Double-Strength CAFFEINE: Fast Template-Free Symbolic Modeling of Analog Circuits via Implicit Canonical Form Functions and Explicit Introns

Trent McConaghy, Georges Gielen

K.U. Leuven, ESAT-MICAS
Kasteelpark Arenberg 10
B-3001 Leuven, Belgium

Abstract

CAFFEINE, introduced previously, automatically generates nonlinear, template-free symbolic performance models of analog circuits from SPICE data. Its key was a directly-interpretable functional form, found via evolutionary search. In application to automated sizing of analog circuits, CAFFEINE was shown to have the best predictive ability from among 10 regression techniques, but was too slow to be used practically in the optimization loop. In this paper, we describe Double-Strength CAFFEINE, which is designed to be fast enough for automated sizing, yet retain good predictive abilities. We design "smooth, uniform" search operators which have been shown to greatly improve efficiency in other domains. Such operators are not straightforward to design; we achieve them in functions by simultaneously making the grammar-constrained functional form implicit, and embedding explicit 'introns' (subfunctions appearing in the candidate that are not expressed). Experimental results on six test problems show that Double-Strength CAFFEINE achieves an average speedup of 5x on the most challenging problems and 3x overall; thus making the technique fast enough for automated sizing.

1 Introduction

Symbolic models of analog circuits increase a designer's understanding of a circuit, leading to better decision-making in sizing, layout, verification, and topology design. Since manually creating such models can be time-consuming and difficult, automated model creation is of interest.

Symbolic analysis approaches [1] directly analyze an input topology, but are limited to linear and weakly nonlinear circuits. *Symbolic modeling* approaches [2,3] use SPICE-like simulation data to generate interpretable mathematical expressions that relate the circuit performances to the design variables, i.e. "performance models". By leveraging SPICE, such approaches can handle nonlinear circuits, environmental effects, manufacturing effects, and different technologies.

Unlike other approaches, CAFFEINE [3] generates symbolic models with *open-ended* functional forms, i.e. without *a priori* supplied templates. CAFFEINE's strategy is to build Canonical Functional Form Expressions in Evolution so that functions are interpretable. Figure 1illustrates the flow.

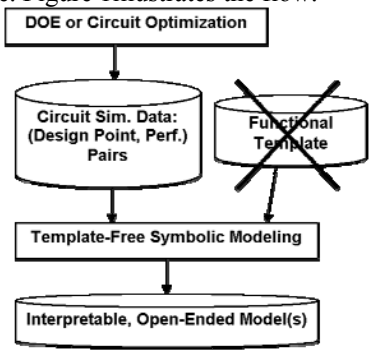

Figure 1: CAFFEINE symbolic modeling flow

CAFFEINE's weakness was speed. In [4], ten modeling techniques were compared on six problems, in the context of circuit optimization. While CAFFEINE had the best prediction ability, it needed to be at least 3x faster to have model building time below 5 minutes; Figure 2 illustrates. Speed would also help behavioral modeling [5].

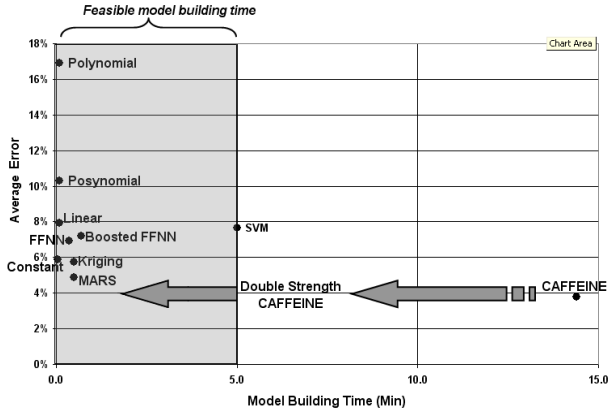

Figure 2: Error vs. modeling time for 10 techniques in the context of a circuit sizing problem, plus the aim of Double Strength CAFFEINE

This paper shows how we have sped up CAFFEINE using two techniques novel to analog CAD and to canonical-form function building: "smooth operators" and "introns". The contributions of this paper are:

