KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT ELEKTROTECHNIEK
Kasteelpark Arenberg 10, B-3001 Leuven

VARIATION-AWARE STRUCTURAL SYNTHESIS AND KNOWLEDGE EXTRACTION OF ANALOG CIRCUITS

Jury:
Prof. G.E. Gielen, promotor

Prof. W.M. Sansen
Prof. W. Dehaene
Prof. J. Vandewalle
Prof. M. Diehl
Dr. G. Smits (Dow Benelux)
Prof. R.A. Rutenbar (CMU, USA)
Dr. G.S. Hornby (NASA Ames, USA)

Thesis submitted to obtain the degree of doctor in engineering sciences

TRENT MCCONAGHY

UDC: 621.3.049.77 November 2008

D/2008/7515/97

ISBN 978-90-5682-987-2

Abstract

This thesis describes new tools for front end analog designers, starting with global variation-aware sizing, and extending to novel variation-aware topology design. The tools aid design through automation, but more importantly, they also aid *designer insight* through automation. We now describe four design tasks, each more general than the previous, and how this thesis contributes design aids and insight aids to each.

The first designer task targeted is global robust sizing. This task is supported by a design tool that does automated, globally-reliable, variation-aware sizing (SANGRIA), and an insight-aiding tool that extracts designer-interpretable whitebox models that relate sizings to circuit performance (CAFFEINE). SANGRIA searches on several levels of problem difficulty simultaneously, from lower cheap-to-evaluate "exploration" layers to higher full-evaluation "exploitation" layers (structural homotopy). SANGRIA makes maximal use of circuit simulations by performing scalable data mining on simulation results to choose new candidate designs. CAFFEINE accomplishes its task by treating function induction as a tree-search problem. It constrains its tree search space via a canonical-functional-form *grammar*, and searches the space with grammatically-constrained genetic programming.

The second designer task is topology selection / topology design. Topology *selection* tools must consider a broad variety of topologies such that an appropriate topology is selected, must easily adapt to new semiconductor process nodes, and readily incorporate new topologies. Topology *design* tools must allow for designers to creatively explore new topology ideas as rapidly as possible. Such tools should not impose new, untrusted topologies that have no logical basis. MOJITO supports both topologyo selection and design. It takes in a pre-specified library of about 30 hierarchically-organized analog building blocks. This library defines *thousands* of possible different circuit opamp topologies from different combinations of the building blocks. The library is independent of process, and does not require input of behavioral models. Therefore, it only has to be specified once. However, designers can readily add new building block ideas to it. MOJITO efficiently globally searches this library's possible topologies and sizings by leveraging the hierarchical nature of the blocks. MOJITO returns ("synthesizes") topologies that are *trustworthy by construction*. MOJITO is multi-objective, i.e. it returns a set of sized topologies that collectively approximate an optimal performance tradeoff curve. Once a single MOJITO run is done at a process node, the results are stored as a database for future queries by other designers. Therefore MOJITO supports a "specs-in sized-topology-out" workflow with immediate turnaround.

This thesis also demonstrates *insight* aids for topology selection and design. By taking a data-mining perspective on this database, it (a) extracts a specs-to-topology decision tree, (b) does global nonlinear sensitivity analysis on topology and sizing variables, and (c) determines analytical expressions of performance tradeoffs.

The third design task combines the previous two: variation-aware topology selection and design. The *design* tool is MOJITO-R, which extends MOJITO with structural homotopy to efficiently handle variation-awareness and return *robust* topologies. The *insight* tools take a data-mining perspective on MOJITO-R's resulting database, so that the designer can explore the relation among topologies, sizings, performances, and *yield*.

The final designer task is about novelty. This thesis explores two tools that can support designers to create designs with novel *functionality* and/or novel *topologies*. The first tool is MOJITO-N. It is targeted towards finding novel topologies for classes of circuits that typically have existing reference designs (i.e. non-novel functionality). Using "trustworthiness tradeoffs", MOJITO-N only adds novelty to known topologies when there is payoff. The other tool is ISCLEs. It finds novel topologies for classes of circuits without recourse to existing reference designs (i.e. novel functionality). ISCLEs *boosts* digitally-sized "weak learner" circuits to create an overall ensemble of circuits. ISCLEs has promise to be *naturally* robust to process variation, and an area footprint that scales with shrinking process geometries.

This thesis had several aims. The first was to bring the role of the designer back into computer-aided design (CAD) tool design, to provide more opportunities for designer-computer interaction such that the strengths of each can be exploited. This drove the work in knowledge extraction tools, and was the key to a trustworthy topology design tool. The second aim was to chart out a possible roadmap that could guide industrial CAD tool rollout, starting with the near term goal of global variation-aware sizing, then hitting successively farther-out goals such as topology design and novel topology design. Each tool was designed to have inputs and outputs as close as possible to industry, and to use off-the-shelf simulators for flexibility and accuracy. The third aim was to bridge fields of analog CAD and genetic programming / evolvable hardware: both fields aimed for topology synthesis, but the jargon and the algorithmic tools were different, and the specific problems in topology synthesis needed clarification and better designer / industrial context. The final aim was to make contributions that generalize beyond analog CAD. It turns out that the roadmap and algorithms developed are general, and can be applied to many of other fields from automotive design to bio-informatics.

Thank Yous

There are many people who have helped make it possible for me to do this work. I do not think it is possible to thank them enough, but I will at least try!

First, I would like to thank my loving wife, Masha, who has been by my side, encouraging me in my thesis work, and having patience when I got too busy for doing much else. You are my inspiration. I would like to thank my parents, Lorne and Norma, who provided me with a strong foundation in life and in education, and have always been very supportive of me as I chase my dreams. To my brothers Troy and Trevis, thank you, for your insights and advice. Masha, dad, mom, Troy, Trevis, I love and treasure each of you.

To Georges, my supervisor, I would like to thank you on multiple counts. First, thank you for the invitation to study at ESAT-MICAS, which led to the thesis work in the first place. Second, thank you for your guidance and advice from both a technical standpoint (with your deep experience in the field) as well as a planning standpoint (with your understanding of human nature). I appreciate the freedom you gave me to pursue the problems in analog CAD and AI that I believed to be the most interesting / challenging, and for the thorough reviews of my papers and my thesis. Thank you for your patience as I balanced my Solido work with my PhD work. Thanks for being my promotor!

A very special thanks to Pieter Palmers, who as a long-term collaborator was instrumental in the success of this PhD. Pieter, thank you for your your help in developing the MOJITO library, in collaborating to refine MOJITO for "real" analog results, and our long conversations about the relation between search algorithms, search spaces, and analog design. It all made a huge difference in making MOJITO such a powerful tool.

I had several other collaborators who I must also thank for their tremendous help. Thank you to Peng Gao, who as my Master's student, co-developed ISCLEs with me. I am deeply indebted to your efforts in designing an ISCLEs library of weak learners, and in refining the ISCLEs system to get "real" analog results. Thank you to Tom Eeckelaert for the many insightful conversations about evolutionary algorithms, multi-objective search, analog design, and living in Leuven. Our discussions, and your own research, have left an unmistakable imprint on the work in this thesis. I'm still waiting for MOBU-2, by the way! Thank you to Jurgen Deveguele for supplying me with the test problem for the IBMG tool.

Thanks to Amit Gupta, who as my business partner in both Solido and ADA, has worked tirelessly to help ensure that my work stays grounded in the needs of semiconductor customers. Amit, I also thank you for your patience during the times when balancing Solido and my PhD was difficult and sometimes Solido-specific would need to be sidelined.

Thanks to all the others in Solido, ADA, and the EDA/semi industry who have all helped me and influenced my work. Of those, special thanks go to Pat Drennan for nicely-opinionated feedback about what designers really need. Thanks to Rob Rutenbar, from our time as competitors to your friendship and advice, and for being on my jury. More special thanks to: Chris Labrecque, Kris Breen, Matthew Raggett, Jim Rutt, Jeff Dyck, Solido's customers / investors / board members, ADA's founders / customers / investors / TAB members / board members, and the analog designers who have used who have used ADA/Synopsys and Solido tools over the years... and the ones who will in the future!

Several people in the genetic programming community deserve special mention. John Koza, thank you on multiple counts – for our long-running discussions about industrial-strength analog synthesis; thanks for welcoming me into the GP community (especially the GPTP workshops); and thanks in regard to research proposals and investments. Greg Hornby, thank you for your engineering-oriented ideas in evolutionary algorithms, and our discussions about AI and design which all strongly influenced this thesis (most obviously ALPS!); and thanks for being on my jury. Thanks to Guido Smits for the great conversations about real-world GP, and for being on my jury. Thanks to Rick Riolo, Sameer Al-Sakran, Lee Jones, Una-May O'Reilly, Varun Aggarwal, Arthur Kordon, Erik Goodman, Maarten Keijzer, Julian Miller, Jason Lohn, Didier Keymeulen, Adrian Stoica, Matt Streeter, Marty Keane, Wolfgang Banzhaf, Lee Spector, Riccardo Poli, Terry Soule, Marc Bedau, Conor Ryan, Mike Korns, and many others who have influenced my work. I appreciate all our conversations!

Thank you to all of my jury members, if I haven't mentioned you already. Thank you to Georges Gielen, Willy Sansen, Wim Dehaine, Joos Vandewalle, Moritz Diehl, Guido Smits, Rob Rutenbar, and Greg Hornby. Thank you all for taking the time to review my thesis, to be at the meetings, and for all your feedback. Special thanks for those who had to travel long distances to be there!

Thanks to the several people who helped me to adjust to life in Leuven, in getting around, in moves, and in translations (being a native English speaker in a Flemish-speaking town isn't always easy!). Thank you to Raf Schoofs, Bert Lenaerts, Bert Serneels, Stijn Swaegers, Stefan Huber, Theo Marescaux, Yi Ke, Phillipe Coppejans, and many, many more.

Thanks to everyone else at ESAT and ESAT-MICAS who has helped me. Thanks to Danielle, Chris, Lut, and the other administrative staff. Thanks to Ben, Frederik, and the other sysadmins. Thanks to Nick van Helleputte for the enjoyable teaching collaborations. Thanks to Moritz Diehl for geting me more involved in OPTEC. Thanks to Willy Sansen for building such a strong research group, you deserve a wonderful retirement! And finally, thanks to all the MICAS members, including my other PhD colleagues, for being such a vibrant, open community that is wonderful to be around.

Finally, thanks again to Masha! I am a very lucky husband.

Trent McConaghy
November 2008

There's only one thing that I'm certain of:
If you don't look for uncertainty, it will come looking for you.

–Anonymous

Analog synthesis is always the holy grail.

–Ron Gyurcsik (as Director of Cadence AMS Group)

For the goal is not the last, but the best.

–Aristotle

Summary of Contents
and Roadmap for Analog CAD

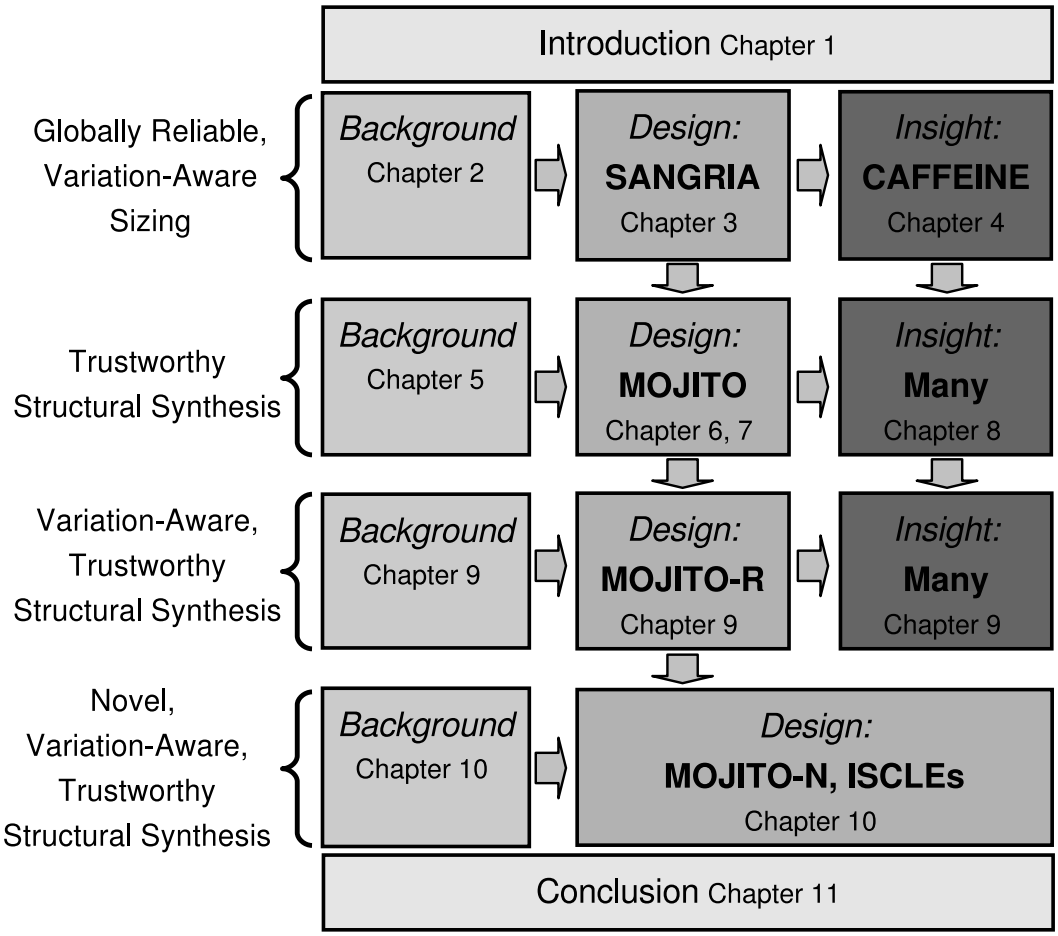


Table of Contents

Acronyms

AC	Alternating Current
ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
ADF	Automatically Defined Function
ADM	Automaticall Defined Macro
ALPS	Age Layered Population Structure
ARF	Adaptive Ranking on Pareto fronts
BB	Building Block
BJT	Bipolar Junction Transistor
CAD	Computer-Aided Design
CAFFEINE	CAnonical Form Function Expressions IN Evolution
CART	Classification And Regression Trees
CMOS	Complementary Metal-Oxide-Semiconductor
CD	Chemical Deposition
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DB	DataBase
DC	Direct Current
DOC	Device Operating Constraint
DOE	Design Of Experiments
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EDA	Electronic Design Automation
EI	Expected Improvement
EMC	ElectroMagnetic Compatibility
ENOB	Effective Number Of Bits
EP	Evolutionary Programming
FFNN	Feed-Forward Neural Network
FU	Unity gain Frequency
GA	Genetic Algorithm
GDR	Gradient Directed Regularization
GENRE	GENerative REpresentation
GP	Genetic Programming
GBW	Gain BandWidth

HFC	Hierarchical Fair Competition
IBMG	Interpretable Behavioral Model Generator
IC	Integrated Circuit
IP	Intellectual Property
ISCLEs	Importance Sampled Circuit Learning Ensembles
ISLEs	Importance Sampled Learning Ensembles
ITRS	International Technology Roadmap for Semiconductors
IS	Importance Sampling
LCB	Least-Constrained Bounds
LHS	Latin Hypercube Sampling
LS	Least-Squares
LS-SVM	Least-Squares Support Vector Machine
LSB	Least Significant Bit
LVS	Layout Versus Schematic
LTI	Linear Time Invariant
MARS	Multivariate Adaptive Regression Splines
MBO	Model Building Optimization
MINLP	Mixed-Integer Non-Linear Programming
MC	Monte Carlo
ML	Machine Learning
MSB	Most Significant Bit
MOEA	Multi-Objective Evolutionary Algorithm
MOJITO	Multi-ObJectIve and -TOpology
MOJITO-N	MOJITO with Novelty
MOJITO-R	MOJITO with Robustness
MOR	Model Order Reduction
MOBU	Multi-Objective Bottom-Up methodology
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MPW	Multi-Project Wafer
NMOS	N-channel MOSFET
NMSE	Normalized Mean Squared Error
NN	Neural Network
NIID	Normal, Independent, and Identically Distributed
NSGA-II	Nondominated Sorting Genetic Algorithm II
OS	Overshoot
OTA	Operational Transconductance Amplifier
PCB	Printed Circuit Board
PDK	Process Design Kit
PM	Phase Margin
PMOS	P-channel MOSFET
PWL	Piece-Wise Linear
PWP	Piece-Wise Polynomial
PVT	Process, Voltage, Temperature
RCX	Resistor Capacitor eXtraction

RF	Radio Frequency
RSM	Response Surface Method
RTL	Run Transferable Library
SANGRIA	Statistical, Accurate, aNd Globally Reliable sIzing Algorithm
SDL	Schematic Driven Layout
SGB	Stochastic Gradient Boosting
SiP	System-in-Package
SoC	System-on-Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
SR	Slew Rate *or* Symbolic Regression
SVM	Support Vector Machine
TDCD	Top-Down Constraint-Driven methodology
THD	Total Harmonic Distortion
TSMC	Taiwan Semiconductor Manufacturing Corporation
VC	Variable Combo (in CAFFEINE)
VOFF	OFFset Voltage
WL	Weak Learner (in ISCLEs)

Notation

General Notation

a, A	scalars are non-bold lowercase and non-bold uppercase
$\boldsymbol{a}$	vectors are bold lowercase
$\boldsymbol{A}$	matrices are bold uppercase
i, j, k, l	i, j, k, l are reserved for indices; e.g. $\bullet_i$ is the i^{th} component of $\bullet$
$I(condition)$	indicator function that returns 1 if $condition$ is True, and 0 otherwise
$[a, b]$	range of continuous values from a to b; $a \leq b$
$\Re^a$	real-valued space of dimension a
$pdf(\boldsymbol{a})$	probability density function for random variables $\boldsymbol{a}$
$E(\bullet)$	expected value of quantity $\bullet$
Δa	a small ("infinitesimal") value of a
$\bullet^*$	$\bullet^*$ is an "optimized" version of $\bullet$
$\bullet'$	$\bullet'$ is an "updated" version of $\bullet$

Circuit / Problem Notation

V, I	voltage, current
R, C, W, L, M	resistance, capacitance, transistor width, transistor length, multiplier
$R_{min}, C_{min}, ...$	minimum value for $R, C, ...$
$R_{max}, C_{max}, ...$	maximum value for $R, C, ...$
T	temperature
V_{dd}, V_{ss}	power supply voltage, voltage rail
Φ	"general" space of possible topologies and sizings
ϕ	"general" design point; $\phi \in \Phi$
$\boldsymbol{d}$	design point as a vector; $\boldsymbol{d} = \{d_1, d_2, ..., d_{N_d}\}$
$\boldsymbol{\theta}$	environmental point (corner); $\boldsymbol{\theta} = \{\theta_1, \theta_2, ..., \theta_{N_\theta}\}$
$\boldsymbol{s}$	process point (corner); $\boldsymbol{s} = \{s_1, s_2, ..., s_{N_s}\}$
N_d, N_θ, N_s	number of design variables, environmental variables, and process variables
D, Θ, S	space of possible design points, environmental points, and process points
Ξ_i	combined environmental and process corner i; i.e. $\Xi_i = \{\boldsymbol{\theta_i}, \boldsymbol{s_i}\}$
$\boldsymbol{\Xi}$	set of combined corners; i.e. $\boldsymbol{\Xi} = \{\Xi_1, \Xi_2, \dots\}$
λ_i	circuit performance specification i; e.g. $gain \geq 60dB$
$\boldsymbol{\lambda}$	list of performance specifications; $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \dots, \lambda_{N_g}\}$
ξ_i	testbench i
N_ξ	number of testbenches
$\boldsymbol{\xi}$	list of testbenches; $\boldsymbol{\xi} = \{\xi_1, \xi_2, \dots, \xi_{N_\xi}\}$

f_i, g_j, h_k	objective i, inequality constraint j, equality constraint k
N_f, N_g, N_h	number of objectives, inequality constraints, equality constraints
$\boldsymbol{f_j}$	list of objective function values for design point ϕ_j; $\boldsymbol{f_j} = \boldsymbol{f}(\phi_j)$

Notation in Sizing and SANGRIA

$Y(\boldsymbol{\phi})$	yield of design point ϕ
$\delta(\boldsymbol{\phi}, ...)$	feasibility
σ	standard deviation
$\widehat{Y_{MC}}$	yield estimate from Monte Carlo sampling
N_{MC}	number of Monte Carlo samples
$\hat{\Theta}$	set of environmental corners to approximate Θ
N_c	number of environmental corners
$Cpk(\boldsymbol{\phi})$	process capability of design point ϕ
$u(\boldsymbol{x})$	uncertainty at point $\boldsymbol{x}$ (of model)
$\Lambda(\boldsymbol{x})$	infill criterion at point $\boldsymbol{x}$ (in model-building optimization)
$\psi, \psi^{(i)}, \psi^{(j)}$	SGB ensemble, SGB model, and CART regression model, respectively
ι	in CART, maximum allowed recursion depth
ι_{min}, ι_{max}	in SGB, minimum and maximum *possible* CART depth, respectively
μ	in SGB, the fraction of overall training samples to use in building a single CART
ϵ_{targ}	in SGB, target training error
H	in homotopy, the homotopy map
η	in homotopy, the degree to which H reflects the true objective function
P_{all}	all individuals from whole SANGRIA run
$\boldsymbol{B}$	in LHS, the bin selections, where $B_{i,j}$ is the bin chosen for variable i in sample j
P_{cand}	in tournament selection, the candidate parents
N_{ens}	number of models in ensemble-based regressor
N_{inner}	the number of new designs from an inner optimization
χ	in DHC, the state of the search. Holds $\rho, \boldsymbol{x}$, etc.
ρ	in DHC, the next action
$\boldsymbol{v}, \boldsymbol{u}$	in DHC, velocity vector and ridge-walking vector
$\boldsymbol{x}, \boldsymbol{xv}, \boldsymbol{xuv}$	in DHC, design points: center, center $+\boldsymbol{v}$, center $+\boldsymbol{u} + \boldsymbol{v}$
V_{list}	in DHC, the list of next $\boldsymbol{v}$'s
v_{init}, v_{min}	in DHC, initial and minimum stepsize
$N_{MC,cand}$	number of Monte Carlo samples from which to choose SANGRIA corners
$N_{MC,chosen}$	number of Monte Carlo samples chosen as SANGRIA (process) corners

Notation in Synthesis and MOJITO

Z	Pareto-optimal set of "general" design points; $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_{N_Z}^*\}$
ϕ_i^*	Pareto-optimal "general" design point; an entry in Z
N_Z	number of points in Z; i.e. $N_M = \|Z\|$
N_T	number of unique topologies in Z

Notation in Regression and CAFFEINE

N	number of data samples

n	model's number of input variables
1	model's number of output variables
$\boldsymbol{X}$	all input data; $\boldsymbol{X} \in \Re^{nxN}$; each column is a sample; one row per variable
$\boldsymbol{y}$	all output data; $\boldsymbol{y} \in \Re^{N}$; each entry is a sample
$\boldsymbol{x_j}$	a single input sample j; $\boldsymbol{x_j} \in \Re^{n}$
y_j	a single output sample; $y_j \in \Re^{1}$
Ψ	search space of possible regression models
ψ	CAFFEINE regression model mapping $\boldsymbol{x}$ to y, i.e. $\hat{y} = \psi(\boldsymbol{x})$; $\psi \in \Psi$
M	Pareto-optimal set of models; $M = \{\psi_1^*, \psi_2^*, \ldots, \psi_{N_M}^*\}$
ψ_i^*	Pareto-optimal design point; an entry in M
N_M	number of points in M; i.e. $N_M = \|M\|$
N_B	number of basis functions in ψ (not counting offset)
$\boldsymbol{a}$	linear weights on basis functions (including offset)
B_i	basis function i
w_b	in CAFFEINE, minimum complexity cost of a basis function
w_{vc}	in CAFFEINE, complexity cost scaling factor of a "variable combo"
ι_i	in CAFFEINE, influence of a basis function B_i
ι_{thr}	in CAFFEINE, target total influence for "useful" expressions
κ	in CAFFEINE, percentage of time that "useful" expressions are chosen
τ	in GDR, target degree of pruning
ν	in GDR, $\boldsymbol{a}$-updating step vector
$\boldsymbol{gr}$	in GDR, gradient of squared-error loss function
$\boldsymbol{hr}$	in GDR, "selective" gradient of squared-error loss function
γ_i	in GDR, indicates if a coefficient is to be selected
N_{scr}	number of scrambles in global sensitivity analysis
$\boldsymbol{X_{scr}}$	like $\boldsymbol{X}$ except one variable's row has been permuted (scrambled)
$\boldsymbol{y_{scr}}$	output of model ψ when simulating $\boldsymbol{X_{scr}}$

Notation in CART / Decision Trees

Υ	set of possible class labels, one per unique topology; $\Upsilon = \{1, 2, ..., N_T\}$
υ_j	topology class label corresponding to individual ϕ_j^*; $\upsilon_j \in \Upsilon$
ω	decision tree mapping $\boldsymbol{f}$ to t, i.e. $\hat{\upsilon} = \omega(\boldsymbol{f})$; $\omega \in \Omega$
N_R	number of disjoint regions created by tree ω
R_i	subregion i created by tree ω

Notation on EA State

N_{gen}	current number of generations
P_{sel}	selected parent individuals (design points)
P_{ch}	children individuals
P_i	in ALPS, the population of individuals in age layer i
$P_{i,j}$	in ALPS, individual j in population P_i
P	in ALPS, all the age layers so far
$\|P\|$	in ALPS, the number of age layers currently in P
$\|P_i\|$	in ALPS, the number of individuals currently in layer P_i

F_i	in NSGA-II, the individuals in nondomination layer i
N_{ND}	in NSGA-II, the number of nondomination layers

Notation on EA Settings

$N_{sim,max}$	maximum number of circuit simulations
$N_{ind,max}$	maximum number of individuals to explore (design points)
$N_{gen,max}$	maximum number of generations
N_{pop}	in CAFFEINE and NSGA-II, the number of individuals in the population
N_L	in ALPS, the number of individuals per age layer
K	in ALPS, the maximum number of age layers
N_a	in ALPS, the "age gap" - the number of generations interval for creating a new age layer 0

Notation in ISCLEs

α	boosting learning rate
$\boldsymbol{y_{overall,target}}$	target output of whole ensemble
$\boldsymbol{y_{current,target}}$	current output of whole ensemble
r_{target}	target correlation between ensemble's output and target output
$r_{current}$	current correlation between ensemble's output and target output
WL_{cand}	candidate weak learner
EL_{cand}	candidate ensemble of weak learners
$\boldsymbol{y_{cand}}$	output of candidate ensemble of weak learners
EL_{chosen}	chosen ensemble of weak learners

Chapter 1

Introduction

No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be.
–Isaac Asimov

1.1 Motivation

The progress of the last half-century can be characterized by the exponential improvements in information technology (IT), due to consistently shrinking transistors (Moore's Law) [Moo1965, Itrs2007] combined with ever-improving algorithms and software [Bro2004]. IT has brought us mainframes, personal computers, video games, cell phones, the internet, smartphones, and cloud computing. It plays a role in almost every aspect of our lives: we snap photos with cell phones and immediately send them to loved ones, rediscover friends on Facebook, call distant relatives for free over the internet, and start a company with headquarters a continent away and customers two continents away. IT is touching every field, from decoding the genome, to brain scanning, to developing alternative fuels. Unsurprisingly, the IT / semiconductor industry has grown into a huge industry, with $255 billion revenue in 2007 [Sia2008]. The accelerating convergence of IT with nanotechnology, bioscience, and cognitive science may have far-reaching benefits [Bai2006, Kur2005].

An IT-enhanced future promises to be bright if such progress continues. However, such progress in IT is far from guaranteed, because the exponential improvement of the *computational substrate* or of the *algorithms* could stop or slow dramatically. Let us examine potential showstoppers for each.

Exponential improvement of computational substrate could stop or slow. Uncontrollable factors in semiconductor manufacturing - process variations - have always existed. Up until recently, the effects would cancel out across the billions or more atoms in a given transistor. But now that transistors have shrunk to atomic scale, Avogadro-size atom counts no longer apply. Even a single atom out of place can affect a transistor's behavior, leading to worsened circuit behavior and even circuit failure. As of 2008, the variation is already large, and as Figure 1.1 shows, it will continue to get worse with future process

technologies. Such variation is particularly problematic for analog circuits, which do not have the abstraction of binary digits to hide small variations. Process variations are not the only problem. Layout parasitics, aging/reliability, electromagnetic compatibility, proximity [Dre2006], and other phenomena can affect circuit behavior. But because of their direct impact on circuit yields, addressing process variations is the most urgent. Design of robustly-behaving analog circuits is difficult and time-consuming. This has caused the analog portion of chips to become the design bottleneck [Itrs2007]. Yet we cannot ignore or bypass analog circuits, since they are crucial for digital circuits to interface with the real world. As of 2006, 70% of systems-on-chips (SoCs) or systems-in-packages (SiPs) have some analog functionality, up from 50% in 2005 and 10% in 1999 [Rat2008]. We need a means to design analog circuits which meet performance goals, have high yield, and with an area that shrink as minimum device lengths shrink. And we need do it fast enough to succeed in tight time-to-market schedules [Gie2005b].

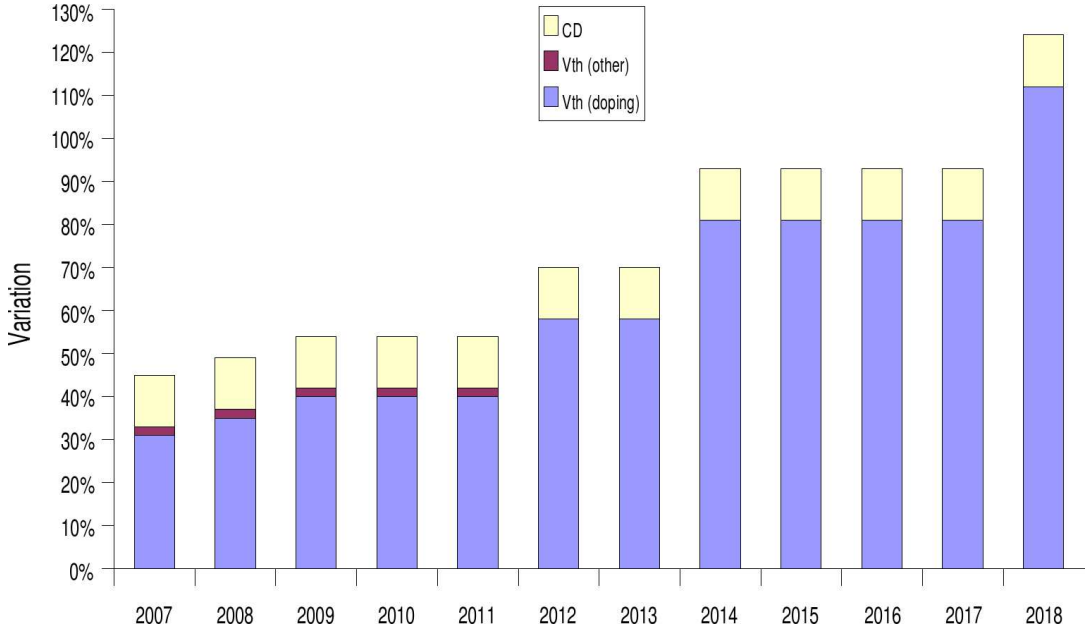


Figure 1.1: *Predicted process variations per year [Itrs2007] (CD is chemical decomposition, V_{th} is threshold voltage).*

Exponential improvement in algorithms / software could stop or slow. Advancements in software design have an odd tendency to originate in the field of artificial intelligence (AI)(e.g. [Rus2003]): a computational problem starts out as a mystery, then some "AI" algorithm to solve it is invented and refined, the solution becomes ubiquitous, and in hindsight the problem and eventual solution looks so simple that it loses the label "AI" (section 12.11 in [Poli2008]). This exact flow was the case for: symbolic manipulation, scheduling, compilers, spreadsheets (!), optimization, regression, and many more modern computer science fields, for problems ranging from stock picking to facial recognition [Kur2005]. Perhaps AI's greatest success is in its ability to spin out new computer-science subfields, once they are sufficiently non-mysterious to fit the label "AI". But one of the big problems that AI has aimed to tackle – design of complex structures – has only seen

limited success so far. The greatest illustration of the failure is to review what's still done almost wholly manually: design of an internal combustion engine, design of an airport, design of music, design of any software with even a modest degree of complexity, and design of analog integrated circuit topologies.[1] But there is hope in some well-defined problems, such as software compilers and digital circuit synthesis leading to millions of lines of code or billions of transistors. Can we go beyond boolean / digital in breaking this apparent complexity wall?

Clearly, progress in semiconductors and in AI face issues. Meeting those challenges is crucial for the progress of both electronics IT and software IT, and the future benefits of IT. This thesis aims to play a role in addressing both issues. From the perspective of circuit design, it presents a suite of techniques to deal with progressively more challenging analog design problems: globally reliable variation-aware sizing, variation-aware structural synthesis, and variation-aware structural synthesis with novelty. In each problem, both automated *design tools* and *knowledge extraction tools* are developed, to leverage the strengths of both the computer *and* the user. From an AI perspective, this thesis presents an approach for automated design of complex structures: it reconciles designer creativity with automated design in a computationally feasible fashion, by using *field-specific, hierarchical building blocks*. This technique is general enough to be applied to other problem domains.

The rest of this chapter is organized as follows. Section 1.2 gives background on analog CAD, and outlines this thesis' contributions from the perspective of analog CAD. Similarly, section 1.3 gives background on AI, and this thesis' contribution to AI. Section 1.4 describes how analog CAD and AI relate. Finally, section 1.5 sketches out the content of the rest of the thesis.

1.2 Background and Contributions to Analog CAD

1.2.1 Analog CAD's Context

As Figure 1.2 shows, the semiconductor industry plays a role in many of the world's largest other industries, from consumer products to automotive. The $300B semiconductor industry can be segregated by type of circuit: wholly digital (25% of market), wholly analog (10%), or mixed-signal which is a combination of both (60%). The $5B Electronic Design Automation (EDA) industry is part of the semiconductor industry, with revenue breakdown of 85% for digital and 15% for analog. EDA is devoted to building computer-aided design (CAD) tools for electrical engineers. Because of the massive size of the semiconductor industry and the constant changes in design constraints due to Moore's Law, EDA is an active industry, with billions in revenue [Edac2006].

Analog computer-aided design (CAD) [Gie2002a, Rut2007] is the subfield of EDA which is devoted to tools for analog circuit designers. Whereas general CAD is crucial for delivering good designs in a timely fashion, *analog* CAD is crucial for delivering good

[1] To be clear, parameter optimization plays a helpful role, but the design of the structure remains mostly manual.

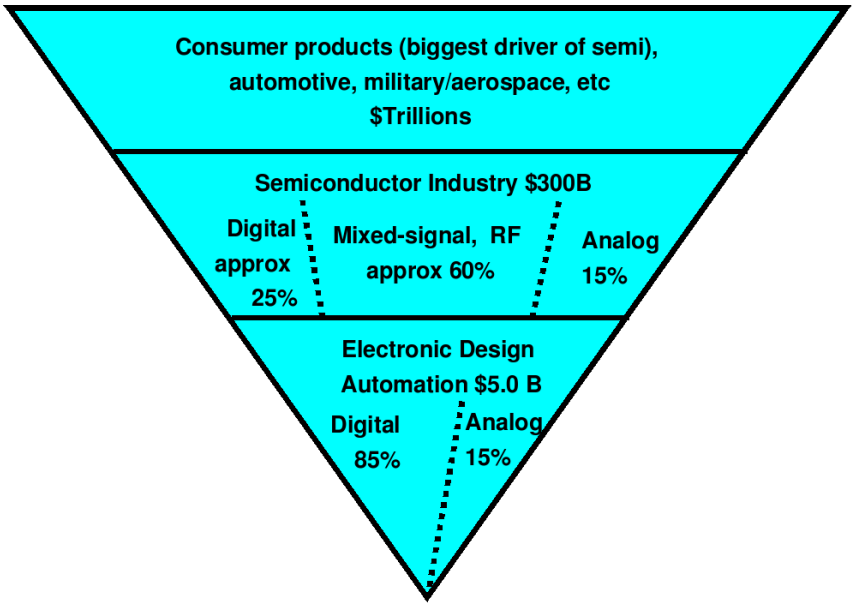


Figure 1.2: *Context of the Analog CAD Industry*

analog designs in a timely fashion. We will review the tool types which are considered part of the standard industrial analog design flows.

1.2.2 Basic Analog Design Flow

For context, we first describe the basic flow that analog circuit designers follow, according to Figure 1.3.

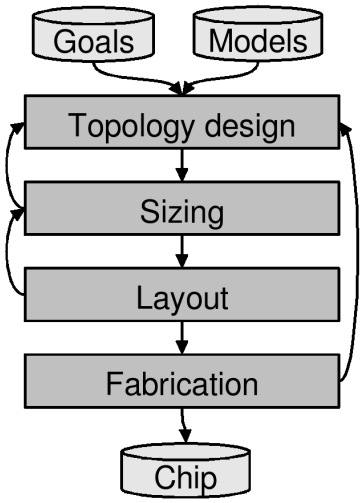


Figure 1.3: *Basic flow for analog circuit design (at a node in the hierarchy).*

The designer (or team) is given a set of design goals such as: “design an opamp with gain > 60 dB, power < 1mW, and minimizing area” on a target fabrication process such

as 65 nm TSMC. In a general sense, the transistor models of the specified process will be used in conjunction with tools to provide estimates of circuit performance. The design proceeds through a series of stages.

First, a topology design is determined, either by selecting an initial topology, or designing a new topology. Then in sizing, the topology's device sizes, such as transistor width, length and biasing, are chosen in a fashion to meet specifications. Then, a layout for the design is created. A layout is basically a set of overlapping polygons, where specific shapes represent specific types of "placed" components and "routed" interconnects and give precise specifications of how to fabricate the design. Layout is labeled "back-end" design, and the steps preceding are "front-end" design. Layout used to be done by manually spreading out polygons on a large surface, and taping down shapes until completion of the layout – "tapeout". That's not practical modern, complex designs, so computers used instead, outputting the industry-standard "GDS-II" format of mask instructions. Those instructions are sent for fabrication. Fabricated chips are tested, then shipped as products, where they are typically integrated as part of an overall system such as a cell phone.

Of course, things can go wrong, which causes re-loops to earlier steps in the flow. For example, if a designer cannot meet the target specifications in the sizing step, a new topology will be considered. If issues are caught at the detailed layout stage (e.g. parasitics), then backtracking to sizing is needed. If the fabricated chips fail the key tests, then a costly "re-spin" is required which involves re-entering the design loop at the front or back end. Sometimes goals are even changed, if the original specifications are too aggressive. Models can change often, as they are refined with increasing knowledge of a given process node.

Design of highly complex chips can fit into variants of this flow, where the front- and back-end design steps are organized into a *hierarchical design methodology*.

1.2.3 Handling Complex Chips via Hierarchical Design

In a hierarchical design methodology, the overall design is decomposed into design of subblocks, and those are further decomposed. Then the blocks are designed according to a hierarchical traversal scheme such as bottom-up or top-down constraint driven [Gie2005].

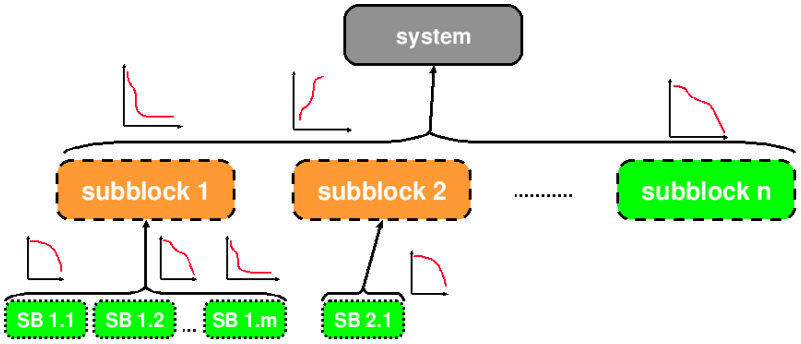


Figure 1.4: *The multi-objective bottom-up (MOBU) hierarchical design methodology.*

In the multi-objective bottom-up approach (MOBU) [Eec2005], a Pareto-optimal performance surface for each lowest-level block is determined using multi-objective optimization. The Pareto-optimal surfaces are used to constrain the design space for the blocks one level up, where more multi-objective optimization is performed to generate Pareto-optimal performance surfaces at that level. This continues until the top level, as Figure 1.4 illustrates.

In a top-down constraint-driven methodology (TDCD) [Cha1997]x, the top level is designed first, which results in specifications for each of its sub-block. Then, each sub-block is designed to meet those specifications, and results in specifications for sub-sub-blocks. The process repeats until lowest-level blocks are designed, as Figure 1.5 illustrates. Finally, the circuit is verified in a bottom-up fashion. The challenge in TDCD is how to model the feasibility regions going downwards. It could be with manually-created models, bottom-up generation of models, use the Pareto-optimal surfaces of MOBU, or otherwise [Gie2005, Gra2007].

Many variants of hierarchical design methodologies exist, as [Gie2005] surveys. With the knowledge that an appropriate hierarchical methodology can be deployed, we can once again focus on how to design at an arbitrary node within the hierarchy (Figure 1.3).

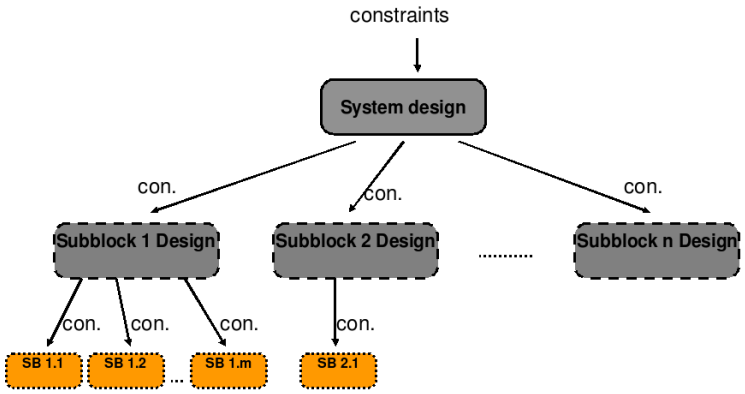


Figure 1.5: *The top-down step of the top-down constraint-driven (TDCD) hierarchical design methodology.*

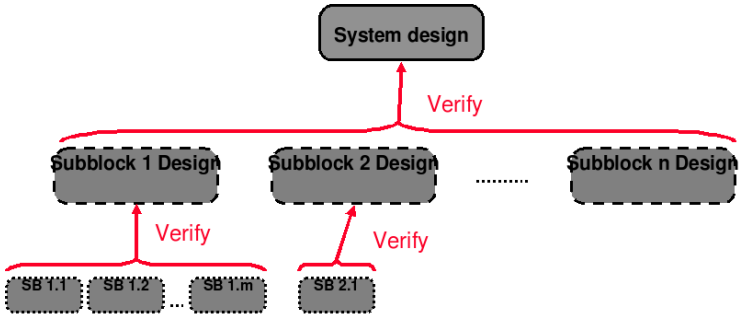


Figure 1.6: *The final verification step step of the TDCD hierarchical design methodology.*

1.2.4 Systematic Analog Design

The sizing step in Figure 1.3 can be handled in a methodical flow that emphasizes the designer's experience [Gie1990, Lak1994]. The approach is to write down equations, set reasonable/safe values for some parameters, then systematically set constraint values until all degrees of freedom are removed (therefore setting device sizes and biases).

The equations include first- and second-order transistor models, Kirchoff's laws, and equations that roughly estimate circuit performance as a function of transistor parameters. Places to set reasonable values without overly constraining the design include: values for gate overdrive voltage v_{od} for each transistor, relative allocation of currents to each current branch according to the overall current (power) budget, and setting device lengths to the process minimum.

Constraint-setting starts with the known specifications; for example if gain is specified > 60dB then the "gain" variable is set to 60 dB, which reduces the degrees of freedom in the overall set of equations. Goals to minimize / maximize are left until last. If just one goal is left, the goal can be maximized analytically (assuming convex). If > 1 goal is left, a manual tradeoff analysis can be performed. SPICE is used for verification of the design, which we now discuss.

1.2.5 SPICE in the Design Flow

The SPICE circuit simulator was introduced the 1970s [Nag1973, Nag1975]. SPICE takes in a "netlist" describing the design's topology, device sizes, bias sources, and device models, and outputs a computational result according to the analysis type. Common analysis types are dc, ac, and transient. Dc analysis reports the dc bias conditions of each device. Ac analysis reports the frequency behavior of the circuit according to small changes in the signal. Transient analysis reports the dynamic, possibly nonlinear behavior over time. The device models allow SPICE simulation to accurately capture the differences in behavior from different circuit fabrication processes.

SPICE is crucial in the design flow for engineers to verify their sized topologies. It also opened the opportunity for a new design style for sizing: an iterative loop in which designers tweak device sizes then re-run SPICE to see the change in performance. Finally, computers have become fast enough that automatic calls to SPICE from an automated design engine, "in the loop", is feasible.

SPICE has proven to be so important that development of improved SPICE simulators continues to this day. Improvements can be in terms of (a) higher speed or greater capacity, such as [Snps2008a, Cdn2008d, Men2008a]), or in terms of (b) enhanced functionality such as RF/noise analyses [Snps2008b, Cdn2008d] or supporting behavioral models / mixed-signal design [Cdn2008e].

1.2.6 Beyond SPICE: Other Industrial Analog CAD Tools

Other tools besides SPICE are key to analog design. Waveform-viewing tools such as [Snps2008c] allow designers to visually examine SPICE outputs, i.e. the simulated dynamic behavior of circuits' voltages and currents, or the results of sweeps over different

parameter values. Schematic editors such as [Cdn2008f, Snps2008f, Men2008e] allow designers to enter circuit topology and sizing information in a visual, intuitive fashion; with automatic translation to netlists to simulate the circuit in SPICE.

Layout editors such as [Cdn2008f, Snps2008e, Men2008f] allow engineers to manually convert a "front-end" design (sized schematic or netlist) into a "back-end" design, a layout. Layout-versus-schematic (LVS) tools like [Cdn2008h, Men2008c, Snps2008g] allow the engineer to cross-reference the front-end design and back-end design to catch discrepancies. Design rule check (DRC) tools like [Cdn2008h, Men2008b, Snps2008g] report if a layout violates any foundry-defined layout constraints. Resistor-capacitor extraction (RCX) tools like [Cdn2008j, Men2008d, Snps2008h] close the front-to-back loop: they extract a netlist back from layout. They are useful because a pre-layout front end design does not have accurate information about unwanted "parasitic" resistances, capacitances, and inductances that occur naturally among wires, devices, and ground.

After layout, a designer can tune an RCX-extracted netlist to soften the negative performance effects of parasitics. Some designers use schematic-driven layout (SDL) tools, where the schematic and the layout are designed simultaneously. This is especially useful when the relative effect of parasitics is large, such as in RF design, low-noise design, or ultra deep submicron design.

