Variation-Aware Structural Synthesis of Analog Circuits via Hierarchical Building Blocks and Structural Homotopy

Trent McConaghy, *Member, IEEE,* Pieter Palmers, *Member, IEEE,* Michiel Steyaert, *Fellow, IEEE,* and Georges G.E. Gielen, *Fellow, IEEE*

***Abstract*—This paper presents MOJITO-R, a tool that performs variation-aware structural synthesis of analog circuits. It returns trustworthy topologies, by searching across a space of thousands of possible topologies defined by hierarchically-organized analog structural building blocks. "Structural homotopy" conducts search at several objective-function tightening levels (numbers of process corners) simultaneously. Multi-objective evolutionary search returns sized topologies which trade off power, area, performances, *and yield*. An experimental validation run returned 78,643 Pareto-optimal designs, having 982 sized topologies with various specification/yield combinations. A decision tree is extracted to visualize the performance-topology relationship.**

***Index Terms*—analog, integrated circuit (IC), design automation, process variation, multi-objective optimization.**

I. Introduction

THE choice of analog circuit topology has a giant impact on circuit power, performance, area, and yield. Figure 1 shows a typical industrial topology selection / sizing flow. The designer starts by manually selecting an off-the-shelf topology. He / she then sizes it, either manually or with an automatic optimizer e.g. [1]. However, even the best optimizers can only produce as good a result as the chosen topology allows [2]. So, sometimes other topologies must be tried, re-looping through the flow. These iterations continue until success is achieved, or topology choices are exhausted. If necessary, a novel topology is designed, but only if the payoff is worth the risk.

Designers typically make the topology selection decision based on experience. Unfortunately, a suboptimal topology choice can occur: the topology may not handle worsening effects due to Moore's law, such as larger statistical variations [3], time-to-market pressure may give the designer too little time to be thorough, or the designer just does not have the experience level to know what might be best. This last point is understandable, as it is well recognized that learning analog circuit design is a process that takes years to get started and decades to master [4]. Hence, it is desirable for CAD tools to help the designer in optimal selection and design of topologies, especially because of the dramatic increase in process varations of recent years [3]. This introduction reviews such tools, then proposes a new tool, upon which the rest of the paper will elaborate.

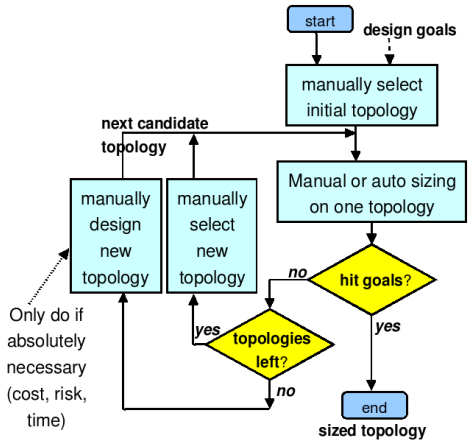

Fig. 1. Status quo industrial flow for topology selection/design and sizing.

One approach is to automate topology selection, by selecting from a given topologies database (DB) according to rules. Rule-based systems like BLADES [5], ISAID [6], OASYS [7], and others [8]–[13] follow this flow. For example, OASYS [7] has a pre-specified decision tree [14] that chooses among 12 different topologies based on input specifications. Unfortunately, these approaches require an up-front setup effort of weeks to months, which must be repeated for *each* new process node on each circuit type. AMGIE [15] aimed to overcome the process node issue by learning selection rules using SPICE in the loop, prior to the main sizing loop. However, it supported few topologies, and rule-learning took substantial computational effort. [16] is more flexible but needs behavioral models to be specified.

For the case of (flat) multi-topology sizing, a *family* of topologies is defined by parameterizing the topologies' possible structures using a fixed-length vector. Each variable in the vector is used to either (a) enable, disable, or choose specific components in a "flat" fashion, or (b) set sizing/biasing values. Such approaches include DARWIN [17] and MINLP [18] for opamps, and [19]–[21] for system-level designs. A key advantage is that only structural information about the

T. McConaghy was with ESAT-MICAS, Katholieke Universiteit, Leuven, Belgium. He is now with Solido Design Automation, Saskatoon, Canada.

P. Palmers was with ESAT-MICAS, Katholieke Universiteit, Leuven, Belgium. He is now with Mephisto Design Automation, Leuven, Belgium.

M. Steyaert and G.G.E. Gielen are with ESAT-MICAS, Katholieke Universiteit, Leuven, Belgium.

Manuscript received November 27, 2008; revised Feb. 25, 2009.

circuit is required, which is independent of the process node. However, each approach relies on a sneaky definition of the search space that is specific to the circuit type. There is not a clear path to generalize. The flat search space makes it difficult to compose libraries with large numbers of topologies. Due to these limitations, DARWIN and MINLP have just 24 and 64 possible opamp topologies, respectively.

Genetic programming (GP) [22] is an evolutionary algorithm that searches across trees and graphs (e.g. topologies). Accordingly, many approaches use GP for the automated design flow [23]–[35]. GP is given a set of devices that it can connect in arbitrary ways without rules – all the building blocks are "invented" (or reinvented) from scratch. This is what gives it the open-ended nature. Early approaches like [23]–[26] had few constraints, but needed prohibitive CPU effort. Even worse, results of almost all approaches were not *trustworthy* because there was no apparent logic behind them [36]. This trust issue was exacerbated because the results often looked strange, e.g. in early papers [23] to recent papers like [35]. A design is only trusted if the designer feels confident enough to commit it to silicon. Researchers aiming for industrially palatable circuits found themselves adding constraints, then re-running the system, and adding more constraints, and so on, in a seemingly non-stop loop. GP-based efforts like [27], [32], [33], [37] and a non-GP effort [37] added constraints using domain knowledge, but still give no guarantee of trustworthy results.

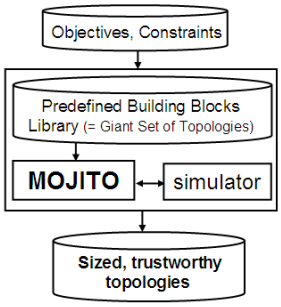

Fig. 2. Proposed flow using MOJITO(-R) for trustworthy-by-construction analog topology synthesis. A small library of analog structural building blocks combine hierarchically to create a *massive* library of possible topologies. MOJITO outputs sized topologies which are Pareto-optimal according to the objectives of area, power, performances, *and yield.*

We propose a new flow, shown in Figure 2. It returns trustworthy circuits (like the CAD approaches), that are specified by structural information only (like flat multi-topology sizing), yet searches through a structurally-diverse search space (like open-ended GP). Because the topologies library has a *sufficiently rich* number of topologies, the designer does not have to intervene in a typical design problem. It uses the same inputs and outputs as existing industrial automated sizers like [1]. It uses SPICE-in-the-loop for accuracy, flexibility, and easy adaptation to new technology processes. Actually, there is even one less input – unlike the sizers, the topology does not need to be specified. The flow is "specs in, sized topologies out". This is industrially-palatable *analog structural synthesis*.

The challenge is specification of a sufficiently rich library. After all, the "flat" topology libraries for opamps had < 100 topologies, with no clear path to get bigger. The answer is in using *hierarchical* analog building blocks. Some building blocks can instantiate into one of *many* building blocks.

There is a further challenge: process variations. Since some topologies are naturally more robust than others, process variations must be considered within the topology selection / design flow. We resolve this with *structural homotopy*, which simultaneously searches across increasing levels of problem thoroughness (more process corners). To our knowledge, this is the first time trustworthy structural synthesis has been combined with process variations.

Since different topologies are useful in different regions of the performance space, multi-objective search is a natural fit. [38], [39] did multi-objective sizing on one topology at a time, and merged the results. However, with thousands of topologies, thousands of runs is infeasibly expensive. Combining multi-objective with variation-awareness, *yield* becomes an additional objective (just like single-topology multi-objective robust sizers like [40]–[43]). This paper combines search across thousands of topologies (trustworthy structural synthesis), accounting for process variations, and multi-objective optimization into one system. Once the search returns a Pareto-optimal set, that set can then be stored, and subsequently queried by designers for an *immediate-turnaround specs-to-sized-topology flow*. The combination of multiple topologies and multiple objectives is where MOJITO earns its label: multi-objective and topology sizing. The "R" in MOJITO-R is for robust design.

Novel contributions of this paper are:

- A topology synthesis search space, specified by structural information only (no rules or behavioral information), with a massive count of possible trustworthy-by-construction topologies. The key is *hierarchical* pre-specified analog building blocks. Using this space, a library for opamps is designed that is *50x larger than past trustworthy structural-only spaces* [17], [18]. A search algorithm traverses the space in a *hierarchy-aware* fashion, using genetic programming [22]. It *avoids premature convergence* via an age-layered population structure (ALPS) [44].
- A means to handle process variation, *structural homotopy*, a novel homotopy approach that conducts search simultaneously on multiple layers of problem tightening, such that higher layers use more process corners. The approach is 10x faster than brute force and just 3x slower than nominal.
- A means to do multi-objective search, returning a Pareto-optimal set of sized topologies which trade off among performances and *yield*. This is done via NSGA-II multi-objective selection [45] at each ALPS layer.
- Finally, a means to help the designer understand the Pareto-optimal results by *automatically constructing* a visual *decision tree* that maps specifications to topologies.

An industrial-strength accurate model of process variations is used [46]. SPICE is used for evaluating circuits, with parallel processing to reduce overall runtime. Combining these items results in MOJITO-R: a variation-aware, multi-objective analog topology synthesis approach having industrially palatable

accuracy, setup requirements, runtime, generality, and results[1].

The rest of this paper is organized as follows. Section II describes the hierarchical search space. Section III describes (nominal) MOJITO, with experimental results in section IV. Section V describes the MOJITO-R extension for process variation, with experimental results in Section VI to validate the overall approach. Section VII describes extraction of a specs-to-topology decision tree, to enhance designer insight. Section VIII discusses other applications. Section IX concludes.

II. The MOJITO Search Space

This section describes a topology space that is specified by structural information only, searchable, trustworthy, and flexible. Its flexibility is due to a hierarchical description having parameter mappings, where the parameter mappings can choose sub-block implementations. We go on to describe an exemplary cell-level library for opamp synthesis to illustrate.

