Contents

1
Automated Extraction of Expert Domain Knowledge from Synthesis Results 1
Trent McConaghy, Pieter Palmers, Georges Gielen, and Michiel Steyaert

Chapter 1

AUTOMATED EXTRACTION OF EXPERT DOMAIN KNOWLEDGE FROM SYNTHESIS RESULTS

Trent McConaghy[1], Pieter Palmers[1], Georges Gielen[1], and Michiel Steyaert [1]
[1]*Katholieke Universiteit Leuven, Leuven, Belgium*

Abstract

Recent work in genetic programming shows how expert domain knowledge can be *input* to a genetic programming (GP) synthesis system, to speed it up by orders of magnitude and give trustworthy results. On the flip side, this paper shows how expert domain knowledge can be *output* from the results of a synthesis run, in forms that are immediately recognizable and transferable for problem domain experts. Specifically, using the application of analog circuit design, this paper presents a methodology to automatically generate a decision tree for navigating from performance specifications to topology choice; a means to extract the relative importances of topology and parameters on performance; and to generate whitebox models that capture tradeoffs among performances. The extraction uses a combination of data-mining and genetic programming technologies. This paper also presents techniques to ensure that the GP-based synthesis system can indeed create a richly-populated, high-performance dataset, including: a parallel-computing, multi-objective age-layered population structure (ALPS) for fast and reliable convergence; average ranking on Pareto fronts (ARF) to handle many objectives; and generating good initial topology sizings via multi-gate constraint satisfaction. Results are shown on operational amplifier synthesis across thousands of topologies that generated a database containing thousands of Pareto-optimal designs across five objectives and dozens of constraints.

Keywords: synthesis, domain knowledge, multi-objective, data mining, analog, integrated circuits, age layered population structure

1. Introduction

Engineers in many fields, from circuit design to automotive design, use their experience and intuition to choose design structures (topologies) and to design new structures to meet design goals. Unfortunately, the structure used may not be optimal, leading to suboptimal final design performances, cost, and robustness. The suboptimal design may be because the design is using new or unfamiliar materials, the designer is too time-constrained to be thorough (via time-to-market pressures), or simply because the designer just doesn't have the experience level to know what might be best. This last point is understandable for many disciplines; e.g. it is well recognized that gaining depth in analog circuit design is a process that takes years to get started and decades to master (Williams, 1991). All said, it still means that a suboptimal design structure may be used.

Hence, it is desirable to provide support for the designer in selection and design of such structures, and ideally to catalyze the designer's learning process. Prior research has focused on automated structure selection and design, but has had little emphasis on giving insight back to the user. This is a problem for two reasons: (1) By deferring control to automated tools, a designer's learning might slow, not to mention being less-equipped to cope when problems arise that the tools cannot handle (2) highly experienced designers can be extremely reluctant to use any automated synthesis tools, because past tools have fallen far short of expectations while the designers have had much success doing their work manually. One can reasonably ask how automation of structural design could possibly be used, if is so potentially troublesome and unwanted?

Knowledge extraction offers a way out: *use automation as a means to accelerate designer insight*. Recent work has shown knowledge extraction for the design-parameters-to-performance mapping (McConaghy et al., 2005), yield modeling (McConaghy and Gielen, 2006), and for design of behavioral / dynamical models (McConaghy and Gielen, 2005a). This paper shows how knowledge extraction can give insights on the relation between structure, parameters, and peformance, which can enhance the engineering field's body of knowledge and catalyze future manual and automation-aided topology design. Figure 1-1 shows the flow, which includes a "multi-objective multi-topology sizing" to synthesize fully-trustworthy structures followed by automated knowledge extraction steps. The output of knowledge extraction tools can be provided to the designers in formats that they can immediately use, as later sections show. This flow can be useful even for the automation-averse designers: the designers who embrace automation can perform the synthesis and extraction, and provide or publish the outputs for their automation-averse colleagues.

Figure 1-1. Target flow. A multi-objective multi-topology sizer generates a database, which is then data-mined to discover relations among design structure (topology), design parameters, and performances.

