Chapter 10

GENETIC PROGRAMMING WITH REUSE OF KNOWN DESIGNS FOR INDUSTRIALLY SCALABLE, NOVEL CIRCUIT DESIGN

Trent McConaghy[1], Pieter Palmers[1], Georges Gielen[1], and Michiel Steyaert [1]

[1]*Katholieke Universiteit Leuven, Leuven, Belgium*

Abstract This paper shows how aggressive reuse of known designs brings orders-of-magnitude reduction in computational effort, and simultaneously resolves trust issues for synthesized designs, for genetic programming applied to automated structural design. Furthermore, it uses trustworthiness tradeoffs to handle addition of novelty in a trackable fashion. It uses a multi-objective algorithm with an age-layered population structure to avoid premature convergence. While the application here is analog circuit design , the methodology is general enough for many other problem domains.

Keywords: synthesis, industrial, analog, integrated circuits, CAD

1. Introduction

Background: GP for Automated Structural Design

A core reason that genetic programming (GP) (Koza, 1992) is interesting is its natural ability to handle search spaces with tree-like and graph-like structures (topologies), which makes it a natural fit for automated invention of structures. One focus has been design of analog circuit topologies, such as those in (Koza et al., 1999; Koza et al., 2003a; Koza et al., 2004; Hu and Goodman, 2004; Lohn and Colombano, 1998; Shibata et al., 2002; Sripramong and Toumazou, 2002; Dastidar and Chakrabarti, 2005). In this domain, GP has evolved several patent-quality circuits (Koza et al., 2003a) essentially "from scratch", which is a remarkable success by almost any measure. It is an especially notable accomplishment from an artificial intelligence perspective because "patent-worthiness" is a good measure of success for testing techniques in automated "creative" design.

GP has been used for structural design in other fields as well: in (Lohn et al., 2004), Hornby and Lohn evolved an antenna design for NASA which was successfully deployed in space. In several works including (Spector, 2004), Spector has evolved quantum circuits. Several groups have used GP as a means to suggest a "design" in the form of a mathematical equation. These designs get manually filtered and tweaked, then deployed in the field, such as: chemical sensors (Castillo et al., 2004), geological exploration (Yu et al., 2006), and financial markets (Becker et al., 2006; Korns, 2006). Unlike the other domains mentioned, GP for circuit design has never been deployed in industry.

GP has not been deployed for circuit design in industry because (a) new designs cost millions of dollars to fabricate and test, and (b) GP-synthesized designs so far have not had the combination of sufficient complexity and trustworthiness to make the cost worth it. If the design fails, then there is not only a new fabrication needed for the revised design ("re-spin"), there is lost time to market. A new analog topology has higher chance of failure due to lack of experience with that topology; it is risky coming from an experienced designer and even more risky coming from an untrusted black box. New topologies only come about if there is no other way – if idea has possible orders of magnitude payoff that it's worth the money to try, or if there is some way to make trying it zero risk. It gets worse: addressing even just robustness (a subset of the trustworthiness issue) on a sufficiently complex problem would take 150 years on a thousand-CPU 1-GHz cluster; faster CPUs with Moore's Law (ITRS, 2007) can't help because the problem becomes more difficult as Moore's Law progresses (McConaghy and Gielen, 2005). Aerospace design has similar resistance to new structural ideas, except there if the new design fails it means that the plane or rocket crashes. Is there a path out?

Background: The Power of Domain Knowledge

Domain knowledge, if applied in the right places, can bring about orders of magnitude reduction in size of the search space, improvement in runtime, or improvement in quality of results. If we are interested in industrial applications then speed and quality of results are of utmost importance, and embedding domain knowledge can be well worth it. Domain knowledge can be applied at multiple levels of generality. We now give some illustrative examples from both evolutionary computation (EC) and other fields. In EC, each of these brought one or more orders of magnitude speedup or improvement in result quality: generative representations and modularity in general, e.g. (Hornby, 2003); permutation design via floating point representations (Rothlauf, 2006); avoiding "danglers" in circuit topology design e.g. (Koza et al., 2003a), machine-code symbolic regression (Nordin, 1994), machine-code digital logic design (Poli and Langdon, 1999), avoiding the need for learning the linear weights in symbolic regression (Keijzer, 2004); thorough exploration of smaller building blocks, e.g. one variable at a time in symbolic regression (Korns, 2006); and more. It has been shown that GP can learn about the structure of the domain in one run to help subsequent runs (Keijzer, 2005). Some interesting examples outside of EC include: in splines, 10x-1000x or more speedup in regression by iteratively updating the least-squares learning matrix rather than doing a full update (Friedman, 1991); 1,000,000x speedup when building behavioral models of circuits, by using knowledge of its connectivity (Phillips, 1998), 1000x by exploiting sparsity in matrices (Lai and Roychowdhury, 2006); 100x space reduction via cheap-to-compute "device operating constraints" in circuits (Ding and Vemuri, 2005), 1,000,000x space reduction by reformulating the independent design variables of a design problem to more natural variables (Bernardinis et al., 2005); and more. For non-trivial practical applications, domain knowledge is key.

Reuse of Structural Domain Knowledge

In (Koza et al., 2003b), Koza et. al note: "Anyone who has ever looked at a blueprint for a building, an electrical circuit, a corporate organizational chart, a musical score, a city map, or a computer program will be struck by the ubiquitous reuse of certain basic substructures within the overall structure...Reuse can accelerate automated learning by avoiding 'reinventing the wheel' on each occasion requiring a particular sequence of already-learned steps. We believe that reuse is the cornerstone of meaningful machine intelligence." All scientific and engineering fields accumulate knowledge of useful structures over time; added new structures are literally advances in the field. For mathematics, this includes new theorems and proofs; for computer science, algorithms; for software engineering, design patterns, and libraries of code; for biology, new theories and models; for analog circuit design, taken to mean new circuit topologies.

Interestingly, "reuse" in GP systems has been reuse of structures that were found by GP during the run, or in a previous run, and not reuse of structural domain knowledge. For automotive design, GP would literally have to reinvent the wheel–and the piston, crankshaft, transmission, etc. Issues which emerge are: reinvention takes a huge amount of computational effort, if it is even tractable at all; there is no guarantee that the functionality will be hit; and because GP does not distinguish the known structures from novel structures, final designs can look very odd and therefore are less trusted.

This paper shows how reuse of structural domain knowledge simultaneously solves the GP issues of computational efficiency and of trust, for those problems which have a sufficient amount of accumulated structural domain knowledge. Figure 10-1 illustrates the general approach to such problems. We demonstrate the approach to analog circuit design, which has accumulated a large amount of structural knowledge over the decades (Razavi, 2000; Sansen, 2006).