A. Search Space Framework

We define the library using hierarchically organized blocks. Each block has external ports and parameters for an interface. Some blocks may have sub-blocks; sub-block parameters are a function of the block's parameters. To generate a netlist for a given block, the only extra information needed is a value for each parameter of the block. Just three block types are needed to define a whole topology library:

- **Atomic Block.** Have no embedded blocks. Being leaf nodes in the building block hierarchy, only these blocks that would show up on a flat netlist. Fig. 3 gives examples.
- **Compound Block.** These have one or more sub-blocks embedded. Sub-blocks can have internal connections among themselves and to the block's external ports. Fig. 4 gives examples.
- **Flexible Block.** These have the special topological parameter $chosen\_part\_index$, which, during netlisting, is used to select one of several candidate embedded blocks and respective wirings, thereby enabling a library. Example: a current mirror which may be simple or cascode ($chosen\_part\_index$ = 0 or 1). Fig. 5 gives an example.

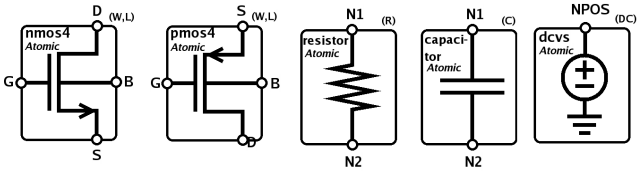

Fig. 3. Example atomic blocks: nmos4 transistor, pmos4 transistor, resistor, capacitor. Example of ports and input parameters: nmos4 has four external ports: *G*, *D*, *S*, and *B*; it has two input parameters, *W* and *L*. Note how *dcvs* (DC-controlled voltage source) has only one external port; the other port ties directly to ground.

Each block has its own parameters, such as transistor widths or branch currents, which fully describe how to implement and size the block and its sub-blocks. Despite the simplicity of such blocks, the combination of blocks means that a given block is its own *library of possible topologies*. A block's search space is merely the possible values that each sizing / topology choice parameter in the block can take. Larger blocks are built up from smaller blocks. To make a whole library, we continue the process to eventually reach the level of the target circuit, such as an operational amplifier (opamp). The blocks can readily be specified in an analog HDL, a circuit schematic editor, or a programming language (we use Python [51]).

[1]An earlier, nominal-only version of MOJITO was presented in [47].

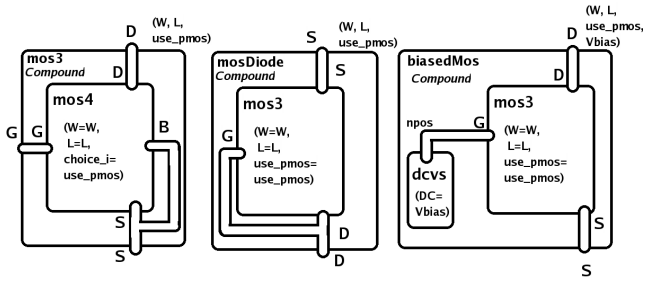

Fig. 4. Example compound blocks. mos3 is a wrapper for mos4, so that the mos4's 'B' node is not seen at higher levels. mosDiode ties together two internal ports to only present two external ports. biasedMos uses a 1-port dcvs (dc-controlled voltage source) block to set its gate bias internally.

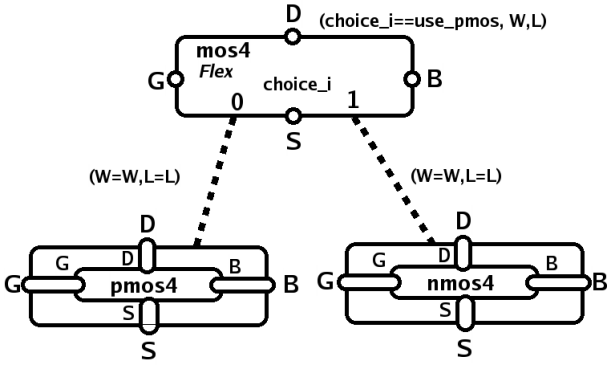

Fig. 5. Example flexible block: mos4 turns the choice of NMOS vs. PMOS into a parameter "*chosen_part_index*". Note how parameters get assigned from mos4 to either of its sub-blocks. In this case both sub-blocks use the mos4's W and L parameters as their own W and L values.

For all these subblocks, instantiation into sets of NMOS vs. PMOS devices is deferred until the very leaf block, as a function of the decision parameters that flow from higher levels. This flexibility allows for a large number of topologies at the top level, without having an excess number of building blocks. It also means that many parameters are shared in the conversion from one block to subblocks, which improves search space locality [50] (more on this later).

The MOJITO example opamp library allows for: one-and two-stage amplifiers, PMOS vs. NMOS loads, PMOS vs. NMOS inputs, stacked vs. folded cascode vs. non-cascode inputs, cascode vs. non-cascode vs. resistor loads, level shifting, different current mirrors, single-ended and differential inputs, and single-ended outputs.

For an industrial setting, this limited initial library would typically be provided by the tool provider, and will require low maintenance because it is generic across all semiconductor processes, and does not rely on any definitions of behavioral

models. Some designers will wish to alter the library themselves, e.g. to try out new ideas of building blocks.

B. Size of Search Space

Table I gives the topology count for the MOJITO opamp library, compared to other approaches[1]. It shows that MOJITO's flexible hierarchical nature increases the number of possible trustworthy opamp topologies in our library by 50x compared to previous publications [17], [18].

TABLE I
SIZE OF OPAMP TOPOLOGY SPACES.

Technique	# topologies	Trustworthy?
GP without reuse, e.g. [29]	$\gg$ billions	NO
DARWIN [17]	24	YES
MINLP [18]	64	YES
GP with reuse: MOJITO	**3528**	**YES**

Having a massive number of options can qualitatively change the designer's perception of the process: rather than doing selection from a few dozen topologies, *the tool is synthesizing* the optimal combination of building blocks from a huge set of possibilities. The number of topologies is sufficiently rich that the designer will feel less need to intervene in a typical design problem. Since the library only needs to be defined once for a given problem type (e.g. opamp), the designer no longer needs to view it as an input to the tool, not even if the process node changes. The space can be expanded even further, via adding even more building blocks. For example, [48] added blocks to support symmetrical transconductance amplifiers [49], which increased the MOJITO library's opamp count to 101,904 topologies (≈1500x more than MINLP [18]).

C. Search Space Locality

Good locality means that small changes to the genotype lead to expected small changes in performance (objective function). Good locality is important for an effective search algorithm [50]. In analog circuits, there is a complication to achieving locality. If a designer makes a small conceptual change to a circuit (genotype) that corresponds to a small change in electrical behavior / performance, there may still be a *dramatic* change in the netlist (phenotype). For example, Figure 6 shows four circuits with similar electrical behavior / performance. However, as we see in the figure, they have *very* different schematics / netlists / phenotypes. The analog design field has many more examples [49], [52].

The MOJITO blocks framework handles this. It leverages the flexible block's *chosen_part_index* parameter, which can be a *function* of one or more higher-level parameters, and choose between sub-blocks that are identical except for how

[1]Topology count is computed by the following rules: the count for an atomic block is one; for a flexible block, it's the sum of the counts of each choice block; for a compound block, it's the product of the counts of each of its sub-blocks. But there are subtleties. Subtlety: for a given choice of flexible block, other choice parameters at that level may not matter. Example: if a one-stage amplifier is chosen, do not count choices related to second stage. Subtlety: one higher-level choice might govern >1 lower-level choices, so don't overcount. Example: a two-transistor current mirror should have two choices (NMOS vs. PMOS), not four (NMOS vs. PMOS x 2).

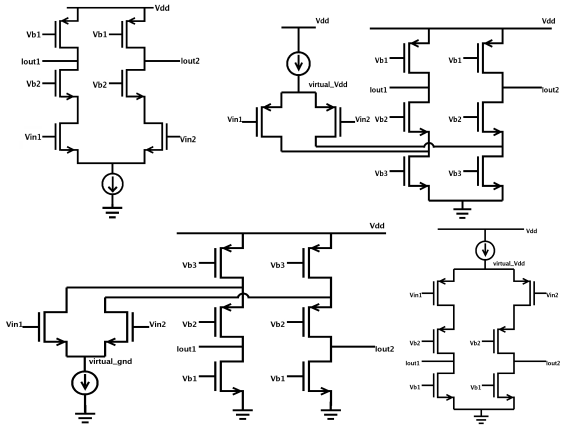

Fig. 6. Four circuits with similar conceptual behavior, but dramatically different phenotypes. The circuits on the left have PMOS inputs, and on the right have NMOS inputs ($input\_is\_PMOS$ = True/False). The load's rail is vdd in the top row, and gnd in the bottom row ($loadrail\_is\_vdd$ = True/False).

those sub-blocks are wired to their parent block. As an example, we describe how the framework handles Figure 6's circuits. Near the top of the library's hierarchy are the parameters *loadrail_is_vdd* ("is the load's rail attached to vdd, not vss?"), and *input_is_pmos* ("are the input devices pmos,not nmos?"). Those values propagate down the hierarchy until a choice for folded vs. cascode must be made: *is_folded* = (*input_is_PMOS* == *loadrail_is_vdd*). Figure 6 illustrates the effects of the four parameter combinations.

D. Worked Example

The search space is a library of circuit building blocks[1]/ Therefore, a point in the search space is a circuit, or in EA terms, an individual. This section illustrates some different ways one can view an individual. The schematic view is both concrete and intuitive (Figure 7). It is annotated to show the hierarchical composition of library blocks. A choice has been made for each Flexible block. The library's root node is dsViAmp2_VddGndPorts (as indicated in the schematic's top left corner). Its sub-blocks are dsViAmp1 (1^{st} stage), ssViAmp1 (2^{nd} stage), and viFeedback (Miller feedback capacitor), which subdivide further until Atomic blocks (leaf nodes) like nmos4, pmos4, and capacitor.

An individual is represented within the synthesis engine's code with a vector for a genotype. All the other representations can be computed from the genotype. The vector representation is an unordered mapping from the root block's variable names to corresponding value. Some variables specify for topology choices (*chosen_part_index*), and others are for setting specific device values (I's and V's which translate to W's and L's) via an operating-point driven formulation [53]. Table II gives the example individual's topology choice values. Each parameter is a choice for a Flexible Block. The first parameter, *chosen_part_index*, decides between one and two stages (value of 1 means two-stage). *stage1_loadrail_is_vdd* = 0 means that stage 1's loadrail is not set to *vdd*, but to *gnd* instead, as

[1]Equivalently, the library is a parameterized grammar.