Data about the semiconductor process that is relevant to design is stored in a process design kit (PDK). A PDK includes the SPICE model files as well as the layout rules. All these tools, plus PDKs, are typically organized into a "design environment" tool like [Cdn2008g, Snps2008d, Men2008e], which facilitates switching among tools and sharing data among tools. More recently, data-sharing has been simplified by industry-standard databases like OpenAccess [Si22008].

1.2.7 Tool Categories and Design Automation

Over the years, these tools have gained broad usage in the designer community because of their almost universally-accepted utility. They can be placed into the following categories:

- *Analysis tools* (e.g. SPICE, DRC, LVS, RCX)
- *Insight tools* (e.g. waveform viewer, schematic viewer)
- *Design-altering tools* (e.g. schematic editor, layout editor, SDL)

There is a fourth category. *Design automation* tools directly change a design to meet design goals. Design automation tools can be further decomposed into *front-end design automation* tools (for automated topology selection, topology design, sizing), and *back-end design automation* tools (for automated device generation, placement, and routing) [Rut2002]. These tools are arguably the hardest to develop, yet have potentially high potential payoff in terms of reduced time-to-market, lower power, lower area, higher yield, and greater performance in general. Digital EDA tools have successfully deployed automation since the late 1980's. Their success is best illustrated by the vendors' revenues [Edac2006].

Because of the challenge and potential payoff, automated analog design tools have attracted the bulk of analog CAD research [Gie2002a, Rut2007]. The popularity of design automation tools in the literature probably explains why the label "analog design automation" (*only* fourth category) is sometimes used to describe all tools in "analog computer-aided design" (all four tool categories).

1.2.8 Design Automation Tools: Adoption Criteria

Adoption of analog design automation tools has been varied. It is a function of how much pain the designer / customer has with the status quo (i.e. if the problem matters), how much improvement the design automation technology brings, and, importantly, how designer-friendly the tool is. The tool's "friendliness" does not only mean a good GUI and low setup time. It also means that the tool acknowledges the designer's creativity and experience [Wil1991]. The tool should allow the designer to leverage those strengths, to remove the tedious work, while explicitly *not* removing the designer's control.

The axes of problem pain, improvement to status quo, and designer-friendliness can guide where certain design automation tool proposals have worked, and others have not. For example, modern automated layout device-generation tools solve their target problem in a fast and reliable fashion, automating a task that was considered tedious and unchallenging. The same is true for point-to-point automated routers. Many designers (or layout engineers) have actually been *willing* to defer all the layout work to automated tools, but traditionally the tools have never solved the problem well enough: automatically-generated layouts were often worse than hand-crafted layouts, or the time to set up the tool was unacceptably long. Perhaps the most recent batch of automated layout tools will finally meet this target.

1.2.9 Front-End Design Automation Tools in Industry

As for front-end design, there have been many failures and some mild successes, as reviewed in [Gie2002a, Rut2007, Mar2008]. Tools that did not use SPICE in the loop or SPICE calibration were too inflexible, and research using them waned (e.g. [Har1992])[1]. Topology selection tools have always had too much setup effort and covered too few topology choices (e.g. [Koh1990]). In general, the "knowledge-based" research approaches of the 1980s waned because of the painfully long setup times (weeks or more) which needed repeating with each new process node.

Research proposals for open-ended structural synthesis tools (e.g. [Koza2003]) take in possible components and automatically "invent" the connectivity and sizings to meet the design goals. These tools fail the adoption criteria on multiple counts. First, existing technology is nowhere near solving the problem well enough because the tools are too slow, and the non-standard results are not trustworthy. Also, "friendliness" gets a zero because the tools aim to be "creative" in topology design which designers justifiably perceive as a direct threat. We will discuss these tools more in chapter 5.

[1] One exception is the posynomial-template flow of Magma Design Automation [Mag2008], formerly of Sabio Labs and Barcelona Design, based on [Her1998]. However, it remains unclear whether this will become more widespread.

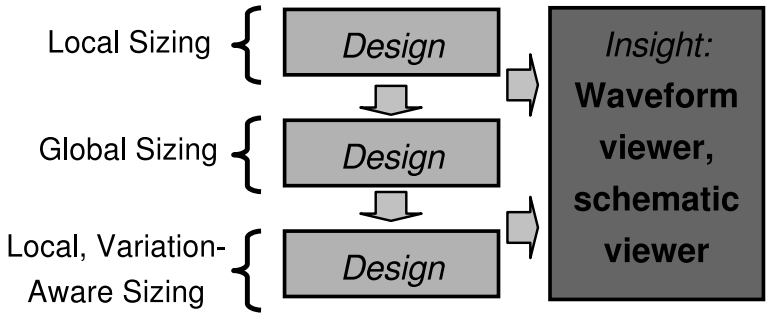


Figure 1.7: *Evolution of front-end analog design automation tools: current state.*

Figure 1.7 describes the evolution of front-end analog design automation tools which have had some degree of adoption in industry, and promise to have further adoption. The common threads are (a) SPICE-in-loop, and (b) for the sizing task. SPICE-in-loop means that the tools can be applied to whatever circuits that can be simulated, which enhances flexibility. The application to sizing is reasonable, as designers do not consider the automation of sizing to be a major creative threat. A challenge is to make the tools fast enough, and so the bulk of research in SPICE-in-the-loop sizers is on designing efficient algorithms.

Of the industrial sizers, first were local optimizers, on nominal or process-voltage-temperature (PVT) corners. They were packaged within simulators like Eldo^{TM} [Men2008g] and HSPICE^{TM} [Snps2008a] from the late 1980s onwards. However, their payoff was often too small to justify the up-front setup effort needed.

The global nominal optimizers followed around 2000, including Analog Design Automation's Creative Genius^{TM} (now Synopsys Circuit Explorer^{TM} [Snps2005]), and Neolinear's NeoCircuit^{TM} (now Cadence Virtuoso NeoCircuit^{TM} [Cdn2005b]). These had better GUIs for shorter setup time, and had higher potential payoff because the global exploration could avoid the need for initial sizings and circuit performance could be improved more. However, one cannot underestimate how quickly experienced analog designers can size circuits; so while there has been a degree of adoption, demand has not been strong.

We have already discussed the issue of process variations [Nas2001]. Analog designers have actually been handling process variations in various ways for a long time. For example, differential topologies are a direct response to avoid the effect of global process variations. Local process variations cause mismatch between differential devices that are intended to have symmetrical behavior. Mismatch has been a limiting factor in many designs for a long time, but recently local process variation has become significantly larger and it is getting worse, as Figure 1.1 showed. Accordingly, performance of analog circuits is threatened. Analog designers have a choice: live with the performance degradation, design more directly for mismatch, or increase area. Performance degradation is unacceptable when there are fixed specifications. Some designers do design for

mismatch, e.g. with architectures that do compensation, or using calibration. But many analog designers do not want to spend their precious time fixing the mismatch nuisance, so they simply make device areas larger. This of course recovers performance, but hurts area and chip cost – sometimes significantly when mismatch is high. While excessive mismatch is a nuisance for designers, it is an *opportunity* for analog CAD tool developers: here is a problem that designers do not want to worry about (or have time to worry about!), and if a tool allows them to bypass those concerns, they might adopt it.

Accordingly, as shown in Figure 1.7, there are recent commercial offerings for local *yield* optimization [Mun2008, Ext2008].

For a baseline of tool inputs, outputs, setup times, runtimes, computational requirements that are acceptable by designers, these industrial sizers can serve as a useful reference.

1.2.10 Motivation for Knowledge Extraction

While adoption of automated design tools is slowly increasing over the years, the vast majority of designers continue to do front-end design manually, to the chagrin of hopeful CAD researchers and managers. If research in automated design is to have a larger impact, the tools need to be adopted more broadly.

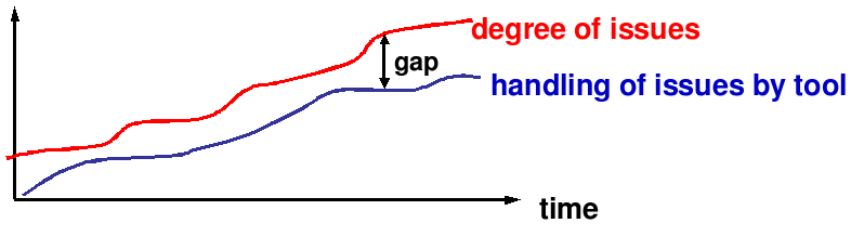


Figure 1.8: *Unforeseen issues mean that tools are perenially behind the issues that designers need to solve (from [Mcc2006b]).*

So, it is important to understand why manual design persists. Key earlier impediments to adoption were the long setup times, hard-to-use GUIs, and long turnaround times for getting results. But with recent tools, those issues have been at least partially mitigated. However, a more fundamental issue remains: risk exposure. When designers use automated design tools, they risk losing control and insight into their designs. If the automated tool cannot solve the problem, the designer needs to have the insight to know how to fix it. Key tasks do not even have industrial tools yet, such as designing or selecting topologies. And even if every task had an automation alternative, some issues are not noticed until one has to design around them, such as with the recent issue of well proximity effects [Dre2006]. Therefore, there are always gaps between what tools can do and what is needed, as Figure 1.8 shows. The key way for designers to manage this persistent tool shortcoming is to maintain *insight*.

The next question is: how well can designers maintain their insights? The challenge is that the design problem keeps changing: every 18 months there is a new process node

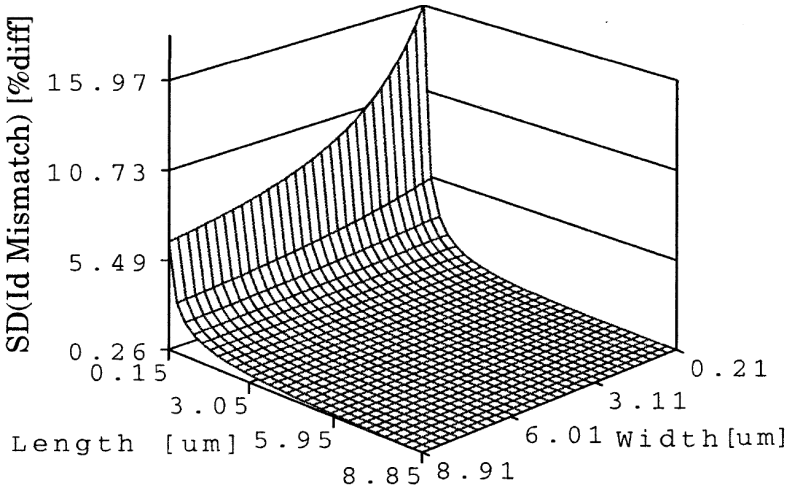


Figure 1.9: *Current mirror mismatch in a 0.13μm technology, as a function of length and width (from [Dre2003]).*

[Itrs2007], where device and therefore circuit behavior is different. Or, there are demands for new types of circuit functionality in which past insights can help little. Robustness issues in particular can kill designer insights. For example, the traditional rule of thumb for mismatch is that mismatch is proportional to 1/WL, so increasing W or L will give proportionate decrease in mismatch. However, the relationship is more complex than that. Figure 1.9 shows a current mirror and its associated mapping from W and L to mismatch. For a large W and L mismatch is low. But note how decreasing W to a tiny value will barely hurt mismatch; and decreasing L only starts to have an effect when the smallest values are hit. Once in the region of small L and very small W, then the mismatch response takes off dramatically. So, the traditional mismatch-WL insight is less valid for modern processes.

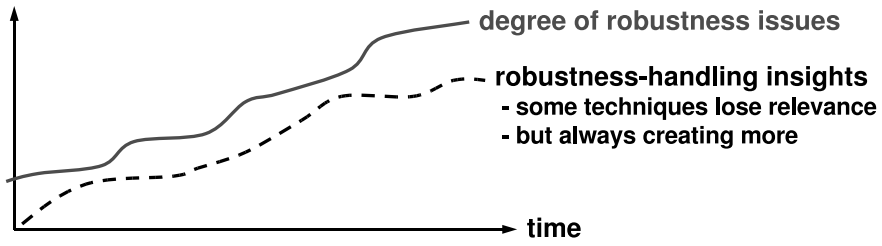


Figure 1.10: *There is an "insight gap" between the time when a robustness issue becomes significant, and when the necessary insights to resolve it are developed (from [Mcc2006b]).*

The general trend is shown in Figure 1.10: the number of robustness-related issues is increasing, and keeping insights up to date is an ever-increasing challenge.

CAD tools can help. Conceptually simple tools to visualize raw data, such as waveform viewers and schematic viewers, are a start. However, we can go much further: we

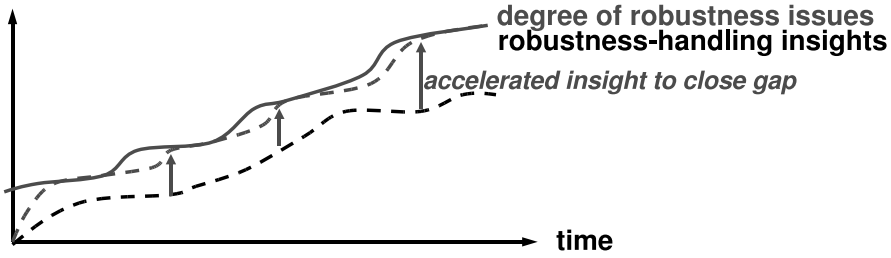


Figure 1.11: *The aim of knowledge extraction is to accelerate insight in order to close the "insight gap" (from [Mcc2006b]).*

can use automation to extract knowledge from analog circuit data, such as SPICE simulation databases. Such *knowledge extraction* tools can help to catalyze the designer's insight-gathering process, leaving him better equipped to solve unforeseen issues. As Figure 1.11 shows, accelerated insight via knowledge extraction can close the insight gap.

1.2.11 Contributions to Analog CAD

Figure 1.7 outlined the evolution of the first three front-end design automation tools: local sizing, global sizing, and local variation-aware sizing (local yield optimization). This thesis proposes and implements four new steps in the evolution of front-end analog design automation, as illustrated in Figure 1.12. The previous three and four new steps collectively cover a range of use cases. Each case is defined by the task's inputs and outputs, and the scope and difficulty of the task. In each case there is not only an optimization component, but also a *knowledge extraction* component. Knowledge extraction helps the designer to *bridge manual techniques with automation-aided techniques*. Furthermore, it helps him to build insight and maintain control over increasingly challenging designs.

A first future step of the roadmap is "Globally Reliable Variation-Aware Sizing". The automated sizing component, SANGRIA, addresses the cases of: (a) when an initial sizing may be in a local optima, (b) there is no decent initial sizing available, and (c) when the best possible yield or performance margin is needed / desired. The knowledge extraction tool, CAFFEINE, generates *whitebox* models which map design variables to performance or Cpk [Nist2006], with near-SPICE accuracy.

A second future step in the roadmap is "Trustworthy Structural Synthesis," using a tool called MOJITO. This approach is to do topology selection across a set of *thousands* of possible topologies – a set so massive that the label "structural synthesis" is appropriate. These topologies remain *trustworthy* so that designers can feel safe when committing them to silicon. The massive topology count, trustworthiness, and computational feasibility are accomplished by the use of pre-specified hierarchical analog circuit building blocks. Because the blocks are structural-only, they only need to be specified once ever (at relatively low cost), and are portable across process nodes. When a multi-objective algorithm searches through this library, it generates a database of sized topologies that collectively approximate the Pareto-optimal tradeoff among performances. Once generated for

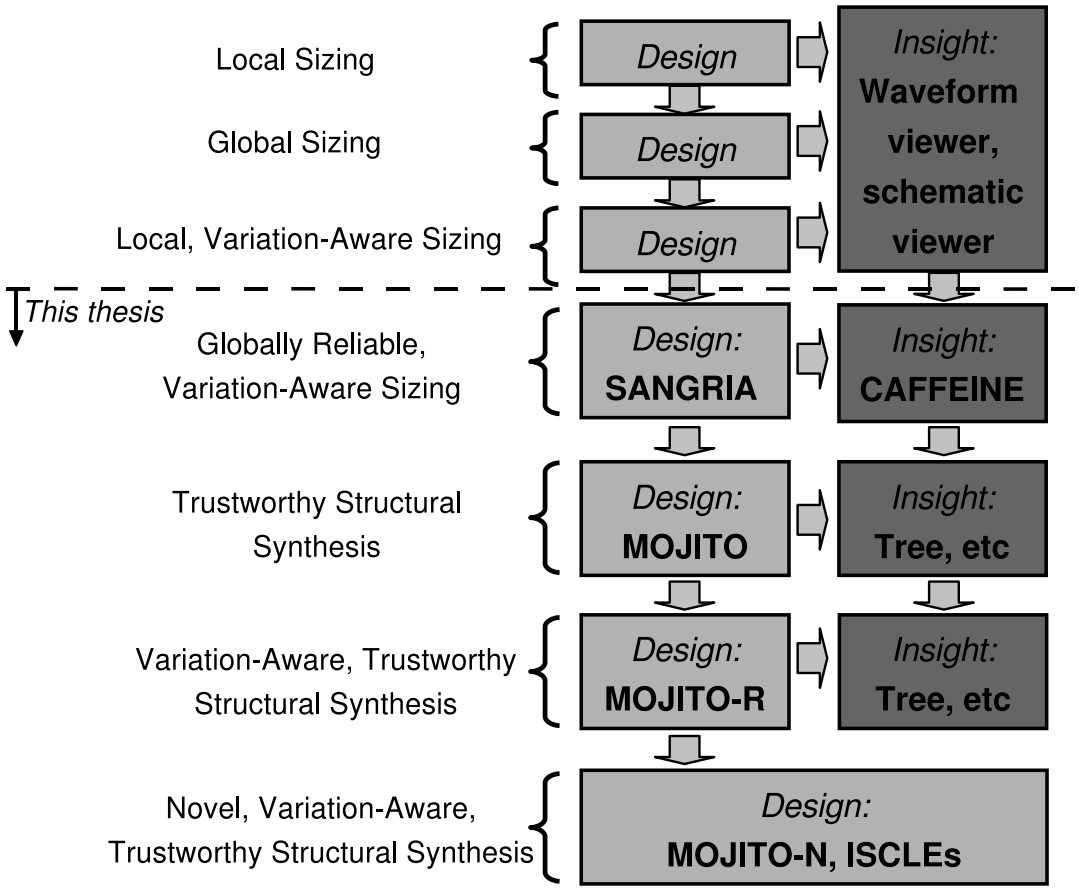


Figure 1.12: *Proposed roadmap in evolution of front-end analog design automation tools (and thesis contributions).*

a given process, the database supports an immediate-turnaround "specifications-in, sized-topology out" flow. This is particularly useful for jellybean IP – simple, non-aggressive, off-the-shelf designs where just a solution is needed, nothing fancy. This tool also supports a flow where the designer wants to try out some new building block ideas within the context of a larger circuit (or one of many larger circuits). The insight portion of the "Trustworthy Structural Synthesis" task the opportunities opened up by this new type of database, which relates topologies, sizings, and performances. It extracts (a) a decision tree for navigating from specs to topology, (b) global nonlinear sensitivities on topology and sizing variables, and (c) analytical performance-tradeoff models across the whole set of topologies.

The next future step in the roadmap is "*Variation-Aware,* Trustworthy Structural Synthesis." The use cases from Trustworthy Structural Synthesis carry over, but with an additional dimension of having *yield* as an objective. MOJITO-R and the revised knowledge extraction approaches perform accordingly.

The final step in the proposed roadmap is "*Novel,* Variation-Aware, Trustworthy Structural Synthesis." MOJITO-N starts with trustworthy structural synthesis, but adds novelty in a trackable fashion. It is useful when a designer / team have exhausted topology ideas,

are willing to take a chance, and are open to topology suggestions from a computer. MOJITO-N is also potentially useful for an young, ambitious semiconductor company (or grad student) that is willing to take some risks to try potentially odd new topologies in order to build up IP or new circuit functionality. ISCLEs takes a different approach: it uses recent advances from machine learning in a novel circuit design technique where the circuits actually *exploit* variation. It has promise to return circuits with area that shrinks as minimum device sizes shrink (i.e. area that scales with Moore's Law).

1.3 Background and Contributions to AI

Whereas the previous section described the thesis' background and the thesis' contributions from the perspective of analog CAD, this section describes the background and contributions from the perspective of artificial intelligence (AI).

1.3.1 Challenges in AI

Sometimes an approach to solve a problem can be overly restrictive, limiting its ability to generalize to other domains. the idea of using *computer programs* to generalize from previous approaches. For example, Kolmogorov proposed to use computer programs as a general way to measure complexity, with the notion of "algorithmic complexity" [Cov2007].

Closer to our interests here, Arthur Samuel proposed using computer programs as a way to describe the goals of AI: "How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?". Later, Samuel re-posed these questions, giving a criterion for success: "The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."

John Koza used these aims as a basis for his work on genetic programming (GP) [Koza1992, Koza2004c]: "Genetic programming (GP) is an evolutionary computation (EC) technique that automatically solves problems without requiring the user to know or specify the form or structure of the solution in advance. At the most abstract level GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done." [Poli2008].

Unlike traditional optimization programs which traverse a vector-valued search space, GP traverses a space of *computer programs* or structures (e.g. trees, graphs, etc). Thus, GP is perhaps the most generalized approach to AI. It is certainly very powerful: even in his original 1992 book, John Koza systematically went through *dozens* of different AI sub-problems, and solved each with GP [Koza1992] with small setup effort and virtually no hints on "how" to solve the problem.

1.3.2 GP and Complex Design

GP has emerged as a vibrant research field and many researchers have enjoyed significant success, especially on problems that can be posed as *structural synthesis* problems.

For example, [Lohn2005] evolved an antenna design for NASA which was successfully deployed in space, [Spe2004] evolved novel quantum circuit designs, and [Kor2007] evolved stock market models which are in daily use for trading. These comprise a few of the 60 *human-competitive* results that genetic programming has achieved since the early 2000s ([Poli2008], ch. 12).

Despite this success, there is still a broad range of structural synthesis problems that GP has not solved successfully enough for widespread industrial use. This is illustrated by what's still done almost wholly manually: design of automobiles, engines, airports, music of appreciable complexity, software with even a modest degree of complexity, and trustworthy analog circuit topologies.

1.3.3 Building Block Reuse in GP

GP researchers have long held the view that *reuse* might be important in approaching challenging problems: "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" [Koza2004a]. There have been a variety of GP approaches for achieving reuse, which we now briefly review.

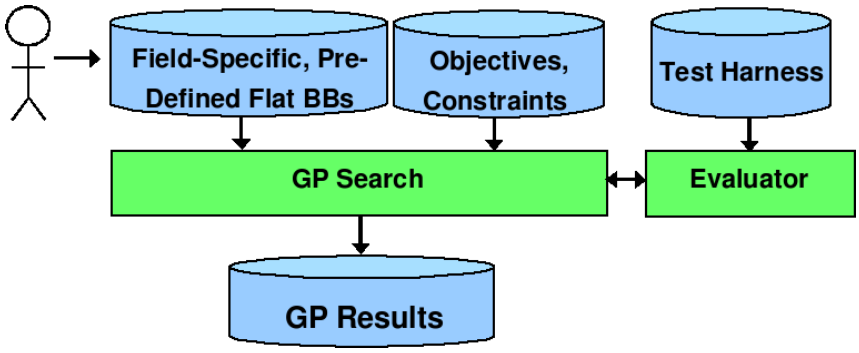


Figure 1.13: *GP reuse: "vanilla" GP implicitly has flat BBs, and no special BB-handling steps.*

First is "plain vanilla" GP; i.e. GP systems based on [Koza1992]. In vanilla GP, the leaf nodes in the tree search ("terminals") are, in fact, building blocks (BBs). They are field-specific, pre-specified by the user, and are flat (i.e. there is no pre-specified hierarchical relationship). GP "functions" are field-specific BBs too, but they too are flat since they do not define a hierarchy of connections. The flow is shown in Figure 1.13. The inputs are "field-specific, predefined, flat BBs" (terminals and functions), objectives and constraints, and a test harness used in conjunction with the evaluator to measure performance of a given design candidate (individual). GP search uses feedback from the evaluator to evolve its design candidates, and eventually returns the resulting design candidate(s).

Some GP approaches do automatic discovery of hierarchical BBs prior to the main GP search, as shown in Figure 1.14. In the first step of the flow, some routine uses

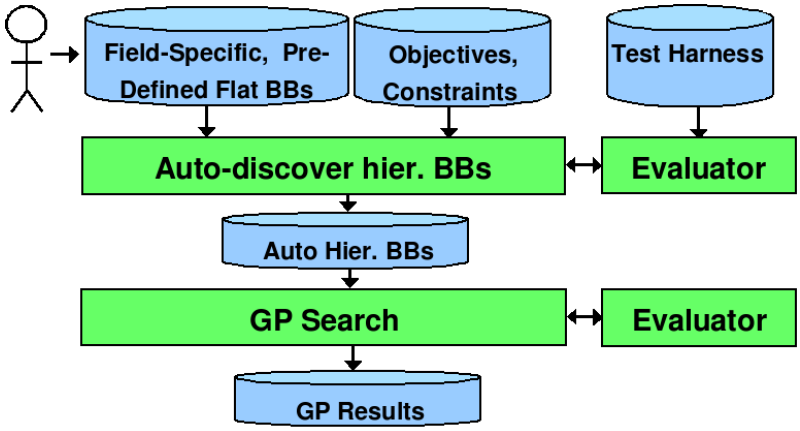


Figure 1.14: *GP reuse: seeding GP with* a priori *auto-discovered BBs.*

lightweight heuristics to automatically extract potentially-useful BBs. For example, in [Kor2006] which does symbolic regression, the first step is to evolve functions with just 1 input variable at a time, and save the best results as BBs. In [Au2003], the authors enumerate through all combinations of tiny sub-functions and keep the most promising ones as BBs. "Cultural algorithms" [Rey2008] automatically discover decision trees in this first step. Once the BBs are discovered, they are input to the GP system.

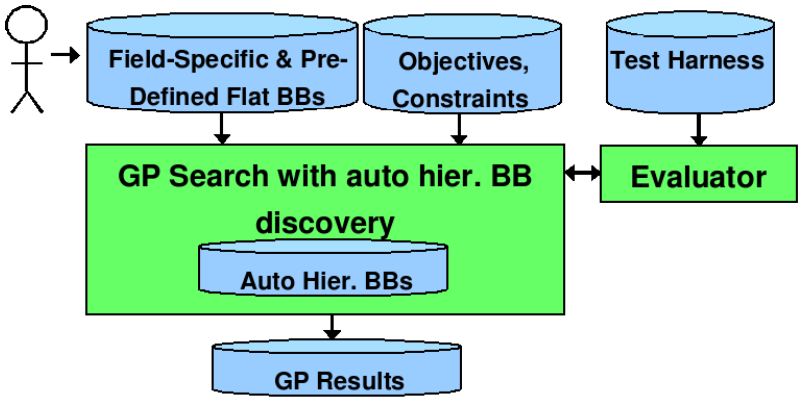


Figure 1.15: *GP reuse: automatically discover and re-use hierarchical BBs during the GP run.*

Figure 1.15 shows a flow where hierarchical BBs are discovered and used within GP search, but not output. This has been a much-explored research theme within GP, with the main aim of speeding up GP search. Approaches include: automatically defined functions (ADFs) [Koza1994], automatically defined macros (ADMs) [Spe1995], generative representations such as GENRE [Hor2003], and stack-based GP like Push [Spe2005]. Several of these approaches illustrated orders-of-magnitude speedup and/or improvement in final objective function value.

Of course, if hierarchical BBs are discovered during a run, it is conceivable that they might be helpful to give future runs a head start, i.e. closing the loop by using past

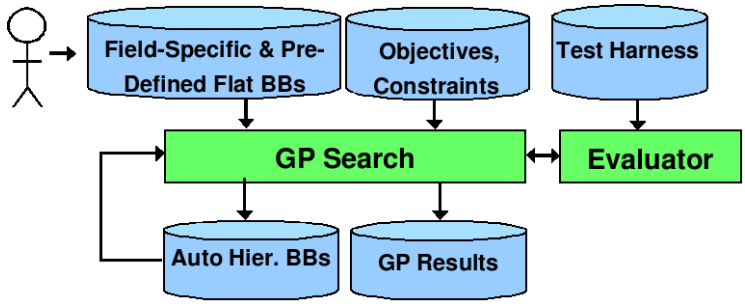


Figure 1.16: *GP reuse: closing the loop with Run Transferable Libraries (RTLs) of BBs.*

problem-solving knowledge to bias future GP searches. This idea was demonstrated with the approach of Run-Transferable Libaries (RTLs) [Ryan2005], as shown in Figure 1.16. After 50 iterative GP runs, the effort needed to solve a problem would be reduced by 2-3 orders of magnitude. The libraries demonstrated an ability to generalize as well. Using a library from simpler problems as input, similar larger problems were attacked and demonstrated similar performance improvements.

1.3.4 Contribution to AI: Complex Design

Despite this ample research into reuse within GP, the design of *complex structures* from automobiles to trustworthy analog circuit topologies remains elusive. Scientific discovery [Kei2001], in which the aim is to identify equations to describe behavior of unknown systems, has a similar "complexity wall." Upon closer inspection, we see that these problems are different from the GP successes in several key ways:

- Designs have >5-10 sub-components (e.g. 20, 100, 500).
- The designs may have modularity, regularity, and symmetry, but they also have substantial heterogeneity.
- Crucially, *most of the design's components, sub-components, sub-sub-components, etc have been long established in the design's respective field.*

For example, in automotive design there are nuts, bolts, wheels, rubber, tires, axles, cylinders, pistons, camshafts, engines, chassis, handles, windows, doors, steering wheel, seats, hood, body, and so on all leading up to the top level of a car. In biology / bioinformatics there is DNA, nuclei, mitochondria, ribosomes, cells, tissues, organs, and so on all leading up to an animal. In analog circuit design there are resistors, capacitors, transistors, current mirrors, differential pairs, cascode stages, input stages, output stages, loads, and so on all leading up to a top-level circuit such as an operational amplifier or data converter.

In fact, it is this collection of *hierarchically-composed* BBs which partially *defines* each respective field. First, it is standard knowledge and terminology shared by practitioners in the field. Second, *development of new BBs*, improvements on existing BB functionality, and gaining further understanding of existing BBs, are all considered *contributions* to each respective field.

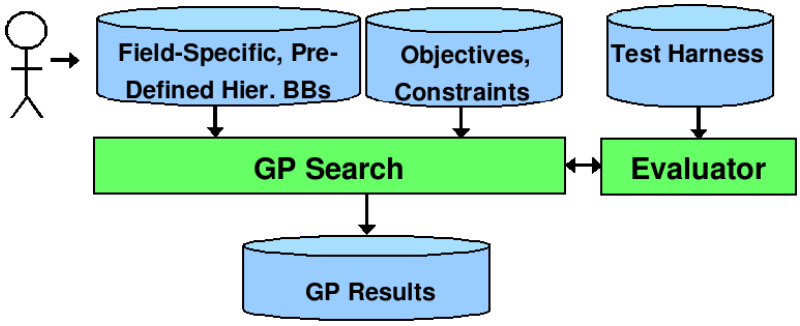


Figure 1.17: *GP reuse: enabling complex design via field-specific, hierarchical, pre-specified BBs.*

We have seen that GP research acknowledges the importance of reuse, and has proposed several reuse-oriented flows. Yet when it comes to applying GP to a particular problem domain, the field's standard, shared domain-specific knowledge is being *ignored*. The result is that GP runs often produce odd-looking designs with unrecognizable sub-blocks, using an excessive amount of computer effort. We argue that this knowledge does not need to be left on the table:

If we use domain knowledge *aggressively* spending the up-front effort to specify the field's known structural building blocks, then we can synthesize complex designs that are *trustworthy by construction*, using dramatically lower computational effort.

The proposed flow is shown in Figure 1.18. Note how the human-input BBs have the required characteristics of: *field-specific*, *pre-defined*, and *hierarchical*. This thesis will describe in detail how such a flow is applied to analog topology synthesis in the MOJITO tool. Systems of even greater complexity are possible by combining MOJITO with hierarchical design methodologies (section 1.2.3).

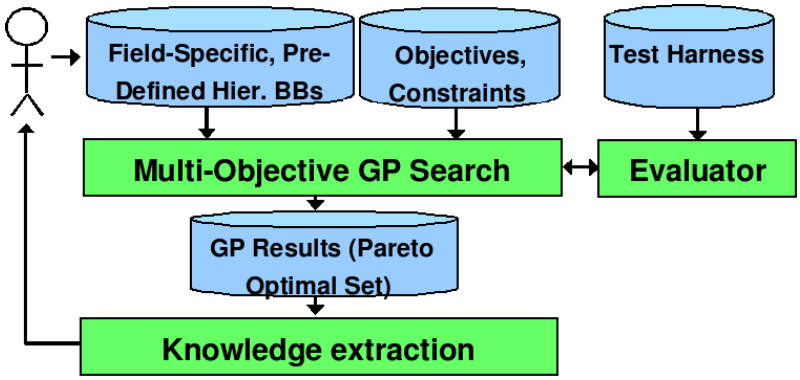


Figure 1.18: *GP reuse: closing the complex design loop via knowledge extraction to give the user new insights for improving the field-specific, hierarchical, pre-specified BB library.*

Finally, we ask how we might *close* the loop for field-specific, hierarchical, pre-specified BBs. Since BBs have to be specified by the user, this means that the user needs to be in the feedback loop. Therefore, the user must gain *insight* into the *relation between structure and performance*. This thesis emphasizes the use of *knowledge extraction* from GP run results. Knowledge extraction returns information in forms that are immediately recognizable by the user, and are transferable to other problems. This adds to the user's knowledge of the field, and closes the loop because it is guidance for trying new BBs. Figure 1.18 illustrates this. This flow can be useful even for the automation-averse designers: the designers who embrace automation can perform the synthesis and extraction, and provide or publish the outputs for their automation-averse colleagues.

1.3.5 Other Contributions to AI

The previous section discussed how this thesis contributes to the AI problems of (a) structural synthesis of complex systems, and (b) how users can use knowledge extraction from synthesis runs to make progress in their field. Besides that, this thesis makes two other contributions of interest to AI: (c) SANGRIA, an efficient approach to global statistical optimization (stochastic programming) which gracefully handles high-dimensionality search spaces, and (d) CAFFEINE, an approach for whitebox regression that gives human-interpretable models and outperforms several other state-of-the-art regression algorithms.

1.4 Analog CAD Is a Fruitfly for AI

This chapter has discussed how the semiconductor industry is important for the progress of IT, and how variation-aware analog design is important for the progress of semiconductors. It has also been discussed how AI is also important for the progress of IT, and how GP is important for the progress of AI. In fact, analog CAD and AI are related more directly: if a fruitfly is considered to be a convenient, representative test problem playing surrogate for a grander goal, then we believe that analog CAD makes an excellent fruitfly for AI. This is because:

- Analog circuit design has current and future industrial relevance, being a field within the massive semiconductor industry and having a continuous stream of design challenges due to changing semiconductor processes [Itrs2007] and performance requirements.
- Candidate designs can be evaluated (to an extent) using off-the-shelf circuit simulators.
- There can be multiple constraints and multiple objectives.
- There are robustness issues such as process variations, environmental variations, and there is an industrial incentive to address them.
- Analog circuit topology design is considered a *creative* endeavor [Wil1991]: designers refer to themselves as "artists", and new topologies are often published in the scientific literature and/or patented.

The combination of these characteristics makes analog circuit design a relevant, challenging application domain for testing AI approaches in general. This is especially true when simultaneously considering the issues of variation-aware design, designer-trustworthy results, and aiding the creative design process. Evenven subsets of those challenges are interesting because of their notable industrial relevance.

Recall the analog design flow of Figure 1.3. This can actually be generalized to a broad set of problem domains, as shown in Figure 1.19. Designing an analog topology generalizes to designing a structure. Sizing and biasing generalizes to designing parameters. Doing layout translates to specifying the implementation, and circuit fabrication generalizes to manufacturing. Just like in analog circuit design, errors can occur in the flow, at which point backtracking is needed.

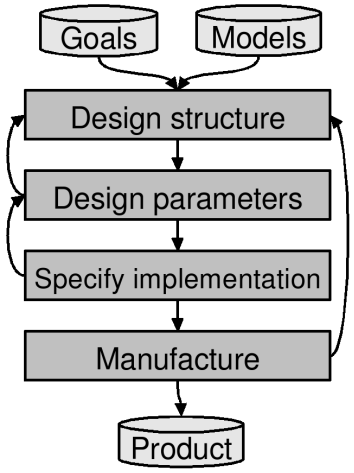


Figure 1.19: *Generalization of circuit design flow of figure 1.3 to include many AI / engineering design domains.*

1.5 Conclusion

This introductory chapter has discussed the importance of information technology (IT), the role that variation-aware analog CAD and GP can play for IT, and has discussed how this thesis contributes to each. The rest of this thesis will elaborate in several ways.

This thesis is broken into three groups of chapters which cover increasing problem scope. Chapters 2-4 cover **robust sizing & insight**: background, a proposed globally-reliable yield optimizer (SANGRIA), and knowledge extraction for robust sizing (CAFFEINE). The second group, chapters 5-8, covers **structural synthesis & insight**: background, a proposed trustworthy synthesis search space and algorithm (MOJITO), and knowledge extraction for the topology-performance-sizings relationship. The third group, chapters 9-10, cover **novel, robust structural synthesis extensions & insight**: robust structural synthesis (MOJITO-R), knowledge extraction from MOJITO-R results, novel structural synthesis (MOJITO-N), and finally novel, robust structural synthesis (ISCLEs) which is also a promising approach for analog designs to scale with Moore's Law .

Chapter 2

Variation-Aware Sizing: Background

A small leak will sink a great ship.
–Benjamin Franklin

2.1 Introduction and Problem Formulation

2.1.1 Introduction

Chapter 1 discussed how the progress of Moore's Law leads to increased process variation, which leads to significantly reduced circuit yields [Itrs2007]. Yield optimization / design centering can play a useful role within a statistical design toolset, to speed a designer's overall robust-design effort. Accordingly, there has been much research effort aimed at building analog circuit yield optimizers. This chapter examines the problem, and reviews past approaches.

This chapter is organized as follows. Section 2.1.2 gives the problem formulation, followed by industrial-level specifications in section 2.1.3. Section 2.2 reviews past approaches, starting with a baseline direct Monte Carlo and proceeding through several more advanced algorithms. Section 2.3 concludes.

2.1.2 Problem Formulation

The aim of yield/robustness optimization is to find a circuit design $\boldsymbol{d}^*$ that meets performance requirements, despite manufacturing and environmental variations. One is given a design space D, process parameter space S with distribution $pdf(\boldsymbol{s})$, environmental space Θ and measurable performances with associated specifications $\boldsymbol{\lambda}$.

We now elaborate. The aim is to find a vector-valued design point $\boldsymbol{d}^*$ that maximizes the objective f, which here is a statistical robustness estimator:

$$\boldsymbol{d}^* = \underset{\boldsymbol{d} \in D}{argmax}\{f(\boldsymbol{d})\} \tag{2.1}$$

where design space $D = \otimes_{i=1}^{N_d}\{D_i\}$. Each variable's design space D_i can be continuous-valued or discrete-valued, i.e. $D_{i,cont} = \{\Re^1 | d_{i,min} \leq d_i \leq d_{i,max}\}$ or $D_{i,disc} = \{d_{i,1}, d_{i,2}, \dots\}$.

Design variables include transistor widths W, transistor lengths L, resistances R, capacitances C, biases, and multipliers M[1]. The range for each variable is determined by technology process constraints and the user's setup.

The objective f is typically yield, Y, the percentage of manufactured circuits that meet all performance constraints across all environmental conditions. Specifically, Y at design point $\boldsymbol{d}$ is the expected proportion E of feasible circuits δ across the distribution of manufacturing variations $pdf(\boldsymbol{s})$:

$$Y(\boldsymbol{d}) = E\{\delta(\boldsymbol{d},\boldsymbol{s})|pdf(\boldsymbol{s})\} = \int_{\boldsymbol{s}\in S} \prod_{i=1}^{N_g} \delta_i(\boldsymbol{d},\boldsymbol{s}) * pdf(\boldsymbol{s})\boldsymbol{ds} \tag{2.2}$$

where possible manufacturing variations $S = \Re^{N_s}$ include variations in oxide thickness t_{ox}, substrate doping concentration N_{sub}, etc. These can be on a per-device level (local), and / or across the circuit or wafer (global). For an accurate model, both local and global variations must be modeled. Section 5.5.3.2 describes manufacturing variations further. $\boldsymbol{s}$ describes the variations in a single manufactured design, i.e. "process corner". Therefore the tuple $\{\boldsymbol{d},\boldsymbol{s}\}$ is an instance of a manufactured design.

δ_i is the feasibility of instance $\{\boldsymbol{d},\boldsymbol{s}\}$ at constraint i. It has a value 1 if feasible and 0 if infeasible:

$$\delta_i(\boldsymbol{d},\boldsymbol{s}) = I(g_{wc,i}(\boldsymbol{d},\boldsymbol{s}) \leq 0) \tag{2.3}$$

where $I(condition)$ is an indicator function which 1 if $condition$ is True, and 0 otherwise. The quantity $g_{wc,i}$ is the worst-case constraint value across possible environmental conditions Θ:

$$g_{wc,i}(\boldsymbol{d},\boldsymbol{s}) = \underset{\theta\in\Theta}{min}\{g_i(\boldsymbol{d},\boldsymbol{s},\boldsymbol{\theta})\} \tag{2.4}$$

where $\Theta = \{\Re^{N_\theta}|\theta_{j,min} \leq \theta_j \leq \theta_{j,max}; j = 1..N_\theta\}$. Environmental variables include temperature T, power supply voltage V_{dd}, and load resistance R_{load}. $\boldsymbol{\theta}$ describes a particular set of environmental conditions, i.e. "environmental corner".

For a wholistic view of the yield optimization problem, we merge equations (2.1) to (2.4):

$$\boldsymbol{d^*} = \underset{\boldsymbol{d}\in D}{argmax} \left\{ \int_{\boldsymbol{s}\in S} \prod_{i=1}^{N_g} I(\underset{\theta\in\Theta}{min}\{g_i(\boldsymbol{d},\boldsymbol{s},\boldsymbol{\theta})\} \leq 0) * pdf(\boldsymbol{s})\boldsymbol{ds} \right\} \tag{2.5}$$

Note how this is a min-max optimization formulation, where the outer objective is a stochastic quantity.

Each constraint g_i corresponds to a performance specification λ_i which has an aim and a threshold, and can be directly translated into an inequality constraint. For example, $\lambda_1 = \{power \leq (1e-3)W\} \mapsto \{g_1 \leq 0; g_1 = power - (1e-3)\}$, and $\lambda_2 = \{gain \geq 60dB\} \mapsto \{g_2 \leq 0; g_2 = -gain + 60\}$. $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, ..., \lambda_{N_g}\}$.

[1]These are variables at cell level in a sizing-driven variable formulation. Variables will be different at system level, or another formulation such as operating-point driven in section 6.4.

Performances can be measured by industry-standard SPICE-style circuit simulation, equations, or other means. SPICE simulates a text-based "netlist", which can be generated from a "testbench", design, process point, and environmental point. A testbench ξ specifies how to extract ≥ 1 performance measures at a given circuit design, process point, and environmental point. It includes current / voltage generator circuitry as stimuli into the design, load circuitry, ≥ 1 analysis instructions (e.g. ac, dc, transient with associated parameters) which return scalars or waveforms, and ".measure" statements which extract scalar performance values from the waveforms and other scalars. Testbenches are typically specified by the tool provider's "template" testbenches or the designer. All testbenches taken together are $\boldsymbol{\xi} = \{\xi_1, \xi_2, \ldots, \xi_j, \ldots, \xi_{N_\xi}\}$, and measure all performances $\boldsymbol{\lambda}$.

The environental space is actually testbench-dependent: $\Theta_j = F(\xi_j)$. For example, some testbenches may have loads that other testbenches do not have. Each testbench ξ_j often has a representative set of environmental corners $\widehat{\Theta_j} \approx \Theta(\xi_j)$ where $\widehat{\Theta_j} = \{\boldsymbol{\theta_{j,k}}\}, k = 1..N_c(j)$. $\widehat{\Theta}$ is usually set *a priori*, either by the designer (typical) or via computation.

Each testbench ξ_j measures a subset of all circuit performances: $\boldsymbol{\lambda_j} = \boldsymbol{\lambda}(\xi_j) = \{\lambda_{j,1}, \lambda_{j,2}, \ldots\} \in \boldsymbol{\lambda}$. On process point i, testbench j, with the testbench's k^{th} environmental corner and l^{th} constraint, we have scalar performance $\lambda_{i,j,k,l}$ and corresponding scalar constraint $g_{i,j,k,l}$.

"Process capability" (Cpk) [Nist2006] is alternative to yield as a statistical robustness estimator to optimize (objective f). Cpk simultaneously captures the worst performance's spread and margin above/below its specification. Therefore, Unlike yield, Cpk can distinguish between two designs having yield of 0%, or between two designs having (estimated) yield of 100%.

Cpk is defined as the worst case Cp_i across all constraints:

$$Cpk(\boldsymbol{d}) = \min_{i \in 1,2,\ldots,N_g} \{Cp_i(\boldsymbol{d})\} \tag{2.6}$$

where Cp_i is:

$$Cp_i(\boldsymbol{d}) = (E(g_{i,wc}(\boldsymbol{d})) - 0)/(3 * \sigma(g_{i,wc}(\boldsymbol{d})) \tag{2.7}$$

where E is the expected value of $g_{i,wc}$ across $\boldsymbol{s}$, and σ is the corresponding standard deviation. The numerator captures how the design compares to the specification (= 0 here), on average; and the denominator captures spread.

2.1.3 Yield Optimization Tool Requirements

To be appropriate for industrial use, a yield optimizer must be accurate, efficient, scalable, flexible, and globally reliable. We now elaborate.

Accuracy: For dependable results, the optimizer must not make any oversimplifying assumptions that can lead to large error margins. That is, the model of the yield problem must be correct:

- Results (evaluation of objective) must be SPICE-accurate.
- Uses an accurate statistical modeling approach, such as [Dre1999], even if that means having $10x$ more process variables.
- Handles environmental variations (changes in temperature, load, etc). If treated as corners, there are typically $N_c(j) = 5 - 10$ corners per testbench ξ_j, though there can be 50 or more.
- Given that circuit performances $\boldsymbol{\lambda}$ are random variables themselves, the shape of $pdf(\boldsymbol{\lambda})$ is unknown. We cannot assume a Gaussian distribution, or make assumptions about the correlation structure among performance variables λ_i.
- The objective function $f(\boldsymbol{d})$ can be nonconvex or discontinuous. There is no free access to derivatives.
- We cannot make assumptions about the nature of the feasibility region $\{\boldsymbol{d}|\delta(\boldsymbol{d}, \boldsymbol{s}) > 0\}$. For example, it may be disjoint.

Efficiency: The optimizer must run fast enough to be useful in an industrial context.

- Return good results overnight, or over a weekend for biggest problems. And of course, faster is even better.
- Exploits parallel computing if available.
- Can refine results; i.e. can make small incremental improvements to $f(\boldsymbol{d})$ via typically small changes to $\boldsymbol{d}$.