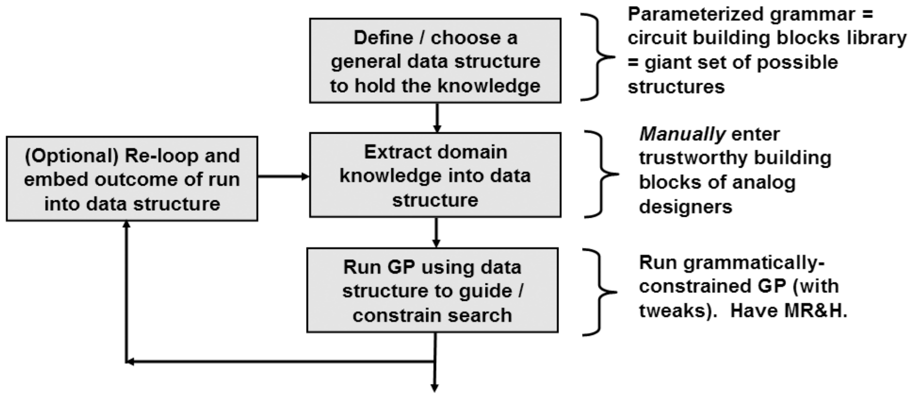
Define / choose a general data structure to hold the knowledge
Parameterized grammar = circuit building blocks library = giant set of possible structures
(Optional) Re-loop and embed outcome of run into data structure
Extract domain knowledge into data structure
Manually enter trustworthy building blocks of analog designers
Run GP using data structure to guide / constrain search
Run grammatically-constrained GP (with tweaks). Have MR&H.

Figure 10-1. A general framework to leverage domain-specific structural knowledge with GP. Our instantiation of the framework for analog circuit design is described with the text on the right.

A Path to Practical Automated Structural Design

We now discuss various approaches to design of structures ("topologies"). The status quo approach to GP for structural design is shown in Figure **??** left. Figure 10-2 middle gives a flow that focuses on optimizing a fixed structure (what circuit designers currently do).

We specify our goals for a structural (topology) design tool. If a topology that is known to be 100% trustworthy will meet design goals, then the tool should return that topology. It should strive to keep the inputs and outputs as close as possible to existing techniques. It should draw on as much prior structural design knowledge as possible, so long as that knowledge is convenient to the user, it doesn't have to be convenient to the tool developer. Only if no existing known topology can meet its goals should the tool resort to adding novelty–to do so otherwise would introduce unnecessary risk. If it does add novelty, it should be easy to track where and how that novelty is added, and what the payoff is.

We now classify "automated topology design" into the following sub-categories, and discuss which of them a designer would want:

1. **Lightweight multi-topology sizing:** Search across predefined, 100% trusted topology space, but the topologies have to be input by designers. The trustworthiness is useful because it means that there is less reliance

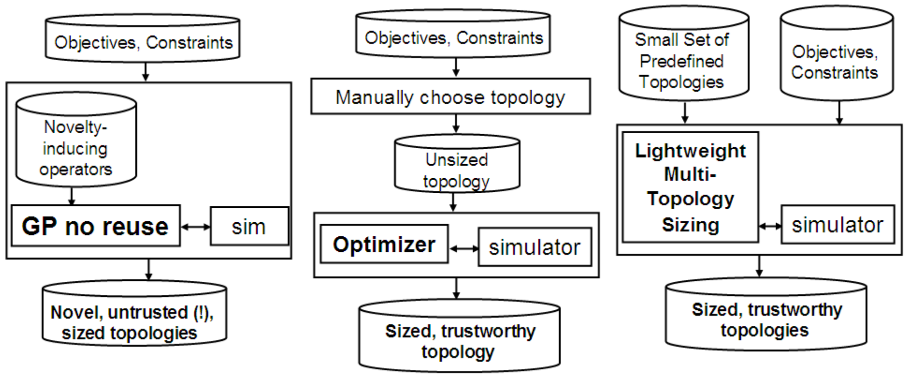
Objectives, Constraints
Novelty-inducing operators
GP no reuse
sim
Novel, untrusted (!), sized topologies
Objectives, Constraints
Manually choose topology
Unsized topology
Optimizer
simulator
Sized, trustworthy topology
Small Set of Predefined Topologies
Objectives, Constraints
Lightweight Multi-Topology Sizing
simulator
Sized, trustworthy topologies

Figure 10-2. Current approaches to get sized topologies. Left: Status quo GP flow having no structural reuse – painful because topologies are untrustworthy, and huge computational burden. Middle: Current industrial flow using optimization (sizing) – painful because topology selection is manual. Right: Earlier approaches to multi-topology sizing – painful because the libraries are small and inflexible, and therefore required designer setup and intervention.

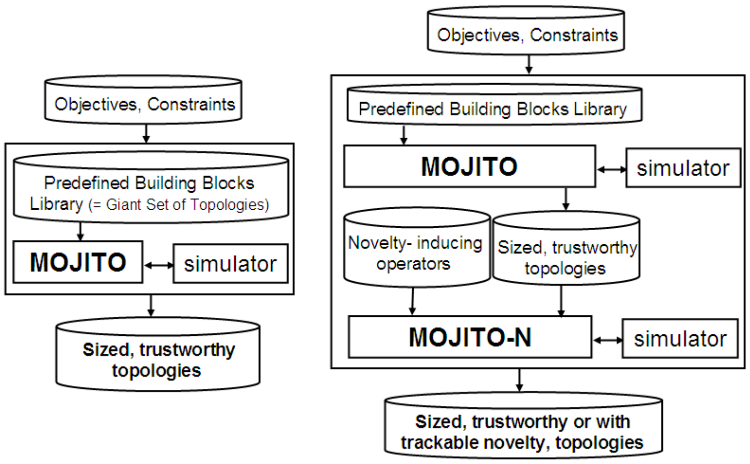
Objectives, Constraints
Predefined Building Blocks Library (= Giant Set of Topologies)
MOJITO
simulator
Sized, trustworthy topologies
Objectives, Constraints
Predefined Building Blocks Library
MOJITO
simulator
Novelty- inducing operators
Sized, trustworthy topologies
MOJITO-N
simulator
Sized, trustworthy or with trackable novelty, topologies

Figure 10-3. Proposed approaches to get sized topologies. Left: MOJITO: Multi-topology sizing – specs-in, sized-circuit out; gives 100% trustworthy results, but not novel designs. Right: MOJITO-N: Multi-topology sizing with novelty – gives trustworthy results *and* designs with measurable novelty.

on detailed measures to guarantee robustness and manufacturability, but it is too tedious to expect a designer to enter more than a few topologies. Even if the topology space is parameterized, it is hard to get beyond a few dozen possible topologies. Figure 10-2 right, illustrates this.

2. **Multi-topology sizing:** Search across predefined, 100% trusted topology space, where the number of topologies is sufficiently rich that the designer can consider it "complete enough" to not have to intervene in a typical design problem (i.e. hidden from view from the perspective of the designer). This is of great interest to them, because it means that it uses the same inputs and outputs as their existing tools, yet they don't have to take the time to select a topology. It is simply "specs in, sized topology out". Interestingly, if one does a (long) multi-topology sizing run with a huge number of goals set as objectives, the result itself is effectively a library of sized results; future queries for sized topologies of certain specifications are a computationally cheap lookup; i.e. it is "specs in, sized topology out, *immediately*." Figure 10-3, left, illustrates.

3. **Multi-topology sizing with innovation:** Search across 100% trusted topology space, and add novelty if there is a performance payoff. That is, "innovate" as needed. This would be of great interest for designers who are searching for new design ideas, if that is what is truly desired or needed. It is especially useful if there is a mechanism to track novelty, and therefore assess how much trust designers have in the design. Figure 10-3, right, illustrates.