- Reconciling so-called "smooth, uniform crossover" and "smooth mutation" with canonical form functions
- Designing non-expressed subfunctions, or "introns", for canonical form functions
- Demonstration of faster creation of template-free interpretable symbolic models, thus creating new opportunities for applying CAFFEINE, most notably in automated circuit sizing

This paper is organized as follows. Section 2 defines the problem. Sections 3-5 give background on GP, CAFFEINE, smooth / uniform operators, and introns. Section 6 describes Double Strength CAFFEINE. Section 7 shows results; section 8 provides conclusions.

2 Problem Formulation

The problem that CAFFEINE addresses [3] is:

- *Given:* A set of $\{\mathbf{x}(t),y(t)\},t=1..N$ data samples where $\mathbf{x}(t)$ is a d-dimensional design point t and $y(t)$ is a corresponding circuit performance value measured from (SPICE) simulation of that design, and **no** model template
- *Determine:* A set of symbolic models $f^* \in F$ that together provide the optimal tradeoff between prediction error and some measure of model complexity (i.e. a multi-objective problem).

This paper has an additional aim: make CAFFEINE fast enough to be used within the context of optimization; specifically, make it at least 3x faster than [3].

We measure speedup as the reduction in number of candidate functions (individuals) to solve test problems. The multi-objective nature of CAFFEINE adds a wrinkle: one can measure performance of multi-objective algorithms [6], but such measures either lose information making multi-objective measures scalar, or provide just qualitative assessment. We took a simpler route appropriate to our problem: have an upper bound of complexity, minimize the training normalized mean-squared error (nmse) with a *single*-objective algorithm, and stop when target nmse is hit. This strategy is fair as long as our speedups are independent of the number of objectives.

3 Background: GP and CAFFEINE

Genetic programming (GP) [7] is an evolutionary algorithm, i.e. a stochastic population-based search technique. GP's distinguishing characteristic is that individuals (points in the design space) are *trees*. A grammar can be used to structure this space of possible trees for GP's search [8, 9].

CAFFEINE uses GP as a starting point, but extends it with a specially designed grammar which allows all functional combinations but in just one canonical form; that form is shown in Figure 3. CAFFEINE uses a multi-objective algorithm [10] in order to provide a tradeoff between prediction error and complexity.

An example function that CAFFEINE might evolve is: f(x) = -10.3 + 3.1*x6 + 1.87 * x1 * log(-1.95 + 10.3 / (x2*x7)/(x5)). The value '-10.3' is the top w_o; the '3.1' and the '1.87' are the weights of basis functions for the top 'weighted linear add'; the 'x6' is the lone instance of top-level standalone 'Poly/Ratl'; the 'x1' is a 'Poly/Ratl' that has a product with the nonlinear function log(). Inside the log() is another weighted linear add subfunction.

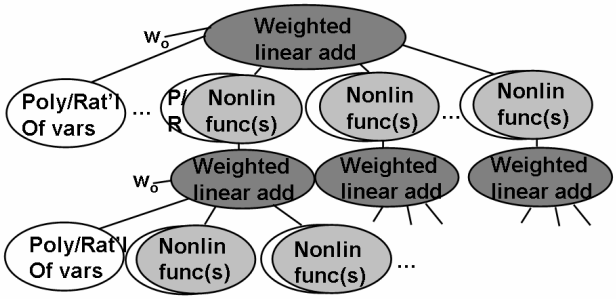

Figure 3: CAFFEINE evolves functions of this canonical form

4 Background: Smooth, Uniform Operators

We now describe why an algorithm designer would aim for operators that are "smooth", that are "uniform," and what those terms mean in the context of search.

When we design a search operator, we are effectively shaping the structure of the space that the search algorithm will be traversing. If we do a good job at operator design, then the space will appear relatively smooth to the search algorithm, have less local optima, and thus be easier to navigate. Conversely, if we do a poor job, then the space will have jagged peaks, crevasses, and a high number of local optima. Algorithms that search in a space of possible structures (such as GP) usually to belong in this latter category – but only because the importance of operator design is underestimated.