Novel contributions of this paper are:

- For augmenting insight of an expert designer, a data-mining perspective on a topology-performance database, leading to (a) a decision tree for navigating from performance specifications to structure, (b) extraction of relative importance among topology and other variables for each performance, via stochastic gradient boosting, and (c) automatic generation of whitebox models to capture performance tradeoffs.
- To generate a high-quality database, an improved multi-topology multi-objective sizing algorithm is designed, which includes: (a) Optimization under many objectives via average ranking on Pareto front, and (b) high-quality starting designs via multi-gate constraint satisfaction.

This paper is organized as follows. Section 2 describes the improved multi-topology multi-objective sizing algorithm. Section 3 describes the experimental setup to generate the sized-topologies database. Section 4 employs data-mining techniques to extract topology-performance insights. Section 5 concludes.

2. Enhanced Multi-Topology Multi-Objective Sizing

This section describes the algorithm to generate richly-populated database of high-performance designs that approximate a performance tradeoff. We start with MOJITO (McConaghy et al., 2007a; McConaghy et al., 2007b), but extend it to overcome difficulties encountered.

Prior Work: Structural Synthesis and MOJITO

There is much interest in using GP for structural synthesis, and many researchers have used analog circuit design as the “fruitfly” for experiments, e.g. (Koza et al., 2003). Past approaches had two key limitations: (1) high computational effort to get human-competitive results, and (2) because the search is so open-ended, designs are not trustworthy enough to fabricate and properly test, which can cost millions of dollars for analog integrated circuits (McConaghy and Gielen, 2005b). MOJITO overcomes both issues simultaneously by traversing a space of fully-trusted building blocks that is still large enough to be interesting, yet maintaining the possibility for novelty in a controlled fashion (McConaghy et al., 2007b).

Because this paper builds on MOJITO, we now outline its key elements:

- It defines the space of possible topologies as a hierarchically composed set of domain-specific building blocks. For analog circuit design, this means starting at transistors and building up through current mirrors and differential pairs, all the way to two-stage amplifiers. Approximately 30 building blocks combine to allow approximately 3000 different amplifier topologies.
- It uses hierarchy-aware search operators in order to naturally mix analog sub-blocks; i.e. grammar-constrained GP operators (Koza, 1992; Whigham, 1995)
- The search algorithm is GP. It used an age-layered population structure, ALPS (Hornby, 2006) to avoid premature convergence, and NSGA-II (Deb et al., 2002) for handling multiple objectives and constraints.
- It is multi-objective. Combined with the multi-topology search, it means that the results can return a variety of topologies, each for a different region of performance space. This final feature opens up the opportunity for deeper analysis of the relation between objectives (performances), topologies, and design parameters.

Challenges and Solutions

MOJITO had issues that prevented it from getting a richly-populated high-performance database with a variety of topologies. We discuss each, along with the solution.

Challenge: High Number of Objectives. Many design problems, including analog circuit design, can have five or eight or more objectives, depending on the functionality (Razavi, 2000). Unfortunately, most optimization approaches do poorly when there are more than two or three objectives (Corne and Knowles, 2007). To improve NSGA-II, we need to understand why it does poorly. When doing selection, NSGA-II sorts the population into nondomination layers: the

1^{st} layer is the nondominated set, the 2^{nd} layer is what would be nondominated if the 1^{st} layer was removed, etc. The selected parents are the ones taken from the 1^{st} layer, 2^{nd} layer, etc. until all parents have been chosen. Because there might not be a clean cut between layers, only some individuals from the last active nondomination layer can be taken. NSGA-II chooses individuals with the highest distance from other individuals in performance space ("crowding"). The problem is that with many objectives, there will be so many nondominated individuals that only a fraction of them can be chosen, and that fraction biases towards the corners of the performance space which are the farthest apart; and not the center points which are close to all. To solve this, we use Adaptive Ranking on Pareto Front (ARF) (Corne and Knowles, 2007). We modify NSGA-II to use it by: if doing selection at the nondominated layer, use the average rank measure AR for selection, instead of crowding distance. AR is defined by:

$$AR(x) = \sum_{k=0}^{num\_objectives} rank(k, x, X) \tag{1.1}$$

where x is an individual, and rank(k, x, X) is the rank of individual x compared to the whole nondominated set X, for objective k. At a given objective, the best individual has a rank value of 1, second-best has rank 2, etc.