4. **Topology invention from scratch:** No structural information is input (status quo GP). That is everything is "invented" (or reinvented) from scratch. Designers would question why this would ever be needed, if (3) exists. After all, why ever reinvent known structures? And they have no idea where the novelty may lie; it may be near-impossible to untangle the circuit to understand it. If they wanted extreme novelty, they would just let (3) run longer. Incidentally, because such a methodology would require a tedious iterative looping of plugging "holes in goals" for each new problem, that makes it more "hands-on" than an approach that has structural reuse. Figure 10-2, left, illustrates.

In this paper, we demonstrate how GP can be used to build the industrially interesting categories (2) and (3). The key to (2) is to aggressively reuse existing structural knowledge. The key to (3) is trustworthiness tradeoffs to ensure that only novel designs that actually give a payoff are rewarded. One might be concerned that the problems (2) and (3) are trivially easy compared to (4). Our responses are that (4) is pointlessly hard, and that one should strive to "trivialize a problem" as much as possible to help ensure its use. And we will see that problems (2) and (3) are challenging in their own right, by no means trivial to solve effectively.

2. MOJITO for Multi-Topology Sizing

MOJITO is a system for multi-objective and topology sizing. Its flow from a user perspective is shown in Figure 3 left (the diagram on the right is for novelty, described later). It actually follows a generally applicable framework for GP in structural design, as Figure 1 describes. This section describes the instantiation of the framework, specifically: how the library of structural design knowledge is defined, the GP search algorithm, and experimental results. Note: some parts of this section were originally reported in (McConaghy et al., 2007).

Background on Multi-Topology Sizing

This section reviews other approaches to multi-topology sizing in the literature. Multi-topology sizing is not a new idea, but it has never been applied to giant topology spaces, nor with SPICE in the loop (both of which greatly increase the difficulty of the problem). The work typically comes out of the analog CAD field (as opposed to an EA field). BLADES (E1-Turky and Nordin, 1986), OASYS (Harjani et al., 1992), and others (Berkcan et al., 1988; Koh et al., 1990; Toumazou et al., 1990; Swings et al., 1991; Ning et al., 1991; Antao and Brodersen, 1995; Kampe, 2000; Doboli and Vemuri, 2003; Martens and Gielen, 2006) depend on rule-based reasoning or abstract models having transforms to structural descriptions, and therefore have an undesirable amount of up-front setup effort. DARWIN (Kruiskamp and Leenaerts, 1995) and others (Maulik et al., 1995; Tang and Doboli, 2006) only require structural information, but rely on a sneaky definition of a flat combinatorial search space to define possible topologies; they do not show a clear path to generalize and are restricted to a few hundred topologies at most.

MOJITO Inputs and Outputs

The core philosophy is to use the inputs and outputs that are acceptable for industrial single-topology multi-objective sizing tools, such as (Synopsys, 2007); but to add the smallest possible amount of extra information in order to enable multi-topology sizing. Instead of a single topology, the tool takes in a set of hierarchically organized building blocks. Just like a single topology, these building blocks can be specified in an industrial circuit schematic editor. Getting such inputs is not unreasonable: such blocks do not appear as anything special to the designer, as they are merely based on well-known building blocks that one can find in any analog design textbook (Razavi, 2000; Sansen, 2006). And in fact, since we have already designed an example library (see following sections), the designers can use that. This makes it is straightforward to switch technologies, or add new building blocks.

MOJITO uses off-the-shelf simulators (e.g. SPICE) rather than specially designed performance estimators. Its output is a tradeoff of sized circuits, for selection by a designer or within a hierarchical methodology like MOBU (Eeckelaert et al., 2007). MOJITO can be seen as a pragmatic fusion of knowledge-based and optimization-based analog CAD (Rutenbar et al., 2002).

Search Space Framework

This section describes a topology space that is specified by structural information only, searchable, trustworthy, and flexible. Its flexibility is due to an intrinsically hierarchical nature which includes parameter mappings; the parameter mappings can choose sub-block implementations. It could be summarized as a parameterized grammar with a generative-representation twist.

Creating a representation for circuits is a design challenge in its own right. We choose to adopt a strongly hierarchical approach, because a flat representation is not conducive to the construction of a library or to larger designs. Analog circuit hierarchies analog be represented by analog hardware description languages (HDLs) (Ashenden et al., 2002; Kundert and Zinke, 2004), analog circuit database representations, even grammars (Ressler, 1984; Tanaka, 1993). With these options already existing in the analog domain, why not just use one of them? The problem is that if a designer makes a small conceptual change to a circuit that corresponds to a small change in performance, there may be a drastic change in the netlist. While this complicates the design of an appropriate search representation, it is needed for changes like folding an input or flipping all NMOS transistors to PMOS. Myriad examples can be found in any analog design textbook. The structural-only op amp approaches (Kruiskamp and Leenaerts, 1995; Maulik et al., 1995) do cover some of these examples, but are designed into a flat space, need special heuristics just to work in their small spaces, and do not readily generalize. The existing grammatical approaches did not provide enough flexibility.

The generative representation GENRE (Hornby, 2003) provided inspiration. A generative representation transforms a genotype to phenotype by executing the genotype commands as if they were a program. Unfortunately, GENRE does not readily allow one to embed known trusted building blocks, and is too flexible in allowing the addition and removal of ports on substructures during search. The MOJITO representation removes some flexibility in order to allow easier embedding of domain knowledge; it has an associated drawing style that both analog designers and computer scientists will understand. It is composed of three simply-defined “Part” types, which we now describe.

Let us define a “Part” as merely a circuit block at any level of the hierarchy. It has a fixed set of arguments in its interface: “port arguments” (nodes available to the outside world) and "number arguments" (parameters which affect its behavior, e.g. a device size). Arguments to a Part's embedded Parts are a function of arguments above. To fully netlist a given Part, the only extra information needed is values for the arguments to that Part. Direct-representation Part types are:

- **Atomic Part Type.** Parts of this type are the leaf nodes in the hierarchy (tree) of Parts. They do not contain any embedded parts. Figure 10-4 gives examples.

- **Compound Part Type.** These have one or more sub-Parts embedded. Sub-parts can have internal connections among themselves and to the Part's external ports. All sub-parts get netlisted. Figure 10-5 gives examples.

We add the following generative Part type. It netlists by executing the Part as a function of a third type of argument in its interface: "topological arguments":

- **Flexible Part Type.** These have the topological argument "choice_index", which during netlisting is used to select one of several candidate embedded parts and respective wirings. The argument values going into the chosen sub-part can be very particular to that sub-part if the mapping function has cases for different choice_index values. Example: a current mirror which may be simple or cascode (choice_index = 0 or 1). Figure 6 gives an example.

Despite the simplicity of Part types, the interactions allow a capture of essential structural domain knowledge of analog building blocks. The generative Parts, especially Flexible Parts, are what turn a Part into its own IC *library of possibilities* rather than merely a representation of a single circuit design. Traversing the topology space merely means changing one or more of the "topological argument" input values.

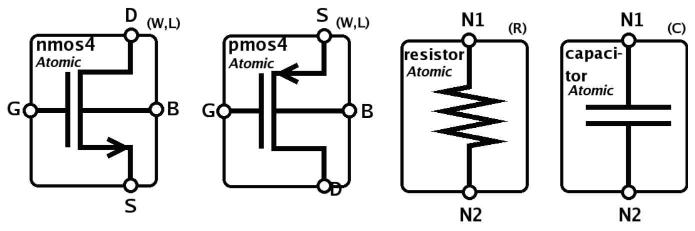

Figure 10-4. Example Atomic Parts: nmos4 transistor, pmos4 transistor, resistor, capacitor.