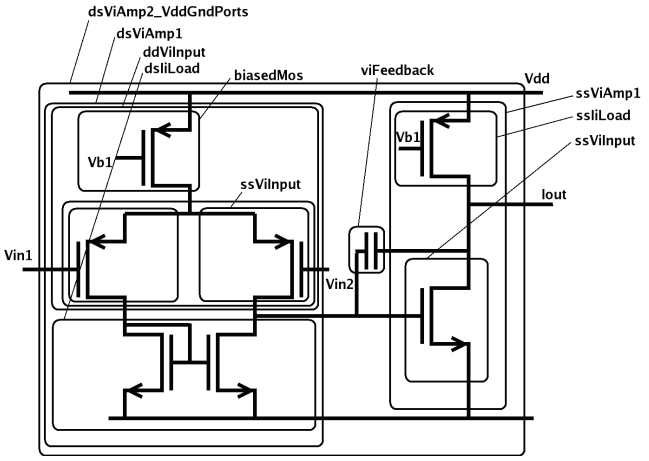

Fig. 7. An example individual ("PMOS-input Miller OTA") shown in schematic form, annotated with MOJITO building block labels.

we already saw. And so on. Note that some variables may be ignored, depending on values of other variables, e.g. if *chosen_part_index*=0 to choose a one-stage opamp, then all variables related to the second stage have no effect.

TABLE II
EXAMPLE INDIVIDUAL: VALUES FOR TOPOLOGY CHOICE VARIABLES.

Variable Name	Value
chosen_part_index	1
stage1_loadrail_is_vdd	0
stage1_input_is_pmos	1
stage1_degen_choice	0
stage1_inputcascode_is_wire	1
stage1_inputcascode_recurse	0
stage1_load_chosen_part_index	1
stage2_loadcascode_recurse	0
stage2_load_part_index	0
stage2_inputcascode_is_wire	1
stage2_loadrail_is_vdd	1
stage2_input_is_pmos	0
stage2_degen_choice	0
stage2_inputcascode_recurse	0

III. THE MOJITO SEARCH ALGORITHM

MOJITO search is a multi-objective evolutionary algorithm (MOEA) that uses an age-layered population structure (ALPS) [44] to balance exploration vs. exploitation. The algorithm's aim is formulated as a constrained multiobjective optimization problem:

$$\begin{array}{rll} minimize & f_i(\phi) & i = 1..N_f \\ s.t. & g_j(\phi) \leq 0 & j = 1..N_g \\ & h_k(\phi) = 0 & k = 1..N_h \\ & \phi \in \Phi & \end{array} \quad (1)$$

where Φ is the "general" space of possible topologies and sizings. The algorithm traverses Φ to return a Pareto-optimal set $Z = \{\phi_1^*, \phi_2^*, \cdots, \phi_{N_{ND}}^*\}$ on N_f objectives, N_g inequality constraints, and N_h equality constraints. Without loss of generality, we can minimize all objectives and have inequality constraints with aim ≤ 0. By definition, a design ϕ is feasible if it meets all constraints: $\{g_j(\phi) \leq 0\}\forall j$, $\{h_k(\phi) = 0\}\forall k$, $\phi \in \Phi$. By definition, all the designs in Z are *nondominated*, i.e. no design ϕ in Z dominates any other design in Z. A feasible design ϕ_a *dominates* another feasible design ϕ_b if $\{f_i(\phi_a) \leq f_i(\phi_b)\}\forall i$, and $\{f_i(\phi_a) < f_i(\phi_b)\}\exists i$.

The next subsection describes the high-level search algorithm, and subsequent sections describe details.

A. High-Level Search Algorithm

We use an evolutionary algorithm (EA) as the base of our search algorithm because EAs can perform constrained multi-objective optimization, e.g. [45], support parallel computing, and offer flexibility in overall algorithm design.

A key issue with most EAs is premature convergence, i.e. the algorithm converges to a local region of the design space too early in the search without having explored the global space sufficiently, which leads to sub-optimal results. Tactics to soften this include huge populations [25], [29], restarting e.g. [54], or diversity measures like crowding e.g. [45]. All these tactics are painful or inadequate [44].

Random injection of individuals might help because fresh building blocks can enter, except they get killed off too quickly during selection. To give random individuals a chance, the age-layered population structure (ALPS) [44] segregates individuals into *age* layers, and restricts competition to within layers. Genetic age is how many generations an individual's oldest genetic material has been around: the age of an initial individual is 0; the age of a child is the maximum of its parents' ages; age is incremented by 1 each generation. Figure 8 illustrates. Each age layer P_k holds N_L individuals. Layer P_1 allows individuals with age $\leq$19; P_2 allows age $\leq$39, and so on; the top level P_K allows ∞ age. If an individual exceeds the maxmium age for a fitness layer, it gets removed from that layer. Selection at an age layer k uses the individuals at that layer k and layer $k-1$ as candidates, such that younger high-fitness individuals can propagate to higher layers.

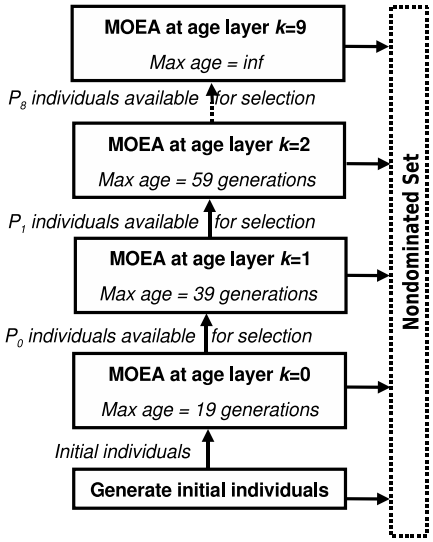

Fig. 8. Structure of MojitoSynthesis(), which uses an age-layered population structure (ALPS) and adds multi-objective support via multi-objective EA selection at each age layer.

TABLE III
PROCEDURE MOJITOSYNTHESIS()

Inputs: Φ, N_a, K, N_L
Outputs: Z
1. $N_{gen} = 0$; $N_{ind} = 0$; $Z = \emptyset$; $P = \emptyset$
2. while $N_{ind} < N_{ind,max}$:
3. if $(N_{gen} \% N_a) = 0$:
4. if $|P| < K$:
5. $P_{|P|+1} = \emptyset$
6. $P_{0,i} = \sim \Phi$, $i = 1..N_L$
7. for k = 1 to $|P|$:
8. (P_k, Z) = OneMOEAGeneration(P_k, P_{k-1}, Z)
9. $N_{gen} = N_{gen} + 1$; $N_{ind} = N_{ind} + \sum_k |P_k|$
10. return Z

We use ALPS in our search algorithm, *MojitoSynthesis()*, which has the pseudocode of Table III. Every N_a ("age gap") generations (Table III, line 3), a new age layer may be added (lines 4-5), and initial individuals enter layer *k*=0 as random individuals (line 6). Only a single-objective ALPS exists in the literature [44]. So, we designed a multi-objective version, by having multi-objective EA (MOEA) at each age layer k, running one generation at a time (line 9 in Table III). Whereas a canonical MOEA would select at just layer k, here the MOEA selection also considers layer $(k-1)$'s individuals. An external archive holding the Pareto-optimal set Z is always maintained. We use a maximum number of individuals, $N_{ind,max}$, as the stopping criterion (line 2).

TABLE IV
PROCEDURE ONEMOEAGENERATION()

Inputs: P_k, P_{k-1}, Z
Outputs: P'_k, Z'
1. P_{sel} = SelectParents($P_k \cup P_{k-1}$)
2. P_{ch} = ApplyOperators(P_{sel})
3. P_{ch} = Evaluate(P_{ch})
4. P'_k = $P_{sel} \cup P_{ch}$
5. Z' = NondominatedFilter($Z \cup P_{ch}$)
6. return (P'_k, Z')

Table IV shows the algorithm for the MOEA at each age layer R, for one generation. Note how individuals from the lower layer $k-1$ are imported for selection. NSGA-II [45] does selection. Therefore NSGA-II's dominance rules are used for handling constraints: a *feasible* design always dominates an *infeasible* design, and if two designs are infeasible then the one with smallest constraint violation is considered dominant. For feasible designs, NSGA-II's usual "nondominated filtering" approach applies [45] .

B. Search Operators

Tree vs. Vector View. Each building block has its own parameters, which fully describe how to implement the block and its sub-blocks. As we build up the hierarchy of building blocks, we eventually reach the level of the block we want to search for, such as the amplifier block. So, the search space for the circuit type (e.g. fully differential amplifier) is merely the possible values that each of the block's parameters can take. Since these parameters can be continuous, discrete, or integer-valued, one could view the problem as a mixed-integer nonlinear programming problem, which one could solve with an off-the-shelf algorithm whether it be a classical MINLP solver or an EA operating on vectors.

But a vector-oriented view does not recognize the hierarchy, which causes issues. One issue is that a change to variable(s) may not change the resulting netlist at all, because those variables are in sub-blocks that are turned off. Another issue is that an n-point or uniform crossover operator could readily disrupt the values of the building blocks in the hierarchy, e.g. the sizes of some sub-blocks' transistors change while others stay the same, thereby hurting the resulting topology's likelihood of having decent behavior. We cannot reconcile this by applying a hierarchical design methodology [55], [56] because there are not complete goals on the sub-blocks, just on the highest-level blocks[1]. To compensate, we design the EA mutation and crossover operators to have both tree- and vector-based aspects.

Mutation Operator. This operator varies one or more parameters. Continuous-valued parameters follow Cauchy mutation [57] which allows for both tuning and exploration. Integer-valued $chosen\_part\_index$ parameters follow a discrete uniform distribution. Other integer and discrete parameters follow discretized Cauchy mutations. While this can be viewed as mutations to the parameter nodes in a tree, it can also be viewed as changes to a vector in multi-dimensional Cartesian space, so that insights from vector-based optimization apply (e.g. evolutionary programming [57]).

Crossover Operator. Because crossover needs two individuals to operate on, the population-based nature of EAs make them a natural choice [22]. Crossover works as follows: given two parent individuals, randomly choose a sub-block S in parent A, identify all the parameters associated with sub-block S, and swap those parameters between parent A and parent B. It is possible that S shares parameters with other sub-blocks that have the same higher-level block as S; and in that case the new parameters for S will affect those other sub-blocks.

IV. MOJITO EXPERIMENTAL RESULTS

A. General Experimental Setup

The search space had N_d = 50 topology selection and sizing variables; there were 3528 possible topologies. MOJITO was implemented in Python [51]. The experimental setup parameters are given in Table V.

B. Experiment: Hit Target Topologies?

In this section, we aim to validate MOJITO's ability to find targeted topologies. The objectives were to maximize GBW, and to minimize power. Three runs were done, the only difference between them being the specified common-mode voltage $V_{cmm,in}$ at the input. We know that for V_{dd} = 1.8V and $V_{cmm,in}$ = 1.5V, topologies should result in an NMOS input pair. For $V_{cmm,in}$ = 0.3V, topologies should result in PMOS inputs. At $V_{cmm,in}$ = 0.9V, there is no restriction between NMOS and PMOS inputs.