Challenge: Uneven Sampling of Topologies. During subsequent runs, we saw that the algorithm was generating a simple class of structures (single-stage amplifiers) just as often as a more complex class (two-stage amplifiers), despite the fact that there are many more possible topology variants of the more complex class. This is because the random sampling views the space "flat", randomly picking a value for each of the topology choice parameters, with equal bias. To fix this, we instead give equal bias to each possible *topology* (Iba, 1996), via the algorithm in Table 1-1.

Table 1-1. Procedure RandomCircuit()

1.	Starting at the bottom building blocks, and proceeding to the top, calculate the possible permutations of topology choice variable values.
2.	To randomly generate a topology, starting at top and proceeding to bottom: draw a set of choice variables, using permutation (count) information to give equal bias to each topology.
3.	Draw value for each design variable from a uniform distribution.

Challenge: Maintain Diversity of Topologies. With further runs, we found that most randomly generated higher-complexity topologies (e.g. folded topologies, 2 stage amplifiers) would die out within a few generations of being generated. While ALPS generated more topologies in later random injection phases, those would die out too. Upon investigation, we found that the randomly-generated complex topologies' performances were much worse than simple ones, and that they did not improve as quickly. This is because the more complex topologies have more design variables to get right to reach a minimal performance bar. We also found that the first feasible topology found would overtake other topologies, further hurting diversity. This is because of NSGA-II's constraint-handling: it lumps all constraints into one overall violation measure, and always prefers feasible individuals over infeasible individuals. It effectively does single-objective search until the feasible individual is found (killing some topology diversity then), and then emphasizes the first feasible individual excessively (because no other topology get there quite as fast). An idea pointed the way: do not make topologies compete strongly against each other until they are at least nearly feasible. It is ok to have them competing once past feasible, because each topology will occupy its own niche in performance space and will therefore be maintained. From this guideline, we designed a series of constraint-satisfaction "gates", where the first earlier gates are cheaper to can prune out many poor design points quickly; and upon exiting the final gate, the topology can be assured to be competitive with other topologies.

The gates we chose are specific to the problem of analog circuit design, but the general concept can be generalized. The rest of this paragraph describes specifics for analog circuits. For the fastest gate, we leverage an operating-point driven formulation (Leyn et al., 1998). This formulation uses I's, V's, and L's as independent variables rather than W's and L's. Its advantages are that designable variables have less nonlinear coupling than a WL formulation; and that one can have "function device operating constraints (DOCs)" in which the DOCs can be measured by simple function calculations on design variable values without need for circuit simulation. To implement it, we need to compute W from the biases, for each device of each candidate design. First- or second-order equations are too inaccurate, and SPICE in the loop is too slow. So we sampled 350,000 points in L, Ids, Vbs, Vds, Vgs space, SPICE simulated each point once on an NMOS and once on a PMOS model, then store all the points in a lookup table. The second gate is simulation-based DOCs (Graeb et al., 2001). The third gate is performance constraints on non-transient testbenches. Table 1-2 gives pseudocode.

In our experiments, we found that the step 2 gate would take about 10-2000 designs to pass, the step 3 gate would take 10-500 designs, and step 4 would take 30-1000 designs. Overall runtime for the procedure was typically less than 10 minutes on a single 2.5-GHz machine. Note that this compares favorably with other recent single-topology circuit sizers, such as (Stehr et al., 2007). We achieved our immediate aim: to reliably generate complex topologies which could compete against simple topologies for multi-objective search, ensuring topology diversity.

Table 1-2. Procedure InitialCircuit()

```
1.  $C$ = RandomCircuit()
2.  While $C$ does not meet function DOCs:
        $C_{new}$ = Gaussian Mutate design variables of $C$
        If funcDocCost($C_{new}$) <= funcDocCost($C$):
            $C$ = $C_{new}$
3.  While $C$ does not meet simulation DOCs:
        $C_{new}$ = Mutate design variables of $C$
        If simDocCost($C_{new}$) <= simDocCost($C$):
            $C$ = $C_{new}$
4.  While $C$ does not meet performance constraints and iterations < max:
        $C_{new}$ = Mutate design variables of $C$
        If perfCost($C_{new}$) <= perfCost($C$):
            $C$ = $C_{new}$
```