Scalability: The optimizer must be effective for circuits that have only a few devices, up to circuits with hundreds of devices. It should also allow extension to support system-level design, in the context of a hierarchical design methodology, to extend to thousands of devices or more.

- It must handle $N_d = 10 - 200$ or more design variables
- It must handle $N_g = 5 - 50$ general performance constraints, plus up to hundreds of device operating constraints; therefore N_g can be ≥ 100.

Flexibility:

- Design variables may be any combination of continuous and discrete / integer / binary
- The optimizer must readily adapt to new types of circuits, e.g. going from opamps to bias generators to system-level designs.
- The optimizer must painlessly adapt to changes in SPICE models, including statistical models. Therefore, as models improve the accuracy of the results can improve without significant new setup time each time. This also allows for easy process migration.
- Continues to do meaningful optimization once yield of $\approx$ 100% is reached

Global / Reliability:

- The user should not need to worry about whether or not the optimizer needs "restarting".
- The optimizer should not get stuck in local optima.
- The user should not need to supply a starting point. However, if the user does supply one, the optimizer can use it as a hint.
- If the supplied initial design point has yield of 0%, the tool will still optimize.
- In a given run, the optimizer will eventually converge to the best solution[1]. (And, practically speaking, in less than infinite time.)

2.1.3.1 Local vs. Global Optimization

This section highlights the importance of *global* statistical optimization.

Many approaches assume a flow of (a) get initial sizing by searching across global design space, evaluating on nominal, then (b) starting at an initial sizing, perform yield optimization with a local optimizer, i.e. "yield tuning".

Unfortunately, this approach misses out on yield / performance opportunities. Figure 2.1 shows a simple yield optimization problem setup, where the nominal performance is a multimodal function of W1. Process variations are modeled by simply adding a Gaussian-distributed random variable to W1, leading to a mapping of W1 to yield as shown in Figure 2.2 bottom.

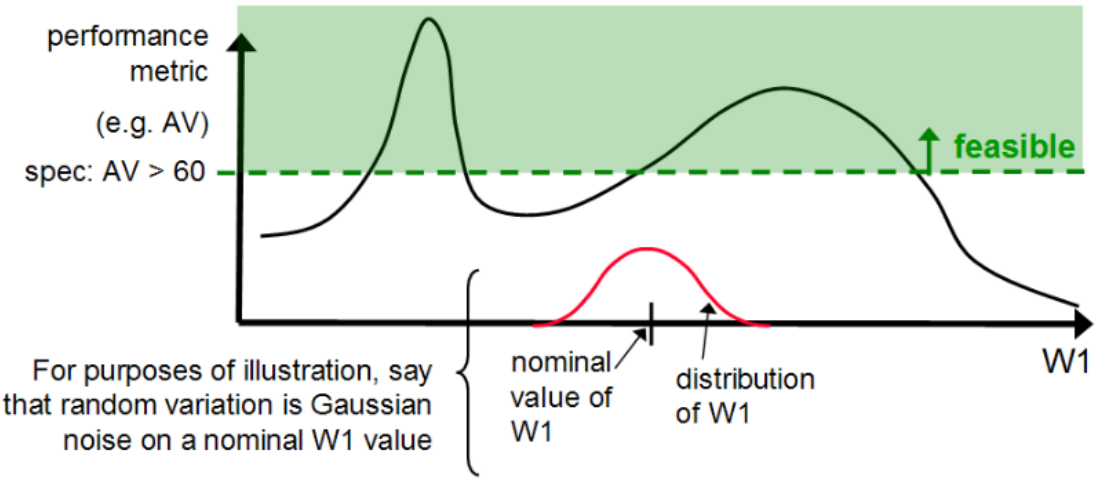


Figure 2.1: *A simple yield optimization problem setup.*

If a local yield optimizer is given the nominal optimum point as the starting point, it will return that as the optimum yield point as well, rather than the true optimum point which is at a very different value of W1. The problem can actually be far worse, when design variables and process variables do not have such a simple additive relationship towards performance.

[1]With a caveat: there is still a chance that "eventually" takes an impractically long time.

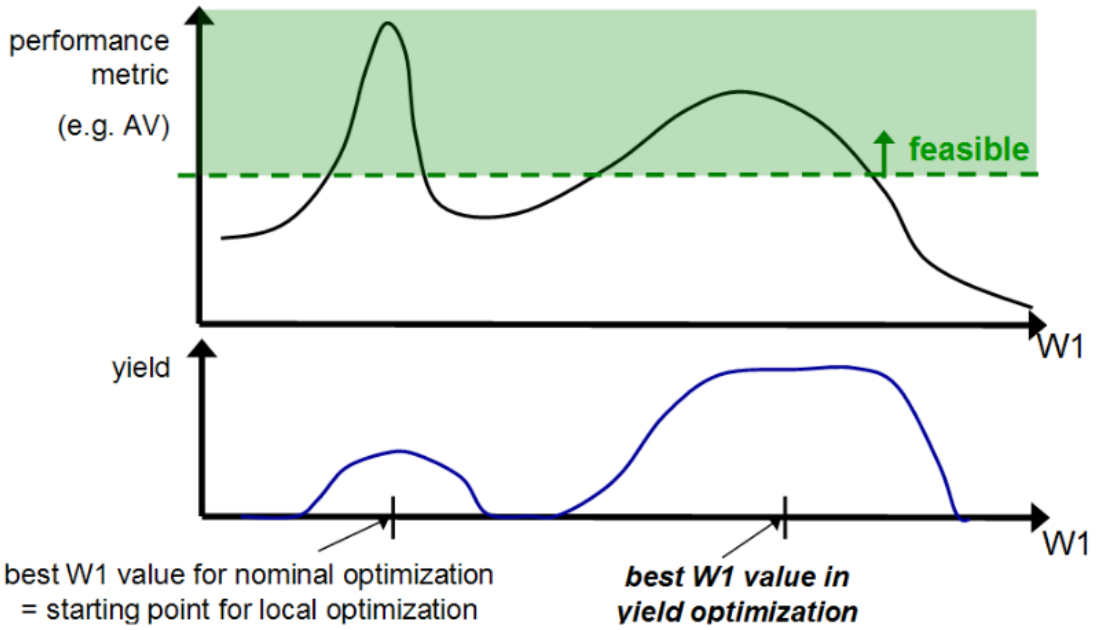


Figure 2.2: *Multimodality in performance space can lead to multimodality in yield space and / or disjoint feasible regions. In this conceptual example, the global nominal optimum will lead to a local optimum for yield.*

This example also illustrates the sensitivity of the problem to the actual value of the performance specification: at some values the problem is unimodal, and at other values the problem is multimodal. In practice it will be very difficult to determine beforehand what the nature of the search space is like.

In sum, constraint sensitivity makes it surprisingly easy to create local optima in a yield optimization problem. More importantly, we saw that a *global nominal optimum* could readily lead to a *local yield optimum*. If we do not want to compromise optimality, i.e. we want to hit the *global yield optimum*, then the search algorithm cannot merely do global nominal followed by local yield optimization.

2.2 Review of Yield Optimization Approaches

2.2.1 Direct Monte Carlo on SPICE

This can be considered a "baseline" approach, because Monte Carlo (MC) sampling is understood by both designers and CAD engineers, and typical commercial optimizers use SPICE in the loop [Cdn2005b, Snps2005]. The general idea is to to evaluate a candidate design point $\boldsymbol{d}$ by doing a full MC sampling on $pdf(\boldsymbol{s})$, invoke SPICE simulation on each $\boldsymbol{s}$ (and environmental corner), compute performances from the simulation results, then to estimate yield $\widehat{Y_{MC,sim}}$.

An outer-loop yield optimization algorithm explores the design space D:

$$\boldsymbol{d^*} = \underset{d \in D}{argmax}(\widehat{Y_{MC,sim}}(\boldsymbol{d})) \tag{2.8}$$

where $\widehat{Y_{MC,sim}}$ is estimated with MC sampling and simulation.

In MC sampling, N_{MC} process points i are drawn from the process distribution $\boldsymbol{s_i} \sim pdf(\boldsymbol{s})$. Simulation is done at design point $\boldsymbol{d}$, for each process point $\boldsymbol{s_i}$, for each testbench ξ_j, for each environmental point $\boldsymbol{\theta_{j,k}}$, giving performance vectors $\boldsymbol{\lambda_{i,j,k}}$ and corresponding constraint-value vectors $\boldsymbol{g_{i,j,k}}$.

From the simulation data, $\widehat{Y_{MC,sim}}$ is the average estimated feasibility $\hat{\delta}_i$ across samples:

$$\widehat{Y_{MC,sim}}(\boldsymbol{d}) = \frac{1}{N_{MC}} * \sum_{i=1}^{N_{MC}} \widehat{\delta_i} \tag{2.9}$$

where $\widehat{\delta_i}$ is the feasibility of sample $\boldsymbol{s_i}$. It is computed across testbenches $\{\xi_j\} \forall j, j = 1..N_\xi$:

$$\widehat{\delta_i} = \widehat{\delta}(\boldsymbol{d}, \boldsymbol{s_i}) = \prod_{k=1}^{N_\xi} \widehat{\delta_{i,j}} \tag{2.10}$$

where $\widehat{\delta_{i,j}}$ is the feasibility of sample $\boldsymbol{s_i}$ on testbench ξ_j. It is computed across ξ_j's environmental corners and constraints:

$$\widehat{\delta_{i,j}} = \widehat{\delta}(\boldsymbol{d}, \boldsymbol{s_i}, \boldsymbol{\xi_j}) = \prod_{l=1}^{N_g(j)} I(\min_k\{g_{i,j,k,l}\} \leq 0) \tag{2.11}$$

Let us examine a typical runtime for direct MC on SPICE:

- Assume that simulation time takes 1 minute for one random instance of one circuit on all testbenches at one environmental corner.
- Assume that on average, there are $N_c = 8$ environmental corners per testbench.
- There are $N_{MC} = 50$ samples of process points.
- There are 5 simulators, each running on a different CPU.
- Therefore, the total simulation time to evaluate 1 design = (1 minute) * 8 * 50 / 5 = 80 minutes.
- It is typical for a relatively well-performing optimization algorithm to explore 1,000 - 10,000 or more designs. Let us say it examines 1,000 designs.
- Then, if we apply direct MC sampling, total optimization time is (80 min / design * 1,000 designs) = 80,000 min = 1,333 h = 55 days.

The advantage of direct MC on SPICE is simplicity and accuracy of results. The major disadvantage is runtime, which makes the approach infeasible to use in practice. Another disadvantage is the inability to refine results because of the limited number of samples possible.

There are really two versions of direct MC on SPICE, which differ with respect to whether blocking is used. In the blocking version, the 30-50 process points are only drawn once. After that, just these process points are used for the whole run. The non-blocking version draws new process points for each new design candidate. The advantage of the blocking version is that there is a better chance to refine designs because there is no noise between design candidates' objective function values. Also, if the mapping from $\{\boldsymbol{d}, \boldsymbol{\theta}, \boldsymbol{s}\}$ to performance is differentiable, then the statistical mapping, from $\boldsymbol{d}$ to the statistical estimator f, remains differentiable. The disadvantage is that the final design gets *tuned* to those 30-50 points, such that the yield estimate on them could be overly optimistic. The non-blocking version has opposite virtues: the yield estimate stays unbiased, but it is difficult to refine design candidates because the noise caused by different process points overwhelms neighbors' differences in objective function values – it breaks differentiability.

2.2.2 Direct Monte Carlo on Symbolic Models

One might consider direct MC if something faster than SPICE is used to evaluate circuit performance. In the AMGIE system [Plas2002] for example, fast-evaluating symbolic expressions were used in combination with direct MC sampling to directly optimize for the symbolic-model estimate of yield $\widehat{Y_{MC,sym}}$:

$$\boldsymbol{d}^* = \underset{d \in D}{argmax}(\widehat{Y_{MC,sym}}(\boldsymbol{d})) \tag{2.12}$$

Because circuit evaluation is cheap, $N_{MC} >> 50$ process points can be drawn. The non-blocking (unbiased) choice of points can be used, because the noise is reduced by such a large N_{MC}.

The runtime for direct MC on symbolic models is reasonable. Unfortunately, symbolic models are quite inaccurate which means that the final design is probably not optimal. Extracting the symbolic model is not always easy [Gie2002b]. Also, the final yield estimate is not accurate, though that can be mitigated by doing a post-optimization MC sampling on SPICE.

2.2.3 Direct Monte Carlo on Precomputed Regression Models

In this approach, the first step is to compute performance regression models using SPICE simulation data. In the second step, direct MC on the models is performed, to return $\boldsymbol{d}^*$:

$$\boldsymbol{d}^* = \underset{d \in D}{argmax}(\widehat{Y_{MC,reg}}(\boldsymbol{d})) \tag{2.13}$$

In detail, the flow is:

1. Do space-filling sampling across the design, process, and environmental variable space $\boldsymbol{x_i} \in (D \cup S \cup \Theta)$.

2. Do SPICE simulation at each sample point $\boldsymbol{x_i}$ to measure performances λ_i.

3. For each performance, build a regression model ψ_i mapping the $\{\boldsymbol{d}.\boldsymbol{s}, \boldsymbol{\theta}\}$ variables to performance value: $\widehat{\lambda}_i = \psi_i(\boldsymbol{x})$.

4. Optimize for yield using the models $\{\psi_i\}\forall i$ as "cheap" statistical simulators.

5. Validate the final design $\boldsymbol{d}^*$ with MC sampling on SPICE.

Space-filling sampling can be done by uniform sampling, Latin Hypercube Sampling [Mck2000], or another Design of Experiments (DOE) [Mon2004] approach.

The flow owes its origins to the DOE literature, in which it is called the Response Surface Method (RSM) and assumes use of polynomials for the regression models [Mon2004]. Based on the analog CAD literature on building regressors with input $\boldsymbol{d}$, one can imagine variants using linear models [Gra2007, Mcc2005c, Li2008c], posynomials [Dae2002, Dae2003, Dae2005, Agg2007], polynomials [Li2007b, Mcc2005c], splines [Wol2004, Mcc2005c], neural networks [Van2001, Wol2003, Mcc2005c], boosted neural networks [Liu2002, Mcc2005c], support vector machines [Ber2003, Kie2004, Ding2005, Ding2005b, Mcc2005c], latent variable regression [Sin2007, Li2008b], kriging [Mcc2005c, Yu2007b], and CAFFEINE template-free functions [Mcc2005a][chapter 4]. One could even combine regression models of performance with classification models of feasibility, as in [Ding2005b, Dae2005].

The major challenge of this approach is the large number of input variables, $n = N_s + N_d + N_\theta$, which the regressor needs to handle. Accurate models of process variations can need 8 or more random variables per device [Dre1999]. Thus, in a 100-device circuit, there can be $N_s = 8 * 100 = 800$ process variables plus the $N_d \approx 100$ design variables and a few environmental variables; i.e. $n \approx 1000$ input variables. This poses a giant problem for scaling in most regression approaches. One recent quadratic-modeling approach [Li2007b] has better scaling: it can build a model in $O(k * n)$ time where k is the rank of the matrix approximating the $O(n^2)$ coefficients of the quadratic model. However, because the quadratic's functional form is severely restricted, it cannot handle many types of nonlinearities, such as discontinuities (e.g. when a transistor switches operating regions). Section 4.5.4 will show how inaccurate quadratic-based prediction can be. There are potential alternative regression approaches which scale in $O(n)$ build time *and* handle arbitrary nonlinearities (e.g. [Fri2002]), but it is extremely difficult for *any* model to build a sufficiently predictive model in just $O(n)$ with $n \approx 1000$ variables without an unreasonable number of SPICE circuit simulations.

2.2.4 Adaptively Updated Regression Models

To overcome the issues of precomputed regression models, one idea is to do an *iterative loop* of sampling, simulation, and model building. New samples in model the input space are chosen based on maximizing the model's uncertainty[1]. This approach has various labels, including "evolutionary DOE", "active learning", and "adaptive sampling" [Wol2004, Ber2003, Ding2006]. Unfortunately, none of the approaches demonstrated an

[1] *Not* minimizing uncertainty. The aim is to find where the model is weakest, i.e. high-uncertainty regions.

ability to scale to significantly larger problems; they only gave proportional improvement compared to a non-active sampling approach.

Another approach [Dae2005, Yu2007b] is similar, except the algorithm chooses new samples by maximizing the predicted performance(s) of the circuit. The issue here is that one can readily get stuck in a local optimum, due to the model's blind spots, as section 3.2 elaborates. Furthermore, [Yu2007b] used kriging, which has poor scalability properties. A related set of approaches is density estimation from regression models, as section 2.2.12 will discuss.

Even better is an approach that accounts for both model uncertainty *and* predicted performance, as section 3.2 also discusses. Unfortunately, even that may not scale well to the 50, 100, or 1000 input variables that such an approach would need for our application.

2.2.5 FF/SS Corners

The core idea of all corners-based approaches is: if corners are "representative" of process and environmental variations, and all corners can be "solved", then the final design's yield will be near-100%:

$$\boldsymbol{d}^* = \underset{d \in D}{argmax}\left(\prod_{\Xi_i \in \boldsymbol{\Xi}} \delta(\boldsymbol{d}, \Xi_i)\right) \mapsto Y(\boldsymbol{d}^*) \approx 100\% \tag{2.14}$$

where to "solve at a corner" means to find a design $\boldsymbol{d} \in D$ which is feasible across all constraints at the corner Ξ_i, i.e. $\delta(\boldsymbol{d}, \Xi_i) = 1$. To "solve at corners" means to find a design that is feasible at all those corners $\{\delta(\boldsymbol{d}, \Xi_1) = 1, \delta(\boldsymbol{d}, \Xi_2) = 1, \dots\}$.

A corner-based problem formulation is more familiar to designers ("solve at corners"), rather than a statistical formulation, which many designers do not have deep experience with. A corner-based formulation usually means a simpler optimizer design. One can actually view direct Monte Carlo optimization (with blocking) as a corner-based approach in which a very large number of corners is naively picked, compared to other corners-based flows which try to find a smaller number of representative corners. There are many possible interpretations of "representative"; one for example is "well-spaced along on the 3-σ contours of performance space".

Typical commercial performance optimizers (e.g. [Cdn2005b, Snps2005]) as well as early-2000's academic research such as ANACONDA [Phe2000] handle both environmental and process variations as corners. Environmental variations are pre-specified as corners $\hat{\Theta}$, as described in section 2.1.2. More dimensions are added to the corners to handle manufacturing variations as "FF/SS" corners of SPICE models. FF/SS process corners (F = "fast", S = "slow" for NMOS/PMOS) are typically supplied by the foundries as part of process design kits. However, FF/SS corners are designed for digital circuits' key performances, not for analog. The SS corner $\boldsymbol{s_{SS}}$ aims to bracket worst-case circuit speed (i.e. slow speed), and the FF corner $\boldsymbol{s_{FF}}$ brackets worst-case power consumption (because fast circuits consume more power). However, analog circuits have a greater variety of performance goals, which are dependent on circuit type (e.g. opamp vs. ADC). Therefore FF/SS corners do not properly bracket analog performance variations.

The runtime of FF/SS corners is reasonable, because the number of corners is reasonable. But the optimization is not accurate because the corners are not representative of the yield optimization problem.

2.2.6 3-σ Hypercube Corners

Rather than having FF/SS corners, one could instead use the extreme $\pm$3-σ value of each process variable to construct corners. The total number of corners, if just extreme values are used, is $2^{(N_\theta+N_s)}$. Such an approach might be considered if there were few random variables. For example, $N_\theta = 3$ environmental variables and $N_s = 3$ random variables would lead to $2^6 = 64$ corners. But accurate randomness models may have 8 or more random variables *per device* [Dre1999], which would mean a 40-device circuit would have $N_s = 320$ random variables and therefore about 2^{320} corners. Furthermore, the corners are extremely pessimistic because the probability of each corner is far lower than the probabilities of most random points from a typical MC sampling. To reduce the corner count, one could do fractional factorial sampling [Mon2004], but this will remain pessimistic and may not give adequate coverage of the process space S.

2.2.7 Inner-Loop Corners

Another approach is to auto-discover some of the "meanest" corners on-the-fly for each performance, keeping the corner count low and keeping corners representative. The approach [Och1999] performs a semi-infinite programming problem: an outer optimization loop traverses the design space D on the current set of $\{\boldsymbol{s}, \boldsymbol{\theta}\}$ corners to maximize performance; periodically, it is paused and an inner optimization traverses $\{S \cup \Theta\}$ on a fixed $\boldsymbol{d}$, *minimizing* performance to find a new $\{\boldsymbol{s}, \boldsymbol{\theta}\}$ corner, which is then added to the outer loop's problem. This can be viewed as "outer approximation" of worst-case performance.

This approach is actually extremely pessimistic, because the inner optimization can find process points that are highly improbable, due to the inner objective being worst-case (as opposed to statistical). Also, the inner optimization process to discover "mean" corners is computationally expensive. This is probably why the approach was never demonstrated beyond circuits with a few transistors.

2.2.8 Inner-Sampling Corners

This approach, present in [Cdn2005c], also auto-discovers the "meanest" corners on the fly for each performance, during temporary pauses of an outer-loop optimization. But instead of a periodic inner-loop optimization, a MC sampling is performed on the best design point so far. Then, more corners are extracted from the new MC results: for each performance, the $\{\boldsymbol{s}, \boldsymbol{\theta}\}$ giving the worst performance value is added to the overall set of corners.

This is an improvement compared to Inner-Loop Corners because the inner corner-discovery is more efficient and less pessimistic. A bonus is that intermediate MC-sampled designs are returned, so that progress can be ratcheted upwards. A disadvantage is that N_g

corners are added with each inner invokation, and N_g is typically ≥ 10 so the approach quickly slows down after even just 1 iteration.

2.2.9 Device Operating Constraints

Device operating constraints (DOCs) are topology-specific constraints to ensure that devices are operating in the intended region (e.g. transistor must be in saturation), and building block behavior is as expected (e.g. currents in current mirrors must match). While this is obvious to an analog designer, it is not obvious to an optimization algorithm unless explicitly measured and enforced. DOCs have been documented in many places, most notably [Gra2001] and also described in [Plas2002, Ber2005, Ding2005b, Das2005, Mcc2006d, Gra2007], under various labels including "feasibility guidance", "operating point constraints" and "structural constraints."

DOCs by themselves are not used for yield optimization *per se*, but rather as a step to take prior to yield optimization, to get a starting design that has a higher chance of being robust. Unsurprisingly, [Gra2001] found that yield using DOCs in optimization is significantly better than yield not using DOCs. [Sch2001] shows that using them within the context of a yield optimizer will improve the optimizer's convergence. In [Ste2003, Ste2007] a faster variant of optimization under DOCs was presented: the optimal design is the maximum-volume ellipsoid bounded by linearly-modeled DOCs in design variable space.

2.2.10 Device Operating Constraints Plus Safety Margins

The authors of [Phe2000] suggested putting safety margins on DOCs, because transistors would stay in their target operating region even under slight perturbations in their operating point due to manufacturing variations. The upside is that decent yield is maintained, runtime is the same as nominal. The downside is that there is a possible reduction in circuit performance, because high-performing analog circuits often perform on the edge of their operating envelopes, and safety margins would prevent that. Also, because DOCs with safety margins are not directly optimizing on yield, yield analysis can only be done as a post-processing step; there is no clear relationship between yield and safety margins. Finally, an open issue is how to choose the size of the safety margins.

2.2.11 Density Estimation from SPICE Monte Carlo

The idea in this approach is to speed up yield estimation and/or improve accuracy at a given design point $\boldsymbol{d}$ using density estimation. The steps are: (a) take a relatively small number of Monte Carlo samples at $\boldsymbol{d}$ via SPICE simulation, (b) estimate the probability density function across the performances space, $\widehat{pdf}(\boldsymbol{\lambda})$, then (c) estimate yield $\widehat{Y_{DE}}$ by computing the volume of the pdf that is within the performance feasibility region:

$$\widehat{Y_{DE}}(\boldsymbol{d}) = E\{\delta(\boldsymbol{d},\boldsymbol{\lambda})|\widehat{pdf}(\boldsymbol{\lambda})\} = \int_{\boldsymbol{\lambda}\in\Re^{N_g}} \prod_{i=1}^{N_g} \delta_i(\lambda_i(\boldsymbol{d})) * \widehat{pdf}(\boldsymbol{\lambda},\boldsymbol{d})d\boldsymbol{\lambda} \qquad (2.15)$$

where $\delta_i(\lambda_i(\boldsymbol{d}))$ is 1 if λ_i is feasible, and 0 otherwise. If the pdf construction works well, then the yield estimates $\widehat{Y_{DE}}$ can be more accurate than yield estimation based on counting the number of feasible samples $\widehat{Y_{MC}}$. Intuitively, this works because it uses the information at a finer degree of granularity. The upper/lower confidence intervals for a density estimator's yield estimate can theoretically converge more rapidly than confidence intervals using a binomial distribution that merely counts feasibility.

There is a tradeoff based on the number of simulations: too many simulations is too computationally expensive, and too few simulations gives an inaccurate yield estimate. Unfortunately, with more performance metrics, more simulations are needed for an accurate estimate; and if we include DOCs there can be *hundreds* of performance metrics. Furthermore, estimates of $pdf(\boldsymbol{\lambda})$ make strong assumptions about the nature of the distribution. For example, the approach [Tiw2006] finds 10 random points "which make the distribution the most Gaussian". This is clearly not representative of the true distribution. Or, the approach [Li2007a] models the frequency distribution as a linear-time-invariant (LTI) system, which makes it subject to oscillation (ringing) at sharp drop-offs in density. Also, [Li2007a] could not handle multivariate distributions, though that now has a workaround in [Li2008].

2.2.12 Density Estimation from Regression Models

This approach builds regression models ψ_i mapping $\{\boldsymbol{d}, \boldsymbol{s}\}$ to each performance λ_i. Then, on a candidate design point $\boldsymbol{d}$, it does Monte Carlo sampling on those performances, simulates each model ψ_i to get performance estimate λ_i, and finally estimates yield using density estimation rather than counting feasibility. The regression models can be built beforehand, or on-the-fly (which gives this approach similar to the approaches of sections 2.2.3 and 2.2.4.

An example of this approach in practice is [Li2007b] where the regression models are quadratics built on-the-fly, and the density estimator is based on an LTI system [Li2007a].

The key problem is building accurate regression models in the $\{\boldsymbol{d}, \boldsymbol{s}\}$ input space, as discussed in section 2.2.3. Approach [Li2007b] aims to bypass the problem by restricting the search to be highly local, which has the optimality-compromising problem discussed in section 2.1.3.1. We actually have difficulty understanding the motivation of using density estimation on cheap models, because the bottleneck is in SPICE simulation, not in doing Monte Carlo sampling on regression models. Using density estimation adds complexity, and introduces the additional issues of section 2.2.11.

2.2.13 Spec-Wise Linearization and Sensitivity Updates

In this approach [Ant2000, Sch2001, Gra2007], the optimizer has a different perception of the yield optimization problem. As shown in Figure 2.3, the optimizer's goal is to find a design point $\boldsymbol{d}$ that shifts the performance contours in process space S (and therefore feasible region) to align favorably with the fixed distribution pdf($\boldsymbol{s}$).

"Favorable" can be:

- "Maximum worst-case-distance" (WCD) [Ant2000]: maximum distance from center of the pdf to the closest feasibility boundary. This can be viewed as a design centering formulation.
- "Maximum yield" [Sch2001]: maximum volume under the pdf that is in the feasible region.

The spec-wise linearized approach models each performance's feasibility δ_i as a linear classifier ψ_i: $\widehat{\delta_i} = \psi_i(\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{\theta})$. Each classifier is built from a sensitivity analysis and SPICE simulations. The linear models are concatenated to form an approximation of the overall feasibility region $\widehat{\boldsymbol{\delta}} = \bigcap_i \{\psi_i(\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{\theta})\}$. By definition, $\widehat{\boldsymbol{\delta}}$ is a convex polytope. Using $\widehat{\boldsymbol{\delta}}$, the algorithm finds a sizing that shifts the polytope approximation to align "favorably" with the fixed pdf($\boldsymbol{s}$). The algorithm then repeats with further sensitivity analyses, etc. Figure 2.3 illustrates.

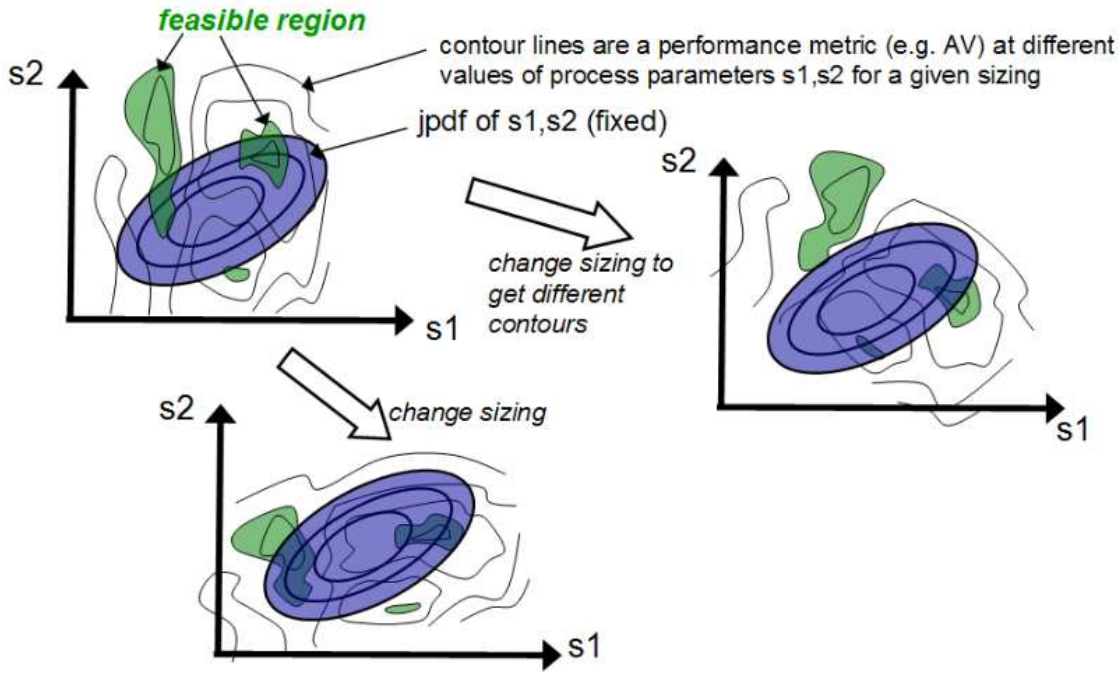


Figure 2.3: *A spec-wise linearized approach to yield optimization, in an ideal case.*

One issue is that the approach can only do local optimization due to its sensitivity-analysis nature. The assumption of linearity has some discomforting ramifications. The authors justify the linear feasibility classifiers with (to paraphrase) "in a DOC-compliant space, circuits are either linear or weakly nonlinear" [Ant2000] which draws on their results from 1994 [Ant1994]. 1994 is certainly *several* process generations ago. To test the assumption's validity on *modern* processes, we performed an experiment using TSMC 0.18μm models. We used two circuits shown in Figures 3.7 and 3.21 – a 10-transistor amplifier and a 50-transistor amplifier, respectively. For each circuit, we picked "reasonable" sizings that met the DOC constraints. We took 2 * N_d latin hypercube samples

[Mck1979, Mck2000] in a union of the $\{\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{\theta}\}$ space, where N_d is the number of input variables, each d_i could vary by 2% of its total range, and each s_i was within $\pm 3\sigma$. On each point, we simulated across 3 testbenches spanning 16 performances. For each performance λ_i of each circuit, we built both a linear model and a nonlinear model (SGB [Fri2002]; see also section 3.3). We used 75% of the data as training data, and the remaining 25% of the data for testing.

Table 2.4 shows the results, which shows the normalized root mean squared error on the testing data (see equation 3.7). We see that the linear models have average prediction errors of 19.0% for the 10-transistor design, and 18.8% for the 50-transistor design. On several performances, the linear models have errors exceeding 40%. In contrast, the nonlinear SGB models have an average error than is about half that of the linear models; and a maximum error that is also half. Section 4.5.4 has additional experiments that show how poorly linear models capture the mapping from design variables to performance.

	10 transistor opamp prediction error (15 design / 89 process / 4 env. vars)		50 transistor opamp pred. error (68 design / 340 process / 4 env vars)	
Performance	**Linear model**	**Nonlinear model**	**Linear model**	**Nonlinear model**
AV	39.1%	15.7%	7.6%	7.6%
BW	40.3%	17.8%	21.7%	13.8%
CMRR	4.8%	4.8%	10.9%	7.1%
FU	24.0%	13.1%	14.0%	11.9%
GBW	23.8%	13.1%	13.6%	10.8%
GM	25.7%	10.8%	8.7%	7.6%
IS	8.7%	5.2%	14.9%	9.9%
PM	29.3%	10.5%	16.5%	10.2%
PSRR	32.6%	12.5%	7.3%	7.3%
PWR	8.4%	4.4%	14.9%	9.5%
THD	16.9%	7.5%	23.4%	13.0%
OS	16.1%	11.3%	46.2%	4.3%
SR	9.2%	0.9%	41.8%	4.2%
ST	14.8%	4.8%	40.8%	3.6%
VOH	0.3%	0.3%	13.3%	6.1%
VOL	10.0%	3.0%	5.2%	3.4%
Average	**19.0%**	**8.5%**	**18.8%**	**8.1%**
Max	**40.3%**	**17.8%**	**46.2%**	**13.8%**

Figure 2.4: *Experimental results to test validity of linear versus nonlinear modeling.*

A final concern is that we have only seen the algorithms of [Gra2007] demonstrated on a process variation model in which threshold voltage V_{th} is assumed to be a varying process parameter. As [Dre1999] elaborates, that means that variation in the parameter β is accounted for twice. We have not seen the approach demonstrated on more accurate models of process variations, which have >10x more random variables and would therefore slow down this sensitivity-analysis-based approach by >10x.

2.2.14 Maximum Volume Ellipsoid

The maximum volume ellipsoid (MVE) approach [Xu2005] has a high-level perspective of the optimization problem similar to that of spec-wise linearization, in which the aim is to shift the performance contours to fit within an approximation of the feasibility region, $\hat{\boldsymbol{\delta}}(\boldsymbol{s})$.

Whereas the spec-wise linearization approach of section 2.2.13 modeled $\hat{\boldsymbol{\delta}}$ as a convex polytope, MVE models it as an ellipsoid. The approach aims to maximize the volume of the ellipsoid. The optimization is performed as follows: (a) different design points are sampled, each returning a different ellipsoid volume, (b) a posynomial mapping from design point to ellipsoid volume, $\boldsymbol{d} \mapsto V$, is automatically constructed, and (c) the optimal design $\boldsymbol{d}^*$ is found via geometric programming in a convex optimization formulation [Boyd2004], using $\boldsymbol{d} \mapsto V$ for feedback.

A related approach from the process control literature is polynomial chaos expansion [Nagy2006]. Here, the spec-wise linearization is generalized to higher order via either power series or polynomimal chaos. We refer the reader to [Nagy2006] for details.

The main issues with this approach are its modeling assumptions. First, an ellipsoid can be a poor approximation of the feasibility region. For example, it assumes convexity, which cannot handle disjoint feasibility regions. Second, posynomials can have poor accuracy in modeling analog circuits, as section 4.5.3 examines in depth.

2.2.15 Tradeoff Response Surface

In HOLMES [Sme2003b], the authors first generate a set of nondominated candidate designs via multiobjective optimization, ignoring statistical variations. Then, they automatically build a polynomial model mapping design variables to performances, using the nondominated designs as samples. Then, a robust optimization is done, using the polynomial models in place of simulations, and incorporating randomness. The problem with this approach is that the polynomial performance models are unaware of random variations; there is no way that the robust optimization step can reconcile this because it continues to use those performance models. In other words, the approach assumes, and relies upon, a tight correlation between nominal and robust designs. [Tiw2006] has a similar approach, with similar issues.

2.3 Conclusion

In this chapter, we have reviewed the problem formulation for yield optimization, and identified a set of requirements for an industrially useful yield optimizer. Then we reviewed the existing approaches to yield optimization.

None of the optimization approaches that we reviewed had both accuracy and reasonable efficiency. The approaches either (a) sped up the yield simulation via faster but less accurate approximations, (b) optimized on a goal that somewhat correlated with yield (e.g. worst-case distances) or (c) most commonly, made potentially dangerous simpli-

fying assumptions. Additionally, some of the optimization approaches only worked for small problems, or imposed constraints on the way circuits could be designed or modeled.

Finally, none of the approaches gave reliable global convergence in yield optimization.

The next chapter proposes our approach, which overcomes these issues to meet the target tool requirements.

Chapter 3

Globally Reliable, Variation-Aware Sizing: SANGRIA

Monte Carlo? I don't know how you design, but my workflow doesn't involve gambling.
–Anonymous

3.1 Introduction

In this chapter, we present SANGRIA: Statistical, accurate, and globally reliable sizing algorithm [Mcc2008e]. SANGRIA was designed with industrial use in mind, to achieve the yield optimizer tool requirements of chapter 2: accuracy, efficiency, scalability, flexibility, and global reliability.

To be accurate and flexible, SANGRIA uses SPICE in the loop, and an accurate model of manufacturing variation [Dre1999]. To be efficient in light of expensive circuit simulations, SANGRIA follows the mantra "every simulation is golden". The mantra is carried out with a combination of *model building optimization* (MBO) and *structural homotopy*. We now elaborate on each.

In MBO [Jon1998], regression models are built on-the-fly during each iteration of the optimization and used to propose new designs. However, as we will see, MBO algorithms in the literature have issues that need to be overcome in order to work for industrial-strength yield optimization. Part of the solution lies in the choice of regressor, which needs to have excellent scalability properties without being constained to a pre-set functional form. A recently-developed approach called stochastic gradient boosting (SGB) [Fri2002] meets these goals.

Homotopy algorithms [Noc1999] work by starting with an easy problem with an obvious solution, then gradually transforming it into the true problem while simultaneously maintaining the solution to the problem. Whereas typical homotopy algorithms tighten *dynamically* towards the true objective function, SANGRIA's "structural homotopy" embeds problem loosening into the data *structure* of the search state, in which searches at various problem difficulty levels are conducted simultaneously and linked via a recent development in evolutionary algorithms (ALPS: age-layered population structure [Hor2006]). For SANGRIA, a "full" problem does full simulation across all testbenches and {process,

environmental} corners; and loosened problems have fewer testbenches and / or corners. Therefore SANGRIA can do exploration cheaply, while promising design regions get evaluated more thoroughly. ALPS has the characteristic of periodically exploring wholly new regions of the search space, in a fashion that shelters new regions from more optimized designs; it is this characteristic that gives SANGRIA global reliability.

For shorter runtimes, SANGRIA leverages parallel processing.

In order to handle infeasible designs, and to perform meaningful optimization beyond $\approx$ 100% yield, SANGRIA optimizes on Cpk (equation (2.6)) rather than yield.

SANGRIA is benchmarked on a suite of circuit yield optimization problems, to demonstrate its accuracy, efficiency, scalability, flexibility, and global reliability.

The rest of this chapter is organized as follows. Sections 3.2, 3.3, and 3.4 review SANGRIA's foundations: model-building optimization, stochastic gradient boosting, and homotopy, respectively. Section 3.5 describes the SANGRIA algorithm in detail. Section 3.6 shows experimental results on a variety of circuits, where some circuits have hundreds of variables. Section 3.7 discusses how system-level circuits would be handled. Section 3.8 concludes.

3.2 Foundations: Model-Building Optimization (MBO)

3.2.1 Introduction to MBO

MBO [Jon1998] is an approach to optimization, in which regression models are built on-the-fly during each iteration of optimization, and used to propose new designs. MBO is a suitable choice for optimization when the cost of evaluating a design is expensive (e.g. > 10 s, and definitely when > 1 min or 1 h), so that the cost of model building and model simulation does not dominate the optimizer runtime. A typical MBO problem formulation is:

$$\begin{array}{rl} \boldsymbol{d}^* = & argmax\{f(\boldsymbol{d})\} \\ s.t. & \boldsymbol{d} \in D \end{array} \tag{3.1}$$

where f is the objective function to be maximized, and D is the bounded design space, e.g. $D = \{\Re^{N_d} | d_{i,min} \leq d_i \leq d_{i,max}; i = 1..N_d\}$. While not shown here, MBO can also apply to optimization problems with other types of constraints and with multiple objectives. MBO is similar to what we termed "adaptively updated regression models" in section 2.2.4, except in we take MBO formulation to be more rigorous, especially in its accounting for uncertainty.

3.2.2 MBO Description

The general MBO algorithm is given in Table 3.1. In line 1, design points are generated to "fill out" the design space, e.g. with uniform sampling, Latin Hypercube Sampling [Mck2000], or another Design of Experiments (DOE) [Mon2004] approach. Typically, the number of initial samples is a linear function of N_d, with a lower bound. For example, $n_{init}(N_d) = max(20, 2 * N_d)$. In line 2, each initial point $\boldsymbol{d_j}$ is simulated: $f_j = f(\boldsymbol{d_j})$.

Table 3.1: *Procedure ModelBuildingOptimization()*

Inputs: f, D
Outputs: $\boldsymbol{d}^*$
1. $\boldsymbol{D}$ = generate $n_{init}(N_d)$ space-filling points in D
2. $\boldsymbol{f}$ = simulate($\boldsymbol{D}$)
3. while stop() $\neq True$:
4. $\quad\psi$ = build model with $\{\boldsymbol{D}, \boldsymbol{f}\}$ training data; $\hat{f} = \psi(\boldsymbol{d})$
5. $\quad\boldsymbol{d_{new}}$ = optimize on ψ according to an infill criterion $\Lambda(\psi(\boldsymbol{d}), u(\boldsymbol{d}))$
6. $\quad f_{new}$ = simulate($\boldsymbol{d_{new}}$)
7. $\quad$add $\boldsymbol{d_{new}}$ to $\boldsymbol{D}$; add f_{new} to $\boldsymbol{f}$
8. $\quad\boldsymbol{d}^*$ = $\boldsymbol{d_j}$ in $\boldsymbol{D}$ with highest f
9. return $\boldsymbol{d}^*$

Line 3 begins the iterative loop. At each iteration, a new design $\boldsymbol{d_{new}}$ is proposed with the help of regression and an inner optimization (lines 4-6). In line 4, a regressor is built (e.g. a neural network). In line 5, an inner optimization is performed by simulating on the regressor at candidate design points. The inner optimization aims to find a $\boldsymbol{d}$ to balance the overall MBO's exploitation with exploration, by using an "infill criterion" objective, Λ. Specifically, Λ combines maximizing $\hat{f}(\boldsymbol{d})$ with reducing the model's blind spots, i.e. maximizing model uncertainty $u(\boldsymbol{d})$:

$$\boldsymbol{d_{new}} = \underset{\boldsymbol{d}\in D}{argmax}\{\Lambda(\psi(\boldsymbol{d}), u(\boldsymbol{d}))\} \tag{3.2}$$

where problem (3.2) is solved with an off-the-shelf single-objective optimizer such as [Yao1999]. The efficiency of this inner optimization is not of great concern here, because the cost of simulating the regressor $\psi(\boldsymbol{d}) = \hat{f}(\boldsymbol{d})$ is typically at least a few orders of magnitude cheaper than evaluating the objective function $f(\boldsymbol{d})$ through simulations.

Step 7 updates the best design so far, $\boldsymbol{d}^*$. MBO returns $\boldsymbol{d}^*$ once a stopping criterion is hit, such as the maximum number of simulations being exceeded, maximum runtime being exceeded, the target objective being hit, or convergence having stagnated.

A conceptual illustration of MBO in action is given in Figures 3.1 and 3.2. There is a single design variable $\boldsymbol{d}$, which is the x-axis. Each diamond is a training datapoint $\{\boldsymbol{d_j}, f_j\}$ from steps 1-2. The smooth curve interpolating the training points is the regressor ψ's predicted values across $\boldsymbol{d}$. Here, the regressor is a feedforward neural network with one hidden layer [Amp2002]. The infill criterion Λ is the curve of "mountains" sitting on top of the regressor's curve. Here, uncertainty is computed as merely the distance to the closest training point: $u(\boldsymbol{d}) = min\{abs(\boldsymbol{d} - \boldsymbol{d_1}), abs(\boldsymbol{d} - \boldsymbol{d_2}), \dots\}$[1].

The infill value is a weighted sum of regressor value and uncertainty:

$$\Lambda(\boldsymbol{d}) = (1 - w_{explore}) * \psi(\boldsymbol{d}) + w_{explore} * u(\boldsymbol{d}) \tag{3.3}$$

[1]Note that using distance is just *one* way to compute uncertainty; as we will see later there are better ways.