[1]We could, however, still apply a hierarchical methodology to the results, and that is exactly how MOJITO would translate to design of system-level analog circuits.

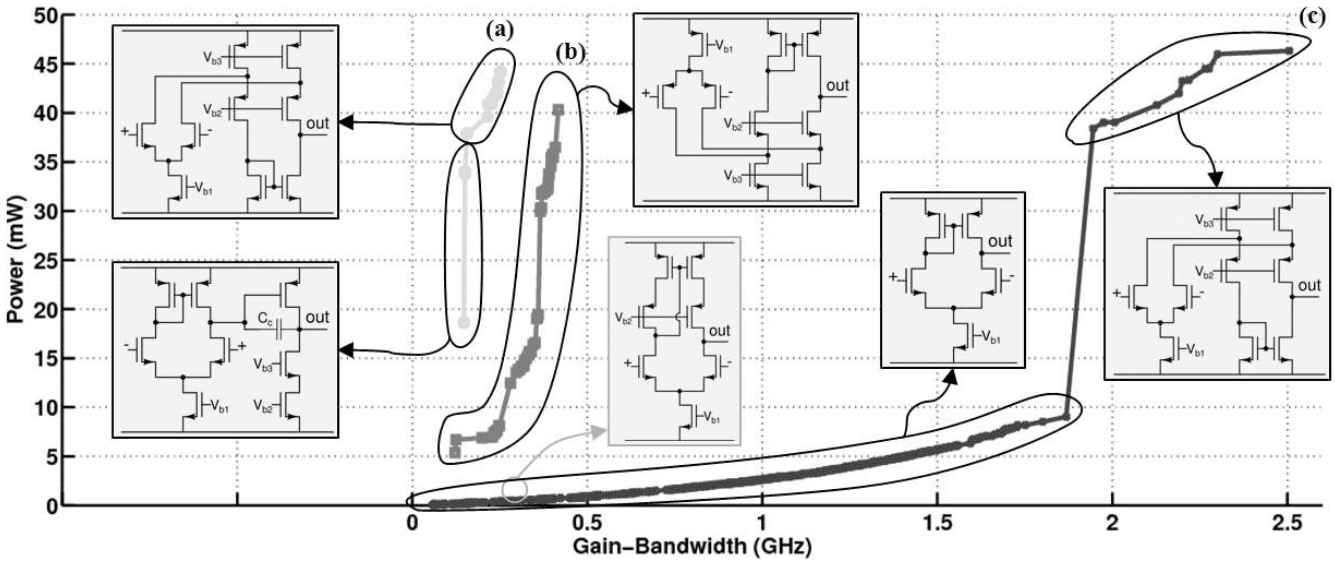

Fig. 9. Combined result plot for 3 synthesis runs. Set (a) shows a Pareto front for $V_{cmm,in}$ = 1.5 V, set (b) is for $V_{cmm,in}$ = 0.3 V and set (c) is for $V_{cmm,in}$ = 0.9 V. Aim is to minimize power and maximize Gain-Bandwidth. Each point is a sized topology; each topology has many different sets of sizings. The expected topologies were found.

TABLE V
EXPERIMENTAL SETUP PARAMETERS

Technology	0.18μm CMOS
Testbench parameters	load capacitance = 1pF, supply voltage = 1.8V, output DC voltage = 0.9V
Simulator	HSPICE^{TM}
Constraints	$PM > 65°$, DC Gain $>$ 30 dB, $GBW > 1$ GHz, power $<$ 100 mW, dynamic range $>$ 0.1 V, $SR >$ 1e6 V/s, dozens of device operating constraints [58]
Objectives	See specific experiment
EA settings	num. age layers K = 10, num. individuals per age layer N_L = 100, age gap N_a = 20, max. num. individuals $N_{ind,max}$ = 100,000

Each run took the equivalent of overnight on ten single-core 2.0 GHz Linux machines, covering about 100,000 individuals. Fig. 9 illustrates the outcome of the experiments. It contains the combined results of the three optimization runs.

Result (a) has $V_{cmm,in}$ = 1.5V, and has indeed only topologies with NMOS inputs. MOJITO chose to use 1-stage and 2-stage amplifiers, depending on the power-GBW tradeoff. Result (b) has $V_{cmm,in}$ = 0.3V, and MOJITO only returns amplifiers with PMOS input pairs. For result (c) a $V_{cmm,in}$ = 0.9V has been specified. Though both amplifiers with NMOS and PMOS input pairs might have arisen, the optimization preferred NMOS inputs.

The curve clearly shows the switch in topology around GBW $\approx$ 1.9GHz, moving from a folded-cascode input (larger GBW) to a simple current-mirror amp (smaller GBW). Note that there would be more amplifiers with folded-cascode input at GBW $<$ 1.9GHz, but they are not part of the Pareto-optimal set and therefore do not show up in the plot. An algorithmic answer is: By definition, the Pareto-optimal set only contains the designs that are no worse than any other designs. So the apparent "jump" is merely a side effect of a maximum achievable GBW of $\approx$ 1.9GHz for the simple current-mirror amp, after which a significantly higher-power amplifier, having the folded-cascode input, is needed. To get deeper insight yet, the designer can use his expertise in circuit analysis, e.g. here he would see that since going from a non-folded to a folded input needs one more current branch, the extra branch has extra current (power) needs. Interestingly, the search retained a stacked current-mirror load for about 250MHz GBW. All in all, this experiment validated that MOJITO did find the topologies that we had expected *a priori*.

V. MOJITO FOR ROBUST DESIGN: MOJITO-R

A. Problem Specification

Nominal MOJITO had many search objectives, but did not include yield. When we add yield as an objective, *each* sized topology has its *own* Pareto-optimal set, trading off performances with yield. For example, let us have gain as one objective (to maximize) and yield as the other. Then, an unattainably large gain specification will give a yield = 0.0%. But yield will rise as we loosen the gain specification (0.0% $<$ yield $<$ 100%), and eventually the specification will always be met (yield = 100%). This whole tradeoff of gain vs. yield is possible from a *single* design. We can generalize to > 1 performance objective, of course. Then, all design candidates' tradeoffs are merged to form the *overall* tradeoff. The problem formulation is:

$$\begin{array}{rll} minimize & f_i(\phi) & i = 1..N_f \\ s.t. & g_j(\phi) \leq 0 & j = 1..N_g \\ & h_k(\phi) = 0 & k = 1..N_h \\ & \phi \in \Phi & \end{array} \quad (2)$$

where all symbols are like section III, except f_1 is the objective to maximize yield. As discussed, the value of yield for a given ϕ is dependent on the values of $f_2, f_3, \cdots, f_{N_f}$. Before we proceed to describe MOJITO-R, we first describe a foundational technology: homotopy.

B. Homotopy

Homotopy or continuation methods (sec. 11.3 of [59]) are an optimization strategy in which the original optimization problem of solving $f(\boldsymbol{d}) = 0$ is not solved directly. Instead, an easy problem with an obvious solution is set up. This easy problem is gradually transformed to the true problem, and during the transformation, the solution to the problem is continuously tracked. Eventually, the problem becomes the true problem, and therefore its solution is the true solution.

Specifically, the *homotopy map* $H(\boldsymbol{d}, \eta)$ is defined as:

$$H(\boldsymbol{d}, \eta) = \eta * f(\boldsymbol{d}) + (1 - \eta) * (\boldsymbol{d} - \boldsymbol{a}) \quad (3)$$

where η is a scalar parameter and $\boldsymbol{a} \in \Re^{N_d}$. When $\eta = 0$, eqn (3) becomes the easy initial problem $H(\boldsymbol{d}, \eta) = \boldsymbol{d} - \boldsymbol{a}$, having the obvious solution of $\boldsymbol{d} = \boldsymbol{a}$. $H(\boldsymbol{d}, \eta)$ becomes the original problem when $\eta = 1$. The steps in between, i.e. the path in the space of $\boldsymbol{d} \cup \eta$ where $H(\boldsymbol{d}, \eta) = 0$ for various values of η, is called the *zero path*.

There are various strategies for shifting from the easy problem at $\eta = 0$ to the true problem at $\eta = 1$. The most obvious is to gradually change η from 0 to 1, and solve at each step along the way. However, this may not always work because the zero path may not always follow monotonically increasing values of η. More successful strategies track the zero path itself, rather than the η value.

C. MOJITO-R Approach

Whereas typical homotopy algorithms tighten *dynamically* towards the true objective function, *structural homotopy* embeds objective function tightening into the search state's *structure*: searches at several different tightening levels are conducted simultaneously. MOJITO-R extends MOJITO with structural homotopy to handle variation issues. It returns a Pareto-optimal set of sized topologies, which trade off yield and performances. Its high-level structure is shown in Figure 10. Its algorithm and sub-algorithms are identical to MOJITO (section III), except:

- Evaluations at ever-higher levels get progressively tightened (i.e. structural homotopy), by adding more Monte-Carlo sampled process points with the counts shown in Figure 10[1]. Each age layer simulates at *all* testbenches.
- For each sized topology, a yield-performances tradeoff is generated by (1) sweeping through all combinations of specifications[2], and computing yield for each vector of performance values, then (2) applying nondominated filtering on the resulting combined yield-performances vectors.
- At every generation, the final nondominated set is updated by merging the tradeoffs of each sized design in the top two layers (layers with full evaluation), then applying further nondominated filtering.

To further help topology diversity, the algorithm of Table VI replaces the random sampling in step 6 of Table III. It defers competition among randomly-generated topologies until each topology is at least close to feasible. It does so by optimizing sizings & biasings in a series of constraint-satisfaction "gates" that are successively more expensive to evaluate: from function-based device operating constraints (DOCs) (lines 2-5), to simulation-based DOCs (lines 6-9), and finally to performance constraints (lines 10-13). In all three gates, mutateSizings() applies Gaussian mutation to all sizing and biasing parameters of the design.

