence Pileggi. Opera: optimization with ellipsoidal uncertainty for robust analog ic design. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 632–637, New York, NY, USA, 2005. ACM Press.

[45] Eckart Zitzler, Marco Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In *EUROGEN 2001, Evolutionary Methods for Design, Optimization and Control with Application to Industrial Problems*, pages 95–100, Athens, Greece, 2002.