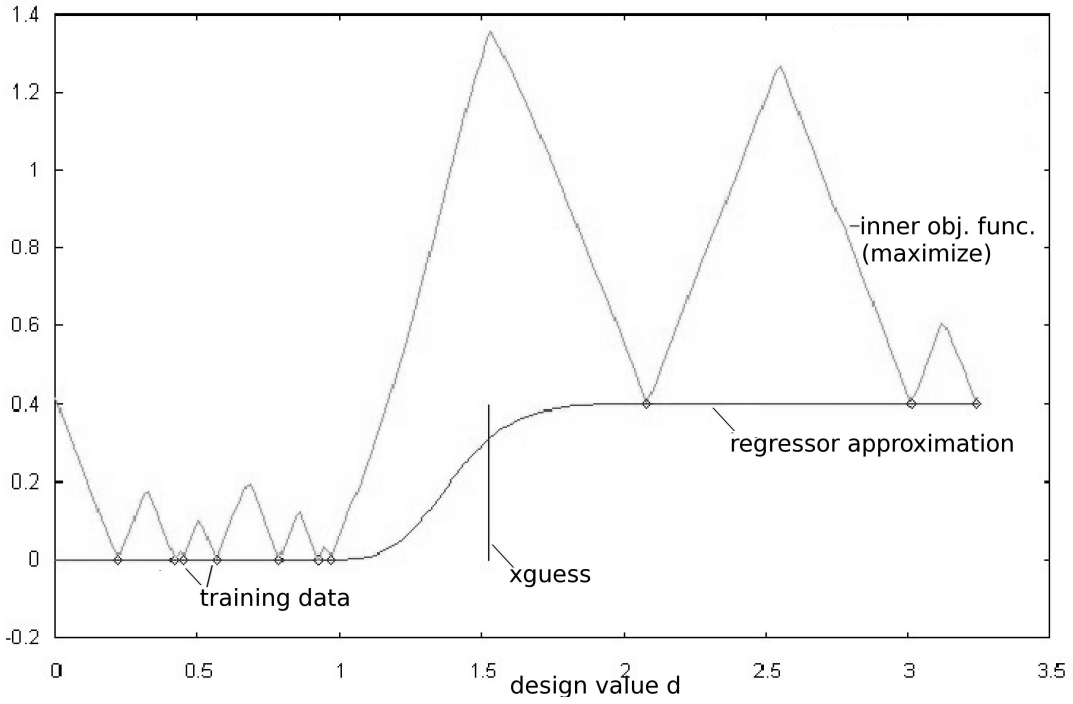


Figure 3.1: *MBO at first iteration.*

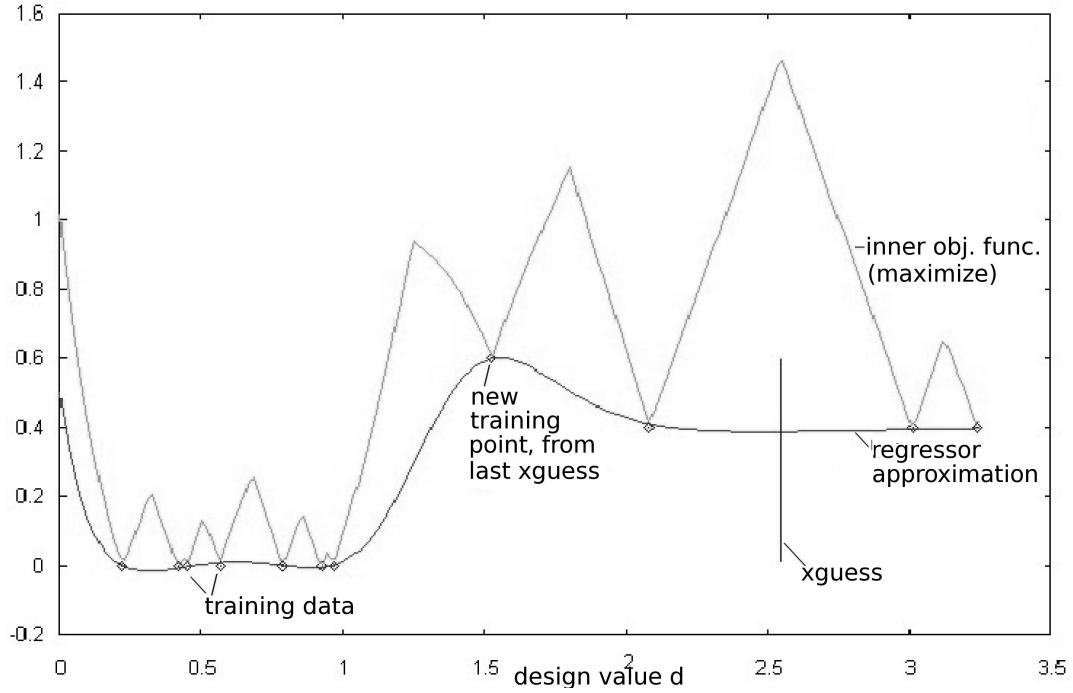


Figure 3.2: *MBO at second iteration.*

where $w_{explore}$ is the relative weight for exploration compared to exploitation; $w_{explore} \in [0, 1]$ and a reasonable choice is in $[0.2, 0.8]$. Therefore at each training point, uncertainty is 0 and the infill curve intersects with the regressor's curve (bottom of each mountain valley). As one moves $\boldsymbol{d}$ away from a training point $\boldsymbol{d_j}$, the uncertainty $u(\boldsymbol{d})$ grows, and with it the infill curve $\Lambda(\boldsymbol{d})$, until a mountain peak where the closest training point to $\boldsymbol{d}$ changes.

Line 5 of Table 3.1 chooses a new $\boldsymbol{d}$-value to maximize Λ. In Figure 3.1, this new $\boldsymbol{d}$-value is indicated by the vertical bar at $\boldsymbol{d_{new}} \approx 1.6$. Notably, $\boldsymbol{d_{new}}$ is not at the maximal value of merely ψ, but instead a combination of ψ and u to account for the model's blind spots (high-uncertainty regions). Once chosen, $\boldsymbol{d_{new}}$ is simulated on f, the training data is updated, the best point is updated, and a new model is built which incorporates the new data. That updated model is shown in Figure 3.2. Note how at $\boldsymbol{d} \approx 1.6$, the simulated value is much higher than the first model (Fig. 3.1) had predicted. This demonstrates how accounting for model blind spots via u is crucial. Now, the next $\boldsymbol{d}$ is chosen at $\boldsymbol{d_{new}} \approx 2.6$, and the loop repeats.

Many variants of MBO algorithms exist in the literature. Classical quasi-Newton-style optimization algorithms [Noc1999] can be considered as MBO algorithms, in which the regressor at each iteration is a quadratic model, the infill criterion only considers model optimality (ignoring model uncertainty), and the inner optimization has extra trust-region bounds. More recent quadratic-modeling approaches, which have less reliance on a cheaply-measured derivative (e.g. NEWUOA [Pow2006]), can also be viewed as an MBO algorithm. But one does not need to restrict the regressor to quadratics, nor the infill criterion to solely exploitation. Furthermore, without trust-region restrictions on the inner optimization, the optimizer gains the property of global search.

The paper [Jon1998] gives a particularly good account of MBO: it uses a kriging regression model which naturally reports prediction uncertainty, and an "expected improvement" (EI) infill criterion that balances exploration and exploitation. The dissertation [Sas2002] analyzes kriging-based MBO more thoroughly, testing various infill criteria approaches combining $\hat{f}(\boldsymbol{d})$ optimality with model uncertainty $u(\boldsymbol{d})$. An important lesson was that EI constructed a $\boldsymbol{d} \mapsto \Lambda$ mapping characterized by large plateaus with periodic spikes. This mapping is very difficult for the (inner) optimizer to search across.

In contrast, the "least-constrained bounds" (LCB) criterion gave better-shaped mappings, which directly led to more reliable convergence of the MBO algorithm [Sas2002]. The LCB criterion was, in fact, already presented as our example, in equation (3.3).

3.2.3 MBO Shortcomings

MBO algorithms are promising because they make maximum use of available data. However, the versions in the literature have several issues, especially in light of the yield optimizer requirements (section 2.1.3):

- **Inadequate regressors.** The typical regressor used, kriging, has terrible scaling properties: the model build time is exponential in the number of dimensions, limiting it to fewer than $\approx$ 10-15 dimensions. The regressor needs to scale well with increasing input dimension and increasing number of samples. While quadratic-based MBOs like

[Pow2006] can scale to more dimensions, they only manage to circumvent the non-flexible structure of quadratics by limiting their application to *local* search, whereas we want to do global search for reasons discussed in section 2.1.3.1.

- **Issues in uncertainty.** Most regressors do not have a natural way to compute uncertainty. Linear models do, but we need to account for nonlinear mappings. Kriging also does, but it scales poorly with input dimension. Density estimation reframed into regression also reports uncertainty, but density estimation algorithms scale poorly with input dimensionality. A *regressor-independent* fashion is to make uncertainty a function of the Euclidian distance from the closest training point(s), as the example in section 3.2.2 had described, leading to the triangle-shaped "mountains." This is fine for a few dimensions, but past 10-15 dimensions, each point is essentially far away from each other point [Has2001] which renders the Euclidean measure ineffective. Furthermore, it is unclear what $distance \mapsto uncertainty$ mapping works best.
- **Sensitivity of infill criterion.** While LCB is relatively robust compared to EI and other criteria, it still shows sensitivity to its $w_{explore}$ setting [Sas2002]. We do not want a poor $w_{explore}$ to constrain the ability to achieve the global optimum.
- **Too few samples for high-dimensional prediction.** Even if we overcome the other issues, if $N_d \geq 100$ dimensions, and the number of simulations is limited, there is simply too little data to make any meaningful prediction at all. In such cases, MBO will degenerate to random search.

3.3 Foundations: Stochastic Gradient Boosting

3.3.1 Introduction

The choice of regressor makes a crucial difference in MBO performance, as section 3.2.3 found. The regressor must be able to handle arbitrary nonlinearities, be constructed quickly, simulate quickly, scale well with high input dimension, perform well with few as well as with many training samples, and provide an uncertainty measure.

After considering several modeling approaches, we found a recent technique called stochastic gradient boosting (SGB) [Fri2002] because it hits all the criteria (except one). SGB handles arbitrary nonlinearities because it is a composed of piecewise-constant functions – an ensemble of *boosted* Classification and Regression Trees (CARTs) [Bre1984]. CARTs are recursively-constructed piecewise constant models, that divide the input space in hypercubes along variable axes. SGB constructs and simulates quickly because CARTs construct and simulate extremely quickly, and the CART-joining part of SGB (a sum) is computationally-cheap. SGB handles high input dimensionality, and few *or* many training samples because the successive-decomposition approach of CART quickly prunes dimensions and training data.

SGB hits all the criteria but one: it does not directly report prediction uncertainty. Therefore, a *bootstrapped ensemble* of SGB models, ψ, is used. The predicted output from the whole ensemble, $cost(\psi, \boldsymbol{d})$, is merely the average value across the SGB models' outputs. Uncertainty $u(\psi, \boldsymbol{d})$ is the standard deviation across the models' outputs. N_{ens}

does not need to be large, because the uncertainty only needs enough fidelity to roughly distinguish regions of uncertainty (e.g. N_{ens} = 5 is reasonable).

Due to its desirable properties, SGB is also used elsewhere in this thesis: section 8.4 uses it for global nonlinear sensitivity analysis, and the ISCLEs algorithm of section 10.4 can be viewed as an adaptation of SGB that replaces CARTs with actual circuits.

The rest of this section is organized as follows. Section 3.3.2 describes the form of SGB ensembles, SGBs, and CARTs. The remaining sections describe how to build each: section 3.3.3 describes SGB ensemble (group of SGBs) construction. Section 3.3.4 describes a single SGB model (group of CARTs) construction. Finally, section 3.3.5 describes single-CART construction.

Following that, we return to the more general discussion of SANGRIA foundations, and then SANGRIA itself.

3.3.2 Form of SGB Ensemble Model

An SGB ensemble ϕ takes the form:

$$\psi(\boldsymbol{x}) = \frac{1}{N_{ens}} * \sum_{i=1}^{N_{ens}} \psi^{(i)}(\boldsymbol{x}) \tag{3.4}$$

where $\boldsymbol{x}$ is an input vector, $\psi^{(i)}$ is the i^{th} SGB model in the SGB ensemble, and N_{ens} is the number of SGB models.

A single SGB model $\psi^{(i)}$ takes the form:

$$\psi^{(i)}(\boldsymbol{x}) = \sum_{j=1}^{N_{ens}^{(i)}} \alpha * \psi^{(j)}(\boldsymbol{x}) \tag{3.5}$$

where $\psi^{(j)}$ is the j^{th} CART model in the SGB model, $N_{ens}^{(i)}$ is the number of CART models in SGB ensemble i, and α is the weight per CART (because α was the learning rate during SGB construction).

A single CART regression model $\psi^{(j)}$ is a *piecewise constant* function, where a different constant value v is output depending on the region R that input vector $\boldsymbol{x}$ is in:

$$\psi^{(j)}(\boldsymbol{x}) = \begin{Bmatrix} v_1^{(j)} & if\ \boldsymbol{x} \in R_1^{(j)} \\ v_2^{(j)} & if\ \boldsymbol{x} \in R_2^{(j)} \\ \vdots & \vdots \\ v_{N_R}^{(j)} & if\ \boldsymbol{x} \in R_{N_R}^{(j)} \end{Bmatrix} \tag{3.6}$$

where the regions $\{R_1^{(j)}, R_2^{(j)}, \cdots\}$ are each defined by a hypercube in input space according to the CART construction algorithm. The regions are disjoint and collectively cover the whole input space.

3.3.3 SGB Ensemble Construction

An SGB ensemble is constructed by *bootstrapping* SGB models. We now give details.

Table 3.2 describes construction of SGB ensemble $\psi : \boldsymbol{X} \mapsto \boldsymbol{y}$, where $\boldsymbol{X} = \{\boldsymbol{x_j}\}, j = 1..N$ are the training inputs, and $\boldsymbol{y} = \{y_j\}, j = 1..N$ are the training outputs.

Line 1 is the outer loop of ensemble construction, encompassing choosing training data (line 2), building an SGB model (3), and updating the SGB ensemble (line 4). It repeats N_{ens} times, resulting in an ensemble holding N_{ens} SGB models.

One can draw sample lists from an unseen distribution of lists, *despite* being only given *one* list as input. This rather surprising ability is due to the technique of *bootstrapping* [Efr1979]. The implementation of bootstrapping is merely sampling with replacement. Accordingly, line 3 generates a "sample list" $\{\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}\}$ from the list $\{\boldsymbol{X}, \boldsymbol{y}\}$ via bootstrapping.

Bootstrapping has another useful property: with each sample list $\{\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}\}$, higher-order computations can be performed to get higher-order "samples". Here, our computation is to build an SGB model from the MC sample list, thereby getting an "SGB sample" $\psi^{(i)}$ (line 4; details in section 3.3.4). The final model ψ is merely a collection of SGB samples (line 5). Simulating the SGB ensemble can be viewed as further higher-order computations on the original bootstrapped data.

Table 3.2: *Procedure BuildSgbEnsemble()*

Inputs: $\boldsymbol{X}, \boldsymbol{y}$
Outputs: ψ
1. $\psi = \emptyset$
2. for i = 1..N_{ens}
3. $\quad \{\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}\}$ = N samples with replacement from $\{\boldsymbol{X}, \boldsymbol{y}\}$
4. $\quad \psi^{(i)}$ = BuildSgb($\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}$)
5. $\quad \psi = \psi \cup \psi^{(i)}$
6. return ψ

The simulation and construction of Random Forests [Bre2001] is similar to SGB ensembles, except Random Forests bootstrap-sample CARTs instead of SGB models.

3.3.4 SGB Construction

An SGB model is constructed by *boosting* CARTs in a stochastic fashion. We now elaborate.

Table 3.3 describes construction of a single SGB model. The training inputs and outputs are $\{\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}\}$, as called from *BuildSgbEnsemble()*. (For use elsewhere, the inputs can be any $\{\boldsymbol{X}, \boldsymbol{y}\}$.)

In line 1, the SGB model is initialized as an empty set. In line 2, the current target output $\boldsymbol{y_{cur}^{(i)}}$ is initialized to the overall target output $\boldsymbol{y}^{(i)}$. $\boldsymbol{y_{cur}^{(i)}}$ can be regarded as the portion of $\boldsymbol{y^{(i)}}$ which still needs to be learned (residual) . Lines 3-7 are the SGB boosting loop. At each iteration, training data is selected (line 4), a CART $\psi^{(j)}$ is built (lines 5-6,

Table 3.3: *Procedure BuildSgb()*

Inputs: $\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}$
Outputs: $\psi^{(i)}$
1. $\psi^{(i)} = \emptyset$
2. $\boldsymbol{y^{(i)}_{cur}} = \boldsymbol{y^{(i)}}$
3. repeat
4. $\quad \{\boldsymbol{X^{(j)}}, \boldsymbol{y^{(j)}_{cur}}\} = \lfloor \mu * N^{(i)} \rfloor$ samples without replacement from $\{\boldsymbol{X^{(i)}}, \boldsymbol{y_{cur^{(i)}}}\}$
5. $\quad \iota \sim U(\{\iota_{min}, \iota_{min}+1, \cdots, \iota_{max}\})$
6. $\quad \psi^{(j)} = \text{BuildCart}(\boldsymbol{X^{(j)}}, \boldsymbol{y^{(j)}_{cur}}, \iota)$
7. $\quad \psi^{(i)} = \psi^{(i)} \cup \{\alpha, \psi^{(j)}\}$
8. $\quad \boldsymbol{y^{(i)}_{cur}} = \psi(\boldsymbol{X^{(i)}}) = \boldsymbol{y^{(i)}_{cur}} - \alpha * \psi^{(j)}(\boldsymbol{X^{(i)}})$
9. until $\epsilon(\psi^{(i)}) \leq \epsilon_{targ}$
10. return $\psi^{(i)}$

details in section 3.3.5) and added to the ensemble (line 7), and $\boldsymbol{y^{(i)}_{cur}}$ is updated (line 8). Line 10 returns the final model. Unless stated otherwise, SGB settings are: ϵ_{targ} = 0.01, ι_{min} = 2, ι_{max} = 7, and α = 0.1.

The loop stops (line 9) when measured error $\epsilon(\psi^{(i)}$ hits target training error ϵ_{targ}. $\epsilon(\psi^{(i)}$ is measured as normalized root mean squared error:

$$\epsilon(\psi^{(i)}) = \sqrt{\frac{1}{N^{(i)}} * \sum_{l=1}^{N^{(i)}} \left(\frac{\psi^{(i)}(\boldsymbol{x_l^{(i)}}) - y_l^{(i)}}{max(\boldsymbol{y^{(i)}}) - min(\boldsymbol{y^{(i)}})} \right)^2} \tag{3.7}$$

where $N^{(i)}$ is the number of samples, and $\{\boldsymbol{x_l^{(i)}}, y_l^{(i)}\}$ is a given {input, output} sample l from the sample data $\{\boldsymbol{X^{(i)}}, \boldsymbol{y^{(i)}}\}$.

SGB converges robustly because it takes an importance sampling (IS) [Hes2003] perspective of modeling space. In typical (Metropolis) Monte Carlo sampling, the sampling distribution is identical to the target distribution. In IS, the sampling distribution is *biased* towards the regions which will give (approximately) the most information for the statistical estimator. Many boosting algorithms do IS implicitly in model space [Fri2003]; SGB is designed to do it *explicitly*. The sampling bias is exactly the difference between $\boldsymbol{y}$ and $\boldsymbol{y_{target}}$, since an unbiased sampler would simply always target $\boldsymbol{y}$.

With this IS perspective, the SGB algorithm takes measures to enhance the effectiveness of the sampling (model building):

- SGB mitigates unwanted biases due to greedily-built CARTs by *injecting randomness*, via choosing a different fraction μ of training data to train each CART (line 4).
- SGB injects further randomness by randomly choosing the maximimum CART depth ι at each iteration (line 5), so that sometimes shallow CARTs are built, and sometimes deep CARTs are built. These so-called "weak learners" are useful because they *only need to be better than random* – the iterative loop "boosts" them into one overall "strong learner" (hence the label "boosting" for the iterative loop).

- SGB follows an IS tactic to avoid premature convergence to a suboptimal model: have a *slow* update of the sampling bias. Specifically, it updates $\boldsymbol{y_{target}}$ by a factor α (e.g. 0.1) rather than a full 1.0 factor (lines 8,9).

Because SGB does IS, it inherits the dimensionality independence of Monte Carlo algorithms, which is the core of the theory showing why SGB has such excellent scalability properties [Fri2002].

3.3.5 CART Construction

This section describes the construction of a single CART model.

CART construction is performed by recursive partitioning of hypercubes in input space [Bre1984, Fri1991]. The starting region is the whole input space. At each partioning iteration, all existing eligible subregions are split into two child subregions. The split of a region R into two child regions R_{left} and R_{right} takes the form:

$$\begin{array}{l} if\ \boldsymbol{x} \in R\ then \\ \quad if\ x_{splitvar} \leq splitval\ then\ \boldsymbol{x} \in R_{left} \\ \quad else \qquad\qquad\qquad\qquad\ \boldsymbol{x} \in R_{right} \end{array} \tag{3.8}$$

where $x_{splitvar}$ is one of the input variables, and $splitval$ is its split value. This split is generated by considering all possible variables and their corresponding split values, and choosing the best according to a "splitting criterion" - in this case minimizing root mean squared error ϵ. Partitioning continues until there are no eligible subregions left. A subregion is eligible to split if it has not hit stopping criteria. Stopping criteria are: maximum number of recursions ι, and no samples left in subregion to split.

A CART takes the form of a binary decision tree. Accordingly, they have a visual interpretation. Section 8.3 exploits this property of CARTs in a classification setting, as an insight aid for designers.

3.3.6 SGB Construction Example

This section illustrates SGB construction in action on the problem of learning a mapping of a sinusoidal function. In the first boosting iteration of SGB (lines 1-9 of Table 3.3), one CART is built to the current target $\boldsymbol{y_{cur}^{(i)}}$. At at this point, $\boldsymbol{y_{cur}^{(i)}}$ is still equivalent to the overall target $\boldsymbol{y^{(i)}}$. The SGB's output is equal to α multiplied by the initial CART's output. Figure 3.3 (a) illustrates. Note how the energy of the SGB response is only about α = 10% of the overall target sinusoid. Also not the piecewise-constant nature of the mapping.

Figure 3.3 (b) illustrates the outcome of the next step. Note how the SGB response moved closer to the target, but not *too* quickly. Figures (c) to (g) illustrate further convergence, at boosting iterations 2, 5, 10, 20, 40, and 77. At iteration 77, the target error ϵ=0.01 was achieved, so SGB stopped.

Figure 3.3 (h) illustrates convergence of SGB error $\epsilon(\phi^{(i)})$ versus boosting iteration. Note the smooth convergence over time. The convergence rate is sharper in earlier iterations, and slows in later iterations; this can be confirmed by inspections of figures (a) to

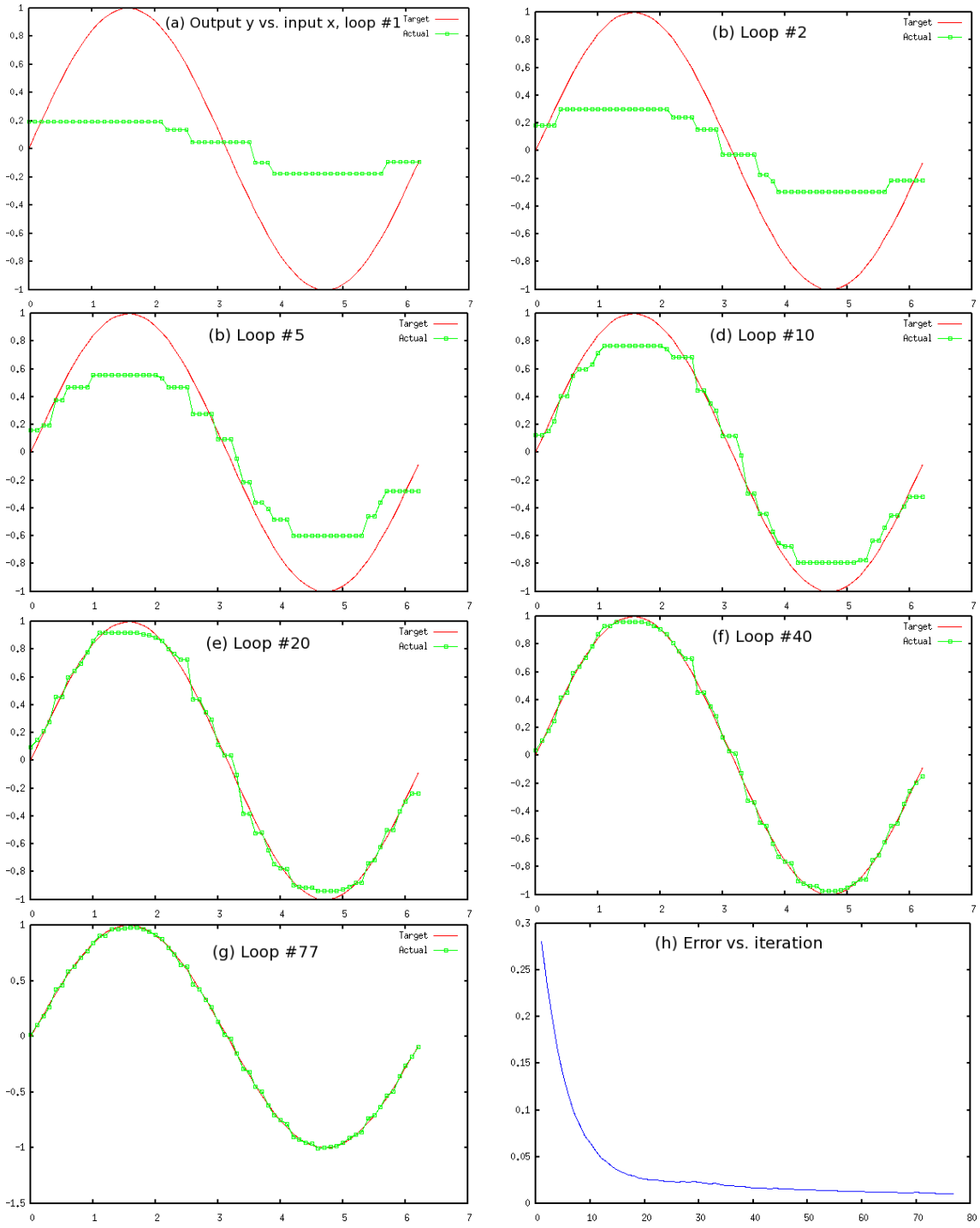


Figure 3.3: *SGB sinusoid learning response; bottom right is convergence of nmse vs. boosting iteration.*

(g) where the difference between the curves is more dramatic for earlier curves such as (a) to (b), and less dramatic for later curves such as (f) to (g).

Interestingly, the ideas behind SGB can apply to *circuits* as well, where small circuits replace CARTs. Section 10.4 explores this in detail. As a sneak peek, note how similarly the sinusoidal mapping is learned, in Figure 3.3 for CARTs, versus Figure 10.8 for circuits.

3.4 Foundations: Homotopy

Homotopy / continuation methods (sec. 11.3 of [Noc1999]) 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 has become the true problem, and therefore its solution is the true solution.

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

$$H(\boldsymbol{d}, \eta) = \eta * f(\boldsymbol{d}) + (1 - \eta) * (\boldsymbol{d} - \boldsymbol{a}) \tag{3.9}$$

where η is a scalar parameter and $\boldsymbol{a} \in \Re^{N_d}$. When $\eta = 0$, equation (3.9) 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.

3.5 SANGRIA Algorithm

Now that we have described some foundations of SANGRIA – model building optimization (MBO), stochastic gradient boosting (SGB), and homotopy – we are prepared to describe the SANGRIA global yield optimization algorithm itself [Mcc2008e]. We start by describing how MBO can be improved, then discuss how that leads into SANGRIA and its other core elements. After that, we will be ready to show the results of SANGRIA doing yield optimization of analog circuits, in section 3.6.

3.5.1 The Beginnings: Improving MBO

MBO is promising because it makes maximal use of simulation information. However, traditional MBO approaches have major issues as section 3.2.3 discussed. This restricts a straighforward application of MBO to yield optimization.

However, we can systematically fix these issues. Table 3.4 summarizes how. To have a regressor with good scaling properties and without a fixed functional template, we start with SGB regressors. But to handle uncertainty too, we use an *ensemble* of SGB regressors, where uncertainty at an input $\boldsymbol{d}$ is the standard deviation across the SGBs' outputs at $\boldsymbol{d}$.

Table 3.4: *Overcoming MBO Issues for SANGRIA.*

MBO Shortcoming	**Solution**
Inadequate regressors	Use stochastic gradient boosting (SGB)
Issues in uncertainty	Use *ensembles* of SGB models
Sensitivity of infill criterion	Make inner optimization *truly* multi-objective
Too few samples for high-dimensional prediction	*Combine* MBO with a gracefully-scaling optimizer

The sensitivity of the infill criterion's exploration vs. exploitation tuning parameter is resolved by *removing* the tuning parameter, and instead a truly multi-objective optimization is used. The multi-objective optimizer minimizes cost and maximizing uncertainty. The single criterion becomes two objectives. The specific algorithm used is NSGA-II [Deb2002].

Finally, for the high-dimensional cases when there are too few samples to predict meaningfully, MBO is always run in parallel with another algorithm that scales gracefully with a large number of search dimensions. Here, the choice is an evolutionary algorithm (EA) with several specific tactics that we will discuss shortly. The EA and MBO parts share each other's design candidates and simulation information. Therefore, behavior in many dimensions is graceful, because the overall SANGRIA algorithm does not *need* MBO, but takes advantage of it when the regressor has enough data to make good predictions.

3.5.1.1 Structure of SANGRIA

The structure of SANGRIA is shown in Figure 3.4.

SANGRIA has the following key elements:

- **High-dimensional model-building optimization.** MBO works with SGB to make maximum use of simulation information. An EA in parallel ensures that search is graceful when there is not sufficient data for good models.
- **Structural homotopy.** Exploration is performed on a loosened objective function (that happens to be cheaper), and exploitation is done on the full objective function (which is more expensive).
- **Local-optimization search operator**. This operator accelerates the EA's convergence by de-randomizing the EA's search steps.

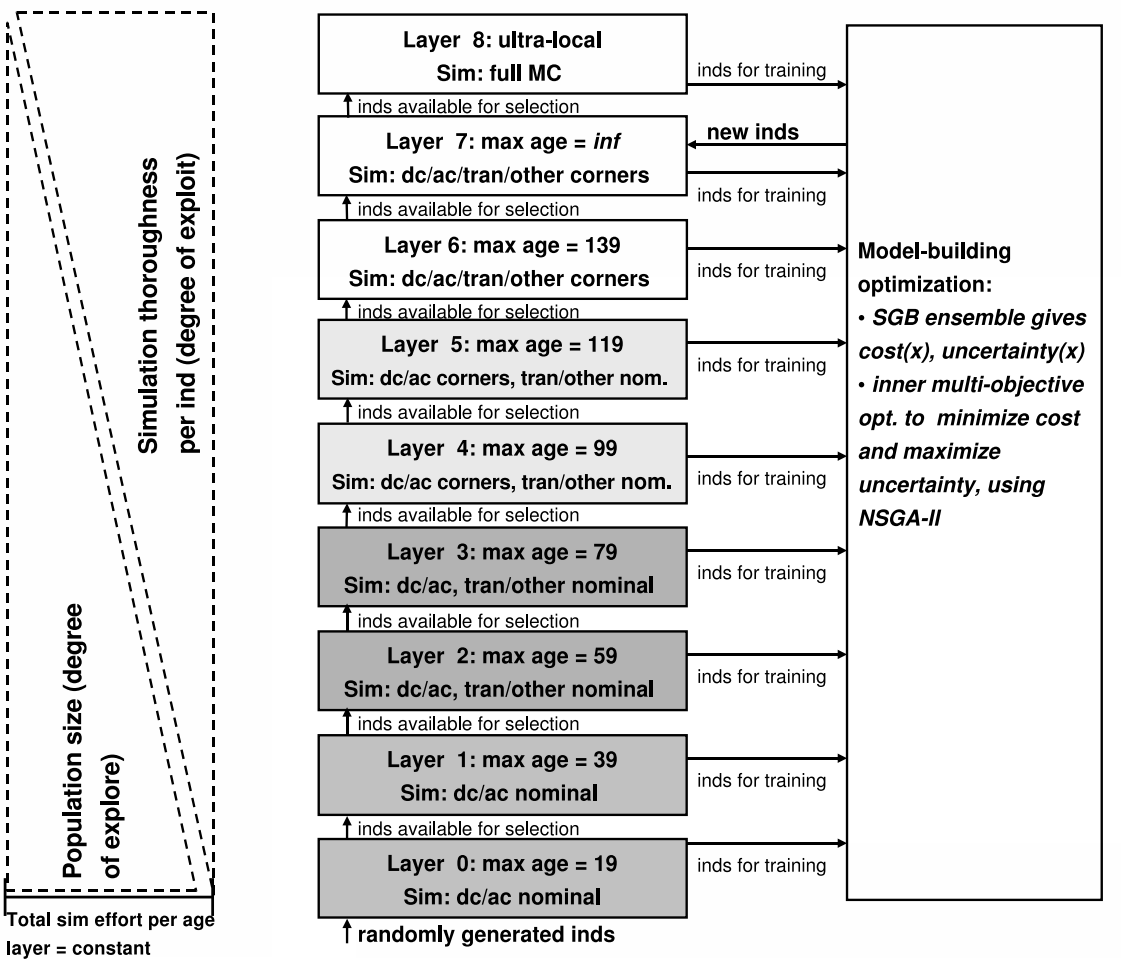


Figure 3.4: *SANGRIA Structure.*

We now discuss structural homotopy and the local search operator further. Then, section 3.5.2 will give a highly detailed description of SANGRIA.

3.5.1.2 Structural Homotopy

SANGRIA has a set of search "layers", where each layer is optimizing a population of candidate designs ("individuals"), as shown in Figure 3.4 middle column. The layers are organized according to the degree to which the candidate designs have been optimized ("age"): randomly-drawn designs enter the lowest layer as zero-age designs, and if they do well they get promoted to ever-higher layers while being further optimized (and aging +1 unit per generation). Each layer has a maximum age: 20 for layer 0, 40 for layer 1, and so on. If a design gets too old for a given layer, then it is ejected from that layer, thereby preventing wasted search effort on a stagnated design. The above concept is called an age-layered population structure (ALPS) [Hor2006].

To enable yield optimization, only design candidates at the highest age layer are fully simulated (across all analyses, process points, and environmental corners). Designs at

lower-age layers have fewer simulations. We call this "structural homotopy." Other homotopy algorithms work by starting with an easier-to-solve loosened version of the problem, then tightening the problem *dynamically*. In contrast, *structural* homotopy embeds the loosening into the algorithm's data structure (state). The solution to each layer can be regarded as a point on the homotopy "zero path". Therefore, we can view structural homotopy as a new approach to traverse the zero path: learn it coarsely to begin with, and refine it over time.

Specifically for circuit yield optimization, see how in Figure 3.4, layer 0 is just simulated at a single process/environmental corner of {dc/ac analyses, nominal process point $\boldsymbol{s}$, typical environmental point $\boldsymbol{\theta}$}. Layer 1 is like layer 0. Then, layer 2 adds transient/other analyses on the single {$\boldsymbol{s}$, $\boldsymbol{\theta}$} corner. Layer 4 adds non-nominal corners for dc/ac, and layer 6 adds non-nominal corners for transient/other. The choice of corners is elaborated in section 3.5.2.6. Finally, layer 8 does a full Monte Carlo simulation (with blocking) on each candidate.

This split of simulations was chosen based on choosing analyses which give many key measures for less simulation cost (ac, dc), and deferring the more expensive analyses which only give incremental measures of quality (transient, and corners). The core idea of structural homotopy does not depend on the exact choice, however. In section 9.5, we will show a different allocation of evaluations that is also effective.

To balance out the simulation cost per age layer, encourage even more exploration at the lower levels, and avoid potentially prohibitive top-layer simulation costs, SANGRIA's lower layers have larger populations which shrink going upwards. Figure 3.4 left illustrates. The top layer has a tiny population, which is why we label it "ultra-local".

To add fidelity at layers with fewer simulations, the non-simulated {$\boldsymbol{s}$, $\boldsymbol{\theta}$} combinations are predicted by adding performance deltas from other {$\boldsymbol{s}, \boldsymbol{\theta}$}'s. Individuals are compared via a cost function that allows meaningful optimization when no simulations are feasible, some simulations are feasible, or all simulations are feasible.

Each layer follows an evolutionary algorithm (EA) framework for updating the population with selection operators and search operators. Selection for layer i is typical EA selection, except individuals at layer $i-1$ are also considered.

Section 3.5.2 will elaborate on specific algorithms, and specific algorithm settings will be given in the results section (3.6).

3.5.1.3 Local-Optimization Search Operator

Each design candidate has its own local-optimization search state χ, allowing it to learn about the local structure of its search space and have accelerated convergence. This is in contrast to a typical EA which uses mutation and crossover operators, which have no memory and do not address fitness. It is also in contrast to "memetic" EAs which run a whole or partial local optimization for each individual's evaluation.

The specific local optimizer used is Dynamic Hill Climbing (DHC) [Yur1994]. DHC was chosen compared to other local optimization algorithms for a few reasons. First, derivatives are costly to compute, which rules out classical nonlinear programming algorithms such as quasi-Newton with BFGS update [Noc1999]. Second, the search space has

continuous and / or discrete elements, ruling out many modern derivative-free algorithms such as NEWUOA [Pow2006] which can only handle continuous design variables.

Nature-inspired algorithms such as simulated annealing (SA), evolutionary algorithms (EAs), and particle swarm optimization (PSO) are derivative-free and can handle mixed continuous/discrete spaces. However, their behavior is aimed towards global optimization, not local, so they are inefficient when the aim is merely local optimization. One exception is the EA variant of covariance-matrix adaptation [Han2001] which has fast convergence and a local optimization focus, but unfortunately it only works in continuous-valued spaces.

Pattern (direct) search algorithms [Kol2003], which include the simplex [Dan1963] and Hooke-Jeeves [Hoo1961] algorithms, are also derivative-free, can handle mixed spaces, and have a local search focus. These are a reasonable choice, and in fact they have been used within other analog CAD optimizers [Phe2000]. DHC [Yur1994] can be viewed as a loosened version of pattern search - loosened because it allows for stepsize growth in order to improve convergence rate, at the expense of losing some pattern search convergence properties. Since we have many local optimizers in parallel, we are less concerned about provable convergence per local optimizer, and more concerned with convergence *rate*; hence we chose DHC.

We are now ready to describe the specific algorithms within SANGRIA, including an elaboration of DHC.

3.5.2 SANGRIA Detailed Description

This section describes specific algorithms and sub-algorithms in SANGRIA, in detail.

3.5.2.1 High-Level Algorithm

SANGRIA's high-level algorithm, *SangriaOptimization()*, is described in Table 3.5. The algorithm's inputs are the search space bounds D, age gap N_a, maximum number of layers K, and number of individuals $N_L(k)$ for each layer k.

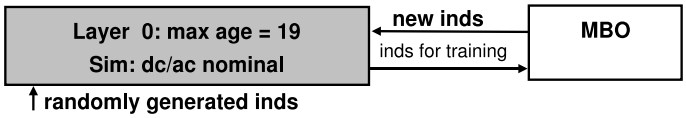
Line 1 initializes the generation count N_{gen}, the data structure P which will hold a population at each age layer P_k, and a list of all individuals encountered so far in the search P_{all}. Lines 2-13 are the generational loop, which repeats until stopping conditions are met. Lines 3-6 handle the case of an "age-gap" generation which happens every N_a generations. In an age-gap generation, the 0^{th} layer gets $N_L(0)$ new space-filling individuals in the N_D-dimensional space D, including a "loose" layer-0 evaluation.

In lines 7-9, each age layer P_i is updated at a time. First, parents are selected from the current or lower layer, and only if they are not too old. Then, each individual's local DHC state χ is updated, including evaluations appropriate to the age layer k (in line with structural homotopy). Section 3.5.2.8 gives details on DHC updating. Line 10 updates all the individuals encountered so far, P_{all}, just in time for the MBO inner optimization to use it (line 11). Lines 12 and 13 do bookkeeping: updating the best design so far $\boldsymbol{d}^*$, and generation count N_{gen}. Once the search terminates, $\boldsymbol{d}^*$ is returned; and of course during search intermediate $\boldsymbol{d}^*$'s can be returned.

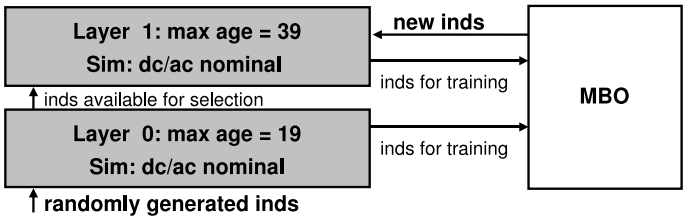
The sections that follow give details on other aspects of SANGRIA, including some of the above routines which were called by *SangriaOptimization()*.

Table 3.5: *Procedure SangriaOptimization()*

Inputs: D, N_a, K, $N_L(k)$
Outputs: $\boldsymbol{d}^*$
1. $N_{gen} = 0$; $P = \emptyset, P_{all} = \emptyset$
2. while stop() $\neq True$:
3. if $(N_{gen} \% N_a) = 0$:
4. if $|P| < K$:
5. $P_{|P|+1} = \emptyset$
6. P_0 = SpaceFillIndividuals($N_L(k), N_D, D$)
7. for k = 1 to $|P|$:
8. P_k = SelectParents($P_k, P_{k-1}, N_L(k)$)
9. $P_{k,j}$ = UpdateLocalOptState($P_{k,j}, k$), j = 1 to $|P_k|$
10. P_{all} = unique($P_{all} \cup P$)
11. $P_{|P|} = P_{|P|} \cup$ InnerOptimize(P_{all}, D, k)
12. $\boldsymbol{d}^*$ = $\boldsymbol{d_i}$ in P_{all} with highest Y or Cpk
13. $N_{gen} = N_{gen} + 1$
14. return $\boldsymbol{d}^*$



(a)



(b)

Figure 3.5: *(a) SANGRIA with just one age layer so far,* $|P| = 1$. *(b) SANGRIA with two age layers so far,* $|P| = 2$.

3.5.2.2 Growth of Age Layers

We point out that P starts out with just one layer, as shown in Figure 3.5(a). At the first "age gap" generation, it grows a new layer, as shown in Figure 3.5(b). At subsequent "age gap" generations, it keeps adding age layers, until it hits steady state with K layers

as Figure 3.4 shows. At steady state, $|P| = K$. MBO always feeds to the current top (non-ultralocal) layer $P_{|P|}$.

3.5.2.3 SANGRIA Individuals

The atomic unit that SANGRIA processes is an "individual". Whereas in most EAs an individual is a single design candidate, here the individual is a local optimization search *state*, χ. χ holds (a) one or more design points, (b) associated circuit evaluations, and (c) local optimizer-specific state information. More information will be provided after section 3.5.2.8 where we discuss the local optimizer, DHC, in more detail.

3.5.2.4 ALPS Selection

Table 3.6 describes tournament selection of parents in SANGRIA. Line 1 determines the candidate parents by merging layer k and layer $k-1$, and only keeping the individuals with age $\leq$ maximum age at layer k. Lines 2-5 fill the selected population: lines 3 and 4 randomly draw parents 1 and 2 with uniform bias from P_{cand}, and line 5 selects the parent with the lowest cost. Line 6 returns the updated population P'_k.

Table 3.6: *Procedure SelectParents()*

Inputs: $P_k, P_{k-1}, N_L(k)$
Outputs: P'_k
1. P_{cand} = ageOk($P_k \cup P_{k-1}$)
2. for i = 1..$N_L(k)$:
3. $\quad par1 \sim unif(P_{cand})$
4. $\quad par2 \sim unif(P_{cand})$
5. $\quad P'_{k,i}$ = best($\{par1, par2\}$)
6. return P'_k

3.5.2.5 SANGRIA Model Building Optimization

This section describes how MBO is deployed within SANGRIA. Table 3.7 describes the high-level MBO algorithm *InnerOptimize()*.

Lines 1 and 2 build the training input and output data, respectively, using the information of all the individuals so far, P_{all}. $P_{all,1}$ is the first individual in this list of all individuals, $P_{all,2}$ is the second, and so on. $P_{all,1}.\boldsymbol{d}$ is the design point of the first individual, and so on.

Line 3 constructs an SGB ensemble ψ from the training data $\{\boldsymbol{X}, \boldsymbol{y}\}$ (see section 3.3). In line 4, an inner optimization is run according to the problem formulation. Since there are two objectives (rather than a single, sensitive infill criterion), a Pareto-optimal set of designs is returned to collectively approximate ψ's exploration-exploitation tradeoff. The multi-objective optimization is performed using NSGA-II [Deb2002].

Multi-objective optimization could return a large number of Pareto-optimal individuals. We do not want to evaluate all of these because it could become computationally

expensive; a better option is to use a representative subset. So, line 5 reduces the number of individuals from $|P_{inner}|$ to N_{inner}, using clustering. SANGRIA employs bottom-up clustering (hierarchical agglomerative clustering) [Jar1968] because it is simple, fast, and reliable. Bottom-up clustering works as follows: (a) each point is assigned its own cluster, (b) measure distance among all clusters, as the Euclidian distance between the closest points in the clusters, (c) merge the two clusters that have shortest distance, (d) if target number of clusters is hit, stop, otherwise goto b.

Model-building time, and inner optimization / model simulation time could become a potential bottleneck. Accordingly, we use a rule of thumb for choosing parameter settings: the computational effort of *InnerOptimize()* cannot exceed the computational effort for circuit simulation.

Table 3.7: *Procedure InnerOptimize()*

Inputs: P_{all}, D, k
Outputs: P_{inner}
1. $\boldsymbol{X}$ = $\{P_{all,1}.\boldsymbol{d}, P_{all,2}.\boldsymbol{d}, \dots\}$
2. $\boldsymbol{y}$ = $\{cost(P_{all,1}, k), cost(P_{all,2}, k), \dots\}$
3. ψ = BuildSgbEnsemble($\boldsymbol{X}$, $\boldsymbol{y}$, N_{ens})
4. P_{inner} = $\left\{ \begin{matrix} minimize\{cost(\psi, \boldsymbol{d})\} \\ maximize\{u(\psi, \boldsymbol{d})\} \end{matrix} \right\}$ s.t. $\boldsymbol{d} \in D$
5. P_{inner} = cluster(P_{inner}, N_{inner})
6. return P_{inner})

3.5.2.6 Setting Corners

SANGRIA uses a corner-based approach to enhance efficiency, viewed as a "lightweight" Monte Carlo simulation. Recall that the core idea of corners-based approaches is: if corners are "representative" of process and environmental variations, and all corners can be "solved", then the final design's yield will be near-100%. We repeat equation (2.14) here:

$$\boldsymbol{d^*} = \underset{d \in D}{argmax}(\prod_{\Xi_i \in \boldsymbol{\Xi}} \delta(\boldsymbol{d}, \Xi_i)) \mapsto Y(\boldsymbol{d^*}) \approx 100\% \tag{3.10}$$

The challenge in SANGRIA is to choose corners that are representative of the performance bounds, but with a minimum count. SANGRIA's approach is to (a) take $N_{MC,cand}$ (e.g. 100) samples of process points, simulate them all at a typical environmental point, then (b) choose $N_{MC,chosen}$ (e.g. 25) representative points (corners). Representative corners are chosen in two steps: (b1) do nondominated filtering towards worst performance values, i.e. nondominated filtering in the opposite directions of optimal, and (b2) if needed, further reduce the points by bottom-up clustering [Jar1968]. Figure 3.6 illustrates. This procedure is not expensive as it is a one-time cost, done prior to starting *SangriaOptimization()*. This also allows it to use designer-specified corners. The procedure is also not overly pessimistic, as it is based on Monte Carlo samples.highest euclide

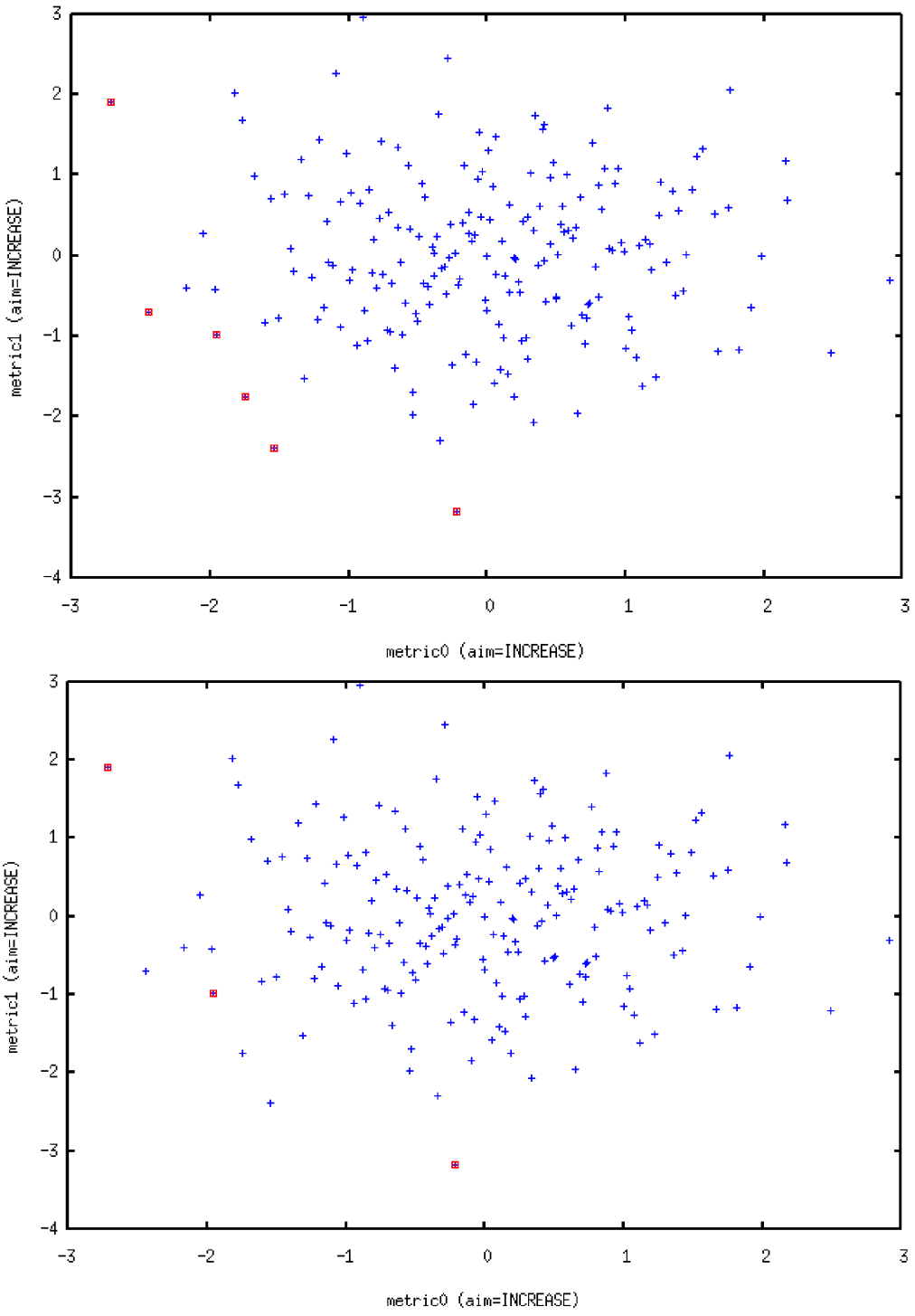


Figure 3.6: *Selecting representative corners. Top: nondominated filtering of "pluses" towards worst performance values gives "squares" in the bottom-left quadrant. Bottom: nondominated filtering followed by clustering; therefore the "squares" in the bottom plot are a subset of the "squares" on the top plot that have the highest Euclidian distance.*

3.5.2.7 Evaluation and Cost Calculation

Table 3.8 describes the evaluation of a population at age layer k, P_k. Each design candidate $\boldsymbol{d}$ at layer k must be evaluated sufficiently for use in selection at layer k *and* at layer $k+1$ (line 2). The $min()$ accounts for the top (K^{th}) layer.