"Smooth operators" in GP are an explicit attempt to make structural change better behaved, to "melt" the jagged peaks into smooth hills. Smooth operators cause small expected change in fitness when an individual is changed slightly. This in turn gives a higher expected probability of a successful search step. A key insight to designing such smooth operators is via "behavioral bridge" [11]: design an operator such that a small change in a design space usually causes a small change in *behavior* space, which usually causes a small change in fitness space. In section 6, we show how we create smooth operators for the search of structures of functions.

Even if one has smooth operators, the space might be structured such that two near-identical designs are at

opposite ends of the space, such that reaching one from the other would take an unreasonable number of mutations. What one needs in search is a way to quickly "tunnel" between two design regions that are similar (though of course not "tunnel" in a way that is catastrophic to the design).

A good tool to consider "similar design regions" is the notion of "building blocks" [12]. In GP, a building block is a subtree; not all nodes of that subtree need to have been chosen yet [13]. Similar points in the GP search space have a large number of similar subtrees. From this perspective, GP's function is to process building blocks, via its sub-processes of selection, reproduction, crossover, and mutation. A key aim is to ensure that building blocks continue to "mix well," which is equivalent to ensuring a good set of tunnels between similar regions in design space. In GP, it has been shown that "uniform crossover" accomplishes such mixing [13]. Recall that "crossover" swaps sub-portions of the design of one parent with the sub-portions of the second parent, to create two children. In *uniform* crossover, each node in each location of the child has *equal probability* of coming from either parent. Compared to other crossover styles, uniform crossover has the most possibility for variety in new designs.

Uniform crossover on trees is hard to implement, as one needs to align operators, handle different depths, and handle functions of different arity. [14] had heuristics to handle these issues, which were complex but worked for its problem domain; their payoff was a decent speedup. It is also challenging to find a way to add smoothness to uniform crossover. We address these in section 6.

We have just described what "smooth" and "uniform" operators are about, and why they can be useful. We now proceed to discuss one more algorithmic tool (introns), after which we describe our proposed algorithm.

5 Background: Introns

Introns are subtrees that are not "expressed", i.e. when such subtrees change the fitness stays the same. They are the equivalent of "junk DNA" in biology. They are usually unintentional and therefore hard to control, but it has been shown that they can be explicitly designed, and that doing so improves search speed and reliability [15].

The explanation from a fitness landscape perspective is that such "neutral" changes connect regions of search space that would otherwise be poorly connected, therefore reducing the probability of getting caught in local optima [16]. So whereas in the last section we showed how an *operator* can be designed to tunnel (crossover), introns actually leverage *representation* to tunnel.

6 Double-Strength CAFFEINE

In this section, we present how to embed smooth operators and introns into CAFFEINE, yet have interpretable functions via a canonical form. We actually retain the whole same algorithmic flow as the original CAFFEINE; we just change the operators and representation.

The trickiest part is designing the crossover operator, most notably making it both uniform *and* smooth. The key to solve this is to notice that a canonical form function *already* imposes a structure – rather than trying to work around it, we can exploit it. We will show how we can abandon a "true" tree representation, and instead use the canonical form to wedge the tree into a bitstring.

It's not straightforward: different subfunctions may be of different depth, arity, etc. Introns provide the answer: put all trees at maximum depth and branchiness, but so that simpler functions remain possible, *we allow subfunctions to be turned off*, by merely setting that subfunction's exponent to zero. The same principle applies to varcombo exponents, and weights. There are still symbols inside the subfunctions, but they do not get expressed. Thus, we have *explicit introns*. Now, since all individuals' trees have the same number of symbols, they can truly be represented as a fixed-length string.

Figure 4 illustrates the representation.

The canonical functional form is now *implicitly* followed. Equivalent types are aligned on a bitstring: the sums of basis functions, the nonlinear operators, the arguments within the operators, etc. Due to this, one can perform simple string-style uniform crossover on them. Furthermore, having the symbols line up means that the "behavioral bridge" is maintained, thereby achieving smoothness in crossover.