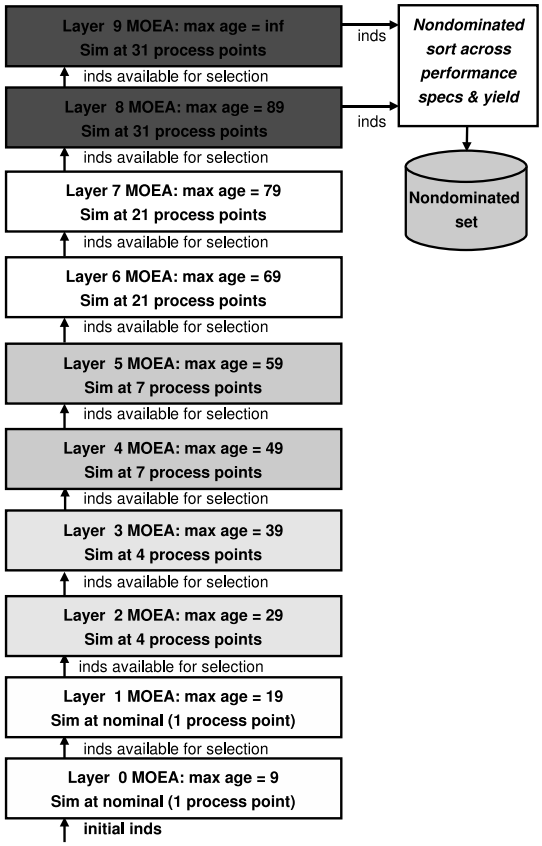

Fig. 10. MOJITO-R adds structural homotopy to MOJITO to achieve variation-aware topology synthesis.

We can estimate the additional simulation cost of running MOJITO-R versus running MOJITO. For simplicity, let us assume one testbench, for one generation at algorithm steady state when all age layers exist, with equal population size per age layer. To start with, we also assume that generating initial individuals comes for free. For a baseline, we assign a cost of 1 evalution-unit / layer for a single age layer with MOJITO, and therefore with (1 + 1 + ... + 1) = 1 * 10 = 10 evaluation-units for MOJITO. In MOJITO-R, upper age layers cost more, giving a cost of: 1 + 1 + 4 + 4 + 7

[1]These numbers were chosen using the following reasoning. 30 process points (+nominal) gives reasonable accuracy for the context of a yield optimizer. The jump from nominal to 4 process points, and from 4 to 7, adds three process points each time which is not a giant jump computationally, but starts to account for process variations. Additional process points have diminishing returns, but 21 is a reasonable middle ground to manage the jump from 7 to 30 process points.

[2]The specification values for each performance were the unique set of observed performance values in simulation. Example: if observed values for 4 process points were AV={61, 60, 60, 62} and $power$={0.02, 0.01, 0.01, 0.02}, then unique values are AV={61, 60, 62} and $power$={0.02, 0.01}, and yield is computated for each of the (AV,$power$) combinations of {(61,0.02), (61,0.01), (60,0.02), (60,0.01), (62,0.02), (62,0.01)}.

TABLE VI
PROCEDURE INITIALCIRCUIT()

Inputs: Φ
Outputs: $\phi \in \Phi$
1. $\phi \sim \Phi$
2. while meetsFuncDOCs(ϕ) $\neq True$:
3. ϕ' = mutateSizings(ϕ)
4. if funcDOCsCost(ϕ') $<$ funcDOCsCost(ϕ):
5. $\phi = \phi'$
6. while meetsSimDOCs(ϕ) $\neq True$:
7. ϕ' = mutateSizings(ϕ)
8. if simDOCsCost(ϕ') $<$ simDOCsCost(ϕ):
9. $\phi = \phi'$
10. while meetsPerfConstraints(ϕ) $\neq True$:
11. ϕ' = mutateSizings(ϕ)
12. if perfCost(ϕ') $<$ perfCost(ϕ):
13. $\phi = \phi'$
14. Return ϕ

+ 7 + 21 + 21 + 31 + 31 = 128. Therefore MOJITO-R is 128 / 10 = 12.8 times slower than MOJITO from these assumptions. However, we cannot ignore the cost of generating initial individuals. Via some ad-hoc tests, we saw that it took *Procedure InitialCircuit()* on average 500 simulations. If initial individuals are generated every N_a = 10 generations, this brings the cost of MOJITO to 500/10 (for init. gen.) + 10 (baseline) = 60 eval.-units, MOJITO-R to 500/10 (init. gen.) + 128 (baseline) = 178 eval.-units, and therefore **MOJITO-R is only 3.0 times slower than nominal MOJITO**. For comparison: a brute-force Monte Carlo (MC) implementation in which all individuals are evaluated on 30 MC samples is 30 times slower than MOJITO, and 10 times slower than MOJITO-R. A brute-force implementation in which all but the initial individuals are evaluated on 30 MC samples has a cost of 10 (30) + 500/10 = 350 eval.-units, which is 350/178 $\approx$ 2 times slower than MOJITO-R. These numbers may be slightly different on other circuit types.

The yield numbers are statistical estimates based on 30 MC samples (the "corners"), a compromise between runtime and accuracy (statistical confidence). On sized topologies of higher interest, if desired, the designer could invoke more accurate yield estimations or even a final sizing-only yield tuner. While the proposed approach cannot be directly applied to high-sigma synthesis, it could be conceivably altered to do so, e.g. via a final step of high-sigma yield-tuning and/or high-sigma corners instead of MC samples.

VI. MOJITO-R EXPERIMENTAL VALIDATION

This section describes experimental results from running MOJITO-R for opamp synthesis. We also describe extraction of a specs-to-topology decision tree from the synthesis results.

A. Experimental Setup and Run

We use the same experimental settings as section IV-A. The problem has seven objectives: maximize yield, minimize power, minimize area, maximize GBW, maximize gain, maximize dynamic range, and maximize slew rate.

The MOJITO-R run took approximately 48 hours on a Linux cluster having 30 cores of 2.5 GHz each (palatable for an industrial setting), covering 242 generations. It returned a database of 78,643 Pareto-optimal points, composed of 982 sized topologies having various specification combinations. 284 of those sized topologies have $\approx$100% yield.

B. Performances on Whole Pareto Front

We first examine each raw point's performance in Figure 11. Each diagonal entry is a histogram for a performance metric (or yield), and other entries in the grid give a 2-D scatterplot of performance / metric values. This plot offers insights into into tradeoffs between performances *and* yield. The histogram for gain is bimodal, with peaks at $\approx$ 55 dB and $\approx$ 110 dB. These modes appear in other plots where gain is on one axis; e.g. the gain vs. power plot has one distinctly higher-power group, and one distinctly lower-power group, plus some outliers. The higher-gain cluster has lower power, indicating that one does not have to compromise gain for power, or vice versa. On the other hand, the plot of GBW versus gain indicates that there can be either high GBW or high gain, but not both.

The yield dimension affects what these insights mean. The yield histogram is in the upper left of Figure 11. All points are not 100% yield, which means some of the performance extremes are only attainable with $<$100% yield. The histogram's peak is 10-20% yield, with most points having $<\approx$ 50%yield; therefore Figure 11 's tradeoffs are are mostly for $\ll$ 100% yield. Let us examine the subplots of Figure 11 where yield is the x-axis. At first glance, these plots seem surprisingly uninteresting because the yield value does not seem to strongly affect the distribution. But it means that each individual performance value is achievable regardless of yield requirement is, but at a tradeoff to other performances. A notable exception is slew rate versus yield (bottom left): the only way to achieve the higheest values of slew rate is with yields of $<$10%.

C. Topologies on the Whole Pareto Front

In the Pareto front, there were nine different topologies. They are illustrated in Figure 12. 982 sized topologies expanded into 78,643 Pareto-optimal points (as discussed, a sized topology can have > 1 Pareto-optimal point because the same design will give a variety of yield-performance tradeoffs). All the topologies are two-stage with NMOS inputs, but their differences end there. In the first stage, some topologies had cascode inputs and some did not. Some topologies had source-degeneration and some did not. The first stage's current-mirror load was either a simple current mirror, a cascode current mirror, or a low-voltage current mirror. The second stage was either PMOS input or NMOS input, sometimes had source degeneration, and sometimes there was a bias transistor in parallel with the input stage.

Table VII gives a count per topology in the whole Pareto front (second column). Topologies 4 and 7 had about 44,000 and 30,000 points respectively, while topologies 2 and 6 had just 98 and 25 points respectively, and the rest are in between.

D. 100%-Yield Pareto Front

On the yield-performances tradeoff, the most interesting subset is the one with 100% (estimated) yield. Designers can

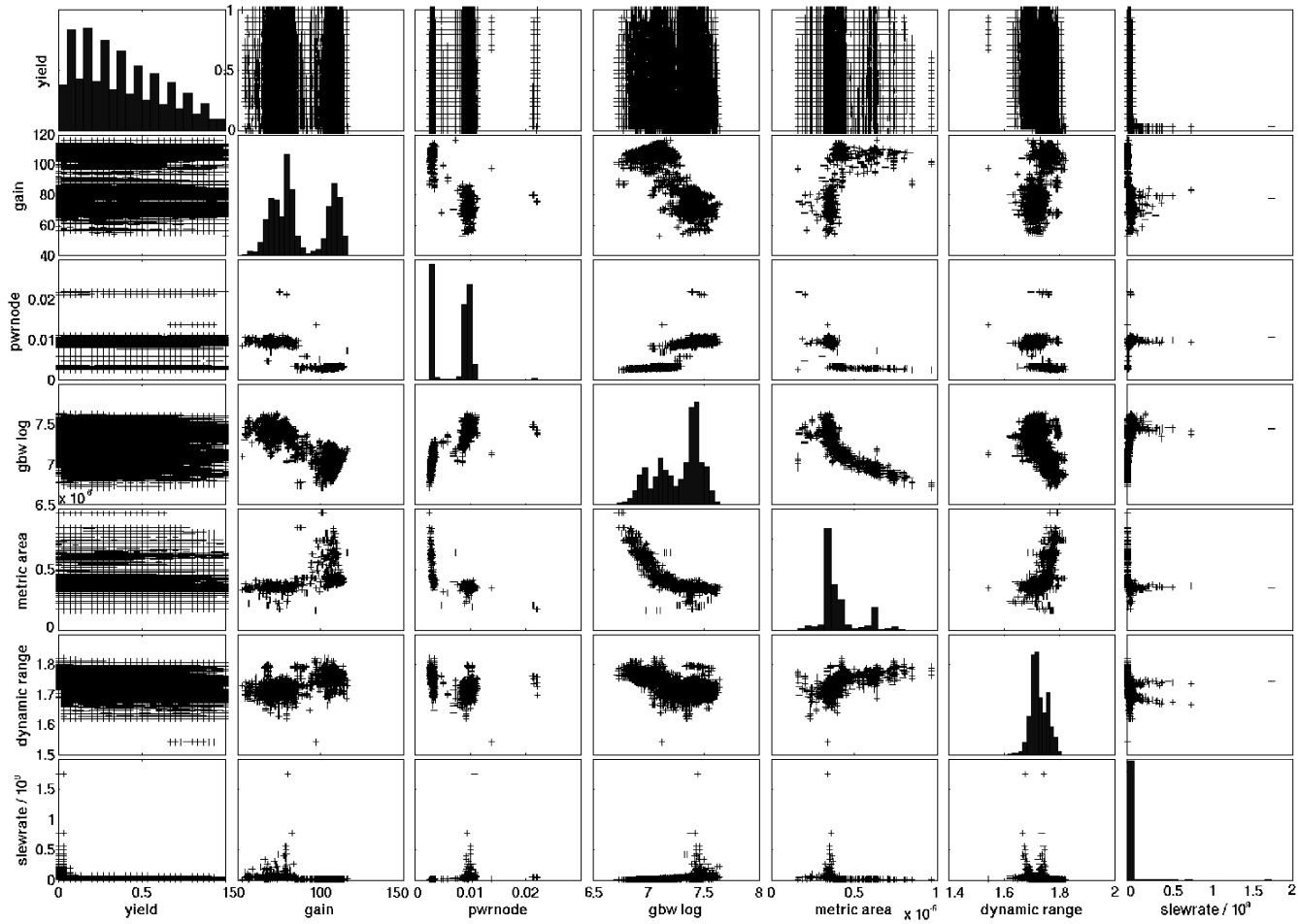

Fig. 11. 2-D scatterplots and histograms of the whole Pareto front. Each "+" is a different entry in the Pareto front. There are 78,643 entries.