SANGRIA's per-layer simulation specifications are shown in Figure's 3.4 middle column. For example, layer 2's specification is {dc/ac nominal, transient/other nominal}. Therefore layer-1 individuals must also be simulated at those specifications, as its individuals are available for selection in layer 2.

Table 3.8: *Procedure Evaluate()*

Inputs: P_k, k, K
Outputs: P'_k
1. for i = 1..$|P_k|$:
2. simulate $P_{k,i}$ for layer $min(k+1, K)$ specifications
3. $P'_k = P_k$; return P'_k

When an individual is evaluated "on nominal", each of its DHC state's $\boldsymbol{d}$'s are simulated at {nominal process point $\boldsymbol{s_{nom}}$, typical environmental point $\boldsymbol{e_{typ}}$}. When evaluated "on corners", it means that the evaluated is simulated at (1) all representative $\boldsymbol{s}$'s with $\boldsymbol{e_{typ}}$, and (2) all $\boldsymbol{e}$'s with $\boldsymbol{s_{nom}}$. This avoids simulating *all* combinations of environmental and process points. Then, the performance λ at a given $\{\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{e}\}$ is estimated as the performance at $\{\boldsymbol{s_{nom}}, \boldsymbol{e_{typ}}\}$, summed with deltas in performance due to $\boldsymbol{s}$ and $\boldsymbol{e}$:

$$\begin{aligned}\hat{\lambda}(\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{e}) \;=\;& \lambda(\boldsymbol{d}, \boldsymbol{s_{nom}}, \boldsymbol{e_{typ}}) \\ &+\; (\lambda(\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{e_{typ}}) - \lambda(\boldsymbol{d}, \boldsymbol{s_{nom}}, \boldsymbol{e_{typ}})) \\ &+\; (\lambda(\boldsymbol{d}, \boldsymbol{s_{nom}}, \boldsymbol{e}) - \lambda(\boldsymbol{d}, \boldsymbol{s_{nom}}, \boldsymbol{e_{typ}}))\end{aligned} \tag{3.11}$$

This setup directly accounts for the interactions of $\{\boldsymbol{d}, \boldsymbol{s}\}$ variables and $\{\boldsymbol{d}, \boldsymbol{e}\}$ variables. It assumes that the interaction of all three together, $\{\boldsymbol{d}, \boldsymbol{s}, \boldsymbol{e}\}$, is less significant. However it can still handle the case when that interaction matters: the top "ultra-local" layer simulates at *all* $\{\boldsymbol{s}, \boldsymbol{e}\}$ combinations for a given $\boldsymbol{d}$.

When the algorithm estimates the cost of an individual, the layer k is important. For example, an individual may have enough simulations for layer 2, but is participating in a layer-1 selection tournament; then its cost calculations only need to use the simulations that layer 1 specifies.

The cost is computed as follows:

$$cost(\boldsymbol{d}) = cost_g(\boldsymbol{d}) + cost_{cpk}(\boldsymbol{d}) \tag{3.12}$$

where $cost_g$ measures the total cost of violating constraints and $cost_{cpk}$ is a contribution from measuring Cpk.

$$cost_g(\boldsymbol{d}) = \sum_i^{N_g} violation(\widehat{g_{wc,i}}(\boldsymbol{d}, \lambda_i)) \tag{3.13}$$

$$violation(g_i, \lambda_i) = \begin{cases} 0 & g_i \leq 0 \\ \frac{g_i - g_{i,min}}{g_{i,max} - g_{i,min}} & otherwise \end{cases} \tag{3.14}$$

where $\widehat{g_{wc,i}}$ is the estimated worst-case value of performance i across all $\{\boldsymbol{s}, \boldsymbol{d}\}$ combinations. Performance is estimated at each $\{\boldsymbol{s}, \boldsymbol{d}\}$ combination with equation (3.11). $g_{i,max}$ and $g_{i,min}$ are the minimum and maximum values of performance g_i seen so far in the optimization run.

The additional $cost_{cpk}$ is activated when all constraints are solved, and pulls cost < 0 depending on how high the Cpk is. It enables the optimizer to increase the margin further once the estimated yield hits 100%:

$$cost_{cpk}(\boldsymbol{d}) = \begin{cases} 0 & cost_g(\boldsymbol{d}) = 0 \\ -(Cpk(\boldsymbol{d}) + cpk_{off}) & otherwise \end{cases} \tag{3.15}$$

where cpk_{off} is a value sufficiently large to ensure that negative values of Cpk do not make the overall value of cost be > 0. Cpk is calculated as in equation (2.6).

3.5.2.8 Dynamic Hill Climbing

Recall from section 3.5.2.3 that a SANGRIA individual is a Dynamic Hill Climbing (DHC) [Yur1994] search *state*, rather than merely a point in design space. DHC is a hillclimber which keeps any improvements found, and when it finds improvements it tries to capitalize on the direction of improvement with acceleration and ridge-walking.

In particular, the DHC state χ_{DHC} maintains and updates (a) three design points {$\boldsymbol{x}$, $\boldsymbol{xv}$, and $\boldsymbol{xuv}$}, (b) simulation / cost info for each design point, and (c) state information of a velocity vector $\boldsymbol{v}$, a ridge-walking vector $\boldsymbol{u}$, V_{list} = possible next $\boldsymbol{v}$'s, and a next action $\rho \in \{TRY\_XV, TRY\_XUV, STOP\}$. $\boldsymbol{x}$ is the current and best point so far, $\boldsymbol{xv}$ is $\boldsymbol{x} + \boldsymbol{v}$, and $\boldsymbol{xuv}$ is $\boldsymbol{x} + \boldsymbol{u} + \boldsymbol{v}$. From SANGRIA's higher-level perspective, it only sees that the (DHC) individual offers a design point ($\boldsymbol{x}$), an associated cost for that point, and a routine to update the individual's local optimization state, *updateLocalOptState()*.

For completeness, we give the algorithm from [Yur1994] in Table 3.9, but recast into a state-machine framework, so that it fits into SANGRIA. At a given iteration, the DHC state χ is updated based on how design proposals $\boldsymbol{xv}$ and $\boldsymbol{xuv}$ perform, i.e. how their costs compared to the center design $\boldsymbol{x}$'s cost. Lines of group 2 enable a "spinning" loop where different random directions from center $\boldsymbol{x}$ are tried, by popping from $\chi.V_{list}$. This loop will repeat unless a special case snaps it out (line groups 3, 4, 5, and 6).

Line groups 3 and 4 handle the case when DHC's $\boldsymbol{xv}$ is worse and it has run out of V_{list} options. If DHC is not at the smallest stepsize, then it shrinks $\boldsymbol{v}$ and creates a new set of options (line group 4). By adaptively shrinking the stepsize when not improving, the probability of improvement $p_{improve}$ goes up. Of course, $p_{improve} \to 0.5$ as $\|\boldsymbol{v}\| \to 0$ when not at a local optimum. This is easy to visualize: as $\|v\| \to 0$, the mapping from $\boldsymbol{x}$ to f in the region of radius $\|\boldsymbol{v}\|$ becomes a first-order Taylor-series approximation (a plane), and exactly half that plane is better than $f(\boldsymbol{x})$. So, the DHC stepsize shrinks adaptively to keep $p_{improve}$ good. Of course, DHC *will* reach a local optimum when $p_{improve} = 0$. Once

Table 3.9: *Procedure UpdateLocalOptState() (state-machine version of DHC).*

Inputs: χ, k. **Outputs:** χ'

1. if $\chi.\rho$ = TRY_XV:
2. if (cost($\chi.\boldsymbol{xv}$) > cost($\chi.\boldsymbol{x}$)) & ($\|\chi.V_{list}\| > 0$):
   # xv worsened; but options left at this stepsize
   $\chi.\boldsymbol{v}$ = pop($\chi.V_{list}$)
   $\chi.\boldsymbol{xv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{v}$
3. else if (cost($\chi.\boldsymbol{xv}$) > cost($\chi.\boldsymbol{x}$)) & ($\|\chi.\boldsymbol{v}\| = v_{min}$):
   $\chi.\rho$ = STOP #xv worsened; no options left
4. else if cost($\chi.\boldsymbol{xv}$) > cost($\chi.\boldsymbol{x}$):
   #xv worsened; but options left at smaller $\|\chi.\boldsymbol{v}\|$
   $\chi.\boldsymbol{v} = \chi.\boldsymbol{v}/2$
   $\chi.V_{list}$ = NewStepsList($\|\chi.\boldsymbol{v}\|$)
   $\chi.\boldsymbol{v}$ = pop($\chi.V_{list}$)
   $\chi.\boldsymbol{xv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{v}$
5. else if $\|\chi.V_{list}\| = 0$
   #xv improved or neutral, without a spin; build off u
   $\chi.\boldsymbol{x} = \chi.\boldsymbol{xv}$
   $\chi.\boldsymbol{u} = \chi.\boldsymbol{u} + \chi.\boldsymbol{v}$
   $\chi.\boldsymbol{v} = \chi.\boldsymbol{v} * 2$
   $\chi.\boldsymbol{xv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{v}$
   $\chi.V_{list} = \{\}$
6. else:#xv improved or neutral, but had to spin to get here; test u
   $\chi.\boldsymbol{xuv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{u} + \chi.\boldsymbol{v}$
   $\chi.\rho$ = TRY_XUV
7. else: # $\chi.\rho$ = TRY_XUV:
   $\chi.\rho$ = TRY_XV
8. if cost($\chi.\boldsymbol{xuv}$) > cost($\chi.\boldsymbol{x}$): #xuv worsened, so just go back to x+v
   $\chi.\boldsymbol{x} = \chi.\boldsymbol{xv}$
   $\chi.\boldsymbol{u} = \chi.\boldsymbol{v}$
   $\chi.\boldsymbol{v} = \chi.\boldsymbol{v} * 2$
   $\chi.\boldsymbol{xv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{v}$
   $\chi.V_{list}$ = NewStepsList($\|\chi.\boldsymbol{v}\|$)
9. else: #xuv improved or neutral, so incorporate u into v and keep going
   $\chi.\boldsymbol{x} = \chi.\boldsymbol{xuv}$
   $\chi.\boldsymbol{u} = \chi.\boldsymbol{u} + \chi.\boldsymbol{v}$
   if $\|\chi.\boldsymbol{u}\| > 0$: $\chi.\boldsymbol{v} = \chi.\boldsymbol{u} * 2$, else $\chi.\boldsymbol{v} = \chi.\boldsymbol{v} * 2$
   $\chi.\boldsymbol{xv} = \chi.\boldsymbol{x} + \chi.\boldsymbol{v}$
   $\chi.V_{list} = \{\}$
10. evaluate(updated subset of $\{\chi.\boldsymbol{x}, \chi.\boldsymbol{xv}, \chi.\boldsymbol{xuv}\}$, k); update cost(subset)
11. χ' = χ; return χ'

it reaches that optimum, it will keep shrinking the stepsize until the minimum stepsize $\boldsymbol{v_{min}}$ is hit, at which point DHC will declare itself converged and stop (line group 3).

Line groups 5 and 6 handle the case when DHC $\boldsymbol{xv}$ is improved or neutral. If it had been building on past successes without needing to do any spinning, then it will try to continue build on those past successes. In line group 5, because successes imply a high $p_{improve}$, $\boldsymbol{v}$'s stepsize is doubled. In line group 6, updating $\boldsymbol{u}$ ties together multiple past successes, with the hope that the aggregate is helpful. If it has not had any recent successes, then it needs to re-initialize its $\boldsymbol{xuv}$ and test it (line group 7).

Line groups 8 and 9 occur when $\boldsymbol{xuv}$ is tested. If $\boldsymbol{xuv}$ was unsuccessful (line group 8), then the search state backtracks. If successful, then once again DHC capitalizes on it by tying together the past successful steps (line group 9).

Line group 10 does evaluation of the newly-generated designs, and updates their corresponding costs for layer k. Note that all costs in this procedure are actually of aware of layer value k.

A speedup not shown in the algorithms is the following: if a layer has solved all its constraints, then it has little need to do more work. Therefore it skips the call to *UpdateLocalOptState()* for that layer, for additional computational savings.

3.5.2.9 Space-Filling Designs

Table 3.10 gives the details of creating individuals (DHC states) to collectively fill out the design space D, using Latin Hypercube Sampling (LHS) [Mck1979]. In lines 1-3, a raw LHS sample matrix is created, which assigns a bin for each design variable of each individual. In lines 4-10, actual design variable values are generated, where each variable must stay within the variable's subspace defined by the bin. Note how it handles any mixture of continuous vs. discrete design variables. In the case of continuous variables, line 8 shows how further random sampling within the bin was needed $\sim U([0, binsize])$.

In line 11, *InitializeDHCstate()* for individual $P_{0,j}$ (state χ_j) involves setting the state χ's attributes as follows: $\boldsymbol{x}$ = the input $\boldsymbol{d}$, $\boldsymbol{v}$ = a random direction having the magnitude of minimum stepsize v_{min}, $\boldsymbol{u} = \{0, 0, \dots\}$, $\boldsymbol{xv} = \boldsymbol{x} + \boldsymbol{v}$, and $\boldsymbol{xuv} = \boldsymbol{x} + \boldsymbol{u} + \boldsymbol{v}$.

Line 12 evaluates the new individuals P_0 sufficiently for level 0, and line 13 returns them.

3.6 SANGRIA Experimental Results

3.6.1 Summary of Test Circuit Problems

We used the test circuits shown in Table 3.11, which includes three opamps of increasing size (from 10 to 50 devices), and a voltage reference circuit ("vref"). The schematics will be shown further on.

Because of its excellent accuracy, and to illustrate the ability of SANGRIA to handle an extremely large number of process variables, we used the process variation randomness model of [Dre1999]. Accordingly, the local variation parameters for each transistor are: NSUB (substrate doping concentration), VFB (flatband voltage), WINT (width variation),

Table 3.10: *Procedure SpaceFillIndividuals()*

Inputs: $N_L(0), N_D, D$
Outputs: P_0
1. $\boldsymbol{B}$ = zeros(N_D, $N_L(0)$)
2. for i = 1..N_D: #for each variable
3. $\quad \boldsymbol{B_i}$ = random permutation of $\{1, 2, \ldots, N_L(0)\}$
4. for j = 1..$N_L(0)$: #for each individual $P_{0,j}$
5. $\quad$ for i = 1..N_D: #for each variable
6. $\quad\quad$ if D_i is continuous:
7. $\quad\quad\quad$ $binsize = (D_{i,max} - D_{i,min})/N_L(0)$
8. $\quad\quad\quad$ $d_{new,i} = D_{i,min} + B_{i,j} * binsize \ + \sim U([0, binsize])$
9. $\quad\quad$ else: # D_i is discrete
10. $\quad\quad\quad$ $d_{new,i} = (B_{i,j})^{th}$ discrete value in $\{D_{i,1}, D_{i,2}, \ldots, D_{i,max}\}$)
11. $\quad$ $(P_{0,j})$ = InitializeDHCstate($\boldsymbol{d_{new}}$)
12. evaluate($P_0, 0$)
13. return P_0

Table 3.11: *Test circuit sizes.*

Label	**# Devices**	**# Design Vars.**	**# Process Vars.**	**# Env. Vars.**	**# Env. Points**	**Test-benches**
10T opamp	10	21	91	5	3	ac, tran, THD
30T opamp	30	56	216	5	3	ac, tran, THD
50T opamp	50	97	342	5	3	ac, tran, THD
vref	12	28	106	3	3	ac, ac

LINT (length variation), U0 (permittivity), RSH (sheet resistance), and TOX (gate oxide thickness). The per-resistor variation parameters are: DRSH (sheet resistance), DXW (width variation), and DXL (length variation); and per-capacitor variation parameters are: DXW (width variation), DXL (length variation), and DTOX (oxide thickness). There is a single global-variation parameter for each of NSUB, VFB, etc. as well. The variables $\boldsymbol{s}$ in the process variations' $pdf(\boldsymbol{s})$, are normal, independent, and identically-distributed (NIID).

Because there are so many variables per device, the total number of process variables is very large. For example, there are 342 process variables for the 50T opamp, which, as we will see, SANGRIA readily handles.

The technology was TSMC 0.18μm CMOS. The simulator was a proprietary SPICE-like simulator of a leading analog semiconductor company, with accuracy and runtime comparable to HSPICE^{TM} [Snps2008a].

In all cases, an initial "rough cut" design is supplied, which took about 10-30 minutes for an expert designer to do. We do this only so that we can have a baseline for comparison, e.g. comparing the yield and performance spread of initial versus resulting designs.

SANGRIA can leverage this, but does not rely on it, because every N_a = 10 generations it will inject randomly-generated designs into age layer 0. These random designs get a chance to refine because they do not have to compete with designs at higher layers, including designs derived from the initial design. As we will see, in several experimental runs it was crucial for SANGRIA to explicitly *not* use the initial design's region and instead build on a region started from random sampling-based exploration. [Hor2006] also observed this.

3.6.2 Algorithm and System Settings

Each run of each circuit problem had identical algorithm parameters. The maximum number of circuit simulations was $N_{sim,max}$ = 100,000, which is easy to run overnight with a modestly-sized computer cluster. (Therefore all the runtimes for each forthcoming SANGRIA run is overnight or less.)

The ALPS settings were as follows. In line with Figure 3.4, there were K = 9 age layers in steady state, a value similar to [Hor2006]. With an age gap N_a = 10, the maximum age per layer was 10, 20, ... for layers 0, 1, ... respectively. Layer 8's maximum age was ∞. [Hor2006] had similar values in the 'linear' age setting. The lowest age layer's population size $N_L(0)$ was 200 individuals (like [Hor2006]). Population size decreased linearly from $N_L(0)$ = 200 down to $N_L(7)$ = 8. The ultra-local layer had $N_L(8)$ = 3 individuals, which allowed some exploration without being overly computationally expensive. The cost offset of cpk_{off} = 10.0, which is more than enough because excellent values of Cpk are >2.0.

$N_{MC,chosen}$ = 25 representative process points were chosen from $N_{MC,cand}$ = 100 candidate points using the approach of section 3.5.2.6.

The MBO optimizer's settings were as follows. SGB parameters were: learning rate α = 0.10, minimum tree depth ι_{min} = 2, maximum tree depth ι_{max} = 7, target training error ϵ_{targ} = 5%. There were N_{ens} = 5 SGBs in an ensemble. SGB parameters were set based on recommendations from [Fri2002]. NSGA-II parameters were: N_{pop} = 25, $N_{gen,max}$ = 50, with a crossover probability of 0.2 which was enough to get near-Pareto optimal results without having the inner optimization risk dominating computational cost. The number of individuals returned from a given inner optimization, N_{inner}, was set to 5, which is large enough to get a good spread of the exploration-vs-exploitation tradeoff without becoming too expensive.

Designs returned as final results had N_{MC} = 30 process points which is quite low, but still provides reasonable resolution for Cpk values.

The whole system was coded in Python [Pyt2008], including Python Numeric [Pyn2008] for handling arrays.

With the exception of the SGB parameters, there was very little tuning of these parameters. The parameters were set based on reasoning, choosing to err on the side of reliability. There is almost certainly opportunity for improving algorithm speed and quality of results via parameter tuning.

3.6.3 Experiments on 10T Opamp Circuit

3.6.3.1 Problem Details

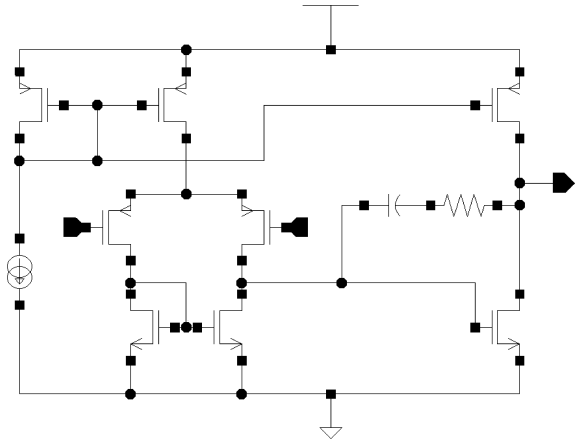


Figure 3.7: *Schematic of 10-device operational amplifier.*

Figure 3.7 shows the schematic for the 10-transistor opamp. It has 21 design variables, 91 process variables, and three testbenches having ac, tran, and THD analyses respectively. Each testbench has 3 environmental points composed of 5 environmental variables. Specifications were: A_V > 65 dB, BW > 1 MHz, GBW > 300 MHz, PM > 56°, GM < -10 dB, settling time ST < 12 ns, SR > 3e8 V/s, overshoot OS < 12%, and THD < -45 dB. We performed four runs with different seeds to the random number generator. We now analyze the results of each run.

3.6.3.2 Detailed Analysis of First Run

This section describes the results from the first run, going into detail to examine the quality of the results, and SANGRIA's convergence behavior.

Figure 3.8 shows the yield vs. generation, and Cpk vs. generation for the first run. Each square in the plot is the result of a full Monte Carlo simulation of the current most-promising SANGRIA design across N_{MC} = 30 process points. We see on the far left of the plot that the initial design's yield is 26.7%, and that the next Monte Carlo sampling happens at generation 60, giving an improved yield of 56.7%. The best yield keeps improving with passing generations, until hitting the maximum of 100% yield at generation 112, making the run a success.

To be precise, the yield numbers are statistical estimates based on the 30 Monte Carlo samples. This means that the lower bound for a "reported" 100% yield is 88.6%, with 95% confidence (using Wilson's confidence interval for a binomial proportion [Wil1927]). But for simplicity, we will just say 100% yield.

Note the squares below the curve of yield vs. generation. These are Monte Carlo sampled results where the candidate design did not do as well as the best so far. It happens when the best design so far on the "ultra-local" layer has already been simulated, so a

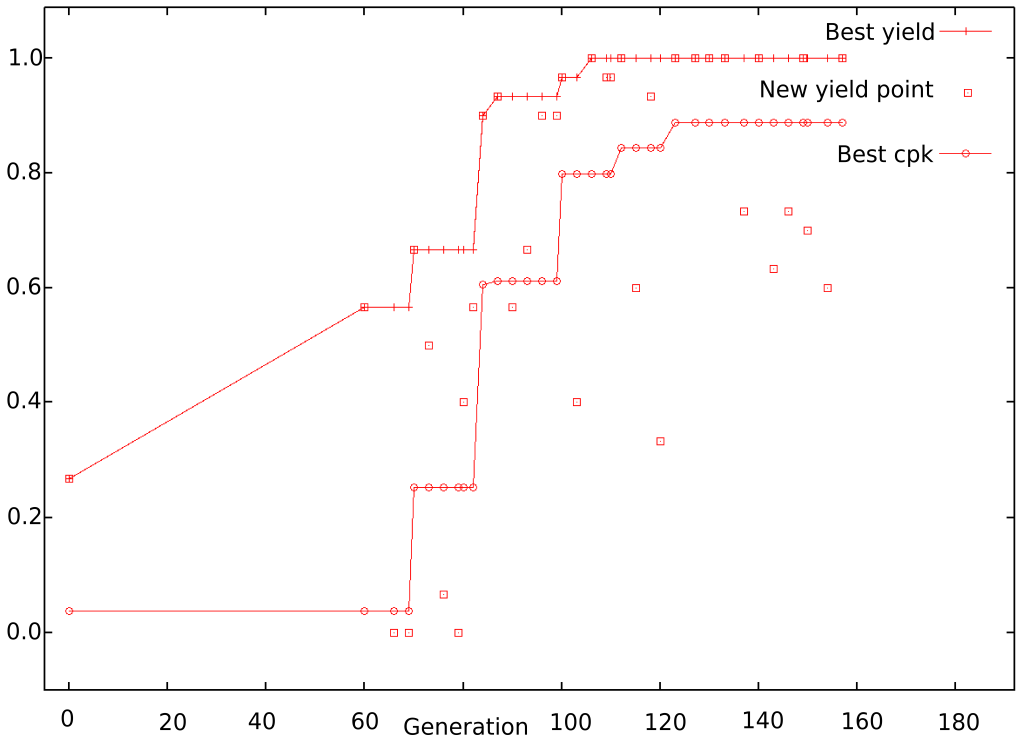


Figure 3.8: *Best yield vs. generation, and best Cpk vs. generation, for SANGRIA run 1 on 10T opamp.*

different design is tried, either from the ultra-local layer or a lower layer. Sometimes they do well, but sometimes they do not.

Once 100% yield is achieved, there is no further optimization to do on the "yield" objective (if using Monte Carlo estimation on 30 feasibility samples). However, SANGRIA continues to do meaningful optimization beyond this stage by maximizing Cpk. Figure 3.8 also shows the best Cpk vs. generation, denoted by the curve with the ○'s. We see that Cpk is steadily increasing prior to achieving 100% yield, but it improves further *after* achieving 100% yield at generation 112. This has the effect of increasing the margins on the performances. The best Cpk value is found in generation 123. The run stopped when the 100,000 simulation budget was hit.

A further illustration of this continued improvement is shown in the boxplots of Figure 3.9. The 3x3 grid holds all 9 performances A_V, BW, etc. Each entry in the grid summarizes the spread for a specific performance on the four designs, each from a Monte Carlo sampling. In each entry, the left box / whiskers is for the initial design, and the three proceeding rightwards are for 100%-yield designs from generation 106, 133, and 167 respectively. A box/whiskers summarizes the performance's distribution as follows. The lower and upper whiskers are the minimum and maximum simulated values, respectively. The lower and upper edges of the box are the 25^{th} and 75^{th} percentiles, respectively. Therefore the box's extremes contain 50% of the data, and the whiskers contain 100% of the data. Each performance's y-axis is actually oriented such that the *top* of the plot has

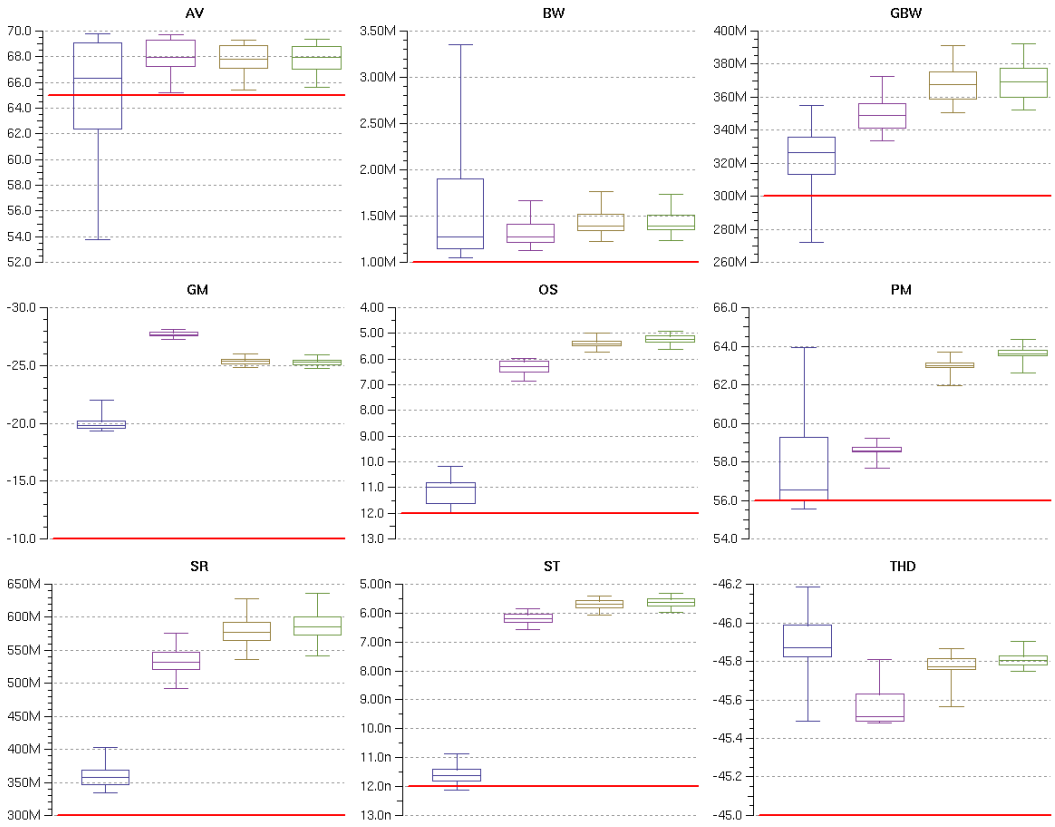


Figure 3.9: *Performance boxplots of four of the designs found in SANGRIA run 1 on 10T opamp: initial, generation 106, generation 133, and generation 167.*

the best values according to the performance's aim. For example, A_V (top left) aims ">", so its top is a larger value (70.0 dB) than its bottom (50.0 dB). Conversely, settling time ST (bottom middle) aims "<", so its top has a smaller value than its bottom (4.0 ns vs. 13.0 ns).

The feasibility threshold is the horizontal bar spanning all four boxes. Therefore it is easy to scan the plots to see which performances are not met and by how much; or which performances are met and their degree of margin. For example, we see that the initial design (left box/whisker) does not meet A_V, GBW, PM, and ST. It it is on the edge for overshoot OS, and is close for BW. It has some margin for SR and significant margin for GM and THD. GM has a very tight spread, and significant margin. The generation-167 design (right box/whisker) is above every feasibility threshold. It basically has equal or tighter spread than the other designs on each performance metric. It has equal or better margin than the other designs as well, except for GM which is better than the initial design but slightly worse than the generation-106 design.

We can also do side-by-side comparisons from the initial design to improved designs. For starters, see that all box/whisker plots of all three 100%-yield designs are fully within the specification range, which they should be by definition. The margin for GBW signif-

icantly improved by tightening the spread. The margins for OS, SR, and ST improved by shifting their respective mean values upwards. A_V was one of the tougher specs, but the three designs significantly tightened the spread to make all samples feasible.

We can also compare the improvement among the three designs. For most performances (BW, GBW, OS, PM, SR, and ST), we see that the margin improved going left-to-right through the generation 106, 133, and 166 designs. Sometimes the margin improved by tightening spread (e.g. THD), and sometimes by improving the mean (e.g. PM). In the case of GM, margin actually decreased a bit but that was not an issue because all designs, even the initial design, had a very high margin.

We can also ask how the area changed from the initial design to other designs. After all, solving a yield issue via a drastic increase to area is not practical for industrial designs. Because we do not use layout information, our area estimate is an approximation: the sum of $W * L$ across devices. It turns out that all three 100% yield designs had *lower* area than the initial design; the smallest 100%-yield design has 23.7% less area than the initial design. Table 3.12 shows the designs' relative area values, along with Cpk values.

Table 3.12: *Cpk and area for four designs in 10T opamp run 1.*

Design Point Label	**Yield**	**Cpk**	**Area Change**
Initial design	26.7%	0.037	0% (baseline)
Run 1 generation 106	100%	0.759	-21.7%
Run 1 generation 133	100%	0.805	-23.7%
Run 1 generation 166	100%	0.834	-23.4%

So far we've seen SANGRIA final results and yield / Cpk convergence. Let us now examine SANGRIA's behavior in more detail, by inspecting the convergence for each SANGRIA layer. Figure 3.10 shows the cost vs. generation for each age layer, which we now explain. At generation 0, only the 0^{th} age layer exists, so only its curve is plotted until generation 10 (age gap N_a = 10). Layer 0's best design was able to immediately meet all the layer-0 constraints, giving it a cost of 0. Therefore the line is merely a horizontal line at y-axis cost=0. At generation 10, layer 1 is added, and it is able to fully solve the design as well because it has the same goals as layer 0. At the next "age-gap generation", generation 20, layer 2 is added, and despite having more goals (tran/other nominal), it was able to solve them so its cost stays at 0.

Interesting things start to happen at generation 30. First, the population formerly at layer 2 gets kicked out, into layer 3. Layer 3 has the same goals as layer 2, and therefore the best cost remains at 0. However, the new individuals going into layer 2 do not immediately solve all the goals at generation 30, so their best cost is >0. In the plot, these are the ∘'s at a cost value of $\approx$ 48 for generations 30-33. But those ∘'s go back to cost=0 at generation 34, which means that the new individuals at layer 2 were able to solve the goals. At generation 40, layer 4 is added. Layer 4 gets immediately solved by the incoming individuals from layer 3. At generation 50, layer 5 is added, and it is solved immediately too. Throughout the whole run, layers 4 and 5 have 0 cost. Since the only difference between them and layer 4 is adding corners on the ac testbench, it implies that

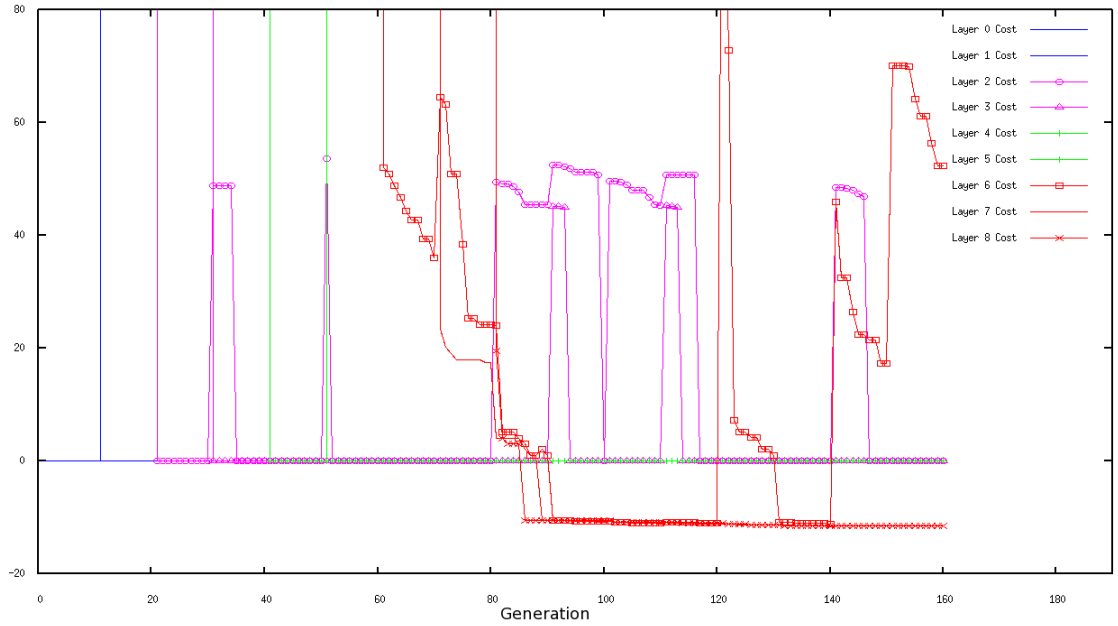


Figure 3.10: *Best cost vs. generation, for each age layer, on SANGRIA run 1 of 10T opamp.*

once a design can solve for the process and environmental variations on ac performances, it can solve for the nominal dc/tran/THD performances. It does not imply that solving on nominal always means solving on corners, however! In fact, we confirm this when layer 6 is added at generation 60. Its cost is initially so high that it is out of the plot's axes, but by generation 62 the cost comes down sufficiently to be visible. These are the squares with cost of ≈ 50 at generation 62. Layer 6's best cost continues to reduce until generation 70.

At generation 70, layer 7 is started, being initialized by layer 6's. Layer 7 continues convergence for generations 70-75, then plateaus for 5 generations. At generation 80, layer 8 is created, starting with the layer 7 population. Layer 8 further reduces the cost, and meets cost=0 at generation 84. Since it is already considering all testbenches and process/environmental variations, then it can aim for a cost of <0, which it does. So from generation 84, it converges with cost values < 0. It steadily reduces cost for the remainder of the run (the stars curve).

Another interesting signature ALPS behavior can be observed in the convergence plot. Note the ○'s (layer 2 curve) appearing at generation 80, with cost ≈ 50. Prior to that generation, layer 2 had solved the problem having cost 0, but its individuals became too old, and the new individuals feeding into it were not good enough to have 0 cost. So layer 2 improves the design in generations 80-90, then hands it to layer 3 at generation 90 which solves it in 3 more generations. At generation 90, the new individuals coming into layer 2 at generation 90 also do not solve the design. So, from generations 90-99, layer 2 improves the best cost, and finally solves it at generation 100.

The spike of cost for layer 6 at generation 120 is another signature ALPS behavior. Once again, its best individuals became too old, were ejected from layer 6, and no new individuals could help. From generations 120-130, layer 6 converges downwards and

feeds into layer 7 with helpful individuals. Similar spikes happen to layer 6 at generation 140 and at generation 150. Note that sometimes spikes do *not* happen, such as in layer 6 for generations 90 - 120. This occurs when the layer's younger individuals are the best.

Recall that each age gap generation (every N_a = 10 generations here), layer 0 gets fresh *randomly-generated* individuals. This means that ALPS is consistently trying new regions of the search space. Through the subsequent refinement in higher layers, these regions get exploited. Individuals in less promising regions die out. ALPS continues this space-filling process over time, and it is this characteristic that gives it the globally reliable behavior, not only theoretically but also in practice. This property is extremely important for handling challenging optimization problems, and we will exploit it in subsequent chapters as well.

We also observe that the best randomly-generated individual of each age gap generation has zero cost (on ac testbench, no process or environmental variations). This means that getting a functional sizing is relatively easy for this problem. Unsurprisingly, we will see on more complex circuits, getting a functional sizing will take more search effort than mere random sampling.

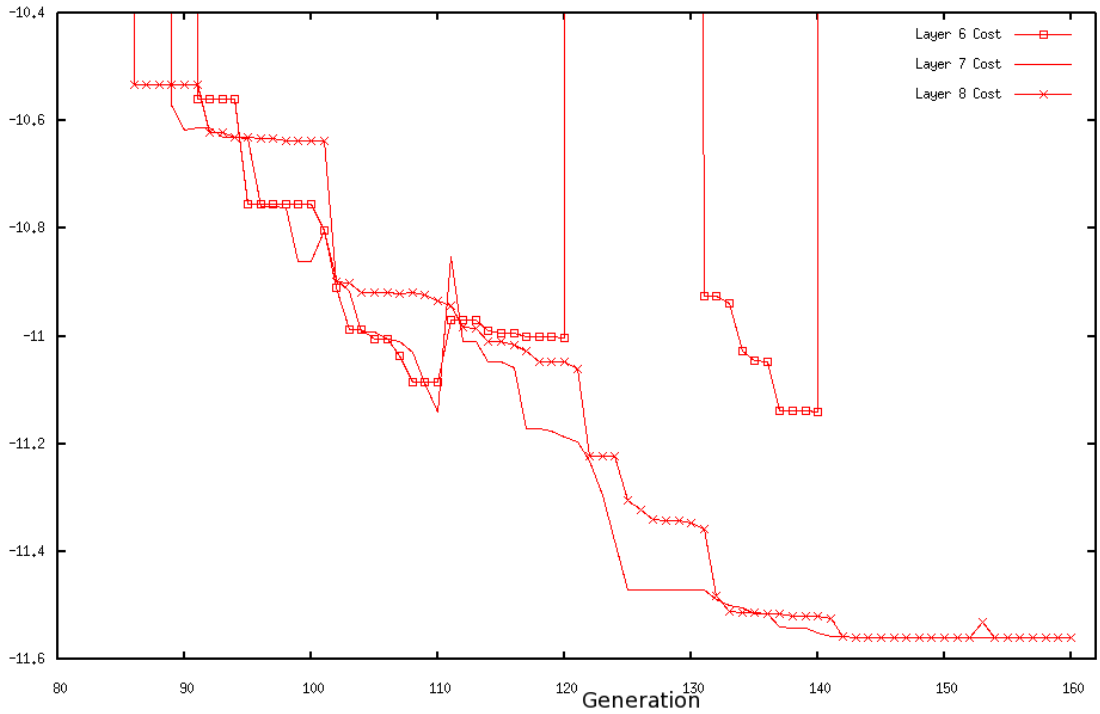


Figure 3.11: *Zoom in on age layers 6-8 in best cost vs. generation, on SANGRIA run 1 of 10T opamp.*

While Figure 3.10 gives us insight into the convergence per layer, it is hard to examine the convergence of the top layers in detail. This is important because ultimately the individuals in these layers become candidates for Monte Carlo sampling. So, in Figure 3.11, we zoom into layers 6, 7, and 8. First, note how layer 8's cost is monotonically decreasing. This is because its maximum age is ∞, which means individuals never get

ejected for being too old, and the best-cost individual remains without regard to age. In contrast, layers 6 and 7 do have upward spikes due to the age limit at these layers. Layer 6 has more drastic spikes than layer 7, because layer 6 solves a tougher problem than layer 5 (which feeds it individuals). In contrast, layer 7's problem is the same as layer 6's, so it has a running start.

Finally, because layer 8's evaluations are based on full Monte Carlo sampling and its cost function is monotonically decreasing, it means that Cpk will be monotonically increasing, which is what we have already observed on the resulting individuals.

3.6.3.3 Second, Third, and Fourth 10T Opamp Runs

Figures 3.12, 3.13, and 3.14 are the results of three subsequent SANGRIA runs on the 10T opamp. In each run, we show the convergence of the yield vs. generation, Cpk vs. generation, and best-cost vs. generation for each age layer.

Each of the three runs (2, 3, 4) achieved a yield of 100% at about generation 100. Also, as Table 3.13 shows, runs 2, 3, and 4 each got Cpks better than run 1 in their 100,000 simulation budgets. We dive deeper to see what the difference in behavior is.

In run 2's Figure 3.12 bottom, we see that the top age layer does not get cost < 0 until generation 110. There was an aborted attempt at generation 70, where the second-highest layer got cost 0, but that design did not translate to the top age layer with low cost. And of course, it also did not translate to good yields, as confirmed by the low-yield results in generations 70-100. The only difference in cost functions between the top two layers is in the accounting for interactions among the process and environmental variables. This means that the early design attempts had stronger process-environmental coupling, and the final, more successful designs, did not. Run 3 had a similar phenomenon, as shown in Figure 3.13. Both these runs illustrate that taking steps from the initial design, no matter how promising, might lead to a local optimum. So, there must be an opportunity to try alternative regions. This reconfirms the need to have *globally* reliable statistical optimization.

Run 4 (Figure 3.14) had a behavior like run 1 in its early generations. As opposed to runs 2 and 3, run 4 had no false starts in getting a cost < 0, and the yields of its Monte-Carlo sampled individuals were steadily improved. Accordingly, run 4 hit a yield of 100% at generation 69, much earlier than run 2's generation 113 and run 3's generation 110. Its Cpk at that point was like run 1 too: <1.0. But at generation 70, run 4 started to behave like runs 2 and 3: its highest layer's cost spikes upwards (to >0), so the search goes elsewhere. Within just a few generations, the layer found a new region with cost < 0, and with its first Monte Carlo sampling at that region (generation 76) it got a design with Cpk > 1.0. The remainder of run 4 was like the last parts of runs 2 and 3: steady improvement to Cpk, driven by steady lowering of the top layer's cost, fed by lower layers' designs.