3. Generation of Database

This section describes the experimental setup to generate the sized-topologies database. Table 1-3 lists the search space and goals. EA settings were: 100 individuals per age layer; 10 age layers, maximum age per layer: 19, 39, ..., 159, 179, infinity. The run took approximately 12 hours on a Linux cluster having 30 cores of 2.5 GHz each. 180 generations were covered. The resulting database had 1576 nondominated points comprising 15 unique topologies. Analog circuit specifics were: process technology of 0.18μm CMOS with 1.8 V supply voltage; output DC voltage of 0.9V; load capacitance 1pF; with Hspice circuit simulator.

Table 1-3. Problem Description

Search Space	50 topology and design parameters 50 variables, comprising 3528 possible opamp topologies (folded, cascode, source degen, 1 & 2 stage, ...)
Objectives	5 objectives: Maximize GBW, minimize power, maximize DC gain, maximize dynamic range, maximize slew rate
Constraints	Function DOCs, simulation DOCs, DC gain > 20 dB, GBW > 1e6, phase margin >65°, pole margin, dynamic range > 0.1, slew rate > 1e6

4. Knowledge Extraction

The previous sections culminated in the generation of a topology-performance-tradeoff database. This section uses that database, employing a suite of data-mining and visualization techniques to extract topology-performance insights.

Figure 1-2. Grid illustrating the Pareto Front of circuit performances. The diagonal entries show histograms of performance; the rest show two-dimensional projections from the five objectives. The squares are 1-stage amplifiers, and pluses are two-stage amplifiers.

Getting Started

We start with some obvious plots to become oriented with the data. In particular, Figure 1-2 shows a grid of 2d scatterplots and histograms for the five performance objectives. From the histograms, we can get a quick picture of the distribution and bounds of performances. From the scatterplots, we begin to understand the limits of combinations of performances and take note of trends. Note how the one-stage topologies only occupy a different region of performance space and follow a markedly performance trend than two-stage. The two-stage topologies have several sub-clusters of performances, hinting at further decomposition of topology types.

Classification Trees

Here, we automatically construct classification (decision) trees of performance values to topology choice. Classification trees have a double use: they can directly suggest a choice based on inputs, yet also expose the series of steps underlying the decision. Such CART trees themselves are nothing new

(Breiman et al., 1984), and are indeed in widespread use, from medicine to operations research. Decision trees for analog circuits are not new either - they have been manually constructed as in (Koh et al., 1990), in which one can start with a given set of specifications and pass through the tree to arrive at topology suggestion(s). The motive for analog design is clear: they capture the tacit analog design knowledge about topology decision-making. What is new is that we construct the decision tree automatically from data. This is only possible now, because a prerequisite to get the data was a competent multi-topology multi-objective sizer that could output a diverse set of topologies. As input to the tree construction, each different topology was assigned a class; the decision variables are the objectives used for generating the dataset; the dataset itself consisted of the non-dominated individuals.

Figure 1-3. A decision tree for going from specifications to topology. This was automatically generated.

Figure 1-3 shows the tree that was automatically generated, annotated with topology choices. It provides insight into what topologies are appropriate for performance ranges, and actually even gives a suggested topology from a set of input specs.

We examine the topology in more detail. While this paragraph is has some analog specifics, it nonetheless can give a reader from any field a general feel for the type of information that is extracted. We see that the objective of low-frequency gain (A_{DC}) is the first variable selected on, and following through the tree, we see that all objectives play a role for selecting some topologies: gain-bandwidth (GBW), power, slew rate (SR), and dynamic range (DR). When specifications require low-gain, the tree suggests single-stage topologies; and two-stage topologies when higher gain is required. In cases where very large gain is required with a limited power budget, a two-stage amplifier with large degrees of cascoding (K) is suggested. If power is less of an issue, one can also use a non-cascoded two-stage amplifier (G). Since only non-dominated individuals are used to generate the tree, the choice for the more power-efficient variant implies lower performance for one or more other metrics (in this case e.g. dynamic range).