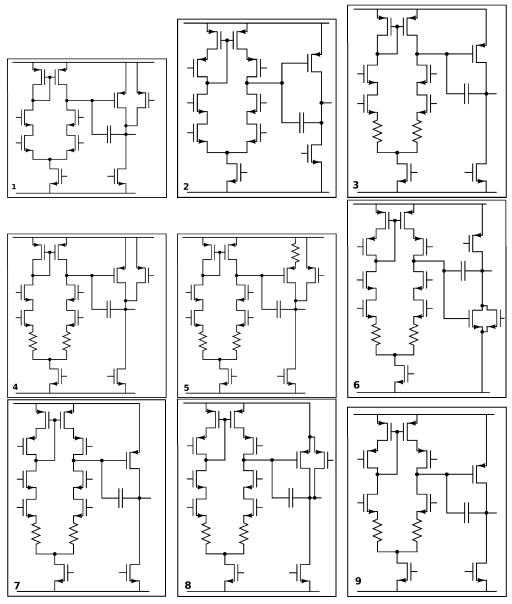

Fig. 12. Topologies in the MOJITO-R experimental run's Pareto front. A hanging gate implies either a bias voltage source connection or input node. Parallel connected transistor pairs are combinations of a second stage amplifier transistor and a biased transistor (a constant current source).

view this as "solving at all corners". Of the 78,643 Pareto-optimal points which were composed of 982 sized topologies having various specification combinations, 284 of those sized topologies have 100% yield (estimated).

Whereas Figure 11 showed all points in the Pareto Front, Figure 13 shows the 284 100%-yield designs. The general tradeoffs are largely the same, including the clusters. The performance values are less aggressive; most notable is that the maximum slew rate is $\approx$ 3x smaller. Some performances have very strong relation in terms of tradeoffs, such as the area-GBW tradeoff. The tradeoffs against slew rate (for lower-performing slew rates) are now more visible. We see that a typical improvement to slew rate will not affect gain, increase power (implying a tradeoff between slew rate and power), increase GBW (a bonus), reduce area (a bonus), and reduce dynamic range (a tradeoff).

The third column of Table VII gives the count for each topology. Some topologies never achieved 100% yield. So, when yield matters, fewer topologies are needed. Topologies with the most 100%-yield entries (third column) are the ones with the most any-yield entries (second column).

So far, we have only examined performances and topologies separately. Let us now examine the topology-performance relation by highlighting specific topologies. Figure 13 illustrates. As hinted before, the topologies break into two clusters of

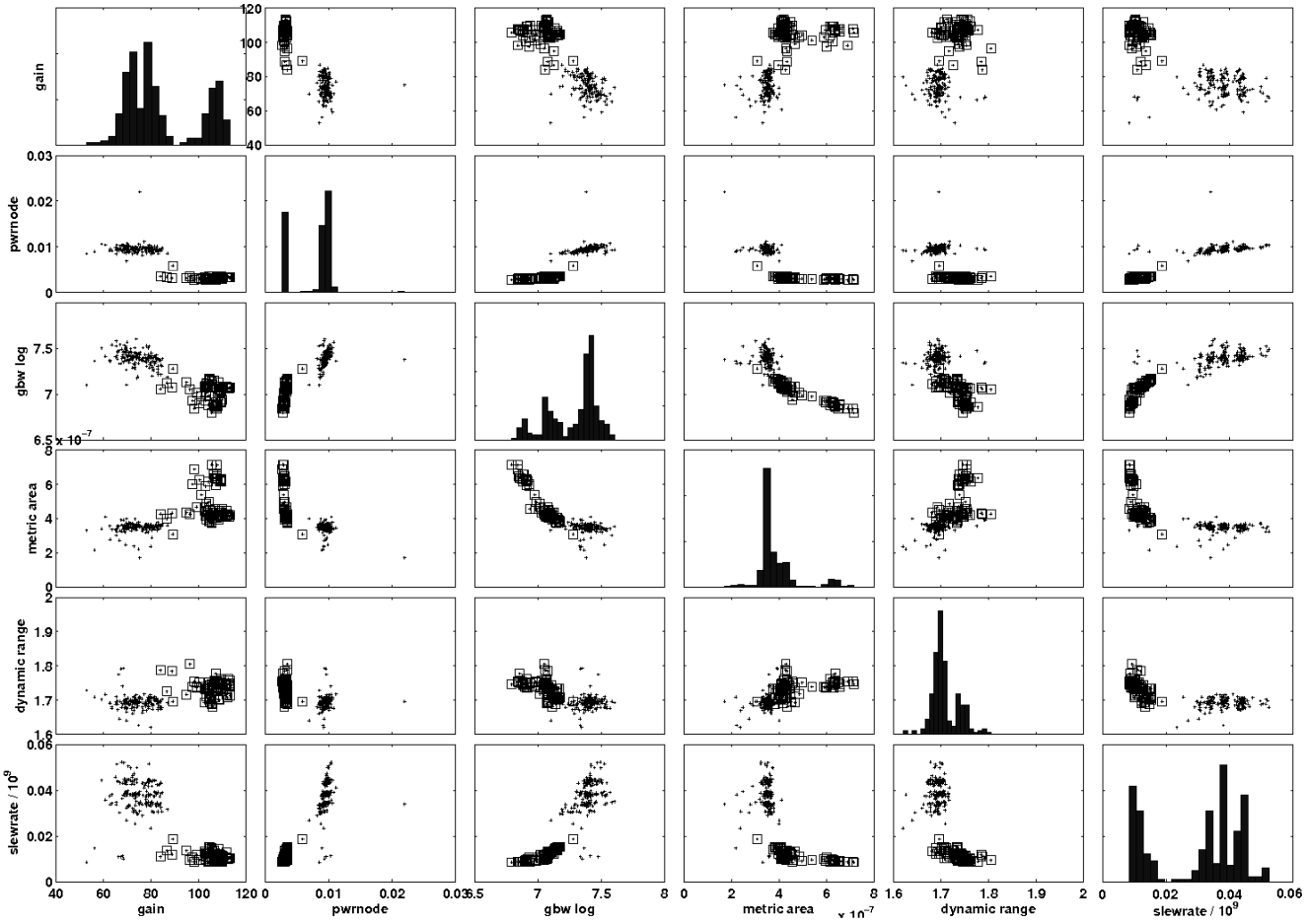

Fig. 13. The 100%-yield points in the Pareto front. The squares highlight topology 7, and the pluses are from topologies 1, 4, 5, and 9.

TABLE VII
TOPOLOGY COUNT IN PARETO FRONT

Topology Label	# Instances in Whole Pareto Front	# Instances in 100%-Yield Front
1	1165	5
2	98	0
3	169	0
4	44037	177
5	346	1
6	25	0
7	29687	89
8	219	0
9	2717	12

performance: topology 7 for one cluster (higher gain, lower power, lower GBW, higher area, higher dynamic range, and lower slew rate), and topologies 1, 4, 5, and 9 for the other. The next section shows a way to further explore the performance-topology relation.

VII. SPECS-TO-TOPOLOGY DECISION TREE EXTRACTION

Decision trees [14] can be used to gain insight into the relation between specifications and topology. These trees return a topology choice, given input specifications. OASYS [7] proposed a decision tree for a topology-choosing expert system, but its tree was manually constructed which took weeks to months of effort, used rules of thumb that became obsolete when the process node changed, and needed updating whenever a new topology was added to its library. In contrast, we construct a decision tree *automatically* from data. This is only possible now, because a prerequisite to get the data was a competent multi-objective, trustworthy-by-construction topology synthesis system. MOJITO(-R) is the first such system. Its output Pareto-optimal set becomes the input for automated tree extraction.

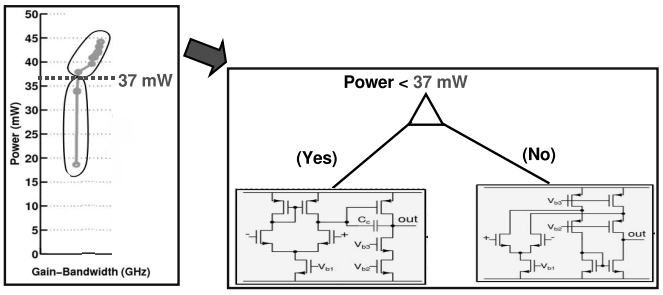

Fig. 14. Left: results of a two-objective run (minimize power, maximize Gain-Bandwidth GBW) which gave two topology choices indicated by the ellipses. Right: corresponding a decision tree to guide the topology choice based on the power input specifications.

Figure 14 gives a 2-objective example mapping from raw

Pareto-optimal data to decision tree. The raw data is from the nominal MOJITO run (a) of Figure 9. The tree reads: if a power $<$ 37 mW is chosen, the 2-stage amplifier is chosen, otherwise a 1-stage amplifier with folded-cascode inputs is chosen. In two dimensions, manual tree construction is trivial. But because visualization past 2-3 dimensions is extremely difficult (see Figure 13), automated extraction is preferred.