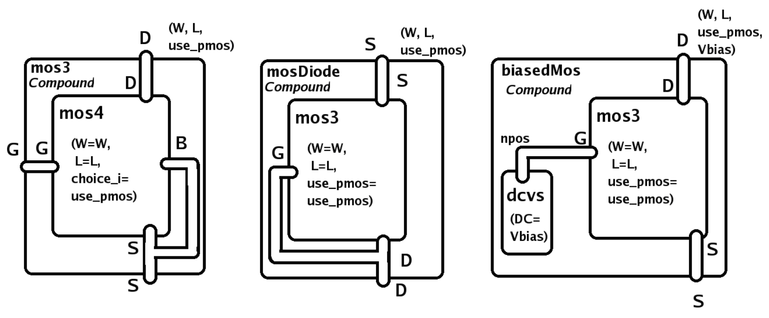

Figure 10-5. Example Compound parts. mos3 is a wrapper for mos4, so that the mos4's 'B' node is not seen at higher levels. mosDiode ties together two internal ports to only present two external ports. biasedMos uses a 1-port dcvs (dc-controlled voltage source) part to set its gate bias internally.

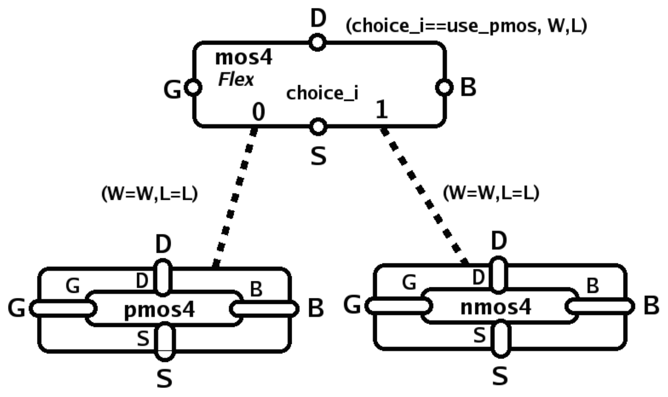

Figure 10-6. Example Flex part: mos4 turns the choice of NMOS vs. PMOS into a parameter "*choice_index*". Note how parameters get assigned from mos4 to either of its sub-blocks. In this case both sub-blocks use the mos4's W and L parameters as their own W and L values.

Remember that for all these subblocks, instantiation into sets of nmos vs. pmos devices is deferred until the very leaf block, based on the parameters that flow through the circuit. This sort of 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 keeps the overall variable count lower than it might have been; this is crucial to the locality of the space and thus the ultimate success of the search algorithm. Figure 7 gives an example of a circuit "sentence" instantiated in the parameterized grammar of MOJITO; this sentence will also be a GP individual tree.

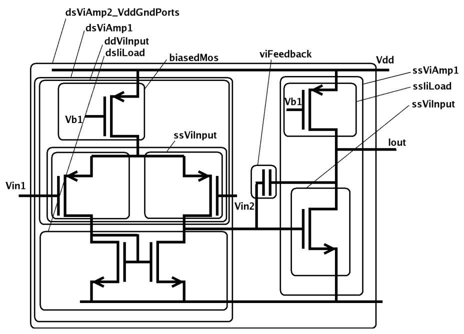

Figure 10-7. Example of MOJITO Building Blocks on a PMOS-input Miller OTA.

3. MOJITO Search Algorithm

We now proceed to describe an algorithm that traverses the MOJITO search space to produce a set of topologies that collectively trade off performances.

The search space is gigantic and diverse, because there can be thousands of possible topologies plus associated sizings. This means a mix of hierarchy and parameters which can be continuous-, discrete-, or integer-valued. SPICE-accurate performance estimation adds computational demand too, compared to the simplified performance estimators that most previous multi-topology sizing approaches used.

- An evolutionary search algorithm that balances exploration with exploitation by grouping individuals by genetic age (ALPS (Hornby, 2006)), and at a nested level achieves multiobjective search by grouping individuals by degree of nondomination (NSGA-II (Deb et al., 2002)).

- Special operators that are designed to exploit the nature of the search space. The crossover operator respects the parameters that should be held together within building blocks, yet still allows sibling building blocks to share parameters (i.e. a mix between vector and tree search spaces). The mutation operator has tactics to avoid stealth mutations (Rothlauf, 2006) on "turned-off" building blocks.

Structure of Search Space from Search Algorithm's Perspective

Each building block has its own parameters, which fully describe how to implement it 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. Thus, 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 evolutionary algorithm (EA) operating on vectors. But a vector-oriented view does not recognize the hierarchy, and so operations on it may have 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. From the perspective of a search algorithm, this means that there are vast regions of neutrality (Huynen et al., 1996); or alternatively the representation is non-uniformly redundant and runs the risk of stealth mutations (Rothlauf, 2006). For EAs, 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. From an EA perspective this means that the "building block mixing" is poor (Goldberg, 2002).

What if we reconcile the hierarchy? We cannot apply a hierarchical design methodology such as (Chang, 1997; Eeckelaert et al., 2005), because there are no goals on the sub-blocks, just the highest-level blocks (we could, however, still apply hierarchal methodology to the results). Neither can we treat it completely as a tree induction problem (to be solved, for example, by grammar-based genetic programming (Whigham, 1995)) because some sibling sub-blocks share the same parent blocks' parameters.

So the search algorithm's perspective of the space has both tree-based and vector-based aspects. We design novel operators that reconcile both aspects, for use within an EA. First, we have a mutation operator which chooses one or more parameters to mutate. Continuous-valued parameters follow Cauchy mutation (Yao et al., 1999) which allows for both tuning and exploration. Integer-valued "part choice" parameters follow a discrete uniform distribution. Other integer and discrete parameters follow discretized Cauchy mutations. To avoid stealth mutations on "turned-off" building blocks, mutations are only kept if the netlist changes; mutation attempts are repeated until this happens. Though "neutral wanderings" of the space has been shown to help exploration in some applications (Vassilev and Miller, 2000; McConaghy et al., 2005), results are mixed and in general make performance more unpredictable (Rothlauf, 2006). We prefer predictability, and rely on ALPS to enhance exploration.

The second operator is crossover. It works as follows: given two parent individuals, randomly choose a sub-block in parent A, identify all the parameters associated with that sub-block, and swap those parameters between parent A and parent B. This will preserve the parameters in the sub-blocks. There will still be some crosstalk because sibling blocks may use those parameters as well, but the crosstalk is relatively small compared to the 100% crosstalk that we'd have if we used standard vector-based crossover. This effectively makes the search a hybrid between tree-based and string-based search (i.e. a cross between a GA and GP).

To generate random individuals, we merely randomly choose a value for each parameter using a uniform distribution.

The Search Algorithm

Even with a search space and operators that are as well-behaved as possible, there is a need for a highly competent search algorithm because the space is so large (there is such a large set of possible topologies and associated sizings), and the performance estimation time for an individual can be on the order of minutes (using SPICE to maintain industrial relevance). We also need multi-objective results. The blow is softened a bit because some degree of parallel computing is allowed (industrial setups for automated sizing typically have 5-30 CPUs).