While it has an underlying hierarchical structure that

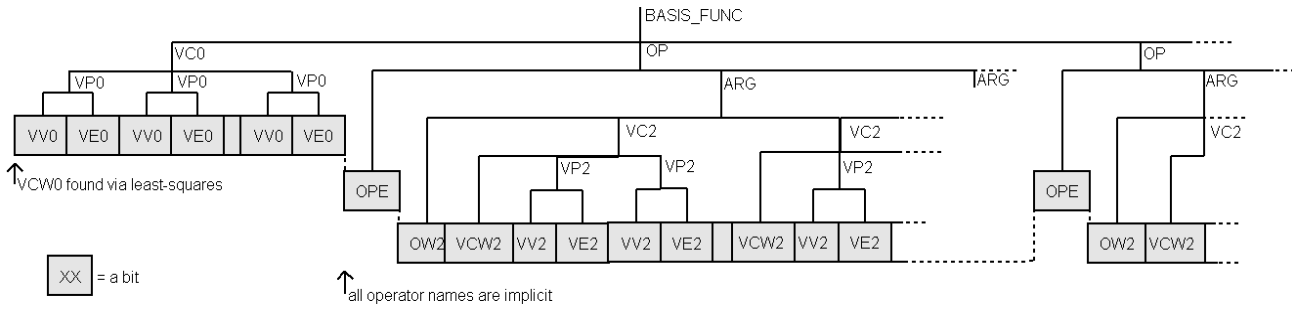

Figure 4: Search-space representation of one basis function of an individual in Double-Strength CAFFEINE.

follows the functional form of Figure 3, it can be treated as a fixed-length string of symbols too, to make uniform crossover possible. This tree/string duality puts it in a gray area between GP and genetic algorithms (which have fixed-length bitstrings).

The VC0 symbol is a depth-0 Poly/Ratl *varcombo*, and VP0 is a *varcombo pair* consisting of a VV0 variable name and VE0 variable exponent. There are *max_num_interacting_vars* of these VP's for each VC. OPE is the operator exponent. The operator name is implicit based on its position; each operator gets a place in each basis function. Inside each operator OP is a list of arguments (ARGs). Each ARG has an offset weight (OW2) and a set of weight-VC tuples (VCW2 plus the usual VC representation). Introns occur anytime a VE0, OPE, VCW2, or VV2 is zero.

A concern is that introns create a search space with too many dimensions. So we use domain knowledge to prune the space. First, we note that our operators always have arity of 1 or 2, which means exponential growth as depth grows is reasonable. Also, even though the focus application is sizing, maintaining the constraint of "interpretability" from [3] acts as a useful surrogate to reduce overfitting. Using that, we choose never to embed one nonlinear operator into another one. Finally, an interpretable design has a limit on how many variables interact, so we change the design of varcombos from an exponent for each variable, to a list of {variable name, variable exponent) pairs.

With our representation in place, we can now design the smooth mutation operators for functions. They are: change real-valued weights by a small amount; delete basis functions with near-zero weights; copy a basis function, then mutate the new basis function (and let linear learning determine weight allocation). Other mutations are less smooth, yet still small: changing the exponent of a single variable or nonlinear operator; changing a variable name, setting any weight to zero (akin to deletion, and simultaneously, intron insertion); copying a varcombo into another varcombo location. Finally are the neutral mutations that naturally occur inside introns: swap the order of whole basis functions within an individual; and swap the order of basis functions inside a nonlinear operator.

7 Experimental Results

7.1 Experimental Setup

We used the same test setup as in the original CAFFEINE paper [3]. The circuit being modeled is a high-voltage CMOS OTA as shown in Figure 5. The goal is to discover expressions for low-frequency gain (A_{LF}), unity-gain frequency (f_u), phase margin (PM), input-referred offset voltage (v_{offset}), and the positive and negative slew rate (SR_p, SR_n). There are 13 design variables. Full orthogonal-hypercube Design-Of-Experiments sampling was used, with scaled range dx=0.1 to have 243 samples from simulation.