Figure 3.15 shows the boxplot for Run 3's best-Cpk design (right box/whisker plots) compared to the initial design (left box/whisker plots). Comparing to the designs in Figure 3.9, we see that the result from run 3 has improved the margin significantly for all performance measures. The run 3 design has 11% smaller area as well, as Table 3.13

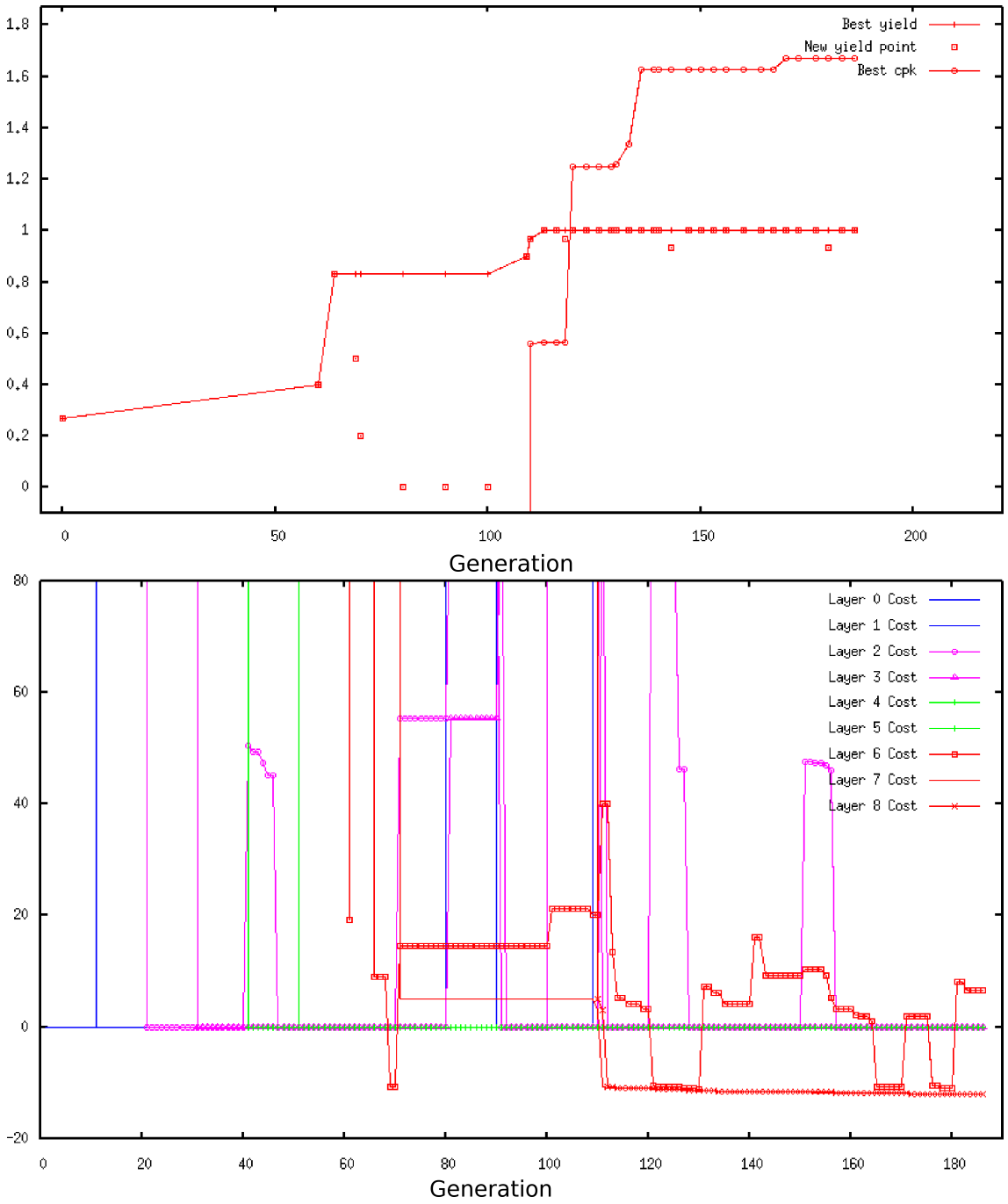


Figure 3.12: *Convergence curves for SANGRIA run 2 on 10T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

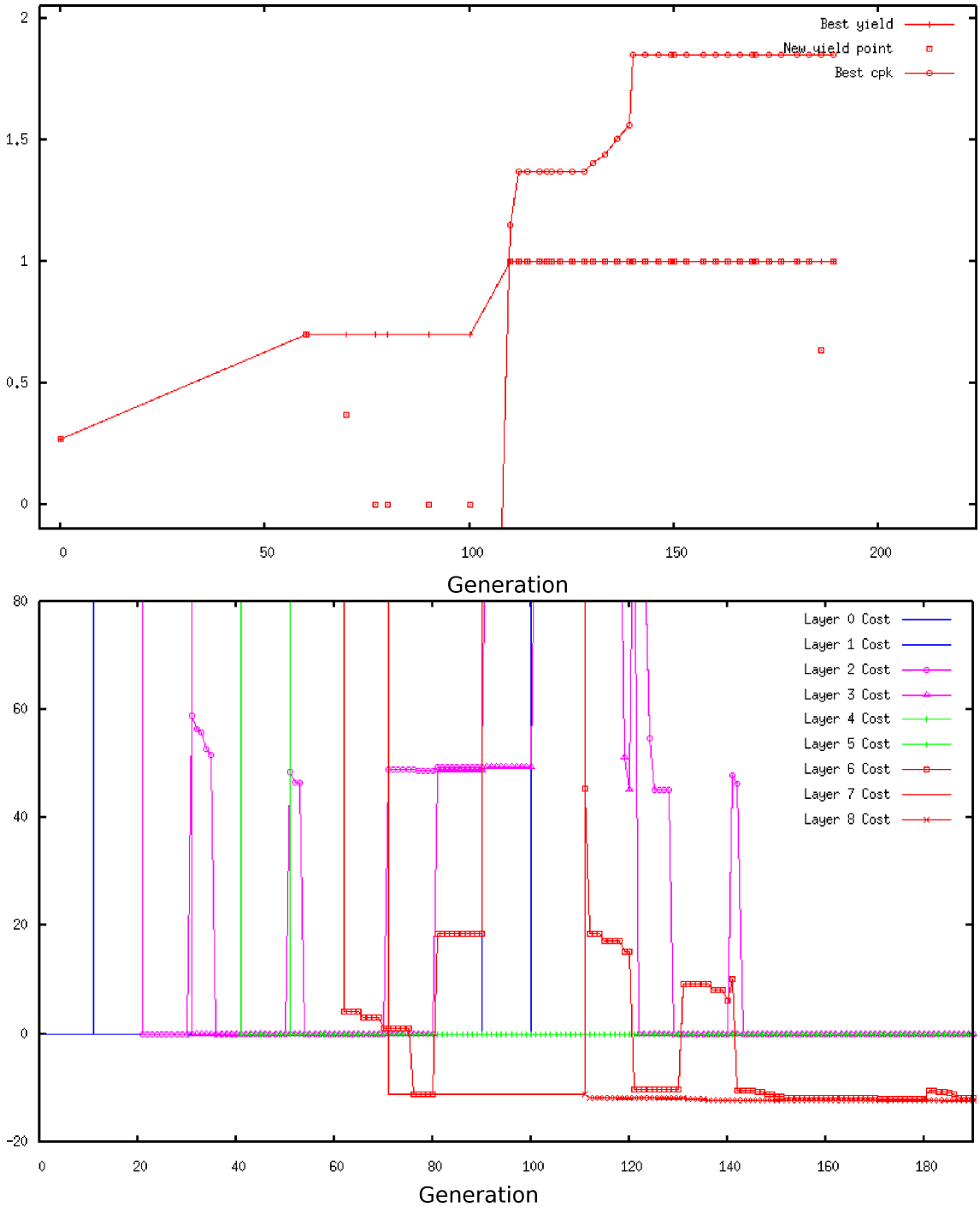


Figure 3.13: *Convergence curves for SANGRIA run 3 on 10T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

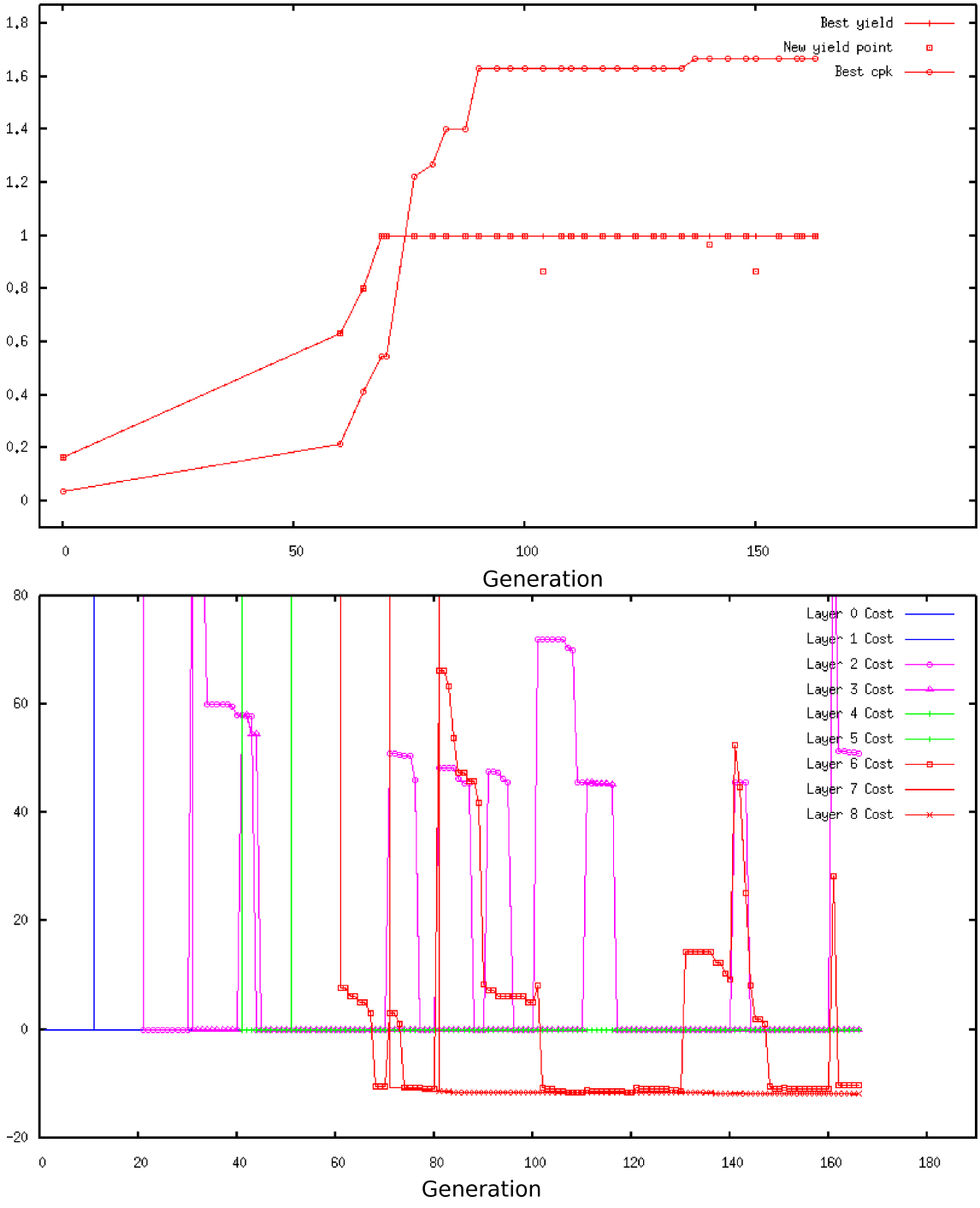


Figure 3.14: *Convergence curves for SANGRIA run 4 on 10T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

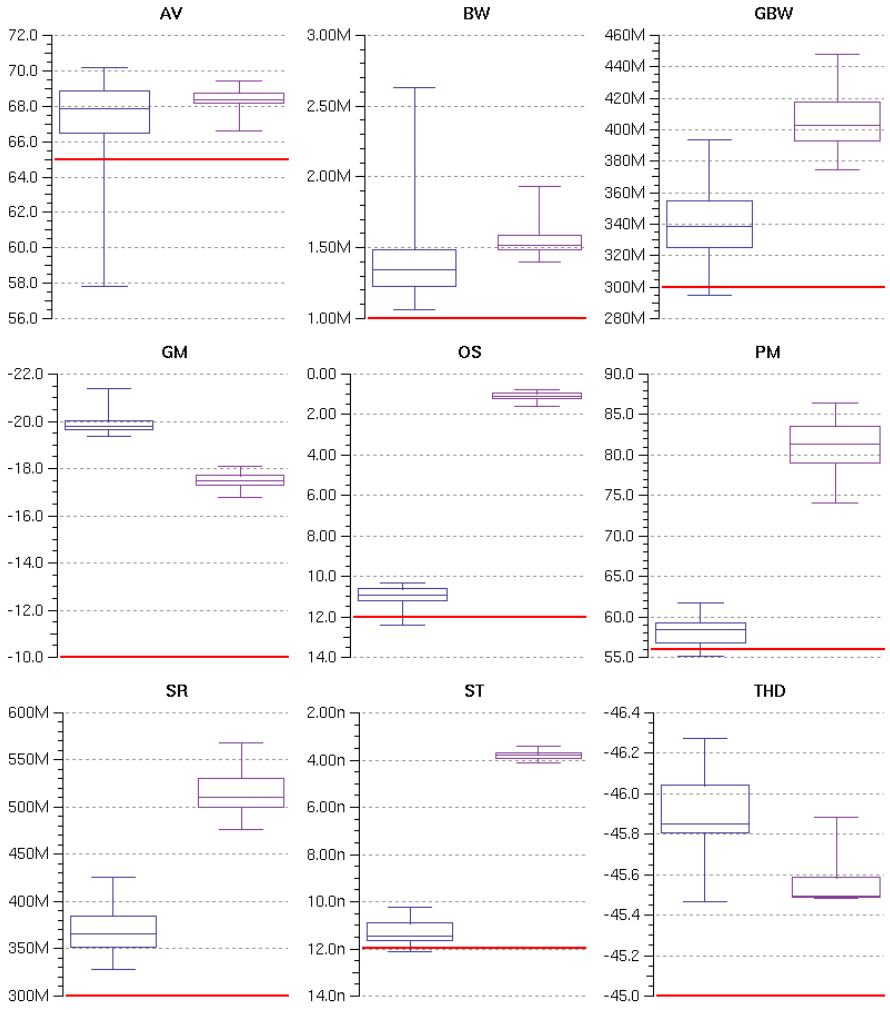


Figure 3.15: *Performance boxplots of two found in SANGRIA run 3 on 10T opamp. In each grid entry, the left boxplot is the initial design, and the right boxplot is the highest-Cpk design.*

shows. Of course, due to the adaptive space-filling nature of ALPS, we expect that if run 1 continued, it would eventually have achieved a high Cpk as well.

Table 3.13: *Best 10T Opamp Designs from Four SANGRIA Runs.*

Label	**Yield**	**Cpk**	**Area (m^2)**
Initial design	26.7%	0.037	11.60e-10
Run 1 Best	100%	0.835	8.88e-10 (-23.4%)
Run 2 Best	100%	1.672	10.25e-10 (-11.6%)
Run 3 Best	100%	1.849	10.32e-10 (-11.0%)
Run 4 Best	100%	1.669	12.04e-10 (+3.80%)

3.6.4 Experiments on 30T Opamp Circuit

Figure 3.16 shows the schematic for the 30-transistor opamp. It has 56 design variables, 216 process variables, and three testbenches having ac, tran, and THD analyses respectively. Performance specifications were: gain A_V > 37.5 dB, bandwidth BW > 13.5 MHz, gain-bandwidth GBW > 300 MHz, phase margin PM > 59°, GM < -10 dB, unity gain frequency FU > 265 MHz, settling time ST < 5 ns, SR > 1.85e8 V/s, overshoot OS < 6%, and total harmonic distortion THD < -40 dB.

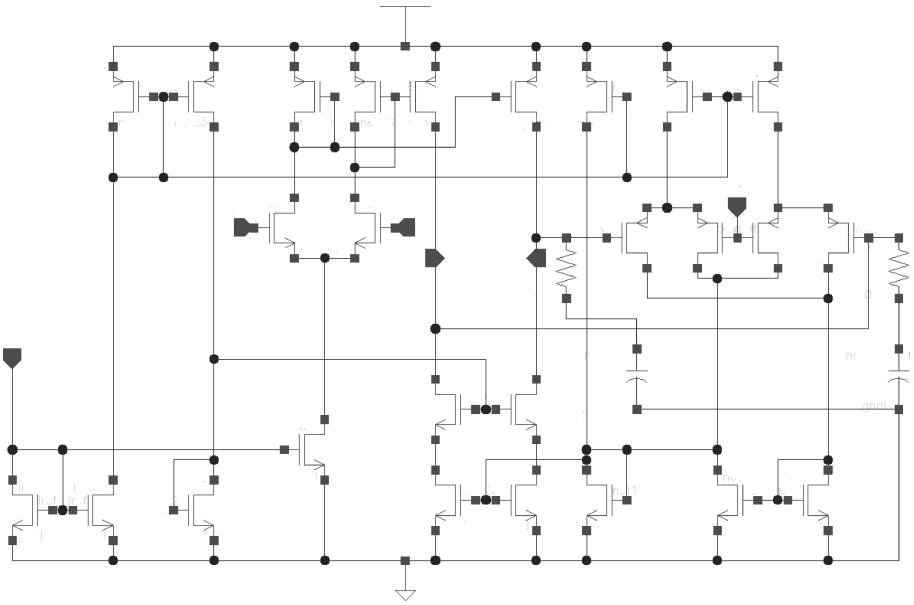


Figure 3.16: *30-device operational amplifier.*

We performed four independent runs, with resulting convergence curves shown in Figures 3.17, 3.18, 3.19, and 3.20 respectively.

In short, all four runs hit 100% yield, and kept improving Cpk significantly beyond. Each of the per-layer cost convergence curves shows the signature behavior that we examined in detail on the 10T problem. Figure 3.19 is particularly interesting, because it only got good results very late in the run; until the good results (and after them) the lower age layers repeatedly try different regions. Finally, a good region was found and the yield and Cpk increased accordingly. This reconfirms the value of SANGRIA's age-layered approach to achieving global reliability.

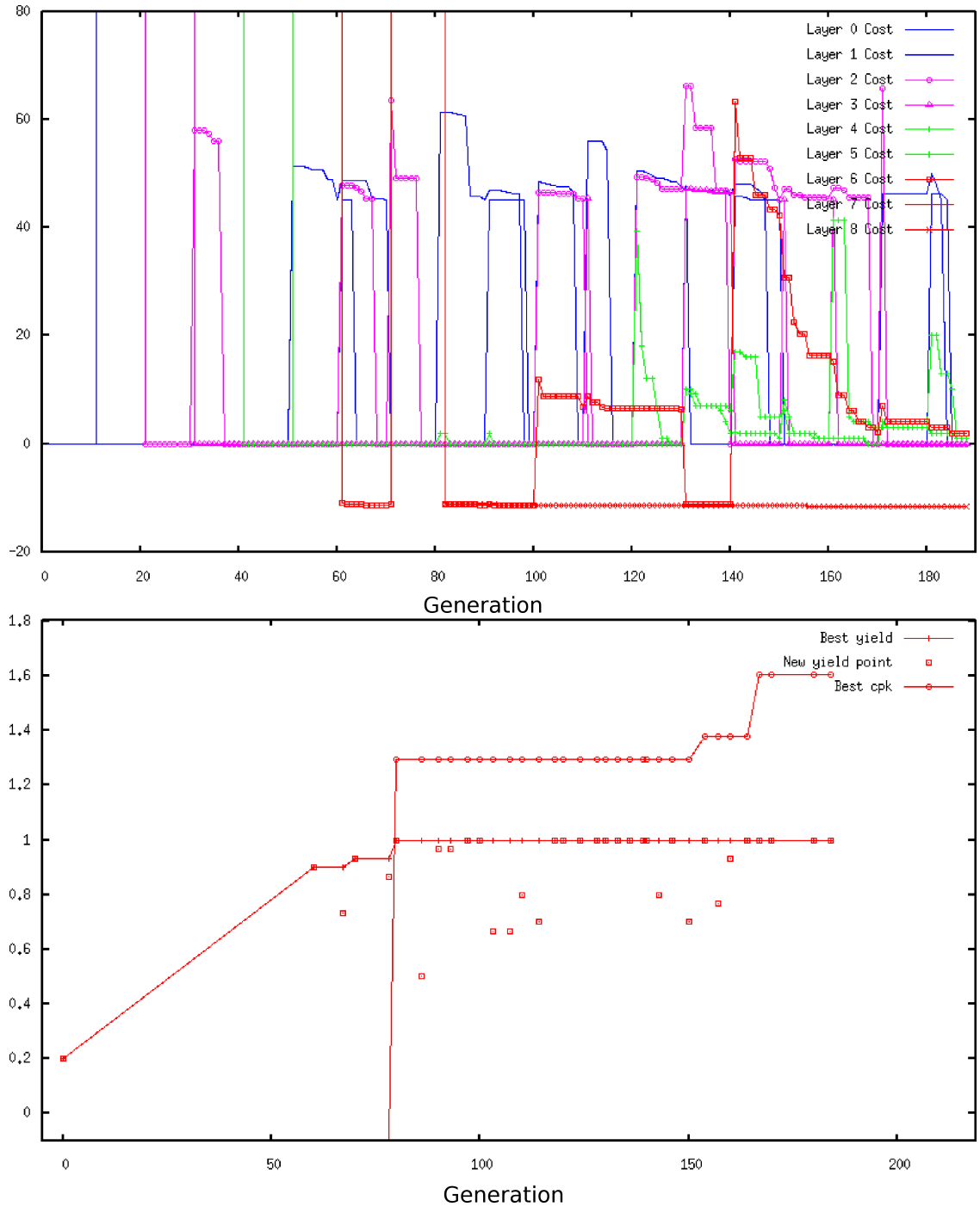


Figure 3.17: *Convergence curves for SANGRIA run 1 on 30T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

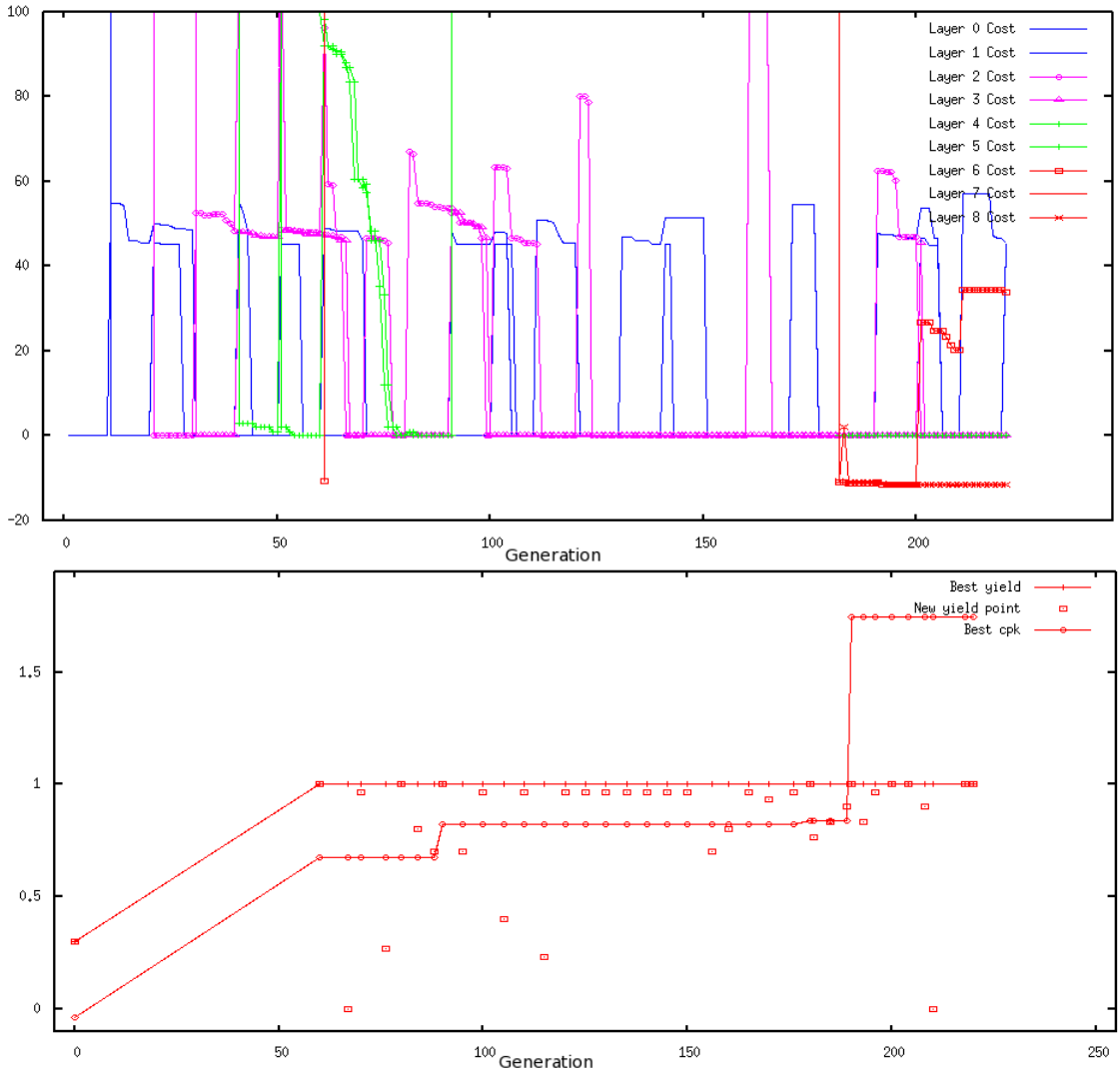


Figure 3.18: *Convergence curves for SANGRIA run 2 on 30T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

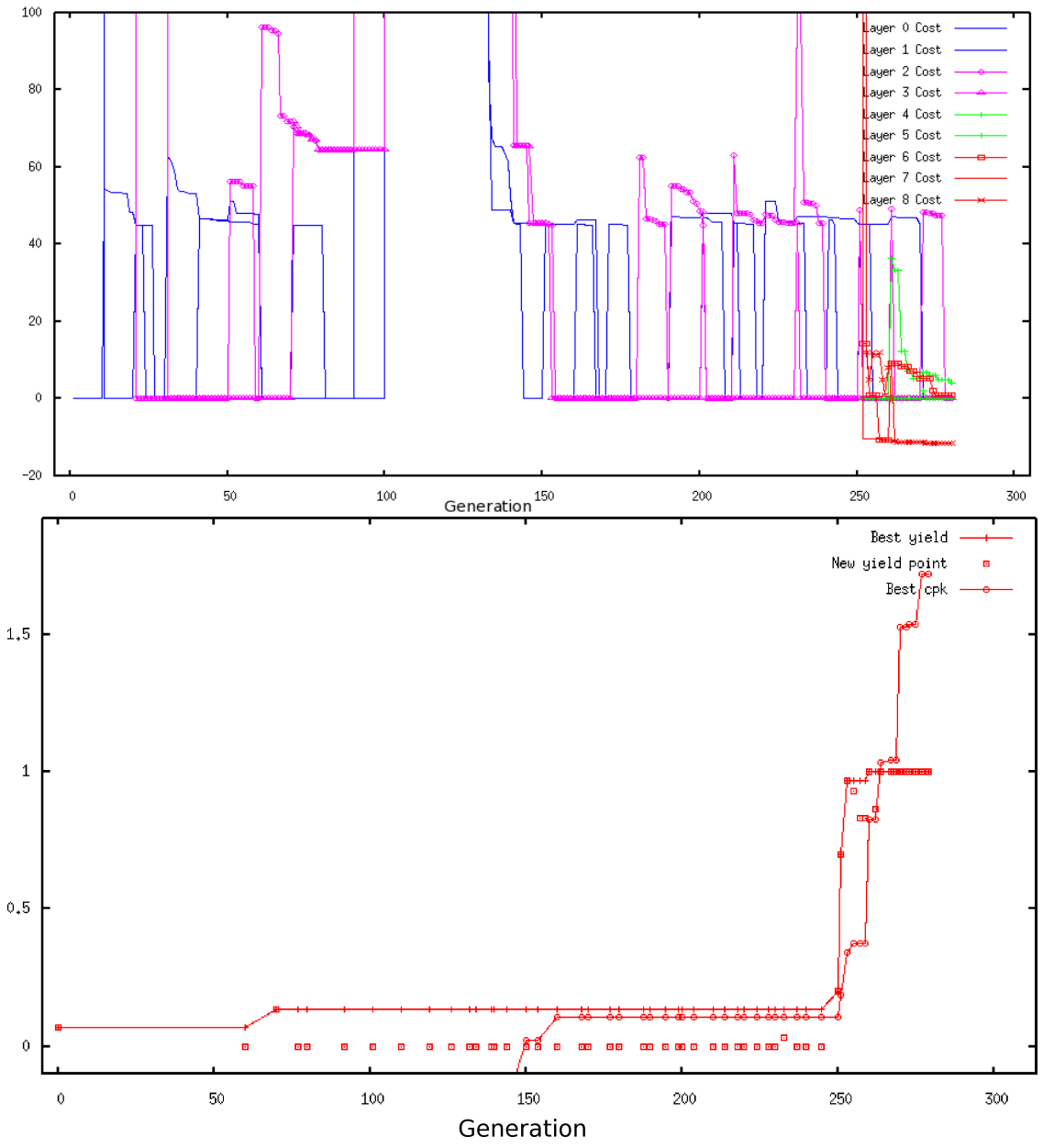


Figure 3.19: *Convergence curves for SANGRIA run 3 on 30T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

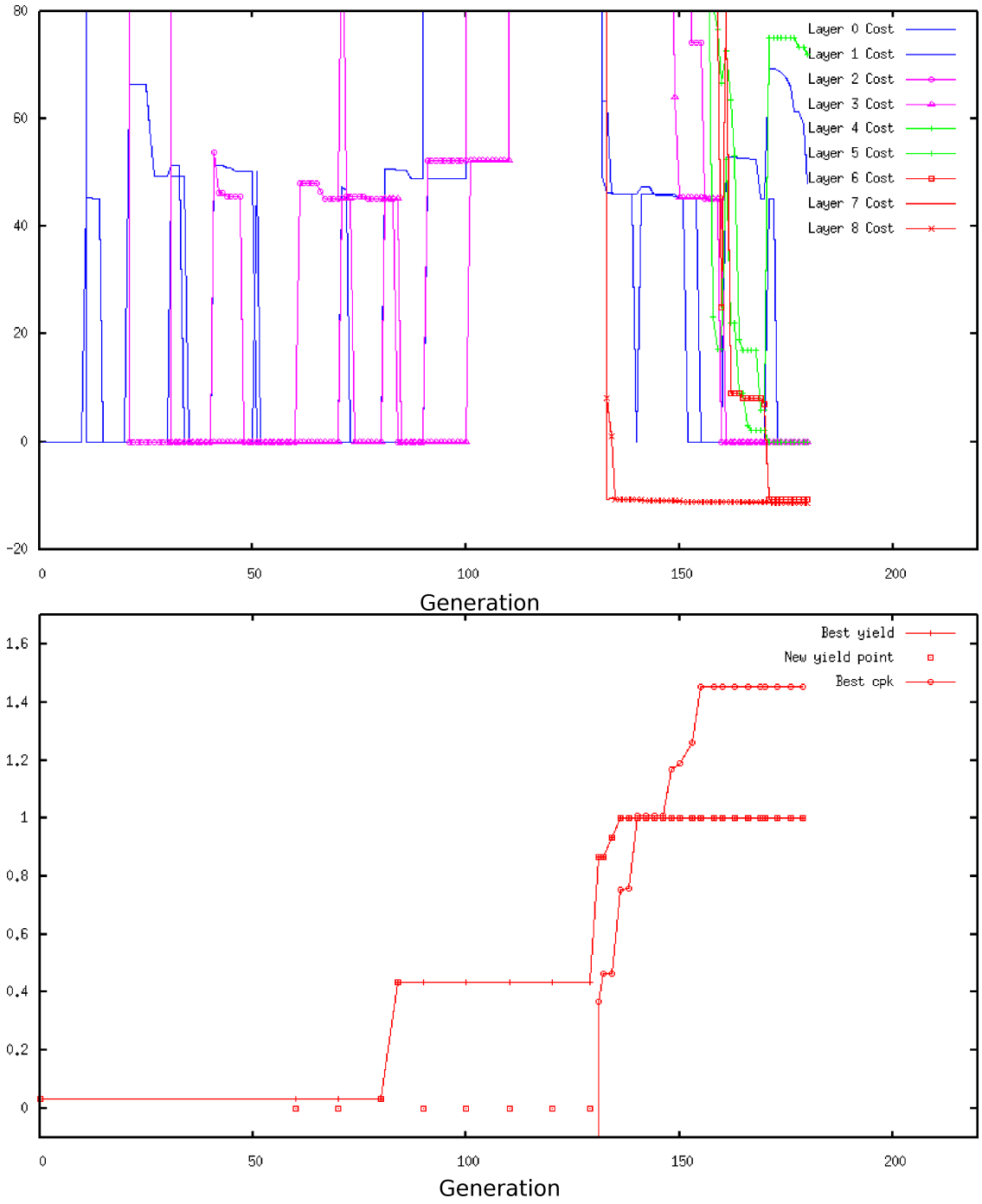


Figure 3.20: *Convergence curves for SANGRIA run 4 on 30T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

3.6.5 Experiments on 50T Opamp Circuit

Figure 3.21 shows the schematic for the 50-transistor opamp. It has 97 design variables (W's, L's, etc), 342 process variables, and three testbenches having ac, tran, and THD analyses respectively. Therefore, these experiments demonstrate the ability of SANGRIA to scale to a very large number of design variables and an even larger number of process variables. Performance specifications were: gain A_V > 30 dB, bandwidth BW > 2.3 MHz, gain-bandwidth GBW > 50 MHz, phase margin PM > 65°, gain margin GM < -5 dB, unity gain frequency FU > 50 MHz, settling time ST < 15 ns, SR > 1.5e8 V/s, overshoot OS < 5%, and total harmonic distortion THD < -40 dB.

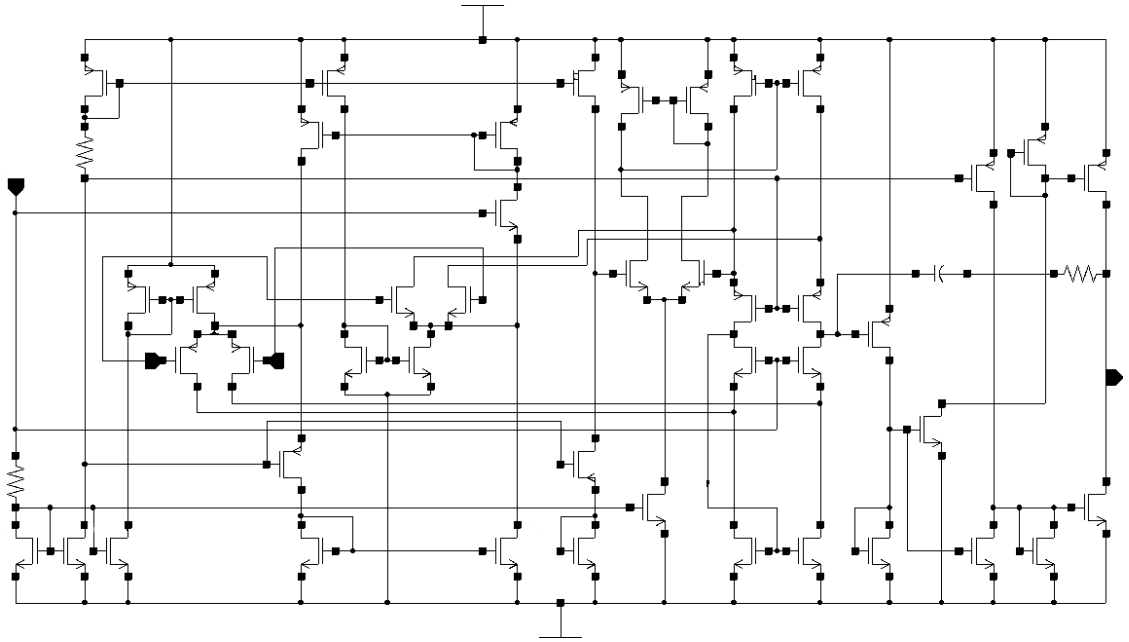


Figure 3.21: *50-device operational amplifier.*

We performed two independent runs, which are shown in Figures 3.22 and 3.23 respectively.

The first run hit a yield of 100% within 80 generations, and kept improving its Cpk significantly beyond that.

The second run almost hit a yield of 100% within its pre-allocated runtime. Upon inspection of the cost convergence curves, we see that almost all the age layers consistently hit a cost of 0 very quickly, and stayed there. Recall that SANGRIA has a "speedup" where if a layer's cost is 0, then it ignores further evolution of that layer until the next age gap. Since fewer age layers are evolving aggressively, there is less opportunity for SANGRIA to explore its way out. There is some evolution, however, indicated by the upward spikes in the right third of the cost convergence curves; so we expect that eventually SANGRIA will hit the target due to its continued space-filling sampling to explore new regions. From a user's perspective, the user would be able to observe the convergence of best cost vs. generation for each age layer, and could therefore observe that progress is

being made. So in an industrial setting, he would just continue to run SANGRIA until he observes that progress has stagnated, or that he has achieved the target yield (the more likely scenario).

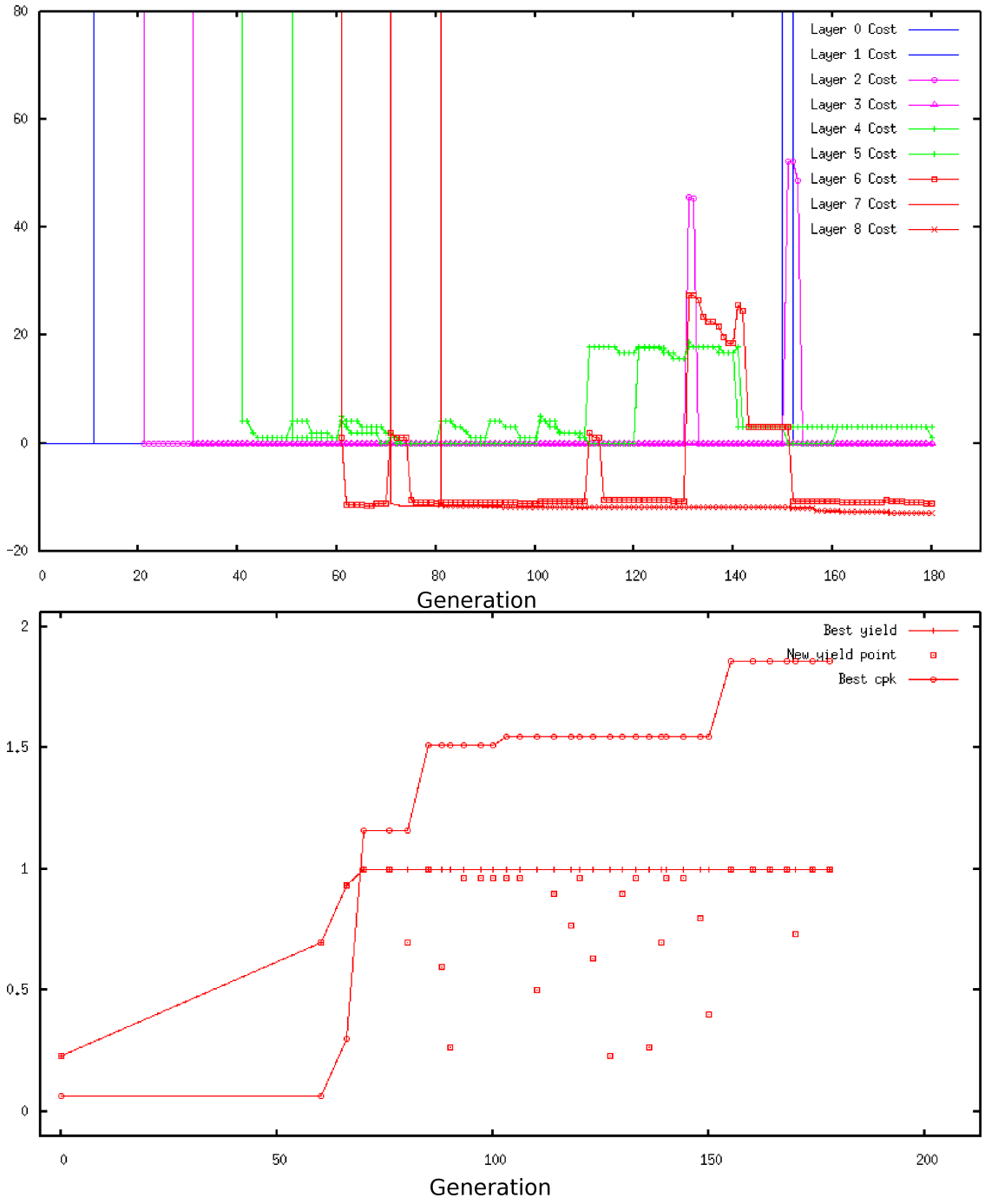


Figure 3.22: *Convergence curves for SANGRIA run 1 on 50T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

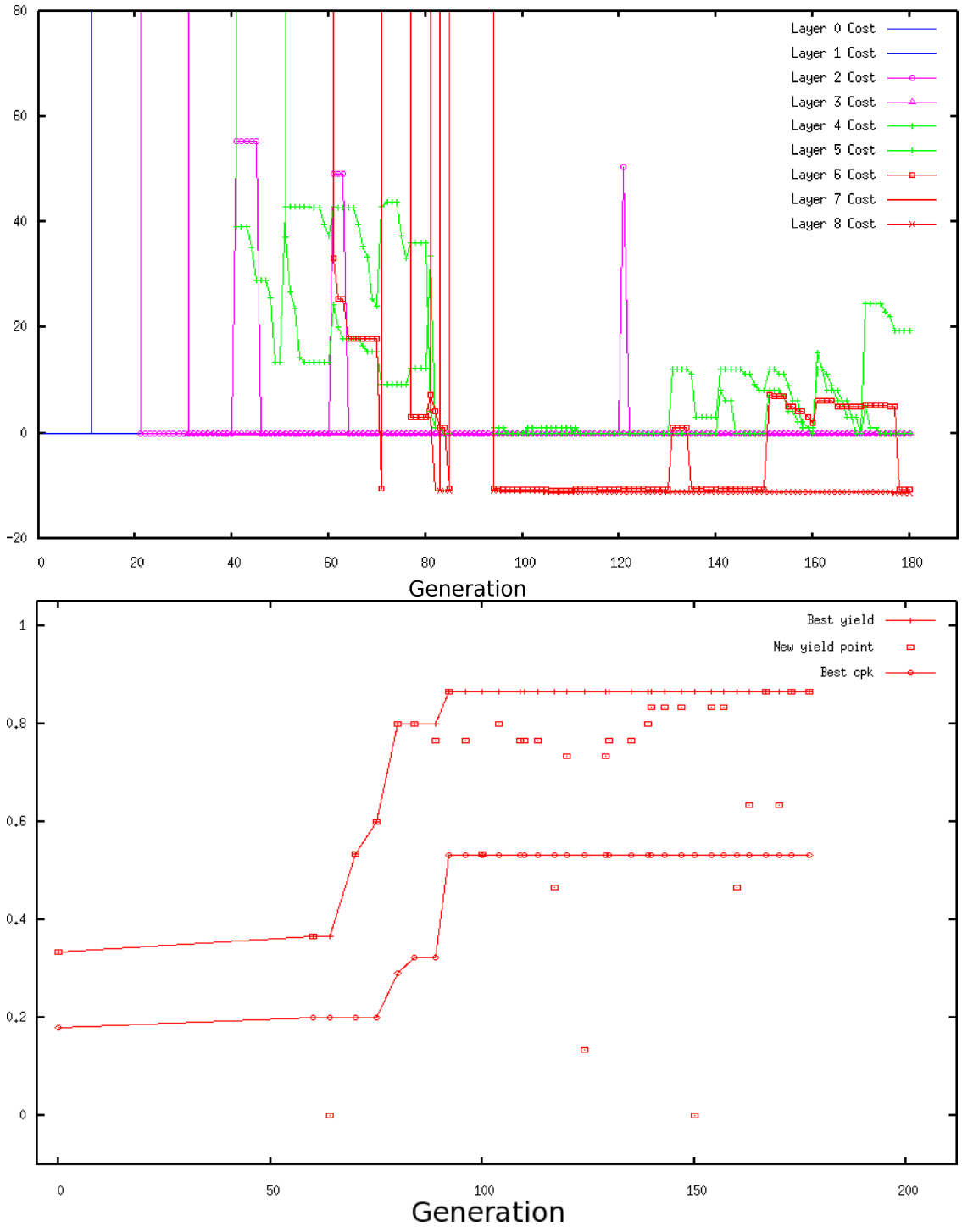


Figure 3.23: *Convergence curves for SANGRIA run 2 on 50T opamp. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

3.6.6 Experiments on Voltage Reference (vref) Circuit

The schematic for the voltage reference is shown in figure 3.24. It has 12 devices, 28 design variables, and 106 process variables. It has two ac testbenches, each with three environmental points having three environmental variables. Performance specifications were: power PWR < 0.111 mW, temperature coefficient TC < -20°C, minimum temperature $TMIN$ < -20°C, maximum temperature $TMAX$ > 85°C, voltage-change reference $DVREF$ < 600, minimum voltage $VMIN$ < 0.78 V, maximum voltage $VMAX$ > 2.8 V.

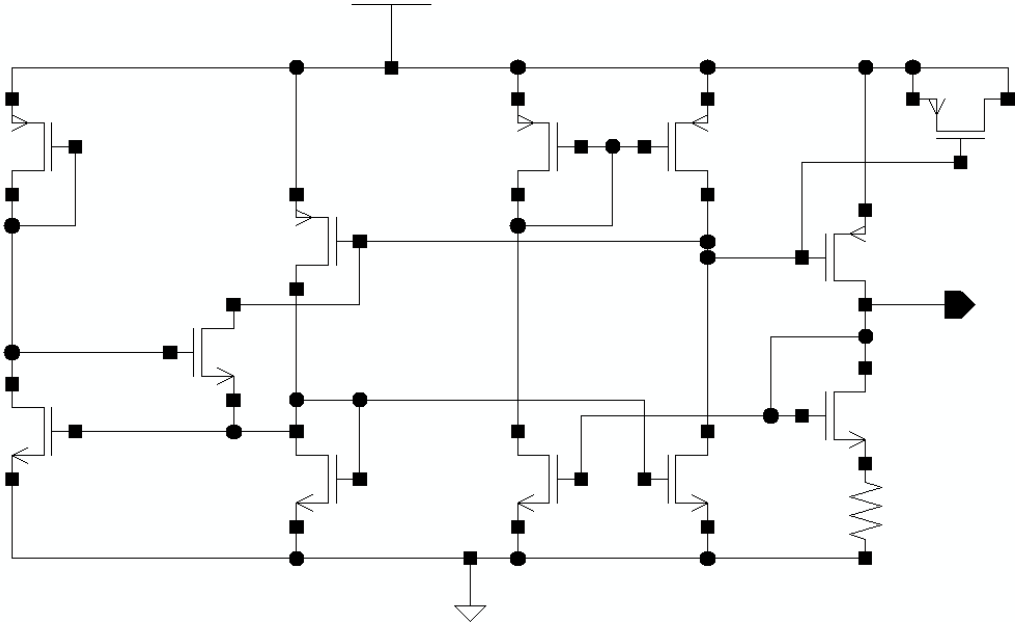


Figure 3.24: *Voltage reference schematic.*

We performed four independent runs. The convergence curves which are shown in Figures 3.25, 3.26, 3.27, and 3.28 respectively. Each Figure shows best yield vs. generation, best Cpk vs. generation, and best cost vs. generation for each age layer.

In short, all four runs hit 100% yield, and kept improving Cpk beyond. Once again, each of the per-layer cost convergence curves shows the signature behavior that we examined in detail on the 10T problem.