A popular, competent EA for multiobjective is NSGA-II (Deb et al., 2002), which sorts individuals by nondomination layer. NSGA-II provides a reasonable starting point for us in the design of our multiobjective EA. We use the constraint-handling approach of NSGA-II as well: a feasible individual always dominates an infeasible one, and for two infeasible individuals the one that dominates is the one with the least total constraint violation (a sum across all constraints' violations).

One key issue with NSGA-II, and most EAs, is that they can converge prematurely. To fix this, one needs to ensure an adequate supply of building blocks (Goldberg, 2002). Tactics include massive population sizes (Koza et al., 2003a), restarting, time-varying population sizes, or diversity measures such as crowding. All tactics are all either inadequate or highly sensitive to parameter settings. Random injection of individuals for fresh new building blocks might help, except they get killed off too quickly during selection. To fix that, HFC (Hu et al., 2005) segregates individuals into similar fitness layers, and restricts competition to within layers, which gives random individuals a reasonable chance. Unfortunately, the choice of fitness thresholds is complicated in practice, and near-stagnation may occur at some fitness levels because the best individuals per level have no competition. The age-layered population structure, ALPS (Hornby, 2006) builds on HFC, but rather than segregate individuals by fitness it segregates by genetic age levels. The age distinction overcomes the issues of HFC. For example, age level 0 might allow individuals with age 0-19, level 1 allows age 0-39, level 2 allows age 0-59, and so on until the top level (e.g. level 9) which allows individuals of any age. Genetic age is the number of generations of an individual's oldest genetic material: the age of a randomly generated individual is 0; the age of a child is the maximum of its parents' ages; age is incremented by 1 each generation. If an individual gets too old for a fitness level, it gets kicked out of that level and given one last chance to compete at the next higher level. Selection at one age level uses the individuals at that level and at one level below as candidates.

Only a single-objective, single-CPU ALPS exists in the literature. In this paper, we make it multi-objective for the first time. There are many conceivable ways to make ALPS multi-objective. We chose a pragmatic approach which is shown in Figure 10-8. There is canonical NSGA-II evolution at each age level, with one difference: for selection at a level l, the individuals at level l and level $l-1$ are candidates (rather than just at level l). In this fashion, younger high-fitness individuals can propagate to higher levels.

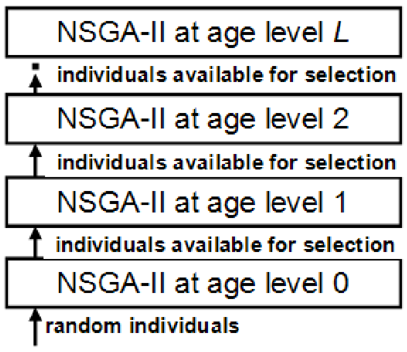

Figure 10-8. Multi-objective ALPS has NSGA-II at each age level.

4. MOJITO Multi-Topology Sizing: Experimental Results

This section describes application of MOJITO to two multi-objective multi-op-amp topology sizing problems.

Problem Setup

The problems were set up as follows. The search space had 50 variables (topology selection variables and sizing variables). EA settings were: 100 individuals per age layer; 10 age layers, maximum age per layer: 9, 19...79, 89, infinity. Each run took approximately 150 hours on a single-core 2.0 GHz Linux machine, covering 100,000 individuals. Search objectives: maximize GBW, minimize power, maximize DC Gain (Experiment Set 2). Constraints: phase margin $> 65°$, all DOCs, DC Gain > 30dB (Experiment Set 1). Simulator was HSPICE. Technology was 0.18μ CMOS; supply voltage 1.8V; load capacitance 1pF.

Experiment Set 1

These runs were to verify the algorithm's ability to traverse the search space and select different topologies. The problem was set up such that the optimization end result was known a priori. Three GP runs were done, with problem setups such that specific output topologies were expected. To summarize results for non-circuit people: it achieved the structures which were expected. The rest of this paragraph gives circuit-specific details. The only difference between the 3 runs is the common mode voltage $(V_{cmm,in})$ at the input. We know that for $V_{cmm,in}$ = 1.5V, topologies must have an NMOS input pair. For $V_{cmm,in} = 0.3V$, topologies must have PMOS inputs. At $V_{cmm,in} = 0.9V$, there is no restriction between NMOS and PMOS inputs. Figure 10-4 illustrates the outcome of the experiments. It contains the combined results of three optimization runs. Result (a) has $V_{cmm,in} = 1.5V$, and has only topologies with NMOS inputs. It 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 PMOS input pairs. Note that result (a) is a result before convergence in order to retain the 2-stage amplifier in the result set. Older generations eliminate the 2-stage amplifier in favor of the folded cascode amplifier, as in result (b). For result (c) a $V_{cmm,in} = 0.9V$ has been specified. Though both NMOS and PMOS input pairs might have arisen, the optimization preferred NMOS inputs. The curve clearly shows the switch in topology around GBW=1.9GHz, moving from a folded cascode input to a simple current-mirror amp. Interestingly, the search retained a stacked current-mirror load for about 250MHz GBW.

Experiment Set 2

In second experiment, one GP run was done, to verify that MOJITO could get interesting groups of topologies in a tradeoff of three objectives. The motivation is as follows: whereas a single-objective multi-topology optimization can only return one topology, the more objectives that one has in a multi-topology search, the more opportunity there is for many topologies to be returned, because different topologies naturally lie in different regions of performance space. Results are shown in Figure 5. We can see that MOJITO found rich and diverse structures as expected. The rest of this paragraph has circuit-specific details. It determined: a folded-cascode op amps gave high gain-bandwidth but with high area, 2-stage amps give high gain but at the cost of high area, the low-voltage current mirror load is a 1-stage with high gain, and there are many other 1 stage topologies which give a broad performance tradeoff. These are all results that a circuit designer would expect.

Incidentally, problems of comparative complexity took status quo GP (i.e. no reuse) 100 million or more individuals (Koza et al., 2003a; Koza et al., 2003b), and the results were not trustworthy; it was estimated that to get to

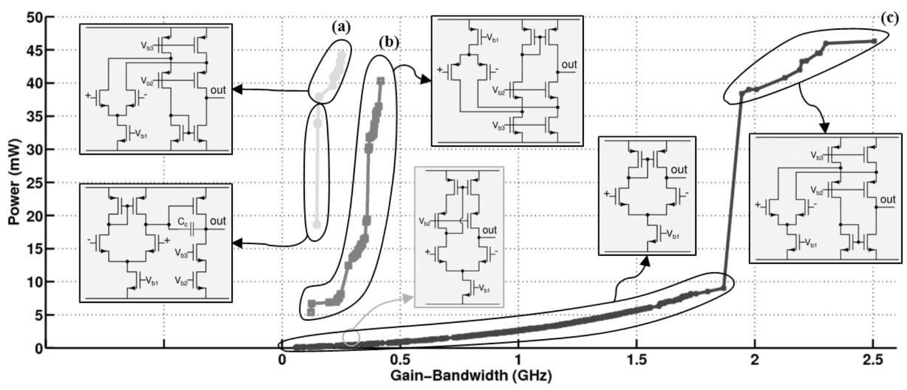

Figure 10-9. Pareto fronts for 3 GP runs a/b/c which had different input settings. The y-axis is an objective to minimize, and the x-axis is an objective to maximize; each point is an individual, which has an associated structure (topology) and parameters (sizings). Some of the specific topologies found are shown; these are the expected topologies.