It is important to remember that the tree is a classifier at its core, which can help avoid reading too much into it. To aid understanding, we describe its construction (Breiman et al., 1984). The algorithm starts with just a root node holding all data points. From among all possible {split_variable, split_value} tuples in the data, it chooses the one from that splits off the most data points using the "gini criterion" . That split creates a left and right child, each getting a subset of the data according to the chosen variable and value. The algorithm recurses, splitting each leaf node until there is just one sample at each leaf node or another stopping criteria is hit. Pruning is done interactively after construction of the full tree, taking just a couple minutes to find a tree that had a nice tradeoff between complexity and detail.

Relative Impacts

Here, we automatically extract the relative impacts of topology & design variables on each objective value. This can give insight into questions on which specific topology choices have influence on specific performances, and by how much, such as (for circuit design) "how much does cascoding affect gain versus number of stages?"

Table 1-4. Procedure ExtractImpacts()

```
let $X$ = matrix where each column holds one sample of {topology,
   design values}
let $y$ = target performance value for each sample
$regr$ = Build regressor mapping $X$ to $y$
$impacts$ = ImpactsFromRegressor($X$, $y$, $regr$)
```

Table 1-5. Procedure ImpactsFromRegressor(X, y, $regr$)

```
$error\_per\_variable$ = $\{\}$
For each variable $v$
    $error$ = 0
    Repeat $num\_scrambles$ times
        $X_{scr}$ = $X$ but randomly permute row of $v$
        $y_{scr}$ = simulate regressor on $X_{scr}$
        $error$ = $error$ + rmse($y$, $y_{scr}$)
    $error\_per\_variable\{v\}$ = $error$
$impacts$ = normalize $error\_per\_variable$
return $impacts$
```

Table 1-4 gives pseudocode for the high-level algorithm, and Table 1-5 for a subroutine. The regressor needs to handle numerical and categorical input variables, handle 50 input variables with 1500 samples, handle nonlinear mappings, and and have high prediction accuracy. We use stochastic gradient boosting (Friedman, 2002), which does importance sampling of CART trees, adaptively shifting the sampling distribution towards regions of high residual error (Friedman and Popescu, 2003). The procedure in Table 1-5 takes its inspiration from chapter 10 of (Hastie et al., 2001). It defines impact for a variable as the relative error that a scrambled input row (variable) will give in predicting, compared to other rows. Its advantages are: non-parametric, robust, and independent of the form of the underlying regressor.

With these algorithms, we extracted the relative importance of variables for each objective. Figure 1-4 illustrates for GBW. We see that the most important variable is “chosen part index”, which is a topological variable that selects one vs. two stages. As expected, design variables that experts commonly associate with this target objective (GBW) of opamps also show up, which help to validate the approach. The rest of this paragraph contains specifics for analog circuits. The GBW design variables are bias current of the first stage and size of compensation capacitance. Interestingly, the figure also indicates a large influence of the length of the transistors in the first stage (input, folding and load). This can be readily explained: these lengths directly influence impedance on the internal nodes, and hence the location of the non-dominant pole. The phase margin requirement >65($^\circ$) translates into the requirement that this non-dominant pole frequency is sufficiently higher than the GBW (approx 2x) (Sansen, 2006).

Figure 1-4. Relative impact of topology, sizing, and biasing variables on GBW, for 10 most important variables

Whitebox Models

The aim here is to extract whitebox models that capture performance tradeoffs. Whitebox models can handle high dimensionality, be analyzed by inspection, and manipulated further by hand. To generate the models, we set one performance as the output, and the rest as inputs; then we applied CAFFEINE (McConaghy et al., 2005) which is GP-based symbolic regression, constrained by a canonical functional form grammar to maintain interpretability of output equations.

Table 1-6 shows results for the objective of GBW. With our circuit expertise, we expected $gain$ to be strongly related to be GBW, and it turns out that just a linear relation between the two will get < 9% training error. But for a better fit, more complex nonlinear relations are needed leading up to an inverse relationship of GBW with $gain$ or $\sqrt{gain}$. The objective of slew rate (SR) is also needed for a reasonable model. Interestingly, the objectives of dynamic range and power are not needed to get within 3.5% training error. Cross-examination with the scatterplots (Figure 1-2) confirms that the strongest tradeoffs are indeed among $gain$, GBW, and SR.