The settings for both CAFFEINE and Double-Strength CAFFEINE were identical. Operators: $\sqrt{x}$, $\log_{10}(x)$, $1/x$, abs(x), x^2, 10^x, max(x_1,x_2), min(x_1,x_2). Maximum number of basis functions = 7, population size 200, stop when 500 generations or target nmse of 0.05 hit, varcombo exponents in [-3,-2,-1,-1/3,-1/2,0,1/3,1/2,1,2,3], and weights in $[-1e+10,-1e-10]\cup[0]\cup[1e-10,1e+10]$. CAFFEINE's maximum tree depth was 7, therefore allowing just one layer of nonlinear operators. All operators had equal probability, except parameter mutation was 5x more likely. Ten runs were done, for each performance goal, on both the old and proposed algorithm, for a total of 10*6*2=120 runs.

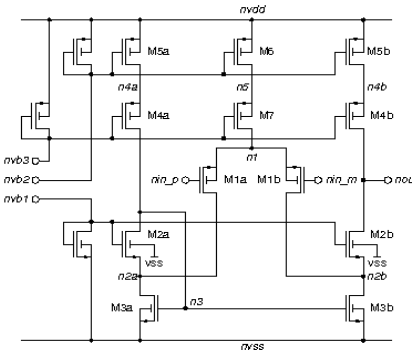

Figure 5: Schematic of high-speed CMOS OTA

7.2 Results and Discussion

Figure 6(a) shows the error vs. time for each of the ten runs, six problems, and both algorithms. We see that both algorithms converge towards the target nmse of 0.05 (5%), but it is apparent that Double-Strength CAFFEINE is doing better especially on the more challenging problems (SR_n, PM, A_{LF}). Whereas CAFFEINE often tapers off, Double-Strength CAFFEINE aggressively charges on. Such behavior is attributable to the combination of smooth, uniform operators and introns (which together, allow refinement of structure, allow easy access to other regions of search space, and bypass local optima). Figure 6(b), which shows the normalized mean square error averaged over all ten runs per performance characteristic, highlights the difference in convergence behavior on the challenging problems.

Our measure of speedups is based on the number of individuals (or equivalently, generations) to meet the target nmse. Figure 6(c) allows us to visualize this measure, incorporated as a probability of success vs. time. In the top-left plot (SR_n) we see that CAFFEINE was successful just twice, snagging the last success in its final generations. In contrast, Double-Strength CAFFEINE already had two successes within 250 generations,