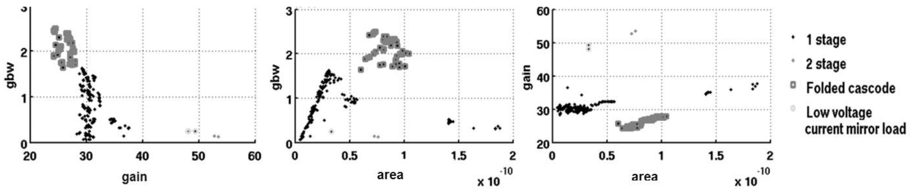

Figure 10-10. Pareto front for a GP run on 3 objectives (maximize gbw, maximize gain, minimize area). Individuals are grouped according to some of their structural characteristics (e.g. 1 stage vs. 2 stage) to illustrate their diversity.

get a reasonable degree of robustness would take 150 years on a 1,000 node 1-Ghz cluster (McConaghy and Gielen, 2005). That is, it would have taken ((150 years * 365 days / year * 24 hours / day) * 1000 CPUS * 1Ghz) / ((150 hours) * 1 CPU * 2 Ghz) = 4.4 million times more computational effort than MOJITO to get comparable results. There's a lot to be said for topology reuse.

5. How Far can Reuse-Only Go? (With No Novelty)

This section describes how huge a fully trustworthy (reuse-only, no novelty) space can become.

The first major question of this subsection is: Can the number of possible topologies be sufficiently rich so that the designer can consider it "complete enough" to not have to intervene in a typical design problem? We calculate the size as follows. 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. Subtlety: one higher-level choice might govern > 1 lower-level choices, so don't overcount. Table 1 shows that MOJITO increases the op amp count by 50x compared to the other reuse-only techniques.

Table 10-1. Size of Op Amp Topology Spaces.

Technique	**# topologies**	**Trustworthy?**
GP without reuse, e.g. (Koza et al., 2003a)	billions	NO
DARWIN (Kruiskamp and Leenaerts, 1995)	24	YES
MINLP (Maulik et al., 1995)	64	YES
GP with reuse: MOJITO (this work)	**3528**	**YES**

The second major question of this subsection is: How big can the space of possible trustworthy topologies for an industrially relevant application get? Compared to what we have just established, we can make the space even larger in many ways, using new techniques, recursion, and system-level design:

- **Add more design techniques.** The field of analog design is a research field in its own right, with its own conferences, journals, etc. Core advances in that field are precisely: new topologies and techniques. One can think of that design effort as (manual) co-evolution of building block topologies. Design opportunities and challenges arise due to new applications, different target specifications, and the steady advance of Moore's Law (ITRS, 2007). Each design technique advance would increase the size of the space by at least 2x, so if we merely took the top 10 advances in op amp design, we would increase the space by at least $2^{10} = 10^3$, bringing the count to $3.5 * 10^6$. And that is a lowball estimate: more realistically one would consider dozens or hundreds of advances, and some advances could be used in multiple places in the design; if we had 10 advances which doubled, 10 which tripled, and 10 which quadrupled, then the space increases by $2 * 10 * 3^{10} * 4^{10} = 6 * 10^{13}$, to total $2 * 10^{17}$ trustworthy op amp designs.

- **Recursion.** Circuits' designs can recurse. For op amps, this is via "gain boosting." One level of recursion brings the count to $(2 * 10^{17})^2 = 4 * 10^{34}$, and two levels of recursion (i.e. gain boosted amps using gain boosted amps) brings the count to $(4 * 10^{34})^2 = 1.6x10^{69}$ trustworthy op amp designs. Yes, designers in industry do actually use two levels of gain boosting, in combination with the latest design techniques.

- **System-level design.** So far we have just talked about an op amp space which is a circuit at lowest level of the design hierarchy (cell level), but higher levels exist too. The next-highest level includes circuits such as data converters (A/Ds, D/As), active filters, and more. These circuits use lower-level blocks like op amps. The level above that is typically the whole analog system, e.g. a radio transceiver like a Bluetooth or Wi-Fi implementation. The level above that would typically combine the analog and digital parts into a mixed-signal system. Each level can have many sub-blocks, and each of those sub-blocks can be any of its combinations. E.g. an A/D might have 8 different op amps. If each op amp had $1.6 * 10^{69}$ possible topologies and even if there was no other topological variation at the A/D level, it means $(1.6 * 10^{69})^8 = 4.2 * 10^{553}$ possible A/D topologies. Let's say the system at one level higher up had an A/D, a D/A, and a couple other parts all with about the same number of topologies; then its size would be $(4.2 * 10^{553})^4 = 3.1x10^{2214}$ possible topologies. (For reference, if just 3528 designs at the cell level, that leads to $((3528)^8)^4 = 10^{113}$ designs).

Combinatorial explosion is a good thing: the more possibilities available for *any* part type, the more possible trustworthy designs you can have. If one can decompose their design into sub-problems (where each sub-problem has its own goals), if one has a competent hierarchical design methodology, if the problem of "massively multi-topology" cell-level sizing design can be cracked, then one can ultimately do system-level 100% trustworthy topology design in spaces with 10^{113} designs, 10^{832} designs, or more.

We *can* do (1) because the decomposition is obvious in circuit design, and the names of sub-blocks are well-established (op amps, bias generators, A/Ds, D/As, filters, phase-locked loops, etc) (Razavi, 2000; Sansen, 2006). We can do (2) because competent hierarchical design methodologies have been demonstrated; and recently it has been demonstrated that they can choose from among different candidate topologies (Eeckelaert et al., 2007). This paper has demonstrated (3).

6. Multi-Topology Sizing with Novelty

Because of the costs of fabricating a design, the motivation for a new topology has to be strong. New topologies only come about if there is no other way, if idea has possible orders of magnitude payoff that it's worth the money to try, or if there is some way to make trying it zero risk. That said, sometimes these motivations exist, and therefore it is of interest to see what sort of effective algorithms can be created. This section describes MOJITO-N, a system for $\underline{\text{m}}$ulti-$\underline{\text{o}}$b$\underline{\text{j}}$ect$\underline{\text{i}}$ve and $\underline{\text{to}}$pology sizing, that adds novelty as needed, with the flow of Figure 10-3, right.

The Search Algorithm

The specifications for such a system, above and beyond (non-novelty) MOJITO, are:

- If a topology that is known to be 100% trustworthy will meet their goals, then the tool should return that.
- Only if no existing known topology can meet their goals should the tool resort to adding novelty.
- If it does add novelty, it should be easy to track where and how that novelty is added, and what the payoff is.

  These specifications are resolved in MOJITO-N as follows:

- Use trustworthy designs as the structural starting points. In fact, do a long 100% trustworthy run first; then add novelty in a follow-on run.
- Create novel designs by: copying an existing part within the parts library, mutating the copy, and then getting a new individual to use that mutated copy. In order to track novelty, remember which parts and choices are novel, and what sort of novelty-mutating operator is used. These altered libraries can be subsequently reused in future runs, therefore closing the loop in the style of run-transferable-libraries (Keijzer, 2005).
- Have a multi-objective framework to manage trustworthiness tradeoffs: trust = $-$novelty, novelty = number of times that a novel part is used, and a novel part is one that has had random structural mutations. Therefore, if novelty does not actually help, it will not show up in the Pareto optimal front (but it will not necessarily be kicked out of the population; that is up to the multiobjective algorithm).
- A novel design will almost certainly be initially worse off than a non-novel design, until it has been sized well enough to be competitive. If not handled explicitly in the EA framework, the novel design will almost certainly die off before its benefit is discovered (if it has a benefit). So that novel designs have a fighting chance, only create novel designs for the easiest-competition age layer 0. Rather than randomly generating the whole individual from a uniform distribution, choose a parent from any age layer, and novelty-mutate it for placement in layer 0. (Note: a plethora of other possible schemes exist here too, but a key enabler is the ALPS structure).