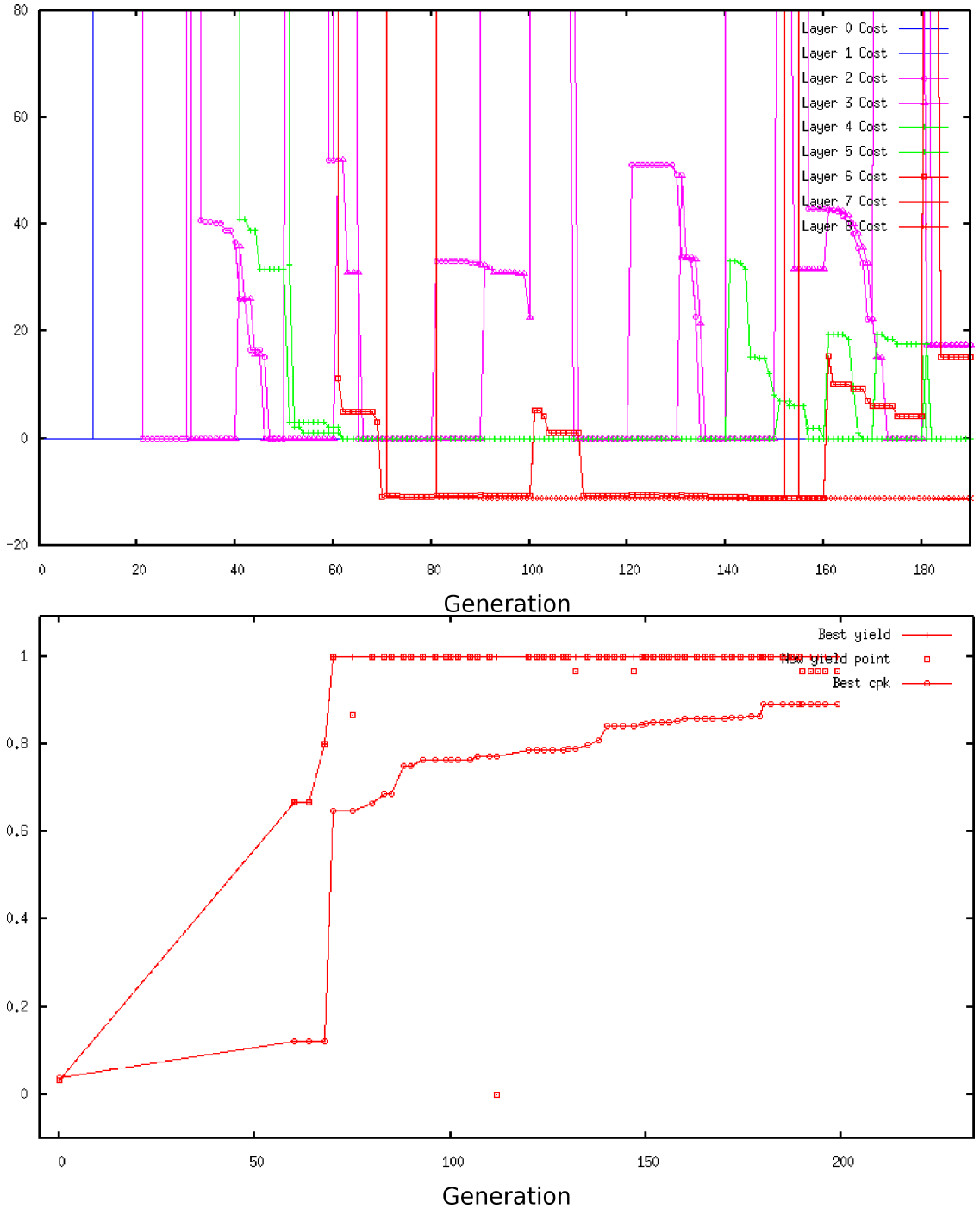


Figure 3.25: *Convergence curves for SANGRIA run 1 on voltage reference. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

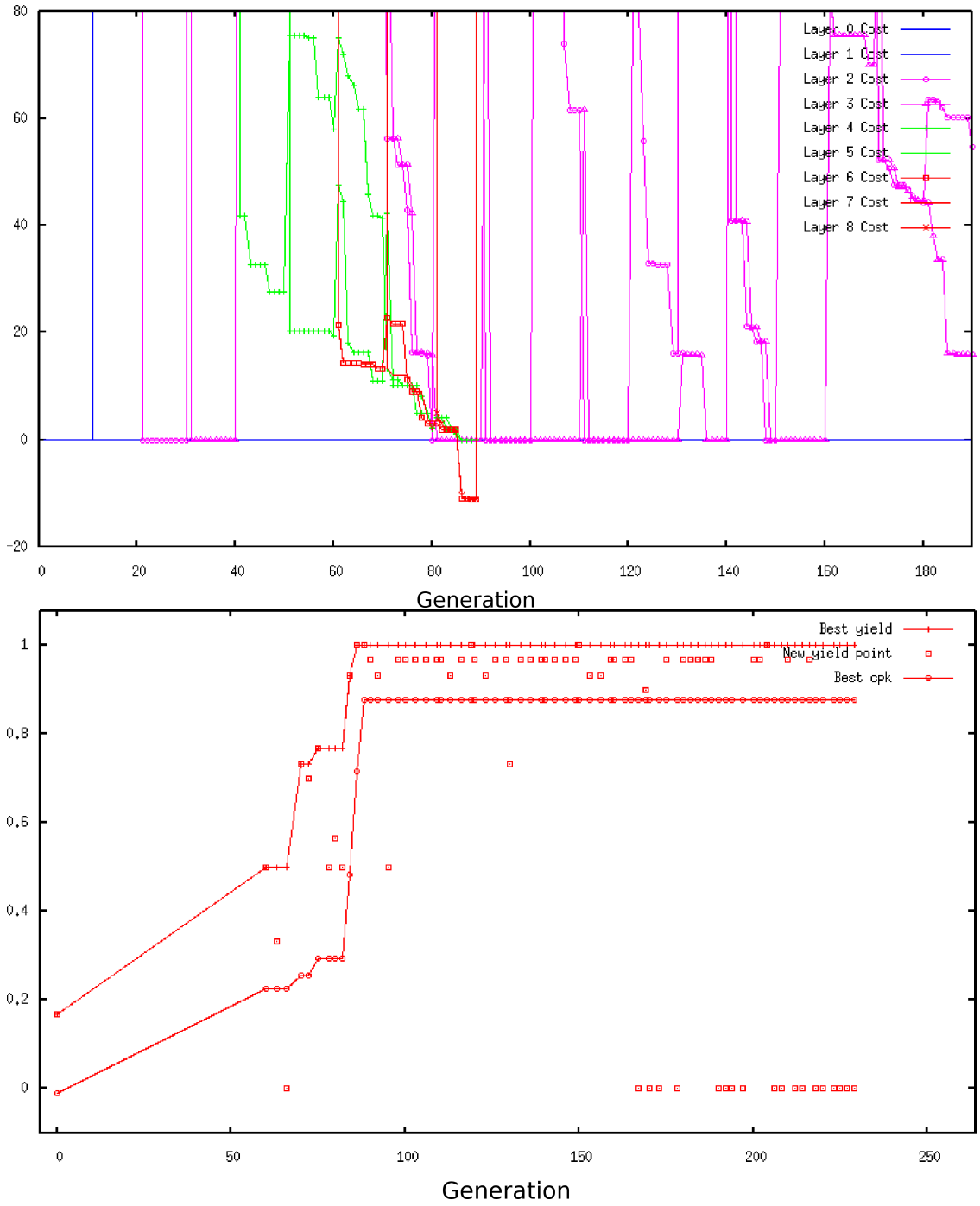


Figure 3.26: *Convergence curves for SANGRIA run 2 on voltage reference. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

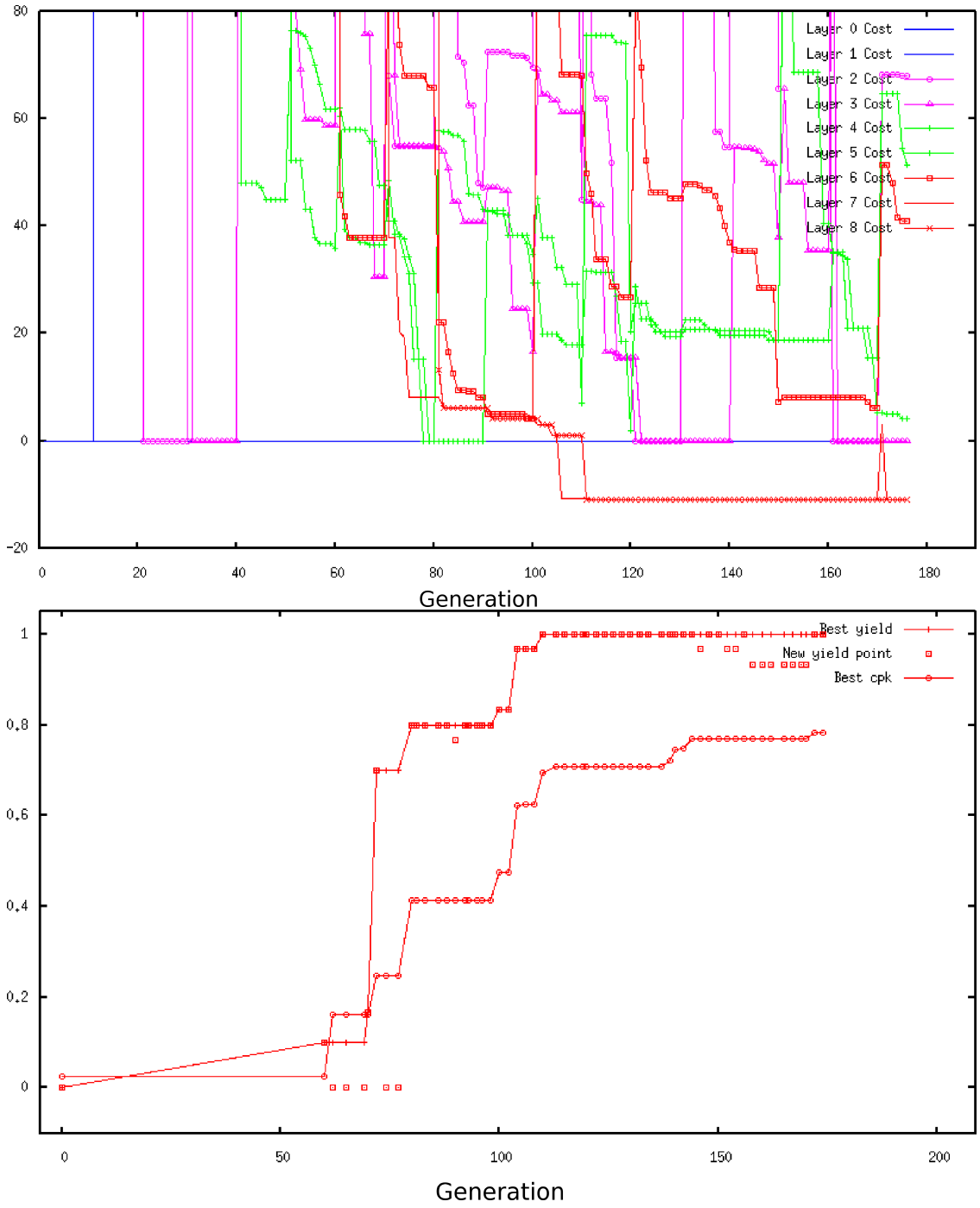


Figure 3.27: *Convergence curves for SANGRIA run 3 on voltage reference. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

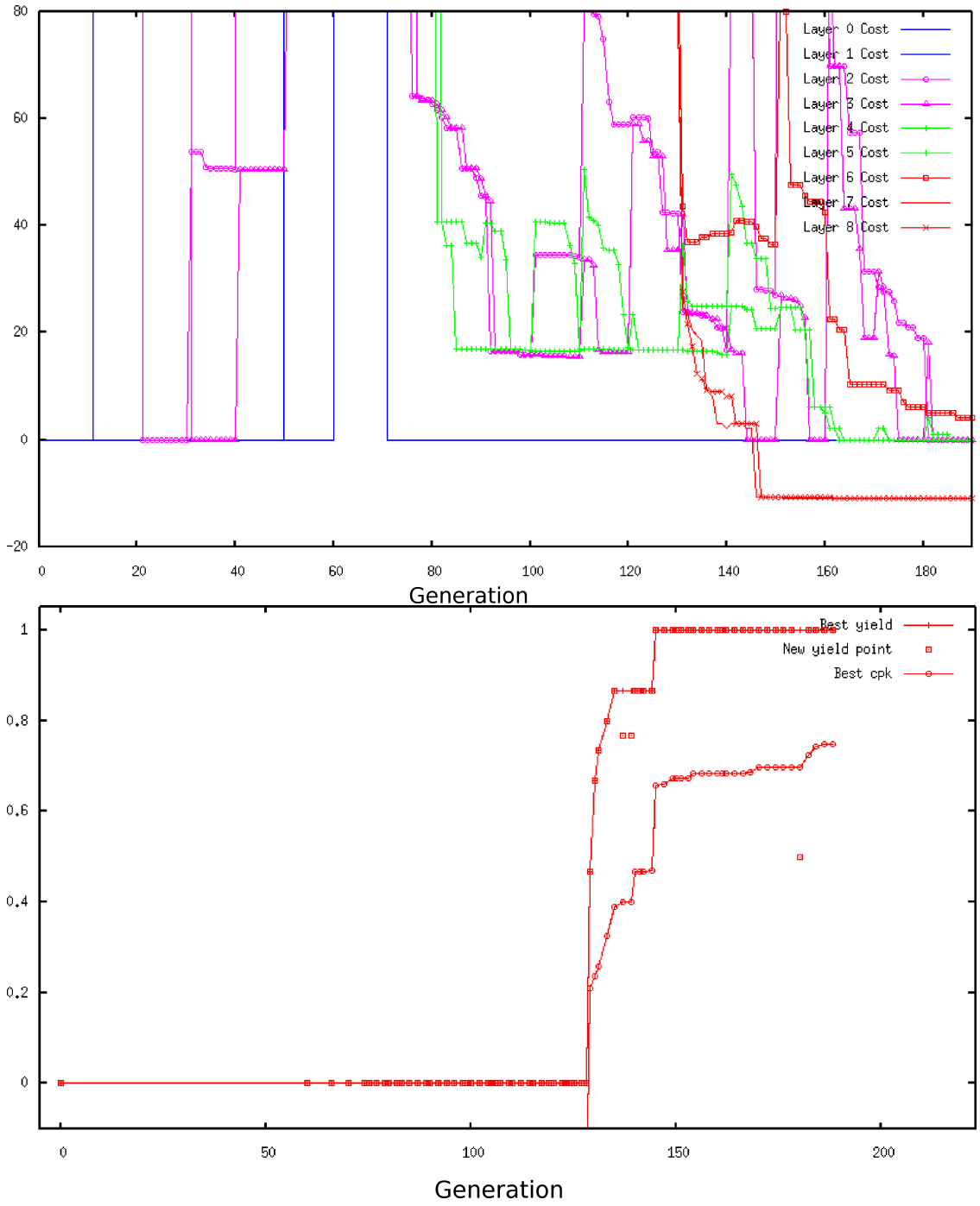


Figure 3.28: *Convergence curves for SANGRIA run 4 on voltage reference. Top: best yield/Cpk vs. generation. Bottom: best cost vs. generation.*

3.6.7 Summary of Experimental Results

Table 3.14 summarizes the experimental results of the 14 SANGRIA runs across four different circuit test problems.

Table 3.14: *Summary of SANGRIA Experimental Results.*

Problem and Run #	# Variables	Initial Yield	Final Yield	Runtime
10T opamp run 1	122	26.7%	100%	< overnight
10T opamp run 2	122	26.7%	100%	< overnight
10T opamp run 3	122	26.7%	100%	< overnight
10T opamp run 4	122	26.7%	100%	< overnight
30T opamp run 1	302	20.0%	100%	< overnight
30T opamp run 2	302	20.0%	100%	< overnight
30T opamp run 3	302	20.0%	100%	< overnight
30T opamp run 4	302	20.0%	100%	< overnight
50T opamp run 1	489	23.3%	100%	< overnight
50T opamp run 2	489	23.3%	83.3%	< overnight
vref run 1	146	16.7%	100%	< overnight
vref run 2	146	16.7%	100%	< overnight
vref run 3	146	16.7%	100%	< overnight
vref run 4	146	16.7%	100%	< overnight

3.7 On Scaling to Larger Circuits

This section discusses how SANGRIA would address circuits with 1000 or 10,000 or more devices, i.e. system-level circuits. The short answer is that system-level designs can be hierarchically decomposed, and that each node in the hierarchy can be explored by SANGRIA (or a modified version).

There are several alternative hierarchical design methodologies, and several ways to estimate performance at each node in the hierarchy. SANGRIA can fit into most combinations. Methodologies include top-down constraint-driven approach (TDCD) [Cha1997], and multi-objective bottom up approach (MOBU) [Eec2005, Eec2007], which section 1.2.3 discussed further. For SANGRIA to handle MOBU or bottom-up computation of feasibility regions, it would be modified to be multi-objective.

Then the question is whether or not SANGRIA can handle the most complex node within a design hierarchy. System-level and higher designs tend to have 5-20 components at their level of the hierarchy, whereas cell-level designs have typically 10-50, (and sometimes as many as 250). So, by demonstrating the ability to handle 50-devices, SANGRIA should be applicable anywhere in the hierarchy, and therefore handle designs with 1000 or 10,000 or more devices.

3.8 Conclusion

Recall from the review in chapter 2.1 that none of the optimization approaches in the literature or industry had a combination of accuracy, reasonable efficiency, and globally reliable convergence. This chapter has presented a solution: SANGRIA. SANGRIA is a tool for globally-reliable, variation-aware sizing of analog circuits [Mcc2008e]. SANGRIA makes no accuracy-compromising assumptions, can handle high-dimensionality statistical SPICE models, and uses simulation in the loop, but achieves industrially-feasible runtime. Most importantly, it has globally reliable convergence, which means that the designer does not need to be concerned about whether the optimization is stuck. Designer confidence is further improved by showing visualizations of best-cost convergence per age layer.

SANGRIA's key elements are: structural homotopy with ALPS, individuals embedding a local-optimization search operator (DHC), and improved model-building optimization (MBO) combining scalable regression (SGB ensembles) and inner multiobjective optimization.

We have tested SANGRIA on four different circuit problems from two different circuit classes (opamp and voltage reference), in a total of 14 runs. The problems ranged from 10-device circuits having 21 design variables and 91 process variables, up to 50-device circuits with 97 design variables and 342 process variables. In 13 / 14 runs, SANGRIA was able to successfully attain 100% yield and further improve Cpk within an industrially feasible number of simulations and runtime, despite the extremely high number of parameters and evidence of multimodality. In contrast, no other approaches have reported global yield optimization results for circuits with this many transistors, and especially not this many process variables.

While this chapter has presented a *design aid* to support global variation-aware sizing via optimization, the next chapter presents an *insight aid* for variation-aware sizing via the extraction of whitebox performance models.

Chapter 4

Knowledge Extraction in Sizing: CAFFEINE

All models are wrong, but some are useful.
–George E.P. Box

4.1 Introduction and Problem Formulation

4.1.1 Chapter Summary

This chapter presents a tool to accelerate designer insight in sizing, by extracting whitebox performance models. The tool is called CAFFEINE [Mcc2005a, Mcc2005b, Mcc2005c, Mcc2006a, Mcc2006b, Mcc2006c, Mcc2008b, Mcc2008g]. CAFFEINE implements a method to automatically generate compact, interpretable symbolic performance models of analog circuits with no prior specification of an equation template. The symbolic models capture mappings of the design variables to individual performances or to Cpk (a robustness measure). CAFFEINE takes SPICE simulation data as input. This enables modeling of whatever SPICE handles: arbitrary nonlinear circuits, arbitrary circuit characteristics (including transient and noise performance measures), modern technology processes, environmental effects, and manufacturing variations. The possible expressions for the model are defined as a set of *canonical form functions*, structured as layers of product-of-sum terms that alternate with layers of sum-of-product terms. These canonical form functions are modeled as a *grammar*, which is subsequently searched via grammatically-constrained genetic programming. Novel evolutionary search operators are designed to exploit the structure of the grammar. By employing multi-objective optimization, CAFFEINE generates a set of symbolic models which collectively provide a tradeoff between error and model complexity.

On six test problems, the compact performance models demonstrate better prediction quality than several other state-of-the-art modeling techniques including posynomials, splines, neural networks, and support vector machines.

We also describe techniques to scale CAFFEINE to handle problems with more than 100 input variables, validated by further experiments.

4.1.2 Motivation

Both *symbolic analysis* and *symbolic modeling* aim to derive human-interpretable expressions of analog circuit behavior [Rut2007]. Symbolic analysis extracts the expressions via topological analysis of the circuit, whereas symbolic modeling extracts the expressions by using SPICE simulation data. These expressions have the same applications: knowledge acquisition and educational / training purposes, analytic model generation for automated circuit sizing, design space exploration, repetitive formula evaluation including statistical analysis, analog fault diagnosis and testability analysis, and analog behavioral model generation [Gie2002b]. In particular, a tool that can help a designer improve his understanding of a circuit is highly valuable, because it leads to better decision-making in circuit sizing, layout, verification, and topology design, regardless of the degree of automation [Mcc2005a]. Therefore, approaches to generate symbolic expressions are of great interest.

Historically, symbolic analysis came first. Notable approaches include ISAAC [San1989, Gie1989], ASAP [Fer1991a, Fer1991b], SYNAP [Sed1988, Sed1992], SAPEC [Man1991], SSPICE [Wie1989], SCYMBAL [Kon1988], SCAPP [Has1989], Analog Insydes [Som1993], and CASCA [Flo1993]. These tools differ in terms of: analysis domain (s-domain, z-domain, dc domain), device-level vs. system-level analysis, use of small-signal linearization, support for mismatching, support for approximations (for better interpretability), support for weakly nonlinear circuits, support for hierarchical analysis (for better scalability), and how the problem is actually formulated (modified nodal analysis, signal-flow graph, admittance matrix, etc.). The paper [Gie2002b] is a tutorial. There has evidently been great interest in symbolic analysis. However, its main weakness has traditionally been the limitation to linearized and weakly nonlinear circuits. This was recently overcome via piecewise-linear/polynomial modeling approaches like [Man2003, Yang2005, Dong2008]. Those new approaches, however, return expressions that are hard to interpret, which is counter to the main motivations of symbolic analysis

Leveraging SPICE simulations in modeling is promising because simulators readily handle nonlinear circuits, environmental effects (e.g. temperature, power supply voltage, loads), manufacturing effects, different technologies, new effects (e.g. proximity [Dre2006]), and more. Simulation data has been used to train many types of regressors, including linear models [Gra2007, Mcc2005c, Li2008c], posynomials [Dae2002, Dae2003, Dae2005, Agg2007], polynomials [Li2007b, Mcc2005c], splines [Wol2004, Mcc2005c], neural networks [Van2001, Wol2003, Mcc2005c], boosted neural networks [Liu2002, Mcc2005c], support vector machines [Ber2003, Kie2004, Ding2005, Ding2005b, Mcc2005c], latent variable regression [Sin2007, Li2008b], and kriging [Mcc2005c, Yu2007b]. However, such models either follow an overly restrictive functional template which limits their applicability, or they are opaque and thus provide no insight to the designer. Less opaque flows exist, such as visualizing CART trees [Bre1984] or extracting rules from neural networks [Tic1999]. However, these approaches do not give the functional relations that symbolic models provide.

The aim of *symbolic modeling* as defined in this thesis is to *use simulation data* to generate interpretable mathematical expressions for circuit applications, typically relating the

circuit performances to the design variables. Symbolic modeling has similar goals to symbolic analysis, but a different core approach to solving the problem. A symbolic model ψ maps an N_d-dimensional input vector of sizings and biasings $\boldsymbol{x}$ to a scalar approximator of circuit performance y. That is: $\psi : \boldsymbol{x} \mapsto \hat{y}$.

In [Dae2002, Dae2003, Agg2007], posynomial-based symbolic models are constructed. The main problem is that the models are constrained to a predefined template, which restricts the functional form. Also, the models have dozens of terms, limiting their interpretability for designers. Finally, the approach assumes that posynomials can fit the data; in analog circuits there is no guarantee of this, and one might never know in advance. There have also been advances in building quadratic polynomial models [Li2006, Li2007b, Feng2006], but polynomials also have a restrictive structure that limit their usefulness.

4.1.3 Approach

The problem we address in this chapter is how to generate symbolic models with more *open-ended* functional forms (i.e. without a pre-defined template), for arbitrary nonlinear circuits and circuit characteristics, and at the same time ensure that the models are *interpretable*. A target flow that reflects these goals is shown in Figure 4.1.

We approach the question by posing it as a search problem in the space of possible functional-form *trees*. An appropriate search algorithm is then genetic programming (GP) [Koza1992], which conducts the search by iterative evolution of a population of points. (A non-population-based approach like [Lan1987] is possible, which examines just one search point at a time. However, because it is single-point, then it cannot swap subexpressions between candidate functions. This compromises its ability to explore the search space effectively.)

GP generates symbolic expressions without using a template, but those functions are overly complex. So, we extend GP via a grammar specifically designed to have simpler but accurate, *interpretable* symbolic models. We name the approach CAFFEINE: Canonical functional form expressions in evolution.

The contributions of this chapter are as follows:

- To the best of our knowledge, the first-ever tool for *template-free symbolic modeling*. Because it uses SPICE simulation data, it allows modeling of any nonlinear circuit characteristic, or analysis (including transient, noise, and more).
- The approach returns models that are compact and understandable, yet with good accuracy. In fact, it returns a *set* of possible models that *trade off* accuracy and complexity.
- A GP-specific contribution is a specially designed grammar and related operators, which ensures that all functions explored follow a *canonical form*, making them directly interpretable. The grammar plugs into any grammatical-GP engine.
- Finally, this chapter proposes techniques to *scale* symbolic modeling to problems with more than 100 input variables. The techniques are: subtree caching [Kei2004], gradient-directed regularization [Fri2004] to simultaneously prune basis functions and

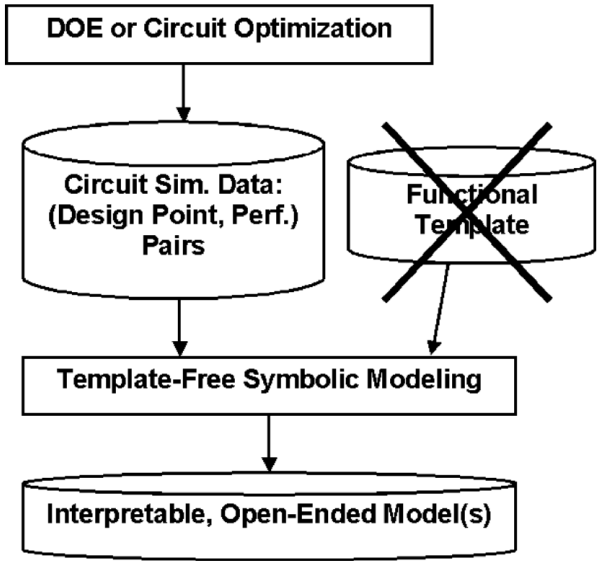


Figure 4.1: *Template-free symbolic modeling flow.*

learn remaining coefficients, a pre-evolution step of filtering single-variable expressions, and always considering all the linear basis functions.

4.1.4 Problem Formulation

The modeling problem that we address has the flow of Figure 4.1. It is formulated as follows:

Given:

- $\boldsymbol{X}$ and $\boldsymbol{y}$: A set of $\{\boldsymbol{x_j}, y_j\}, j = 1..N$ data samples where $\boldsymbol{x_j}$ is a N_d-dimensional design point j and y_j is a corresponding circuit performance value measured from SPICE simulation of that design. Design of experiments (DOE) *or* circuit optimization can be used to generate the data samples.
- **No** model template

Determine:

- A set of symbolic models M that together provide the Pareto-optimal tradeoff between minimizing model complexity f_1 and minimizing future model prediction error f_2.

The formulation is a constrained optimization problem:

$$M = minimize \left\{ \begin{matrix} f_1 & = & complexity(\psi) \\ f_2 & = & E_{x,y}L(y, F(\boldsymbol{x}; \psi)) \end{matrix} \right\} s.t. \psi \in \Psi \tag{4.1}$$

where Ψ is the space of template-free symbolic models. The algorithm will traverse Ψ to return a Pareto-optimal set $M = \{\psi_1^*, \psi_2^*, \ldots, \psi_{N_M}^*\}$. Each model ψ maps an N_d-dimensional input $\boldsymbol{x}$ to a scalar circuit performance y, i.e. $\hat{y} = \psi(\boldsymbol{x})$. Equivalently,

$y = F(\boldsymbol{x}; \psi)$. Complexity is *some* measure that differentiates the degrees of freedom between different models. Details are in equation (4.5).

$E_{x,y}L$ is the expected loss for a given ψ over future predictions in the distribution $pdf(\boldsymbol{x})$, where L is the squared-error loss function [Fri2003]:

$$L(y, F(\boldsymbol{x}; \psi)) = (y - F(\boldsymbol{x}; \psi))^2)/2 \tag{4.2}$$

Section 4.4.1 will describe how an approximation for $L()$ is computed.

By definition, no model in the Pareto-optimal set M dominates any other model. A model ψ_a "dominates" another model ψ_b if $\{f_j(\psi_a) \leq f_j(\psi_b)\}\forall j$, and $\{f_j(\psi_a) < f_j(\psi_b)\}\exists j$; $j = \{1, 2\}$ in our case. That is, to be Pareto-optimal, a model must be at least as good as any model on both objectives, and better than any model in one objective.

4.1.5 Chapter Outline

The rest of this chapter is organized as follows.

Section 4.2 presents background on genetic programming / symbolic regression, and identifies specific issues with the status quo approaches. Section 4.3 introduces the heart of CAFFEINE: canonical form functions. Section 4.4 describes the reference search algorithm, which uses multi-objective genetic programming and a grammar to constrain to canonical form functions. Section 4.5 describes the first round of experiments. Section 4.6 describes how to scale up CAFFEINE to larger problems, with corresponding experiments in section 4.7. Section 4.8 describes other applications of CAFFEINE. Section 4.9 discusses the sensitivity of canonical form functions to the search algorithm employed. Section 4.10 concludes.

4.2 Background: GP and Symbolic Regression

4.2.1 Background: High-Level Issues

Genetic Programming (GP) [Koza1992] is an evolutionary algorithm, with the distinguishing characteristic that GP individuals (points in the design space) are *trees*. Since a symbolic model is a function and can be represented as a tree, the search for template-free models can be accomplished by GP search. In the GP literature, this is called *symbolic regression* (SR).

The functional form of results from canonical GP is completely unrestricted. While this sounds promising compared to the restrictions of fixed-template regression, it actually goes a little too far: an unrestricted form is almost always difficult to analyze. GP-evolved functions can be notoriously *complex* and *un-interpretable*. For example, [Koza1992] showed functions so bloated [Sou2002] that they take up a full page of dense text. A recent paper complains: "[GP-evolved] expressions can get, as we have seen, quite complex, and it is often extremely difficult to understand them without a fair bit of interaction with a tool such as *Mathematica*" [Kir2004].

We can see for ourselves. Using a dataset from section 4.5, canonical GP evolution returned the following "optimized" expression:

$-1.40 * (vsg1 + max(vsg5, max(max(max(vsg5, max(vsg3 + vgs2, min(vsg3, abs(1/vds2))) - log10(vsd5)), min(ib2, abs(sqrt(abs(id1))))) - log10(vsd5), max(id2, min(vsg3, abs(sqrt(abs(log10(id2)))))) + log10(vsd5)) - min(vsg3, abs(sqrt(abs(id1)))) - log10(vsd5)))$

Improvements are clearly needed. The first step is to identify and enumerate the *specific* issues that SR has.

4.2.2 Background: Specific SR Issues

This section examines SR challenges and GP approaches (or lack of approaches) to handle each. Most of them are specific to SR.

Managing Complexity. Occam's Razor is the guide here: the simplest model that describes the data is usually the correct one. Complexity is typically dependent on measures like tree depth and node count. In GP, expression-simplification processes are of two varieties: non-SR and SR-specific.

Non-SR techniques include:

- penalizing complex solutions ("parsimony pressure") [Koza1992],
- having complexity as a second objective and using a multi-objective algorithm [Smi2005, Kor2006],
- maximum tree depth [Koza1992],
- uniform operators such that depths never grow [Poli1999b], and
- other "bloat control" methods, e.g. [Pan2004].

The SR-specific approach is to do symbolic simplification, either (a) automatically during or after evolution with a symbolic math tool like Mathematica, or (b) manually after evolution.

Excessive Compounding of Nonlinear Operators. GP gives equal treatment to function operators, whether they are linear or nonlinear (e.g. '+' vs. log()). The result is that even a very small tree which would pass GP-parsimony standards could be not interpretable by humans. An example is $tan(exp(sin(x)))$: three compounded nonlinear operators is too much, and even two is questionable. Maximum tree depth might handle this, but unfortunately the tree must still be large enough to handle other reasonable combinations of expressions such as polynomials.

Finding Coefficient Values. Induced expressions might have real-valued coefficients which must be determined during GP search. Coefficients can be either the linear "weights" on each basis function (along with the offset), or the nonlinear coefficients inside basis functions.

Linear weights can be handled by: inclusion with nonlinear coefficients; linear regression [Mck1999]; or having just one overall basis function and a simple correlation calculation to sidestep linear regression until after evolution [Kei2004b].

Nonlinear coefficients can be handled by "ephemeral random constants" [Koza1992]; by constant perturbation on a distribution, e.g. uniform [Spe2004] or Gaussian [Ang1996]; via nonlinear optimization [Top2001]; within a constant-creation grammar such as digit concatenation [Dem2005]; or even combining multiple strategies [Dem2005].

Log-Range of Coefficient Values. For some problems, coefficient values should also be able to take on a wide range of possible values that may vary by many orders of magnitude: large positive numbers like 1.0e+10, small positive numbers like 1.0e-10, zero, small negative numbers, and big negative numbers.

Some SR approaches handle it implicitly by allowing $log()$ and/or $power()$ operators to act directly on the constants, or by choosing from a discrete set of log-range variables. We have not been able to identify any work that directly addresses log-valued constants for continuous-valued numbers.

Coefficient values are just one side of the "coefficient coin"; GP must also determine where in the expression to insert each constant. Thus, in contrast to much research on coefficient values, with the exception of the linear/nonlinear distinction, there is little discussion of coefficient placement in the GP literature. Unfortunately, this means that GP-evolved equations can end up having too few constants in some places and too many in others; i.e. shortages *and* excesses.

Coefficient Shortages. Consider the expression $f(x) = log(x)$, which might appear in a typical SR run. We can rewrite it as $f(x) = w_0 + w_1 * log(w_2 + w_3 * x)$ in which it has four implicit coefficients: w_0 = 0.0, w_1 = 1.0, w_2 = 0.0, and w_3 = 1.0. The first two coefficients w_0 and w_1 are linear; the others are nonlinear.

As Figure 4.2 illustrates, GP should be able to make small steps in the space of the function's behavior by having all relevant coefficients readily available. If there is a coefficient shortage, tunability of the function is compromised.

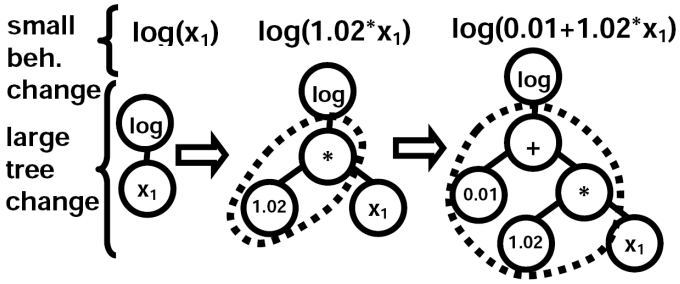


Figure 4.2: *Coefficients can be difficult to insert if not already present, even if the behavioral change is small.*

Coefficient Overabundance. Missing constants in some places is one issue, and having too many in other places is another. The GP system is evolving more parameters than it needs to. Figure 4.3 illustrates one of many examples.

Non-compact Polynomials and Rationals. In GP, it takes many terms to build up a polynomial, and sometimes those terms cancel each other out causing redundant terms,

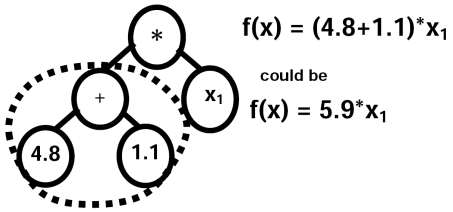


Figure 4.3: *An example of coefficient overabundance.*

as Figure 4.4 shows. In the literature, this is also handled implicitly as part of symbolic simplification.

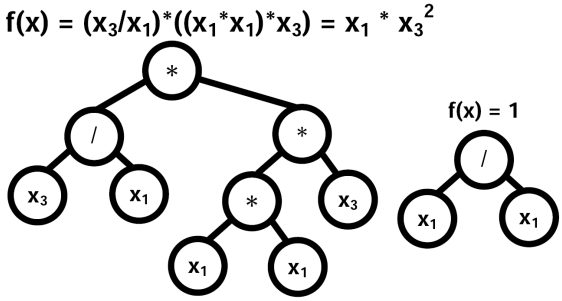


Figure 4.4: *It can take many nodes to get a simple polynomial or rational expression. They can even cancel each other out.*

Dimensional Awareness. In real-world use, functions describe something, and that "something" has units of measurement. Each input variable, and the target variable, has its own unit, such as "m/s" for a velocity variable. For a GP-evolved function to be physically meaningful, the units have to align, e.g. only like units can add, and the input variables must propagate through to the output such that the correct output unit is hit. Most SR systems ignore this, but the work of Keijzer is a notable exception. He demonstrated one system that used dimensionless values, another that biased evolution towards correct units, and a third system that had correct-by-construction units [Kei1999, Kei2001]. Keijzer noted that if there is a coefficient in front of an expression, that coefficient could conceivably have "corrective" units such that the input units translated properly into the output units. Interestingly, the existence of coefficients everywhere (implicit or explicit) causes *implicit* corrective unit transformations!

Bounded Ranges for Expression Outputs. For a given problem, each unit of measurement has a range of reasonableness. For example, velocity of a car can safely be bounded between 0 and 500 km/h. An ideal function would never allow intermediate or final expressions that go beyond unreasonable unit ranges. Most GP research ignores this, though Keijzer handles this via interval arithmetic in GP [Kei2001, Kei2003].

Bounded Ranges for Operators. Some mathematical operators are only valid for specific ranges, e.g. division "/" can only have a nonzero denominator, and $log()$ needs a pos-

itive argument. GP research typically handles this by "protected operators" [Koza1992] or simple exception handling, though the safest and most elegant way is probably interval arithmetic [Kei2001, Kei2003].

The challenge is to find a way to restrict the functional form enough to overcome each of these SR problems, without constraining away any possible forms. The next section discusses CAFFEINE, and how it overcomes these issues.

4.3 CAFFEINE Canonical Form Functions

The design of CAFFEINE follows two guidelines:

- ensure maximum expressiveness per node, and
- make all candidate functions directly interpretable.

Figure 4.5 shows the general structure of a CAFFEINE function. It alternates between levels of *sum-of-product* expressions and *product-of-sum* expressions. Each sum-of-product expression is a weighted linear add of an overall offset term plus weighted basis functions. A basis function is a combination of product terms, where each product term is a polynomial/rational, zero or more nonlinear operators, and zero or more unity operators. Each product term acts as a "gate" to the next sum-of-products layer.

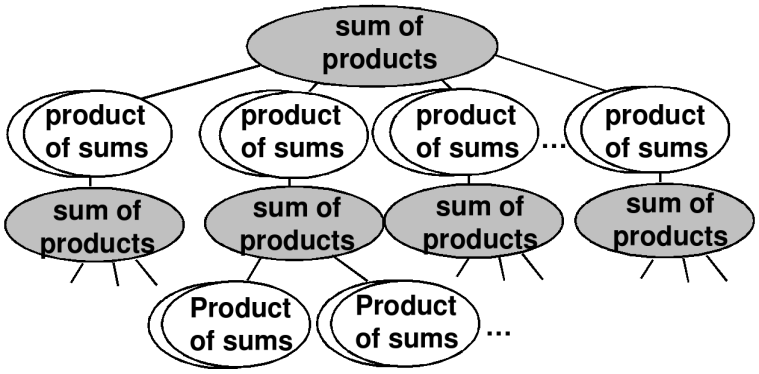


Figure 4.5: *CAFFEINE evolves functions of this canonical form. While it can go deeper indefinitely, it is typically only as deep as shown in order to retain human interpretability.*

An example function is shown in Figure 4.6. We now describe how the function aligns with the CAFFEINE object, a tree. In the "$7.1/x_3$" part of the function, the 7.1 is the tree's top left "w_0" and the "$1/x_3$" is its neighboring "poly/rat'l of vars". The "1.8" corresponds to top "w_1", and the "x_1" is the its neighboring "poly/rat'l of vars".

The function's "log" corresponds to "nonlinear func", which in the tree holds the "weighted linear add" term "$-1.9 + 8.0/x_1 + 1.4 * x_2^2/x_3$". That term itself breaks down: function's the "-1.9" is the tree's lower "w_{offset}"; "$8.0/x_1$" corresponds to the tree's lower left "w_0" * "poly/rat'l of vars"; and "$1.4 * x_2^2/x_3$" corresponds to the tree's lower right "w_1" * "poly/rat'l of vars".

Note how CAFFEINE places coefficients only where they are needed, and nowhere else. This characteristic, which is distinct from traditional GP approaches to symbolic regression, is critical to generating interpretable functions. A coefficient on everything also means that they can be "dimensional transforms" to resolve dimensional awareness.

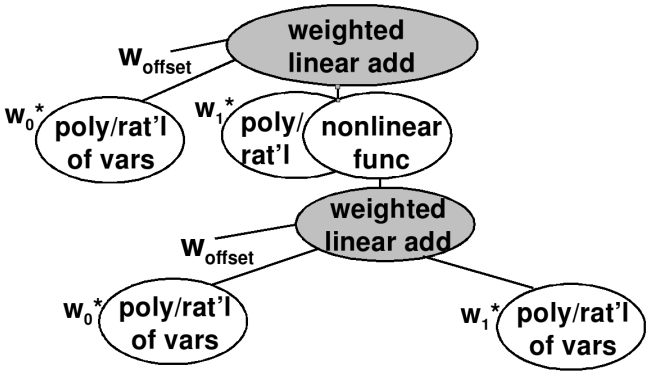


Figure 4.6: *Example of a function in text form, and its corresponding CAFFEINE tree form.*

Figure 4.7 gives an example which has unity functions for product terms. Specifically, note how there is *no* nonlinear function that gates one layer of linear adds to the next. In this fashion, CAFFEINE supports a product-of-sums formulation.

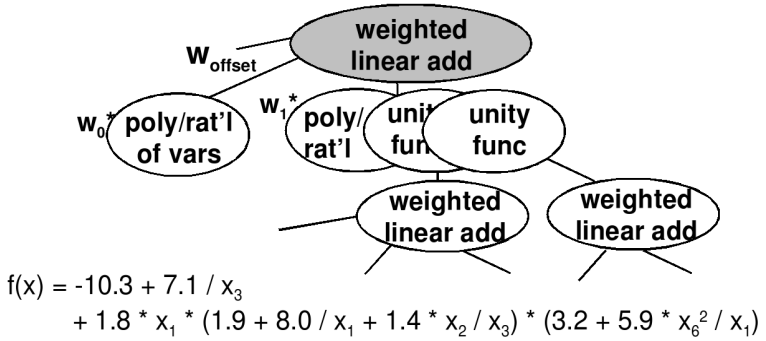


Figure 4.7: *Example where CAFFEINE product terms include unity functions.*

Typical usage of CAFFEINE would restrict the number of product term layers to just one or two, which is a more effective “maximum depth” constraint, therefore ensuring that there is not an excessive compounding of nonlinear components such as $log(sin(exp(x)))$. There is a limit on the number of basis functions. Due to the use of a canonical form, all evolved functions are immediately interpretable, with no symbolic manipulation needed.

Such constraints on functions directly resolve excessive complexity including GP bloat [Sou2002]. Furthermore, they can be used in a complementary fashion with other complexity-reducing tactics, e.g. having a second objective of complexity within multi-objective search.

4.4 CAFFEINE Search Algorithm

This section describes the search algorithm used on CAFFEINE functions. CAFFEINE search uses genetic programming (GP) as a starting point, but extends it in order to properly address template-free symbolic modeling. It attacks the issues of model complexity and interpretability in two main ways: a multi-objective approach that provides a tradeoff between error and complexity, and a specially designed grammar / operators to constrain the search to specific functional forms without restricting good solutions.

As described in the previous section, in CAFFEINE the overall expression is a linear function of basis functions $B_i; i = 1, 2, ..., N_B$:

$$y = F(\boldsymbol{x}; \psi) = a_0 + \sum_{i=1}^{N_B} a_i * B_i(\boldsymbol{x}) \tag{4.3}$$

A CAFFEINE individual ψ has one GP tree to define each basis function: $\psi = \{B_1, B_2, ..., B_{N_B}\}$. The linear coefficients a_i are determined on-the-fly using linear regression on the least-squares eror cost function.

4.4.1 Multi-Objective Approach

CAFFEINE uses a state-of-the-art *multi-objective* evolutionary algorithm, namely NSGA-II [Deb2002]. NSGA-II returns a set of individuals that, collectively, trade off model error and complexity. Error and complexity are objectives f_1 and f_2 in equation (4.1).

Error (expected loss $E_{x,y}L$) is approximated by "training error" ϵ_{tr}, which is is the normalized root mean squared error of individual ψ on training data:

$$\epsilon_{tr}(\psi) = \sqrt{\frac{1}{N_{tr}} * \sum_{i=1}^{N_{tr}} \left(\frac{\widehat{y_{tr,i}} - y_{tr,i}}{max(\boldsymbol{y}) - min(\boldsymbol{y})} \right)^2} \tag{4.4}$$

where N_{tr} is the number of training samples, $y_{tr,i}$ is sample i of training outputs $\boldsymbol{y_{tr}}$, $\widehat{y_{tr,i}} = F(\boldsymbol{x_{tr,i}}; \psi)$, and $\boldsymbol{x_{tr,i}}$ is sample i of training inputs $\boldsymbol{X_{tr}}$. Note that the y-values are scaled by $\boldsymbol{y}$, not $\boldsymbol{y_{tr}}$. ϵ_{test} has a similar formula, except the N_{tr} training points $\{\boldsymbol{y_{tr}}, \boldsymbol{X_{tr}}\}$ are replaced by the N_{test} testing points $\{\boldsymbol{y_{test}}, \boldsymbol{X_{test}}\}$.

Complexity is measured from the number of basis functions, the number of nodes in each tree, and the exponents of "variable combos" (VCs), according to:

$$complexity(\psi) = \sum_{j=1}^{N_B} (w_b + nnodes_j + \sum_{k=1}^{nvc(j)} vccost(vc_{k,j})) \tag{4.5}$$

where w_b is a constant to give a minimum cost to each basis function, $nnodes(j)$ is the number of tree nodes of basis function j, and $nvc(j)$ is the number of VCs of basis function j, with cost:

$$vccost(vc) = w_{vc} * \sum_{i=1}^{d} |vc(i)| \tag{4.6}$$

Using the terminology of [Wam1995], the approach accomplishes *simplification during generation*. It does so by maintaining evolutionary pressure towards lower complexity. The user avoids an *a priori* decision on error or complexity because the algorithm generates a set of models that provide tradeoffs of alternatives, rather than producing just one model.