Table 1-6. Whitebox models Capturing Performance Tradeoff

Train error	$Log(GBW)$ Expression
8.7 %	$10.28 - 0.049 * gain$
7.8 %	$12.57 - 0.69 * \sqrt{gain}$
7.3 %	$5.65 + 86.5/gain + 2.92e - 11 * SR$
6.8 %	$5.72 + 80.2/gain + 4.75e - 06 * \sqrt{SR}$
5.7 %	$7.30 + 47.76/gain - 3430/\sqrt{SR}$
4.1 %	$4.48 + 24.9/\sqrt{gain} - 8.60e6/(gain^2 * \sqrt{SR})$
3.5 %	$16.9/(1 + 0.15 * gain + 1.44e - 22 * SR^2 + 2.56e6/(gain^2 * \sqrt{SR}))$

5. Conclusion

This paper showed how the results of a GP synthesis run can be data-mined to extract domain knowledge that can be immediately used by domain experts. Specifically, it demonstrated a flow and tools to aid domain experts to efficiently choose structures (topologies) and parameters for a given design problem; and also accelerate insight into the relationship among topologies, design variables, and performances. The prerequisite was to generate a high-quality database of topology-performance tradeoffs; to that end, we showed how to enhance multi-topology multi-objective sizing to handle many objectives (via ARF) and generate a variety of topologies (via properly biased random sampling, and a multi-gate constraint satisfaction strategy) on top of a parallel-computing, multi-objective ALPS system. With the database, we applied data-mining based knowledge extraction tools: the automated creation of a specs-to-topology decision tree, stochastic gradient boosting with variable-scrambling to identify relative impacts of topology & design variables on performance, and CAFFEINE symbolic regression to generate whitebox models of tradeoffs.

6. Acknowledgment

Funding for the reported research results is acknowledged from IWT/Medea+ Uppermost, Solido Design Automation and FWO Flanders.

References

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees*. Chapman & Hall.

Corne, D. and Knowles, J. (2007). Techniques for highly multiobjective optimization: Some nondominated points are better than others. In Thierens, Dirk and et al., editors, *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 773–780.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evolutionary Computation*, 6(2):182–197.

Friedman, J.H. (2002). Stochastic gradient boosting. *Journal of Computational Statistics & Data Analysis*, 38(4):367–378.

Friedman, J.H. and Popescu, B. (2003). Importance sampled learning ensembles.

Graeb, H.E., Zizala, S., Eckmueller, J., and Antreich, K. (2001). The sizing rules method for analog integrated circuit design. In *International Conference on Computer-Aided Design*, pages 343–349.

Hastie, T., Tibshirani, R., and Friedman, J.H. (2001). *The Elements of Statistical Learning*. Springer.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten, Cattolico, Mike, Arnold, Dirk, Babovic, Vladan, Blum, Christian, Bosman, Peter, Butz, Martin V., Coello Coello, Carlos, Dasgupta, Dipankar, Ficici, Sevan G., Foster, James, Hernandez-Aguirre, Arturo, Hornby, Greg, Lipson, Hod, McMinn, Phil, Moore, Jason, Raidl, Guenther, Rothlauf, Franz, Ryan, Conor, and Thierens, Dirk, editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Iba, Hitoshi (1996). Random tree generation for genetic programming. In Voigt, Hans-Michael, Ebeling, Werner, Rechenberg, Ingo, and Schwefel, Hans-Paul, editors, *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pages 144–153, Berlin, Germany. Springer Verlag.

Koh, H.Y., Séquin, C.H., and Gray, P.R. (1990). Opasyn: A compiler for cmos operational amplifiers. *IEEE Transactions on Computer-Aided Design*, 9:113–125.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Koza, John R., Keane, Martin A., Streeter, Matthew J., Mydlowec, William, Yu, Jessen, and Lanza, Guido (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

Leyn, F., Gielen, G., and Sansen, W. (1998). An efficient dc root solving algorithm with guaranteed convergence for analog integrated cmos circuits. In *International Conference on Computer-Aided Design*, pages 304–307.

McConaghy, T., Eeckelaert, T., and Gielen, G. (2005). Caffeine: Template-free symbolic model generation of analog circuits via canonical form functions