Experiment

The experimental setup was the same as for the non-novelty MOJITO, except for the following differences. The 100 trustworthy results from the MOJITO "Experiment Set 2" run were used as the inputs to the MOJITO-N run. MOJITO-N was run for 15 more generations (15 * 10 * 100 = 15000 more individuals), which took about 25 hours. The novelty-mutating operators were: add two-port series, add two-port parallel, add n-port parallel. The two-port parts available for add were: capacitors, resistors, nmos/pmos diodes, and biased nmos/pmos devices (a biased mos is merely transistor with a pre-set voltage bias). One more search objective was added: minimize novelty.

With the results, we output the nondominated set, and first examined if any novel individuals existed. Some did. With each novel individual, we queried its data structure to find which parts were novel, and how they were than their original part. It turns out that so far in this run, they all had the same change: the feedback capacitor Cc had been mutated to include a resistor in series. Figure 9 illustrates. This is actually a well-known design technique that one can find in many analog design textbooks: what it does is increase the effective gain from feedback; it does not help the feedforward gain as much because the feedforward path does not get its gain amplified.

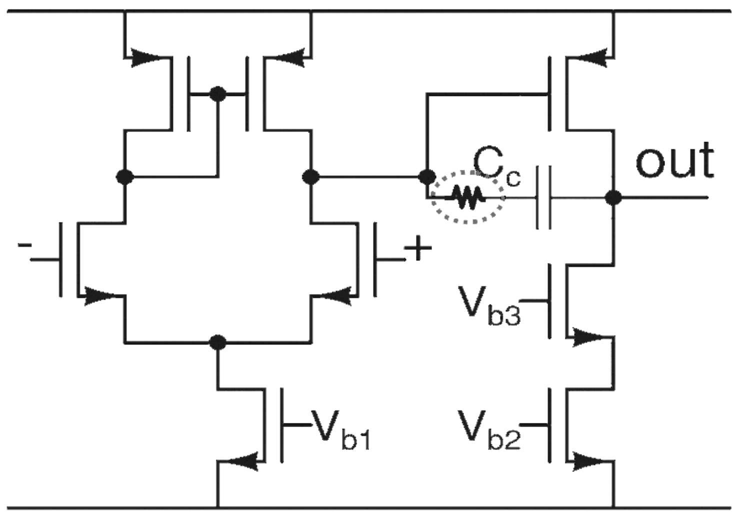

Figure 10-11. Circuit which MOJITO-N successfully re-invented. The circled resistor in the feedback path was not in the library; MOJITO-N added it; this is a well-known design technique.

7. Conclusion

This paper showed how aggressive reuse of known designs brings a vast reduction in computational effort in GP applied to automated structural design. It presented a complementary pair of approaches that incorporate reuse:

- MOJITO automatically designs 100% trustworthy structures of industrially relevant complexity, with commercially reasonable computational effort. MOJITO's effectiveness was demonstrated in two separate experiments, showing how it hit the target designs as expected, from a library of more than 3000 possible topologies.
- MOJITO-N adds novelty to the trustworthy designs, and returns circuits that trade off novelty with performance, also with commercially reasonable computational effort. The novelty is fully trackable, so all changes can be readily understood. MOJITO-N successfully re-invented a known design of industrially relevant complexity.

To properly capture the relevant knowledge to reuse, we designed a parameterized generative representation , and then used the representation to encode a library of building blocks for the specific problem (in our case, operational amplifier design). The key to manage trustworthiness in the presence of novelty was to add an extra objective of "minimize novelty" within a multi-objective optimization framework, which results in trustworthiness tradeoffs. "Novelty" is the number of structural mutation steps taken from a 100% trustworthy design. We view our novelty-approach as "automated innovation" rather than "automated invention" because it builds on existing knowledge – but note that patents are awarded for innovations too.

This work also used state-of-the-art ideas in EA design. It had a hybridized tree/vector view of the search space, implemented as operators having those two perspectives. It was guided by recent advances in theory of EA representations (Rothlauf, 2006). To avoid premature convergence and minimize sensitivity to population size setting, we employed the age-layered population structure (ALPS) (Hornby, 2006), and embedded NSGA-II (Deb et al., 2002) into each age layer of ALPS to make it multiobjective.

These techniques can be readily extended to other GP problem domains of interest, and are complementary with many other recent advances in GP.

References

Antao, B.A.A. and Brodersen, A.J. (1995). Archgen: Automated synthesis of analog systems. *IEEE Transactions on Very Large Scale Integrated Circuits*, 3(2):231–244.

Ashenden, Peter J., Peterson, Gregory D., and Teegarden, Darrell A. (2002). *The System Designer's Guide to VHDL-AMS*. Morgan Kaufmann.

Becker, Ying, Fei, Peng, and Lester, Anna M. (2006). Stock selection : An innovative application of genetic programming methodology. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages –. Springer, Ann Arbor.

Berkcan, E., d'Abreu, M., and Laughton, W. (1988). Analog compilation based on successive decompositions. In *Design Automation Conference*, pages 369–375.

Bernardinis, F. De, Nuzzo, P., and Sangiovanni-Vincentelli, A.L. (2005). Mixed signal design space exploration through analog platforms. In *Design Automation Conference*, pages 875–880.

Castillo, Flor, Kordon, Arthur, Sweeney, Jeff, and Zirk, Wayne (2004). Using genetic programming in industrial statistical model building. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 3, pages 31–48. Springer, Ann Arbor.

Chang, Henry (1997). *A Top Down, Constraint Driven Design Methodology for Analog Integrated Circuits*. Kluwer.

Dastidar, T.R. and Chakrabarti, P.P. (2005). A synthesis system for analog circuits based on evolutionary search and topological reuse. *IEEE Transactions on Evolutionary Computation*, 9(2):2005.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Ding, Mengmeng and Vemuri, Ranga (2005). A combined feasibility and performance macromodel for analog circuits. In *Design Automation Conference*, pages 63–68.

Doboli, Alex and Vemuri, Ranga (2003). Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. *IEEE Transactions on Computer-Aided Design*, 22(11).

E1-Turky, F.M. and Nordin, R.A. (1986). Blades: An expert system for analog circuit design. In *International Conference on Circuits and Systems*, pages 552–555.

Eeckelaert, Tom, McConaghy, Trent, and Gielen, Georges G. E. (2005). Efficient multiobjective synthesis of analog circuits using hierarchical paretoÂ–optimal performance hypersurfaces. In *Design Automation and Test Europe*.

Eeckelaert, Tom, Schoofs, Raf, Gielen, Georges G. E., and Steyaert, Michiel (2007). An efficient methodology for hierarchical synthesis of mixed-signal systems with fully integrated building block topology selection. In *Design Automation and Test Europe*.

Friedman, Jerome H. (1991). Multivariate adaptive regression splines. *Annals of Statistics*, 19(1-141).