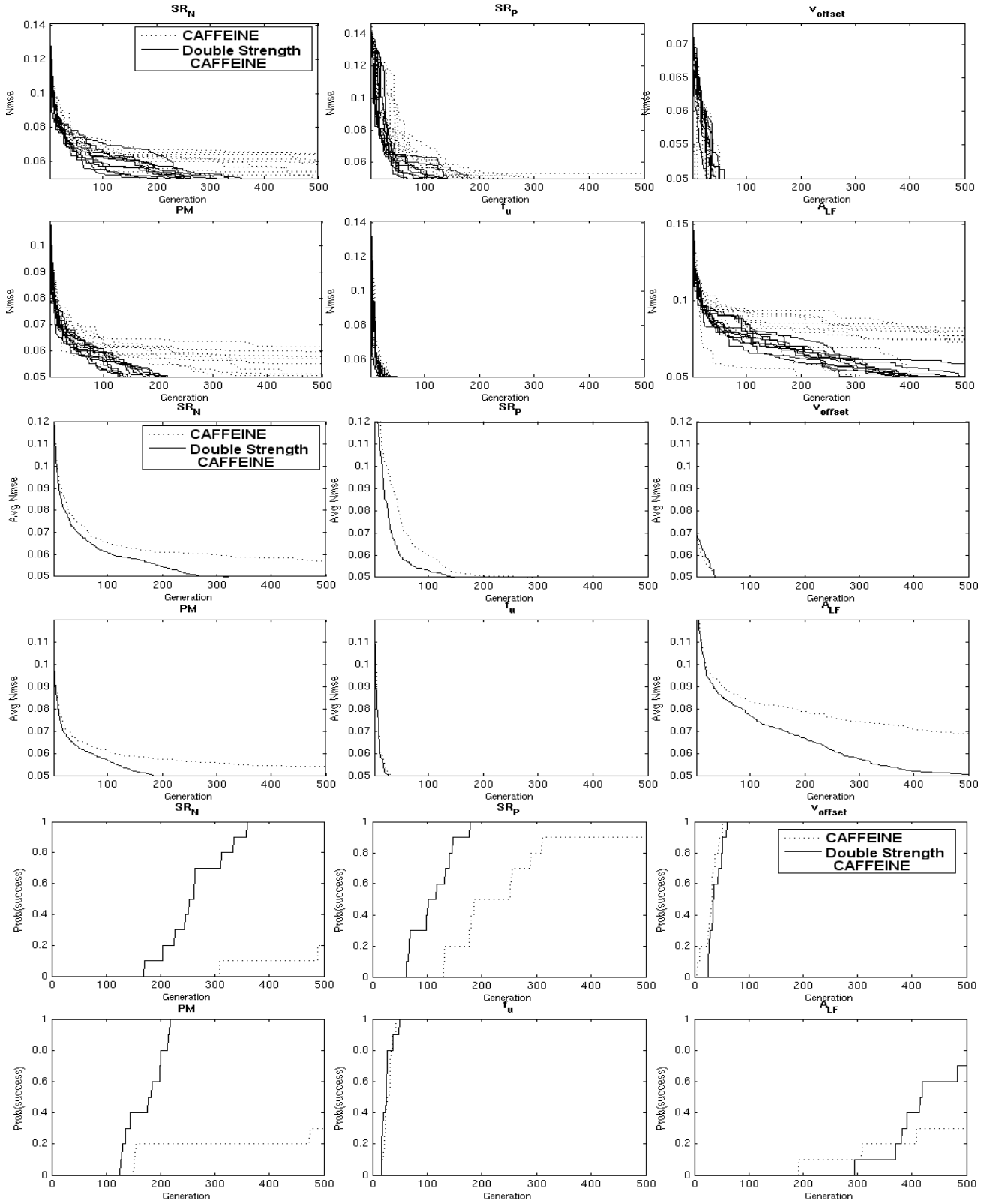

Figure 6: Comparison of CAFFEINE to Double-Strength CAFFEINE. (a) Top two rows are normalized mean-squared error (nmse) vs. generation for each of ten runs, for each performance characteristic (b) Middle two rows are average nmse vs. generation (c) Bottom two rows are probability of successfully hitting target nmse of 0.05 vs. generation

and by generation 400 was 100% successful. The difference was even more pronounced in the PM modeling problem, where Double-Strength CAFFEINE had chalked up 100% success by generation 250, whereas CAFFEINE barely chalked up 30% success in running twice as long. In A_{LF}, CAFFEINE for once stole an early lead, but had forfeited that by generation 400 and by generation 500 had only half the success rate of Double-Strength CAFFEINE.

Table 1 shows the average number of generations to success (in successful runs), the probability of success, and divides them to get the effective number of generations to solve a given problem. Speedup is the ratio of the effective number of generations, from old to new. The average speedup is 3.0x, and on challenging problems the average speedup is 5.0x. Figure 7 shows that, roughly speaking, the harder a problem was for old CAFFEINE, the more the speedup.

Table 1: Speedups. Average is 3.0x; on challenging problems (*) average is 5.0x.