We formulate decision tree induction as a *classification* problem, using a Pareto-optimal set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_j^*, \ldots, \phi_{N_Z}^*\}$ resulting from a MOJITO-R run. Within Z, there are N_T unique topologies ($N_T \leq N_Z$) with corresponding class labels set as $\Upsilon = \{1, 2, \ldots, N_T\}$. For individual ϕ_j^*, let υ_j be its topology class label; $\upsilon_j \in \Upsilon$. Let $\boldsymbol{f_j}$ be the objective function values corresponding to υ_j: $\boldsymbol{f_j} = \{f_1(\phi_j^*), f_2(\phi_j^*), \ldots, f_{N_f}(\phi_j^*)\}$, an N_f – dimensional vector. Tree induction constructs a classifier ω that maps from $\boldsymbol{f_j}$ to υ_j, i.e. $\widehat{\upsilon_j} = \omega(\boldsymbol{f_j})$[1]. ω can be viewed as a collection of N_R disjoint hypercube regions R_i, $i = 1..N_R$; where each region R_i has an associated class $\upsilon_i \in \Upsilon$.

Tree construction using the CART algorithm [14] finds a tree ω in the space of possible trees Ω using a greedy algorithm. It begins with just a root node holding all data points $\{\boldsymbol{f_j}, \upsilon_j\}, j = 1..N_Z$ and therefore is represented by a single region R_1 covering all of input $\boldsymbol{f}$ space. Each objective i is a possible split variable, and the values $f_{i,j}$ for that objective comprise the possible split values (with duplicates removed). From among all possible $\{split\_var, split\_value\}$ tuples in the data, the algorithm chooses the tuple which splits off the most data points ("gini" criterion) [14]. That split creates a left and right child, where left child is assigned data points and region meeting $split\_var \leq split\_value$, and the right child is assigned the remaining data points and region. The algorithm recurses, splitting each leaf node until a leaf node has just one class left.

We used this approach on the data of Figure 13, to extract the tree shown in Figure 15 (100%-yield Pareto front)[2]. Extraction took $<$5 s. At the top node, a power requirement of $<$ 6.4 mW will lead to selecting topology 7. Then, if dynamic range must be $\geq$ 1.72, then topology 9 is chosen. The rest of the tree distinguishes between topology 1 (one case), topology 5 (one case), and topology 9 (other cases) using area and gain as decision factors. Topology 5 takes the most decisions to get to, implying that it occupies a tiny region of performances space, as confirmed by its single entry in Table VII. In summary, an automatically-extracted decision tree is a new means for a designer to gain insight into the process-specific performance-topology relationship.

VIII. Application to Other Circuit Types

To apply MOJITO to other circuit problems, one only needs to modify the library and objectives / constraints. The library can be readily modified by building up blocks and/or using a different block as the root node. In [61], MOJITO was applied to designing current mirrors robust to electromagnetic compatibility (EMC) effects. [62] used simple voltage-in, voltage-out computational circuits as "weak learners" within the context of a boosting algorithm [63] to realize complex computational circuits and flash analog-to-digital converters.

[1]The $\widehat{\ }$ illuminates that the classifer makes a *guess* of the true class.

[2]The earlier paper [60] does decision-tree extraction on nominal MOJITO-generated Pareto fronts.

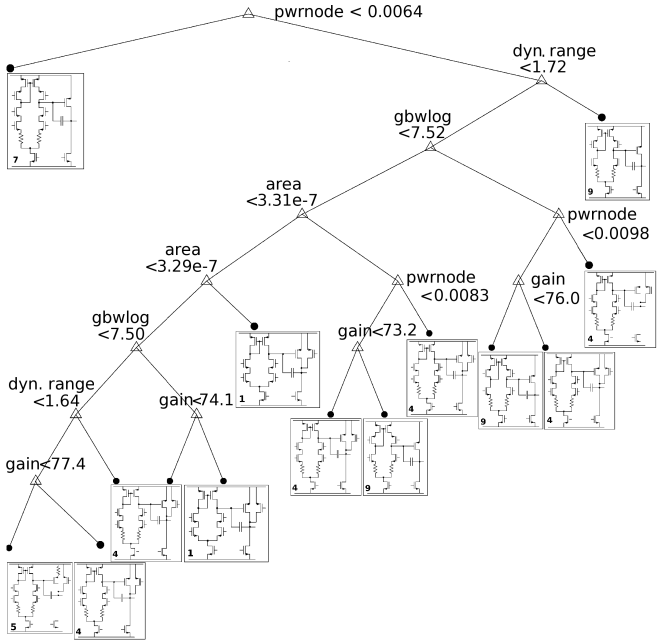

Fig. 15. A decision tree for topology choices in the 100%-yield Pareto front. This was automatically constructed from the results of a MOJITO-R run. Technology is 0.18μm CMOS.

IX. Conclusion

This paper has presented MOJITO-R, a novel approach for variation-aware structural synthesis of analog circuits. MOJITO-R returns sized topologies that are trustworthy by construction, using building blocks that are specified by structural information only, which combine to form thousands of possible topologies. MOJITO-R uses SPICE in the loop, and an accurate model of process variations [46]. These attributes allow MOJITO-R to be used in any process technology, with no additional setup effort. MOJITO-R handles variation-awareness efficiently via *structural homotopy*, in which searches at several different tightening levels (number of process corners) are conducted simultaneously. MOJITO-R performs multi-objective search to return a Pareto-optimal set of sized topologies which trade off among power, area, performances, *and yield*. The novel evolutionary algorithm implementing the search has an age-layered population structure to avoid premature convergence, multi-objective search, and parallel computing. MOJITO-R is experimentally validated in a synthesis run that searches across 3528 different one- and two-stage opamp topologies simultaneously, generating a Pareto-optimal set holding 78,643 Pareto-optimal points composed of 982 sized topologies, of which 284 have 100% yield. A decision tree is extracted to visualize the performance-topology relationship, capturing decisions in a high-dimensional input space.

ACKNOWLEDGMENT

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

REFERENCES

[1] Cadence Design Systems, Inc. (2005). Product page: Virtuoso NeoCircuit. http://www.cadence.com, last accessed 2005.

[2] R.A. Rutenbar, G.G.E. Gielen, and B.A.A. Antao, Eds.. *Computer aided design of analog integrated circuits and systems*. Piscataway, NJ: IEEE Press, pp. 3–30, 2002.

[3] International Technology Roadmap for Semiconductors. http://public.itrs.net, last accessed April 2008.

[4] J. Williams, Ed. *Analog design: Art, science, and personalities*. Newnes Press, ISBN: 0750696400, 1991.

[5] F.M. E1-Turky and R.A. Nordin, "BLADES: An expert system for analog circuit design," in *Proc. Intern. Conf. Circuits and Syst.*, pp. 552–555, 1986.

[6] C. Toumazou, C.A. Makris, and C.M. Berrah, "ISAID: A methodology for automated analog IC design," in *Proc. Intern. Symp. Circuits and Syst.*, vol. 1, pp. 531–555, 1990.

[7] R. Harjani, R.A. Rutenbar, and L.R. Carley, "OASYS: A framework for analog circuit synthesis," in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 12, pp. 1247–1266, 1992.

[8] E. Berkcan, M. d'Abreu, and W. Laughton, "Analog compilation based on successive decompositions," in *Proc. Des. Autom. Conf.*, 1988.

[9] H.Y. Koh, C.H. Séquin, and P.R. Gray, "OPASYN: A compiler for CMOS operational amplifiers," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 9, pp. 113–125, 1990

[10] Z. Ning, A.J. Mouthaan, and H. Wallinga, "SEAS: A simulated evolution approach for analog circuit synthesis," in *Proc. Custom Integr. Circuits Conf.*, pp. 5.2-1-4, 1991.

[11] K. Swings, S. Donnay, and W.M.C. Sansen, "HECTOR: A hierarchical topology-construction program for analog circuits based on a declarative approach to circuit modeling," in *Proc. Custom Integr. Circuits Conf.*, 1991.

[12] B.A.A. Antao and A.J. Brodersen, "ARCHGEN: Automated synthesis of analog systems," in *IEEE Trans. Very Large Scale Integr. Circuits*, vol. 3, no. 2, pp. 231–244, 1995.

[13] N.C. Horta and J.E. Franca, "Algorithm-driven synthesis of data conversion architectures," in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 10(16), Oct. 1997, pp. 1116–1135.

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

[15] G. van der Plas, G.G.E. Gielen, and W.M.C. Sansen, *A computer-aided design and synthesis environment for analog integrated circuits*. Springer, ISBN:0792376978, 2002.

[16] E. Martens and G.G.E. Gielen, *High-level modeling and synthesis of analog integrated systems.* Springer. ISBN: 9781402068010, 2008.

[17] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Proc. Des. Autom. Conf.*, 1995.

[18] P.C. Maulik, L.R. Carley, and R.A. Rutenbar, "Integer programming based topology selection of cell level analog circuits," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 4, 1995.

[19] K. Francken, P. Vancorenland, and G.G.E. Gielen, "DAISY: A simulation-based high-level synthesis tool for delta-sigma modulators," in *Proc. Intern. Conf. Comput. Aided Design*, pp. 188–192, 2000.

[20] A. Doboli and R. Vemuri, "Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 22(11), 2003.

[21] H. Tang and A. Doboli, "High-level synthesis of delta-sigma modulator topologies optimized for complexity, sensitivity, and power consumption," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 25(3), pp. 597–607, 2005.

[22] J.R. Koza. *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, MA: MIT Press, 1992.

[23] J.R. Koza et al., "Automated synthesis of analog integrated circuits by means of genetic programming," *IEEE Trans. Evol. Comput.* 1(2), July 1997, pp. 109–128.

[24] J.D. Lohn and S.P. Colombano, "Automated analog circuit synthesis using a linear representation," in *Proc. Second Intern. Conf. Evolv. Syst.: From Biology To Hardware*, pp. 125–133. Springer-Verlag, 1991.

[25] J.R. Koza, D. Andre, F.H. Bennett III, and M. Keane. *Genetic programming 3: Darwinian invention and problem solving*. Morgan Kaufman, 1999.

[26] H. Shibata, S. Mori, and N. Fujii (2002), "Automated design of analog circuits using cell-based structure," in *Proc. Nasa/DoD Conf. Evolv. Hardware*, 2002.

[27] T. Sripramong and C. Toumazou, "The invention of CMOS amplifiers using genetic programming and current-flow analysis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 21(11), 2000.

[28] R. Zebulum, M. Pacheco, and M. Vellasco. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. CRC Press, 2002.

[29] J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Publishers, 2003.

[30] S. Ando, M. Ishizuka, and H. Iba, "Evolving analog circuits by variable length chromosomes," in *Advances in evolutionary computing*, A. Ghosh and S. Tsutsui, Eds. New York: Springer, pp. 643–662, 2003.

[31] J. Hu and E. Goodman, "Robust and efficient genetic algorithms with hierarchical niching and sustainable evolutionary computation model," in *Proc. Genetic and Evol. Computing Conf.*, 2004.