Goldberg, David E. (2002). *The Design of Innovation*. Springer.

Harjani, R., Rutenbar, R., and Carley, L. (1992). Oasys: A framework for analog circuit synthesis. *IEEE Transactions on Computer-Aided Design*, 8(12):1247–1266.

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

Hornby, Gregory Scott (2003). *Generative Representations for Evolutionary Design Automation*. PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA.

Hu, Jianjun and Goodman, Erik (2004). Topological synthesis of robust dynamic systems by sustainable genetic programming. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 9, pages ??–157. Springer, Ann Arbor. pages missing.

Hu, Jianjun, Goodman, Erik, Seo, Kisung, Fan, Zhun, and Rosenberg, Rondal (2005). The hierarchical fair competition framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2):241–277.

Huynen, M.A., Stadler, P., and Fontana, W. (1996). Smoothness within ruggedness: The role of neutrality in adaptation. *National Academy of Sciences USA*, 93:397–401.

ITRS (2007). International technology roadmap for semiconductors.

Kampe, Jurgen (2000). A new approach for the structural synthesis of analog subsystems. In *International Workshop on Symbolic Methods and Applications in Circuit Design*, pages 33–38.

Keijzer, Maarten (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269.

Keijzer, Maarten (2005). Run transferable libraries. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*. Kluwer.

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

Korns, Michael F. (2006). Large-scale, time-constrained symbolic regression. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Pro-*

gramming Theory and Practice IV, volume 5 of *Genetic and Evolutionary Computation*, chapter 16, pages –. Springer, Ann Arbor.

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

Koza, John R., Andre, David, Bennett III, Forrest H, and Keane, Martin (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman.

Koza, John R., Jones, Lee W., Keane, Martin A., and Streeter, Matthew J. (2004). Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 8, pages 120–?? Springer, Ann Arbor. pages missing?

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

Koza, John R., Streeter, Matthew J., and Keane, Martin A. (2003b). Automated synthesis by means of genetic programming of complex structures incorporating reuse, hierarchies, development, and parameterized toplogies. In Riolo, Rick L. and Worzel, Bill, editors, *Genetic Programming Theory and Practise*, chapter 14, pages 221–237. Kluwer.

Kruiskamp, Wim and Leenaerts, Domine (1995). Darwin: Cmos opamp synthesis by means of a genetic algorithm. In *Design Automation Conference*.

Kundert, K. and Zinke, O. (2004). *The Designer's Guide to Verilog-AMS*. Kluwer.

Lai, X. and Roychowdhury, Jaijeet (2006). Macromodeling oscillators using krylov-subspace methods. In *Asia And South Pacific Design Automation Conference*.

Lohn, Jason, Hornby, Gregory, and Linden, Derek (2004). Evolutionary antenna design for a NASA spacecraft. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 18, pages 301–315. Springer, Ann Arbor.

Lohn, Jason D. and Colombano, S.P. (1998). Automated analog circuit synthesis using a linear representation. In *International Conference on Evolvable Systems*, pages 125–133.

Martens, Ewout and Gielen, Georges G.E. (2006). Top-down heterogeneous synthesis of analog and mixed-signal systems. In *Design Automation and Test Europe*, pages 275–280.

Maulik, Peter C., Carley, L.R., and Rutenbar, R. (1995). Integer programming based topology selection of cell level analog circuits. *IEEE Transactions on Computer-Aided Design*, 14(4).

McConaghy, Trent, Eeckelaert, Tom, and Gielen, Georges (2005). CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming. In *Proceedings of the Design Automation and Test Europe (DATE) Conference*, volume 2, pages 1082–1087, Munich.

McConaghy, Trent and Gielen, Georges (2005). Genetic programming in industrial analog CAD: Applications and challenges. In Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 19, pages 291–306. Springer, Ann Arbor.

McConaghy, Trent, Palmers, Pieter, Gielen, Georges G.E., and Steyaert, Michiel (2007). Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *Design Automation Conference*.

Ning, Z., Mouthaan, A.J., and Wallinga, H. (1991). Seas: A simulated evolution approach for analog circuit synthesis. In *Custom Integrated Circuits Conference*.

Nordin, Peter (1994). A compiling genetic programming system that directly manipulates the machine code. In Kinnear, Jr., Kenneth E., editor, *Advances in Genetic Programming*, chapter 14, pages 311–331. MIT Press.

Phillips, Joel R. (1998). Model reduction of time-varying linear systems using approximate multipoint krylov-subspace projectors. In *International Conference on Computer-Aided Design*, pages 96–102.

Poli, Riccardo and Langdon, William B. (1999). Sub-machine-code genetic programming. In Spector, Lee, Langdon, William B., O'Reilly, Una-May, and Angeline, Peter J., editors, *Advances in Genetic Programming 3*, chapter 13, pages 301–323. MIT Press, Cambridge, MA, USA.

Razavi, Behzad (2000). *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

Ressler, Andrew L. (1984). *A Circuit Grammar for Operational Amplifier Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Rothlauf, Franz (2006). *Representations for genetic and evolutionary algorithms*. Springer-Verlag, pub-SV:adr, second edition. First published 2002, 2nd edition available electronically.

Rutenbar, R.A., Gielen, Georges G.E., and Antao, B.A. (2002). *Computer-Aided Design of Analog Integrated Circuits and Systems*. IEEE Press, Piscataway, NJ, USA.

Sansen, Willy (2006). *Analog Design Essentials*. Springer.

Shibata, Hajime, Mori, Soji, and Fujii, Nobuo (2002). Automated design of analog circuits using cell-based structure. In *Nasa/DoD Conference on Evolvable Hardware*.

Spector, Lee (2004). *Automatic Quantum Computer Programming: A Genetic Programming Approach*, volume 7 of *Genetic Programming*. Kluwer Academic Publishers, Boston/Dordrecht/New York/London.

Sripramong, Thanwa and Toumazou, Christofer (2002). The invention of CMOS amplifiers using genetic programming and current-flow analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1237–1252.

Swings, K., Donnay, S., and Sansen, W. (1991). Hector: a hierarchical topology-construction program for analog circuits based on a declarative approach to circuit modeling. In *Custom Integrated Circuits Conference*.

Synopsys (2007). Circuit explorer product. *Website of Synopsys Inc.*

Tanaka, T. (1993). Parsing electronic circuits in a logic grammar. *IEEE Transactions Knowledge and Data Engineering*, 5(2):225–239.

Tang, H. and Doboli, A. (2006). High-level synthesis of delta-sigma modulator topologies optimized for complexity, sensitivity, and power consumption. *IEEE Transactions on Computer-Aided Design*, 25(3):597–607.

Toumazou, Chris, Makris, C.A., and Berrah, C.M. (1990). Isaid - a methodology for automated analog ic design. In *International Symposium on Circuits and Systems*, volume 1, pages 531–555.

Vassilev, Vesselin K. and Miller, Julian F. (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems*, pages 252–263. Springer-Verlag.

Whigham, P. A. (1995). Grammatically-based genetic programming. In Rosca, Justinian P., editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA.

Yao, Xin, Liu, Yong, and Lin, Guangming (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2).

Yu, Tina, Wilkinson, Dave, and Castellini, Alexandre (2006). Applying genetic programming to reservoir history matching problem. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 6, pages –. Springer, Ann Arbor.