	CAFF			*Fast Acting CAFF*			
Metric	**Avg # gen when succ**	**p(succ)**	**Eff. # gen**	**Avg # gen when succ**	**p(succ)**	**Eff. # gen**	**Spd-up**
ALF*	303	0.3	1010	394	0.7	563	**1.8x**
PM*	260	0.3	867	173	1.0	173	**5.0x**
SRn*	399	0.2	1995	263	1.0	263	**7.6x**
SRp	213	0.9	237	111	1.0	111	**2.1x**
Voffst	30	1.0	30	39	1.0	39	**0.8x**
fu	29	1.0	29	26	1.0	26	**1.1x**

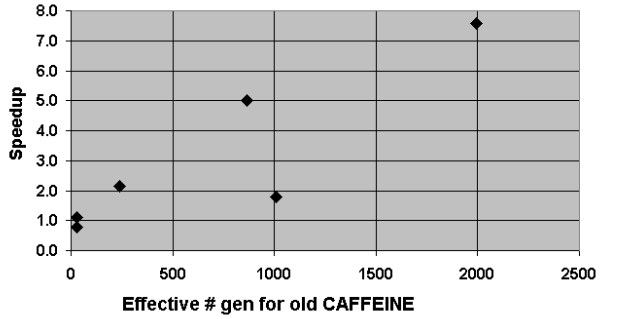

Figure 7: Speedup vs. Difficulty for old CAFFEINE

8 Conclusion

CAFFEINE generates interpretable, template-free symbolic models of nonlinear analog circuit performance characteristics as a function of design variables. This paper has presented an improved version, Double Strength CAFFEINE, that over all test problems had an average speedup over CAFFEINE of 3.0x, and 5.0x over challenging problems. This makes the approach fast enough for automated circuit sizing applications.

The key to speed is in achieving "smooth, uniform crossover" and "smooth mutation", via the special combination of an *implicit* canonical form function and *explicit* introns which allowed us to view candidate functions as fixed-length strings, and manipulate accordingly.

9 References

[1] G. E. Gielen, "Techniques and Applications of Symbolic Analysis for Analog Integrated Circuits: A Tutorial Overview", in Computer Aided Design of Analog Integrated Circuits And Systems, R.A. Rutenbar et al., eds., IEEE, 2002, pp. 245-261

[2] W. Daems, G. Gielen, W. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits," *IEEE Trans. CAD* 22(5), May 2003, pp. 517-534

[3] T. McConaghy, T. Eeckelaert, G. Gielen, "CAFFEINE: Template-Free Symbolic Model Generation of Analog Circuits via Canonical Form Functions and Genetic Programming", *Proc. DATE 2005*, March 2005

[4] T. McConaghy, G. Gielen, "Analysis of Simulation-Driven Numerical Performance Modeling Techniques for Application to Analog Circuit Optimization," *Proc. ISCAS 05*, May 2005

[5] T. McConaghy, G. Gielen, "IBMG: Interpretable Behavioral Model Generator for Nonlinear Analog Circuits via Canonical Form Functions and Genetic Programming," *Proc. ISCAS 05*, May 2005

[6] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation 8(2)*, Summer 2000, pp. 173-195

[7] J. R. Koza. Genetic Programming. MIT Press, 1992.

[8] P. A. Whigham, "Grammatically-based Genetic Programming," *Proc. Workshop on GP:Theory to Real-World Applications*, J. Rosca ed, 1995

[9] M. O'Neill, C. Ryan. Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer, 2003.

[10] K. Deb, S. Agrawal, A. Pratap, T.A. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," *Proc. PPSN VI*, Sept. 2000, pp. 849-858

[11] T. McConaghy, "Smooth Operators in Optimization of Circuit Structures," US Patent # 6,859,914, granted Feb. 2005

[12] D.E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

[13] W.B. Langdon, R. Poli. Foundations of Genetic Programming. Springer, 2002.

[14] R. Poli, J. Page, W. B. Langdon, "Smooth Uniform Crossover, Sub-Machine Code GP and Demes," *Proc. Genetic and Ev. Comp. Conf. (GECCO)*, vol 2, July 1999, pp. 1162-1169

[15] V. Vassilev, J. Miller, "The Advantages of Landscape Neutrality in Digital Circuit Evolution," *ICES, 2000*,pp. 252-263

[16] W. Banzhaf, "Genotype-Phenotype Mapping and Neutral Variation – A Case Study in Genetic Programming," In Y. Davidor et al, eds., *Proc. PPSN III*, 1994, pp. 322-332