[32] T.R. Dastidar, P.P. Chakrabarti, and P. Ray, "A synthesis system for analog circuits based on evolutionary search and topological reuse," in *IEEE Trans. Evol. Comput.*, vol. 9, no. 2, pp. 211–224, 2005.

[33] C. Mattiussi and D. Floreano, "Analog genetic encoding for the evolution of circuits and networks," in *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 596–607, 2007.

[34] A. Das, R. Vemuri, "Topology synthesis of analog circuits based on adaptively generated building blocks," *Proc. Design Autom. Conf.*, 2008.

[35] Y. Sapargaliyev and T.G. Kalganova, "Unconstrained evolution of analogue computational 'QR' circuit with oscillating length representation," in G.S. Hornby et al., eds, *Proc. Intern. Conf. Evolv. Syst.*, 2008.

[36] T. McConaghy and G.G.E. Gielen, "Genetic programming in industrial analog CAD: Applications and challenges," in *Genetic Programming Theory and Practice III*, T. Yu, R.L. Riolo, and B. Worzel, Eds., ch. 19, pp. 291–306. Springer, 2005.

[37] X. Wang and L. Hedrich, "An approach to topology synthesis of analog circuits using hierarchical blocks and symbolic analysis," in *Proc. Asia South Pac. Design Autom. Conf.*, 2006.

[38] B. De Smedt, and G.G.E. Gielen, "WATSON: Design space boundary exploration and model generation for analog and RF/IC design" *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 22(2), 2003.

[39] T. Eeckelaert, R. Schoofs, G.G.E. Gielen, and M. Steyaert, "An efficient methodology for hierarchical synthesis of mixed-signal systems with fully integrated building block topology selection," in *Proc. Design Autom. and Test Europe Conf.*, pp. 81–86, 2007.

[40] B. De Smedt, and G.G.E. Gielen, "HOLMES: Capturing the yield-optimized design space boundaries of analog and RF integrated circuits," in *Proc. Design Autom. and Test Europe Conf*, 2003.

[41] S.K. Tiwary, P.K. Tiwary, R.A. Rutenbar, "Generation of yield-aware Pareto surfaces for hierarchical circuit design space exploration," in *Proc. Design Autom. Conf.*, pp. 31–36, 2006.

[42] H.E. Graeb, *Analog design centering and sizing.* Springer, ISBN-10: 1402060033, 2007.

[43] G. Yu, P. Li, "Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling," in *Proc. Intern. Conf. Comput. Aided Design*, pp. 464–469, 2007.

[44] G.S. Hornby, "ALPS: The age-layered population structure for reducing the problem of premature convergence," in M. Keijzer, et al., Eds., *Proc. Conf. Genetic and Evol. Comput.*, vol. 1, pp. 815–822, 2006.

[45] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[46] P. Drennan and C. McAndrew, "A comprehensive MOSFET mismatch model," in *Proc. Intl. Electron Dev. Meeting*, 1999.

[47] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert, "Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies," in *Proc. Design Autom. Conf. (DAC)*, 2007.

[48] P. Palmers, T. McConaghy, M. Steyaert, and G.G.E. Gielen, "Massively multi-topology sizing of analog integrated circuits," in *Proc. Design Autom. and Test in Europe Conf.*, 2009.

[49] W.M.C. Sansen. *Analog design essentials*. Springer, 2006.

[50] F. Rothlauf. *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2^{nd} edition. ISBN 3-540-25059, 2006.

[51] Python language. *http://www.python.org*, last accessed April 23, 2008.

[52] B. Razavi. *Design of Analog CMOS Integrated Circuits*. McGraw-Hill, ISBN: 00711883982000.

[53] F. Leyn, G.G.E. Gielen, and W.M.C. Sansen, "An efficient dc root solving algorithm with guaranteed convergence for analog integrated CMOS circuits," in *Proc. Intern. Conf. Comput. Aided Design*, pp. 304–307, 1998.
[54] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *Proc. IEEE Congress Evol. Comput.*, pp. 1769–1776, 2005.
[55] H. Chang et al., *A Top Down, Constraint Driven Design Methodology for Analog Integrated Circuits*. Kluwer. ISBN: 0792397940, 1997.
[56] G.G.E. Gielen, T. McConaghy, and T. Eeckelaert, "Performance space modeling for hierarchical synthesis of analog circuits," in *Proc. Des. Autom. Conf.*, pp. 1070–1075, 2005.
[57] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," in *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, pp. 82–102, July 1999.
[58] T. Massier, H. Graeb, and U. Schlichtmann. "The sizing rules method for CMOS and bipolar analog integrated circuit synthesis," in *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 12, pp. 2209–2222, Dec. 2008.
[59] J. Nocedal, S. Wright, *Numerical optimization*. Springer. ISBN: 0387987932, 1999.
[60] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert, "Automated extraction of expert knowledge in analog topology selection and sizing," in *Proc. Intern. Conf. Comput. Aided Design*, San Jose, November 2008.
[61] J. Loeckx, T. Deman, T. McConaghy, and G.G.E. Gielen, "A novel EMI-immune current mirror topology obtained by genetic evolution," in *Proc. Conf. in Electro Magnetic Compatibility*, Zurich, 2009.
[62] P. Gao, T. McConaghy, and G.G.E. Gielen, "ISCLEs: Importance sampled circuit learning ensembles for robust analog IC design," in *Proc. Intern. Conf. Comput. Aided Design*, San Jose, Nov. 2008.
[63] J.H. Friedman, "Stochastic gradient boosting," *Journ. Comput. Stats. & Data Analysis*, 38(4), pp. 367–378, 2002.

Trent McConaghy (S'95-M'99) is co-founder and Chief Scientific Officer of Solido Design Automation Inc. He was a co-founder and Chief Scientist of Analog Design Automation Inc., which was acquired by Synopsys Inc. in 2004. Prior to that, he did research for the Canadian Department of National Defense. He received his PhD degree in Electrical Engineering from the Katholieke Universiteit Leuven, Belgium, in 2008. He received a Bachelor's in Engineering (with great distinction), and a Bachelor's in Computer Science (with great distinction), both from the University of Saskatchewan, Canada, in 1999. He has about 40 peer-reviewed technical papers and patents granted / pending. He has given invited talks / tutorials at many labs, universities, and conferences such as JPL, MIT, ICCAD, and DAC. He is regularly a technical program committee member and reviewer in both the CAD and intelligent systems fields, such as IEEE Trans. CAD, ACM TODAES, Electronics Letters, to the Journal of Genetic Programming and Evolvable Machines, GPTP, GECCO, ICES, etc. His research interest is in statistical machine learning / intelligent systems, with transistor-level CAD applications such as variation-aware design, analog topology design, automated sizing, knowledge extraction, and symbolic modeling.

Pieter Palmers (S'04-M'09) was born in 1980 in Leuven, Belgium. He received his masters degree in electronic engineering in 2003 at the Katholieke Universiteit Leuven, Leuven, Belgium. He recently completed his PhD at ESAT-MICAS, Katholieke Universiteit Leuven, Belgium, and is now at Mephisto Design Automation, Leuven, Belgium. His main research interests are in the field of high speed data converter design and analog design automation.

Michiel Steyaert (SM'92-F'04) was born in Aalst, Belgium, in 1959. He received the masters degree in electrical-mechanical engineering and the Ph.D. degree in electronics from the Katholieke Universiteit Leuven (K.U.Leuven), Heverlee, Belgium in 1983 and 1987, respectively. From 1983 to 1986 he obtained an IWNOL fellowship (Belgian National Fundation for Industrial Research) which allowed him to work as a Research Assistant at the Laboratory ESAT at K.U.Leuven. In 1987 he was responsible for several industrial projects in the field of analog micropower circuits at the Laboratory ESAT as an IWONL Project Researcher. In 1988 he was a Visiting Assistant Professor at the University of California, Los Angeles. In 1989 he was appointed by the National Fund of Scientific Research (Belgium) as Research Associate, in 1992 as a Senior Research Associate and in 1996 as a Research Director at the Laboratory ESAT, K.U.Leuven. Between 1989 and 1996 he was also a part-time Associate Professor. He is now a Full Professor at the K.U.Leuven. His current research interests are in high-performance and high-frequency analog integrated circuits for telecommunication systems and analog signal processing. Prof.Steyaert received the 1990 and 2001 European Solid-State Circuits Conference Best Paper Award. He received the 1991 and the 2000 NFWO Alcatel-Bell-Telephone award for innovative work in integrated circuits for telecommunications. Prof.Steyaert received the 1995 and 1997 IEEE-ISSCC Evening Session Award, the 1999 IEEE Circuit and Systems Society Guillemin-Cauer Award and is currently an IEEE-Fellow.

Georges G.E. Gielen (S'87-M'92-SM-'99-F'02) received the MSc and PhD degrees in Electrical Engineering from the Katholieke Universiteit Leuven, Belgium, in 1986 and 1990, respectively. He currently is a Full Professor at the Katholieke Universiteit Leuven. His research interests are in the design of analog and mixed-signal integrated circuits, and especially in analog and mixed-signal CAD tools and design automation (modeling, simulation and symbolic analysis, analog synthesis, analog layout generation, analog and mixed-signal testing). He is coordinator or partner of several (industrial) research projects in this area, including several European projects (EU, MEDEA, ESA). He has authored or coauthored five books and more than 300 papers in edited books, international journals and conference proceedings. He regularly is a member of the Program Committees of international conferences (DAC, ICCAD, ISCAS, DATE, CICC...), and served as General Chair of the DATE conference in 2006 and of the International Conference on Computer-Aided Design in 2007. He serves regularly as member of editorial boards of international journals (IEEE Transactions on Circuits and Systems, Springer international journal on Analog Integrated Circuits and Signal Processing, Elsevier Integration). He received the 1995 Best Paper Award in the John Wiley international journal on Circuit Theory and Applications, and was the 1997 Laureate of the Belgian Royal Academy on Sciences, Literature and Arts in the discipline of Engineering. He received the 2000 Alcatel Award from the Belgian National Fund of Scientific Research for his innovative research in telecommunications, and won the DATE 2004 Best Paper Award. He is a Fellow of the IEEE, served as elected member of the Board of Governors of the IEEE Circuits And Systems (CAS) society and as chairman of the IEEE Benelux CAS chapter. He served as the President of the IEEE Circuits And Systems (CAS) Society in 2005. He was elected DATE Fellow in 2007, and received the IEEE Computer Society Outstanding Contribution Award and the IEEE Circuits and Systems Society Meritorious Service Award in 2007.