Note that specific parameter settings are given in the experiments (section 4.5).

4.4.2 Grammar Implementation of Canonical Form Functions

In GP, a means of constraining search is via a grammar, as in [Whi1995]. Tree-based evolutionary operators such as crossover and mutation must respect the derivation rules of the grammar.

Even though grammars can usefully constrain search, none have yet been carefully designed for functional forms. In designing such a grammar, it is important to allow all functional combinations (even if just in one canonical form).

The CAFFEINE grammar, shown in Table 4.1 is explicitly designed to create separate layers of linear and nonlinear functions and to place coefficients and variables carefully; in adherence with Figure 4.5

Table 4.1: *CAFFEINE Grammar.*

```
REPVC ↦ VC | REPVC * REPOP | REPOP
REPOP ↦ REPOP * REPOP | OP_1ARG ( W + REPADD) |
        OP_2ARG ( 2ARGS ) | ...  3OP, 4OP, etc
2ARGS ↦ W + REPADD, MAYBEW | MAYBEW, W + REPADD
MAYBEW ↦ W | W + REPADD
REPADD ↦ W * REPVC | REPADD + REPADD
OP_2ARG ↦ DIVIDE | POW | MAX | etc
OP_1ARG ↦ INV | LOG10 | etc
```

First, we describe the notation of Table 4.1. The nonterminal symbols are in bold-case (terminal symbols are not). Each line (or two) shows the possible expressions that a nonterminal symbol on the left can map (↦) into. The possible expressions, i.e. "derivation rules" are separated by the OR operator '|'.

We now explain how the grammar implements canonical form functions. REP is short for "repeating", such as "repeating operators" REPOP and "repeating variable combo" REPVC, which are explained further. The start symbol is REPVC, which expands into one basis function (remember that an individual has several root-level basis functions). Note the strong distinction among operators. The root is a product of variables (REPVC) and / or nonlinear functions (REPOP). Within each nonlinear function is REPADD, the weighted sum of next-level basis functions.

A VC is a "variable combo", intended to maintain a compact representation of polynomials/rationals. Its expansion could have been implemented directly within the grammar; though in our baseline system we store a vector holding an integer value per design variable as the variable's exponent. An example vector is [1,0,-2,1], which means

$(x_1 * x_4)/(x_3)^2$, and according to equation (4.6) has cost $|1| + |0| + |-2| + |1| = 4$. This approach guarantees compactness and allows for special operators on the vector.

In determining coefficient values, we distinguish between linear and nonlinear coefficients. As described, a CAFFEINE individual is a set of basis functions which are linearly added. Each basis function is a tree of grammatical derivations. Linear coefficients are found by evaluating each tree across all input samples to get a matrix of basis function outputs, then to apply least-squares regression with that matrix and the target output vector to find the optimal linear weights.

With each nonlinear coefficient `W` in the tree (i.e. ones that are not found via linear regression), a real value will accompany it, taking a value in the range $[-2 * B, +2 * B]$. During interpretation of the tree the value is transformed into $[-1e+B, -1e-B] \cup [0.0] \cup [1e-B, 1e+B]$.

`POW(a,b)` is a^b. When the symbol `2ARGS` expands to include `MAYBEW`, either the base or the exponent (but not both) can be constants.

The designer can turn off any of the rules in the grammar of Table 4.1, if they are considered unwanted or unneeded. For example, he could easily restrict the search to polynomials or rationals, or remove potentially difficult-to-interpret functions such as *sin* and *cos*. He could also change or extend the operators or inputs, e.g. include W_i, L_i, and W_i/L_i.

Table 4.2: *Procedure ExtractSymbolicCaffeineModels()*

Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$
Outputs: M
1. $M = \emptyset$; $P = \emptyset$; $Q = \emptyset$
2. for $i = 1..N_{pop}$:
3. $\quad P_i \sim \Psi$
4. for $N_{gen} = 1..N_{gen,max}$:
5. $\quad \{P, Q\}$ = OneNsgaiiGeneration(P, Q)
6. $\quad M$ = nondominatedFilter($M \cup P \cup Q$)
7. return M

4.4.3 High-Level CAFFEINE Algorithms

This section describes the CAFFEINE model extraction algorithms in pseudocode. Table 4.2 shows the highest-level routine, *ExtractSymbolicCaffeineModels()*. It takes in the training inputs $\boldsymbol{X}$ and training outputs $\boldsymbol{y}$. It will output a Pareto-optimal set of models, M.

Line 1 of Table 4.2 initializes M, as well as the current set of parents P and current set of children Q, all to empty sets. Lines 2 loops across the population size N_{pop} to randomly draw each individual P_i from the space of possible canonical form functions Ψ.

Line 4 begins the EA's generational loop of lines 5 and 6. The loop stops when the target number of generations $N_{gen,max}$ is hit. Line 5 does the main EA work, which here is a single generation of the NSGA-II [Deb2002] multi-objective EA. Line 6 updates

the external archive of Pareto-optimal individuals, M, by nondominated-filtering on the existing M with the recently updated parents P and children Q.

Line 7 of Table 4.2 concludes the *ExtractSymbolicCaffeineModels()* routine, by returning the Pareto-optimal symbolic models, M.

Table 4.3 gives the pseudocode for one generation of the NSGA-II algorithm [Deb2002]. It inputs the parents P and children Q, and returns respective updated versions P' and Q'.

In line 1 of Table 4.3, the procedure merges the two inputs into one overall population of candidate parents, R. In line 2, it sorts R into *nondomination layers* F_i, $i = 1..N_{ND}$, where F_1 is the nondominated set, F_2 is what would be nondominated if F_1 was removed, F_3 is what would be nondominated if $F_1 \cup F_2$ was removed, etc. F contains all the candidates with no duplicates $F_1 \cup F_2 \cup \cdots F_{ND} = P_k \cup P_{k-1}$; $F_1 \cap F_2 \cap \cdots \cap F_{ND} = \emptyset$.

Table 4.3: *Procedure OneNsgaiiGeneration()*

Inputs: $\{P, Q\}$
Outputs: $\{P', Q'\}$
1. R = $P \cup Q$
2. F = fast-nondominated-sort(R)
3. P' = $\emptyset$; i = 1
4. until $|P'| + |F_i| \leq N_{pop}$:
5. crowding-distance-assignment(F_i)
6. P' = $P' \cup F_i$
7. i = i + 1
8. N_{fill} = N_{pop} - $|P'|$
9. F_i = sort F_i in descending order of crowding distance
10. P' = $P' \cup \{F_{i,1}, F_{i,2}, \cdots, F_{i,N_{fill}}\}$
11. Q' = ApplyOperators(P')
12. Q' = Evaluate(Q')
13. return $\{P', Q'\}$

The aim of lines 3-10 is to fill up the selected parents, P'.

It begins by initializing P' to an empty set in line 3. Then lines 4-7 iterate by filling up each nondomination layer, but *only* if the whole nondomination layer F_i fits. Specifically, it first adds all individuals from F_1, if they all fit, i.e. if if $|P_{sel}| + |F_1| \leq N_L$. If there is space left, it then adds all individuals from F_2 if they all fit. If there is space left, then adds all individuals from F_2 if they all fit. And so on.

Lines 8-10 of Table 4.3 cover the case when P' may not be full yet. This occurs when the last nondomination layer F_i did not fit perfectly into P''s remaining space of N_{fill} individuals. In that case, line 9 sorts the nondomination layer F_i according to "crowding distance". Crowding distance of an individual is the maximum Euclidian distance (in performance space) between that individual and its closest neighbor. Line 10 takes the N_{fill} individuals in F_i that have the greatest spacing, and adds them to P'.

Now that the parents have been selected, children can be created. Line 11 of Table 4.3 applies evolutionary search operators to the parents P' to create children Q'. These

operators are grammar / CAFFEINE-specific, so the operator details are given in section 4.4.4.

Line 12 evaluates the children Q'. In this case, evaluation is of the two CAFFEINE objectives, model training error ϵ_{tr} in equation (4.4), and model complexity in equation (4.5). To measures training error, recall that least-squares regression must first be done to determine the linear coefficients a_i of each of the basis functions B_i.

Line 13 returns the updated parents P' and children Q', and the routine concludes.

4.4.4 Evolutionary Search Operators

We now describe how trees are randomly generated, and explain the search operators on the trees. The search operators are grouped by the aspect of search representation that they concern: grammar, real-valued coefficient, VCs, and basis functions.

Random generation of trees and subtrees from a given symbol involves merely randomly picking one of the derivations of one of the symbols, and recursing the (sub) tree until terminal symbols are encountered (subject to tree depth limits).

Grammatical restrictions on the trees lead to a natural grammar-obeying crossover operator and mutation operator, as described by Whigham [Whi1995]. Whigham-style crossover works as follows: it randomly picks a node on the first parent, then randomly picks a node on the second parent with the constraint that it must be the same grammatical symbol (e.g. REPOP) as the first node, and finally swaps the subtrees corresponding to each node. Whigham-style mutation involves randomly picking a node, then replacing its subtree with a randomly-generated subtree (as in the generation of initial trees).

Real-valued coefficients are mutated according to a Cauchy distribution [Yao1999], which cleanly combines aggressive local tuning with the occasional large change.

The specialized structure of VCs get appropriate operators, which include: one point crossover, and randomly adding or subtracting to an exponent value.

Since each individual has a list of basis functions, this leads to special operators: creating a new individual by randomly choosing > 0 basis function from each of 2 parents; deleting a random basis function; adding a randomly generated tree as a basis function; copying a subtree from one individual to make a new basis function for another.

4.5 CAFFEINE Results

This section describes the application of CAFFEINE to building symbolic models for analog circuits that map design variables to performances, for problems with 13 input variables. It shows the actual symbolic models generated, the measured error vs. complexity tradeoffs, how prediction error and complexity compare to posynomials, and how prediction error compares to other state-of-the-art (blackbox) regression approaches. The extension of CAFFEINE to larger problems is described in section 4.6.

4.5.1 Experimental Setup

Unary operators allowed are: $\sqrt{(}x)$, $log_{10}(x)$, $1/x$, x^2, $sin(x)$, $cos(x)$, $tan(x)$, $max(0, x)$, $min(0, x)$, 2^x, and 10^x, where x is an expression. Binary operators allowed are $x_1 + x_2$, $x_1 * x_2$, $max(x_1, x_2)$, $min(x_1, x_2)$, $power(x_1, x_2)$, and x_1/x_2. Conditional operators included $\leq (testExpr, condExpr, exprIfLessThanCond, elseExpr)$ and $\leq (testExpr, 0, exprIfLessThanCond, elseExpr)$. Any input variable could have an exponent in the range {..., -1, 1, 2, ...}. While real-valued exponents could have been used, that would have harmed interpretability.

The circuit being modeled in this example is a high-speed CMOS OTA as shown in Figure 4.8. The goal is to discover expressions for the low-frequency gain (A_{LF}), unity-gain frequency (FU), phase margin (PM), input-referred offset voltage ($VOFF$), and the positive and negative slew rate (SR_p, SR_n), To allow a direct comparison to the posynomial approach [Dae2002], an almost-identical problem setup was used, as well as identical simulation data. The only difference is that, because scaling makes the model less interpretable, neither the inputs nor the outputs were scaled. The one exception is that FU is log-scaled so that the mean-squared error calculations and linear learning are not wrongly biased towards high-magnitude samples of FU. The technology is 0.7μm CMOS. The supply voltage is 5V. $V_{th,nom}$ is 0.76V and -0.75V for the NMOS and PMOS devices, respectively. The load capacitance is 10 pF.

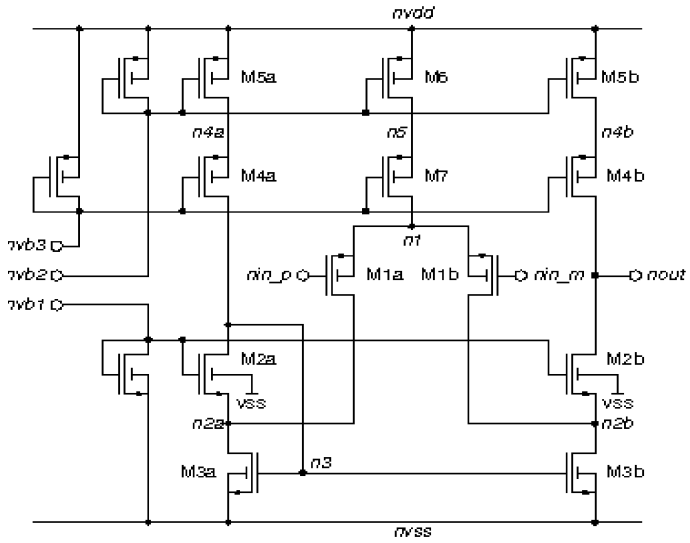


Figure 4.8: *CMOS high-speed OTA.*

Good training data is essential to the methodology. The choice of design variables and sampling methodology determines the extent to which the designer can make inferences about the physical basis, and what regions of the design space the model is valid in. We used an operating-point driven formulation [Leyn1998] , where currents and transistor gate drive voltages comprise the design variables (13 variables in our case). Device sizings could have been used as design variables instead; it depends on designer preference and other reasons (see section 6.4).

Full orthogonal-hypercube Design-Of-Experiments (DOE) [Mon2004] sampling of design points was used, with scaled dx=0.1 (the simpler problem of dx=0.01 from [Dae2002] is ignored in this chapter) to have 243 samples. The simulation time for one sample was about 1 s, or 4 min for all samples; this is fully dependent on the circuit, analyses, and experimental design method being used. These samples, otherwise unfiltered, were used as training data inputs. Testing data inputs were also sampled with full orthogonal-hypercube DOE and 243 samples, but with dx=0.03. Thus, in this experiment we are creating a somewhat localized model; one could just as readily model a broader design space, but the above choice allows us to compare the results to [Dae2002].

The run settings were: N_B = maximum number of basis functions = 15 (any larger is definitely non-interpretable), N_{pop} = population size = 200 (like NSGA-II's default), $N_{gen,max}$ = 5000 generations (more than enough time to converge), maximum tree depth = 8 (so that each basis function has exactly one layer of nonlinear operators), and "W" coefficients range $[-1e+10, -1e-10] \cup [0.0] \cup [1e-10, 1e+10]$ (so coefficients can cover 20 orders of magnitude, both positive and negative).

All operators had equal probability (a reliable setting), except parameter mutation was 5x more likely (to encourage tuning of a compact function). Complexity measure settings were w_b = 10, w_{vc} = 0.25. That is, the cost of adding a basis function is relatively high compared to the cost of adding another variable combo.

One run was done for each performance goal, for 6 runs total. Each run took about 12 hours on a 3 GHz Pentium IV Linux workstation. (Note that this was on a slow Matlab-based system with extensive pass-by-value functions. The implementation of section 4.6 is significantly faster because it has pass-by-reference functions and more improvements).

We calculate normalized mean-squared error on the training data and on the separate testing data: ϵ_{tr} and ϵ_{test} as described in equation (4.4). These are standard measurements of model quality in regression literature. The testing error ϵ_{test} is ultimately the more important measure, because it measures the model's ability to generalize to unseen data. These measures are identical to two of the three posynomial "quality of fit" measures in [Dae2002]: its measure "worst-case quality" q_{wc} *is* the training error ϵ_{tr}, and its measure "typical case quality" q_{tc} *is* the testing error ϵ_{test} (as long as long as the constant 'c' in the denominator is set to zero, which [Dae2002] did.)

4.5.2 Results: Whitebox Models and Tradeoffs

Let us first examine some symbolic models generated by CAFFEINE. We ask: "what are the symbolic models having less than 10% training and testing error, with the lowest complexity?"

Table 4.4 shows those functions. (Note that FU has been converted to its true form by putting the generated function to the power of 10). We see that each form has up to four basis functions, not including the constant. For $VOFF$, a constant was sufficient to keep the error within 10%. We see that a rational functional form was favored heavily; at these target errors only one nonlinear function, ln( ), appears (for A_{LF}). That expression effectively says that the *order of magnitude* of some input variables is useful because it deals in logarithmic scales.

Table 4.4: *CAFFEINE-generated symbolic circuit models with < 10% training error and <10% testing error.*

Perf. Char.	**Expression**
ALF	$-10.3 + 7.08e\text{-}5/i_{d1} + 1.87 * ln(-1.95e\text{+}9 + 1.00e\text{+}10/(v_{sg1} * v_{sg3}) + 1.42e\text{+}9 * (v_{ds2} * v_{ds5})/(v_{sg1} * v_{gs2} * v_{gs5} * i_{d2}))$
FU	$10^{(5.68 - 0.03 * v_{gs1}/v_{ds2} - 55.43 * i_{d1} + 5.63e-6/i_{d1})}$
PM	$90.5 + 190.6 * i_{d1}/v_{gs1} + 22.2 * i_{d2}/v_{ds2}$
$VOFF$	$-2.0e\text{-}3$
SR_p	$2.36e\text{+}7 + 1.95e\text{+}4 * i_{d2}/i_{d1} - 104.7/i_{d2} + 2.15e\text{+}9 * i_{d2} + 4.63e\text{+}8 * i_{d1}$
SR_n	$-5.72e\text{+}7 - 2.50e\text{+}11 * (i_{d1} * i_{d2})/v_{gs2} + 5.53e\text{+}6 * v_{ds2}/v_{gs2} + 109.7/i_{d1}$

One can examine the equations in more detail to gain an understanding of how design variables in the topology affect performance. For example, A_{LF} is inversely proportional to i_{dl}, the current at the OTA's differential pair. Or, SR_p is solely dependent on i_{d1} and i_{d2} and the ratio i_{d1}/i_{d2}. Or, within the design region sampled, the nonlinear coupling among the design variables is quite weak, typically only as ratios for variables of the same transistor. Or, that each expression only contains a (sometimes small) subset of design variables. Or, that transistor pairs $M1$ and $M2$ are the only devices affecting five of the six performances (within 10%).

We now examine the CAFFEINE-generated tradeoffs between training error ϵ_{tr} (q_{wc}) and complexity. Figure 4.9 illustrates. All models in the tradeoff of training error vs. complexity are shown: as complexity increases, the training error decreases. In each performance instance, CAFFEINE generates a tradeoff of about 50 different models. As expected, a zero-complexity model (i.e. a constant) has the highest training error of 10-25%. The highest-complexity models have the lowest training error, of 1-3%.

We can also examine the curves relating complexity to the number of basis functions. Recall that complexity is a function of both number of basis functions, and the complexity of each tree within each basis function. In the curves, we see that the number of basis functions usually increases with the complexity. However, sometimes complexity increases by having larger trees within existing basis functions, rather than adding more basis functions. This can be seen in the curves: as complexity increases, the number of bases temporarily levels off, or even decreases.

The testing error ϵ_{test} (q_{tc}) is also shown in Figure 4.9. We see that unlike the training error, it is not monotonically decreasing as complexity rises. This means that some less complex models are more predictive than more complex ones. However, we can prune the models down to the ones that give a tradeoff between testing error and complexity, as shown in Figure 4.10. These are the most interesting and useful.

It is notable that the testing error is lower than the training error in almost all cases. This sounds promising, but such behavior is rare in the regression literature, and made us question what was happening. It turns out that there is a valid reason: recall that the training data is from extreme points of the sampling hypercube (scaled dx=0.10), and the

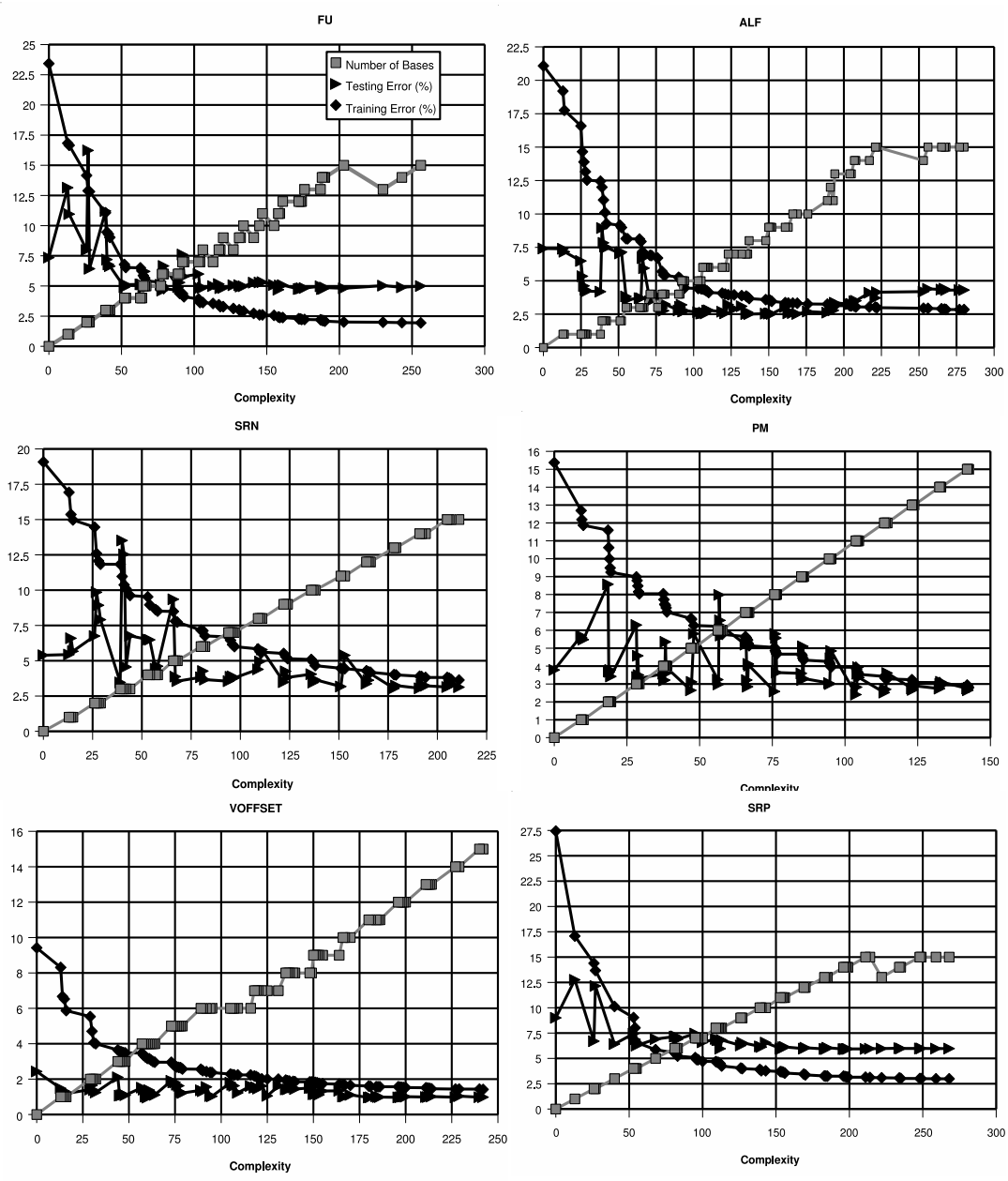


Figure 4.9: *Plots of models' training error, testing error, and number of bases vs. the complexity for each performance goal for the opamp of Figure 4.8. Every* $(diamond, triangle, square)$ *triplet corresponds to a symbolic model at a given complexity.*

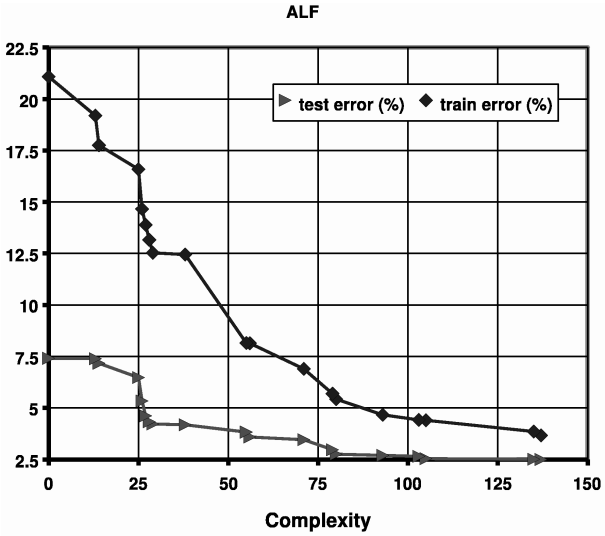


Figure 4.10: *Every* $(diamond, triangle)$ *is a symbolic model for* A_{LF} *like Figure 4.9, except filtered to only keep models on the tradeoff of* testing *error vs. complexity.*

testing data is internal to the hypercube (dx=0.03). This testing data tests the *interpolation* ability. Thus, models that really *are* predictive should be able to interpolate well, even at the cost of a perfect fit to the extreme points. In any case, validly having the testing error lower than the training error demonstrates the strength of the CAFFEINE approach.

By only putting the relevant variables into a model, the approach demonstrates the potential to provide expressions for circuits with significantly more variables (see next section).

One may improve their understanding of the basic dependencies in a circuit in another fashion: by examining expressions of varying complexity for a single performance characteristic. Low-complexity models will show the macro-effects; alterations to get improved error show how the model is refined to handle second-order effects. Table 4.5 shows models generated for the phase margin (PM) for decreasing training and testing error. A constant of 90.2, while giving 15 % training error, had only 4% test error. For better prediction, CAFFEINE injected two more basis functions; one basis being the current into the differential pair i_{d1}, the other basis, i_{d2}/v_{ds2}, being the ratio of the current to the drain-source voltage of $M2$; i.e. $M2$'s small-signal output conductance ($1/r_{out2}$). The next model turns the input current term into a ratio i_{d1}/v_{gs1}; i.e. M1's transconductance, inverted ($1/g_{m1}$). Interestingly, and reassuringly, almost all ratios use the same transistor in the numerator and denominator.

Such analyses achieve one of the aims of the CAFFEINE symbolic modeling tool: demonstrating how to gain insight into the topology.

Table 4.5: *CAFFEINE-generated models of PM, in order of decreasing error and increasing complexity.*

Test error (%)	**Train error** (%)	PM **Expression**
3.98	15.4	90.2
3.71	10.6	$90.5 + 186.6 * i_{d1} + 22.1 * i_{d2}/v_{ds2}$
3.68	10.0	$90.5 + 190.6 * i_{d1}/v_{gs1} + 22.2 * i_{d2}/v_{ds2}$
3.39	8.8	$90.1 + 156.85 * i_{d1}/v_{gs1} - 2.06e\text{-}3 * i_{d2}/i_{d1} + 0.04 * vgs2/v_{ds2}$
3.31	8.0	$91.1 - 2.05e\text{-}3 * i_{d2}/i_{d1} + 145.8 * i_{d1} + 0.04 * v_{gs2}/v_{ds2} - 1.14/v_{gs1}$
3.20	7.7	$90.7 - 2.13e\text{-}3 * i_{d2}/i_{d1} + 144.2 * i_{d1} + 0.04 * v_{gs2}/v_{ds2}$ $-1.00/(v_{gs1} * v_{gs3})$
2.65	6.7	$90.8 - 2.08e\text{-}3 * i_{d2}/i_{d1} + 136.2 * i_{d1} + 0.04 * v_{gs2}/v_{ds2}$ $-1.14/v_{gs1} + 0.04 * v_{gs3}/v_{ds5}$
2.41	3.9	$91.1 - 5.91e\text{-}4 * (v_{gs1} * i_{d2})/i_{d1} + 119.79 * i_{d1} + 0.03 * v_{gs2}/v_{ds2}$ $-0.78/v_{gs1} + 0.03 * v_{gs1}/v_{ds5} - 2.72e\text{-}7/(v_{ds2} * v_{ds5} * i_{d1})$ $+7.11 * (v_{gs2} * v_{gs4} * i_{d2}) - 0.37/vsg5 - 0.58/v_{gs3} - 3.75e\text{-}6/i_{d2} - 5.52e\text{-}6/i_{d1}$

4.5.3 Results: Comparison to Posynomial-Based Symbolic Modeling

We also compared CAFFEINE to the posynomial approach using the posynomial results in [Dae2002]. We first compare model complexity. To pick the models to compare, we first choose the CAFFEINE model which meets the reported posynomial training and test error of [Dae2002], then we compare the number of posynomial coefficients to the number of coefficients appearing in the CAFFEINE expressions (this is reasonable when the CAFFEINE expressions are largely rationals; more complex symbolic models would be less appropriate). As Figure 4.11 shows, the CAFFEINE models are 1.3 to 6.4 times more compact than the posynomial models. And, in $VOFF$, the only performance that the posynomials had slightly better prediction error than CAFFEINE (see Figure 4.12), the CAFFEINE model is 6.2x more compact.

We can also compare the prediction abilities of CAFFEINE to posynomials. To pick a model from a CAFFEINE-generated tradeoff for comparison, we fixed the training error to what the posynomial achieved, then compared the testing errors. The results are in Figure 4.12. In one case, $VOFF$, CAFFEINE did not meet the posynomial training error (0.4%), although it probably could have with more basis functions; we instead picked an expression which very nearly matched the posynomial approach's testing error of 0.8%. What we saw in the previous data, and what we see again here, is that CAFFEINE has a lower testing error than training error, which provides great confidence to the models. In contrast, in all cases but $VOFF$, the posynomials had a higher testing error than training error, even on this interpolative data set. The CAFFEINE models' testing errors were *2x to 5x* lower than ones from the posynomial models. The exception is $VOFF$, where

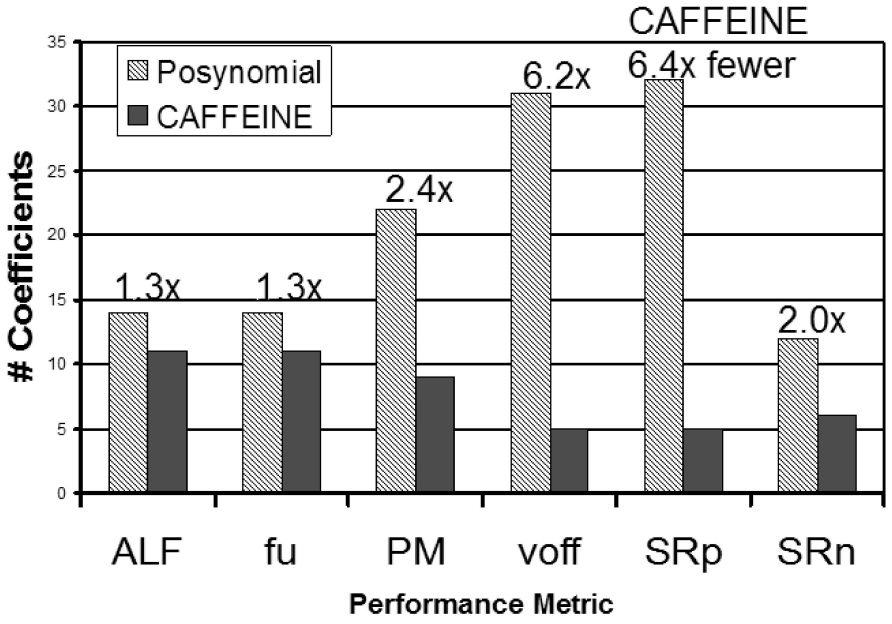


Figure 4.11: *Comparison of the complexity of CAFFEINE models to posynomial models [Dae2002]. Method: (1) choose CAFFEINE model that meets posynomial training and test error, then (2) compare number of coefficients.*

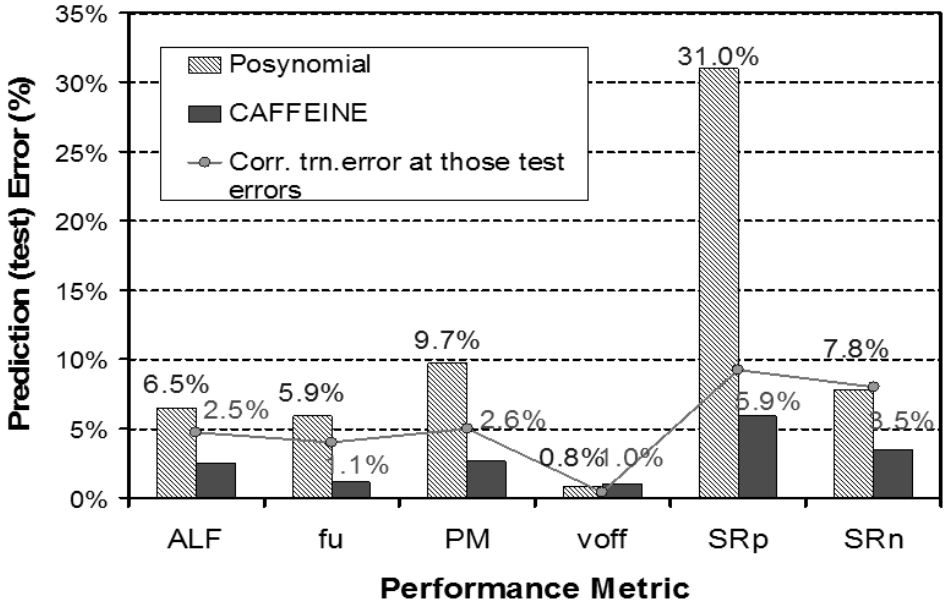


Figure 4.12: *Comparison of CAFFEINE testing error to posynomial testing error; and to the training error.*

the posynomial achieves 0.8% testing error compared to 0.95% for CAFFEINE. In short, posynomials have poor prediction ability even in interpolation. CAFFEINE models predict far better, and with more compact models. Given this, one can reasonably question the trustworthiness of constraining analog circuit performance models to posynomials.

4.5.4 Results: Comparison to State-of-the-Art Blackbox Regression Approaches

While other modeling techniques may produce models that are opaque (and therefore not interpretable), it is still instructive to see how well CAFFEINE compares to them in terms of prediction ability.

So, on the 6 problems already described in section 4.5.1, we tested the following regression techniques: a constant, linear models with least-squares fit, full quadratic models with least-squares fit, projection-based quadratic (PROBE) [Li2006], posynomial [Dae2002], state-of-the-art feedforward neural networks (FFNN) [Amp2002], boosting [Sch2002] the FFNNs, multivariate adaptive regression splines (MARS) (i.e. piecewise polynomial with stepwise construction) [Fri1991], least-squares support vector machines (LS-SVM) [Suy2002], and kriging [Jon1998].

Model builders were coded and configured as follows.

- The code to build constant, linear, and full quadratic models was about 25 lines of Matlab. The model building time was a few seconds, at most.
- The code to build PROBE was about 100 lines of python, using Numeric / LAPACK for least-squares regression and maximum rank of 2. The model building time was a few seconds, at most.
- The posynomial results were taken directly from [Dae2002]; it reports that the model building time was 1-4 minutes (on a slower machine).
- The FFNN is trained via an adaptive Levenberg-Marquardt optimization scheme (OLMAM); we used the Matlab code referenced in [Amp2002]. Settings were *NumRestarts* = 10, *MaxEpochs* = 5000. The time to build a single network was about 10 s. A suitable error was typically found in the first or second restart of about 3 hidden neurons. Therefore the total model building time was about (10 s) * (10 restarts) * (first 2 neurons) + (10 s) * (2 restarts) * (1 final neuron) = 10*10*2 + 10*2 = 220 s = 3.7 min.
- The boosted FFNN was Matlab code wrapping the OLMAM code. Settings were *NumModels* = 20. The model building time was about (220 s to discover *NumHid*) + (10 s)*(20 models) = 220 s + 200 s = 420 s = 7.0 min. A 10x speedup via a C implementation would make this 42 s.
- The MARS model builder was about 500 lines of Matlab code; model building time was about 5 minutes.
- The SVM is trained using the least-squares strategy (LS-SVM); we used the Matlab code from [Suy2002], with all settings at “fully automatic”; the model building time was about 5 minutes.

- The kriging model builder was about 200 lines of Matlab code, with Θ_{min} = 0.0, Θ_{max} = 10.0, p_{min} = 0.0, p_{max} = 1.99. The model building time was about 5 minutes.

Figure 4.13 shows the resulting test errors for the 6 performances (adapted from [Mcc2005c]).

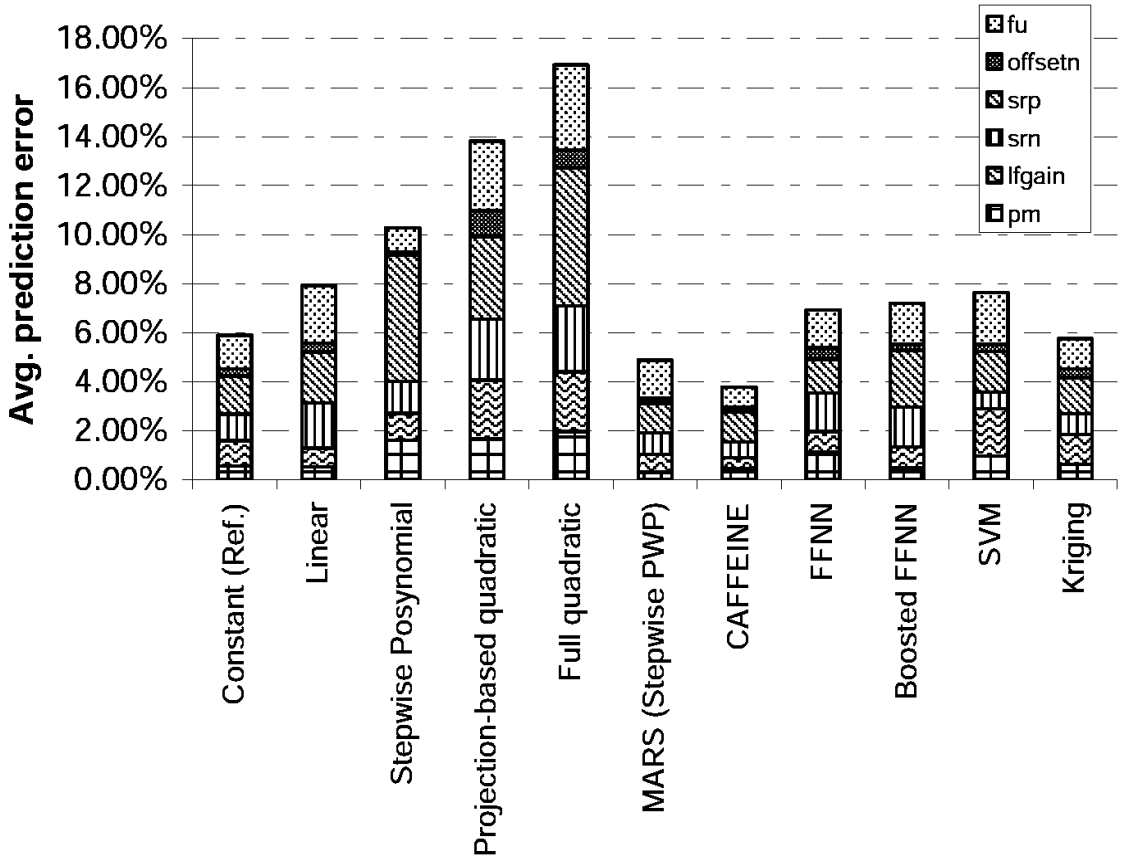


Figure 4.13: *Comparison of prediction ability of CAFFEINE to state-of-the-art modeling techniques.*

On this dataset, CAFFEINE does the best. MARS comes in very close. Kriging is the next-best. The FFNN, boosted FFNN, and SVM are all very close, and perform about the same as the linear model. The quadratic and posynomial approaches and posynomial approaches perform the worst.

The results on different regressors inform us about the nature of the data. Progressing across the spectrum of polynomial complexity – from the simplest linear models to posynomials to projection-based quadratic to full quadratic – the prediction error continually worsens. It turns out that the polynomials even capture the *training* error poorly; for example the projection-based quadratic had a training error of about 10% for each performance. Since the prediction error became lower the more constrained the polynomial model was, this indicates that where the models do attempt to use the added flexibility to predict better, it backfires. In general, this is indicative that a polynomial functional template is not appropriate for circuit performance mappings, even for this relatively simple OTA circuit.

CAFFEINE only selects input variables that really matter. It is biased towards the axes of the input variables rather than being affine-invariant. That is, CAFFEINE expressions and search operators work on one or a few input variables at a time, as opposed to using all variables in a weighted sum. MARS did similarly, because its stepwise-forward nature makes it also biased towards the axes and is selective of input variables. While CAFFEINE had the best or near-best prediction error on 5 of the 6 performance goals, MARS had the best or near-best on 3. As we shall see, the other approaches lose prediction performance because they have different biases.

Kriging performed fairly admirably in this setting. This is not surprising because it tends to perform well when the input samples have relatively uniform spacing, as they do here with the DOE sampling. Kriging, FFNNs, and boosted FFNNs did worse than CAFFEINE and MARS, most likely because they did not have the helpful (for this application) bias towards the input axes. The boosted FFNN did not have noticeably superior performance to the FFNN, which means that overfitting was likely not an issue with the FFNN. The SVM's performance was poor, probably because it treated the variables it selected too uniformly. Also, the support vector at the center of the sampling hypercube has to reconcile all the other samples, which it does not really have enough parameters to do properly. Because kriging did substantially better than SVMs, the choice of kernel distance function was likely not an issue. Interestingly, only three approaches, namely CAFFEINE, MARS, and kriging, did better at prediction than a constant. This is not because constants are good predictors *per se*, but because other predictors failed for the various reasons described. Put in another way, the other predictors' attempts to predict outputs from unseen (testing) inputs did poorly because the models generalized in poor directions that caused more extreme error values, whereas the constant never had extreme error values.

In summary, CAFFEINE demonstrated that it could out-predict all the state-of-the-art approaches tested (on the given circuit-test suite), in addition to being the *only* approach that outputs *template-free*, *symbolic* models. CAFFEINE's construction time is longer than the other methods, but is still fast enough for the *insight* applications that it was designed for.

4.6 Scaling Up CAFFEINE: Algorithm

We ran the algorithm described in section 4.4 on larger circuits – problems with more than 100 input variables. The results were disappointing: despite good performance on smaller problems, CAFFEINE was too slow to return interesting results on these larger problems in reasonable time. That experience motivates this section. The aim is to alter the search algorithm so that it can scale to problems of 100 variables. The specific aims are to (a) run in a reasonable time – hours or minutes, (b) have predictive models, and (c) have interpretable models.

The improved CAFFEINE leverages four complementary techniques:

- Subtree caching

- Gradient-directed regularization to simultaneously prune basis functions and find coefficients for the remaining basis functions
- Filter single-variable expressions in a pre-evolution step
- Always consider all linear basis functions

We now describe each technique in detail.

4.6.1 Subtree Caching

In the original implementation of CAFFEINE, every time a tree was changed, it would have to be *fully* re-evaluated. The technique of sub-tree caching [Kei2004] sidesteps evaluations in some nodes of the tree. Given that the training dataset does not change, when a new tree is created from parent tree(s) via the search operators, only *part* of the new tree is different. Therefore, we evaluate just the nodes of the tree that have changed, and their parent nodes, and *cache* the results. The "evaluation" for other nodes merely uses the evaluated results that have been cached previously. Note that to do this cleanly, CAFFEINE was re-implemented in Python, whereas the previous implementation was in Matlab. This improved runtime further because Python passes function values by reference, whereas Matlab passes by value.

4.6.2 On-the-fly Pruning with Gradient-Directed Regularization

In previous subsections, the linear coefficients **a** of equation 4.3 were learned by minimizing the least-squares (LS) loss function on the training data. But for larger problems having potentially more basis functions, the LS predictions can be unstable because there is higher variance in the range of possible parameters. Furthermore, to keep the complexity down, it is desirable to have a more aggressive way to prune the basis functions. Regularization is promising because it explicitly accounts for parameter variance and can implicitly prune basis functions on-the-fly. Historically, the main regularization choices have been ridge regression [Hor1970] and the lasso [Tib1997]. Unfortunately, ridge regression does little pruning, and the lasso prunes *too* aggressively. Fortunately, a new technique, gradient-directed regularization (GDR) [Fri2004], strikes a compromise. GDR does gradient-based optimization on the loss function f_2 in equation (4.1) according to the coefficient update rule:

$$\hat{\boldsymbol{a}}(\nu + \Delta\nu) = \hat{\boldsymbol{a}} + \Delta\nu * \boldsymbol{hr}(\nu) \tag{4.7}$$

where $\Delta\nu$ is small ("infinitesimal") value and $\boldsymbol{hr}$ is the direction of the next step. The starting value of $\boldsymbol{a}$ is [0, 0, ..., 0]. The gradient to the loss function is:

$$\boldsymbol{gr}(\nu) = -\frac{\boldsymbol{d}}{\boldsymbol{d}\hat{\boldsymbol{a}}} \frac{1}{N_B} \sum_{i=1}^{N_B} L(y_i, F(x_i; \hat{\boldsymbol{a}})) \tag{4.8}$$

where L is given in equation (4.2).

One could directly optimize using $\boldsymbol{gr}$ instead of $\boldsymbol{hr}$ in (4.7), but little pruning would happen, and collinear or near-collinear bases get similar values (like ridge regression).

Instead, GDR encourages diversity by selectively updating coefficients. Specifically, it changes a_i at a given step only if $|gr_i|$ is sufficiently large:

$$\boldsymbol{hr}(\nu) = \{hr_i(\nu)\}\forall i = \{\gamma_i(\nu) * gr_i(\nu)\}\forall i; i = 1, 2, \ldots, N_B \tag{4.9}$$

$$\gamma_i(\nu) = I(|gr_i(\nu)|) \geq \tau * \max_{0 \leq k \leq N_B} |gr_k(\nu)|) \tag{4.10}$$

where γ_i is an indicator function that returns 0 or 1, and hr_i either outputs 0 or gr_i as it combines the indicator function and the gradient. τ is a parameter which controls the degree of pruning: 0.0 is like ridge regression, 1.0 is like lasso, and values in between strike a compromise.

We employ GDR here (with settings given in section 4.7.1). The result is that we can have CAFFEINE individuals with a large number of basis functions, and in a single pass GDR will drive many linear coefficients to zero (i.e. prune the basis functions), and set robust values for the remaining linear coefficients. GDR is fast too: our 300-line python implementation of GDR has about the same runtime as the highly-optimized LAPACK linear LS solver.

4.6.3 Pre-Evolution Filtering of Single-Variable Expressions

The third scalability-improving technique focuses the search towards the most promising single-variable nonlinear expressions. It determines those expressions with the routine *ExtractUsefulExpressions()* shown in Table 4.6, prior to the evolutionary run (i.e. right before line 2 in the procedure of Table 4.2). *ExtractUsefulExpressions()* considers a large set of possible *single-variable* expressions at once, and extracts the most promising ones.

We now describe *ExtractUsefulExpressions()* of Table 4.6 in detail. It takes as inputs the target training inputs $\boldsymbol{X}$ and corresponding outputs $\boldsymbol{y}$. It also takes in ι_{thr}, which governs the final number of expressions returned. It will return a set of chosen expressions, $\boldsymbol{B_{useful}}$.

In lines 1-6, *ExtractUsefulExpressions()* constructs the candidate expressions $\boldsymbol{B}$, by enumerating through all combinations of input variables (line 2), operators (line 3), and exponents (line 4).

Line 7 simulates each candidate expression on each of the training input vectors in $\boldsymbol{X}$. Each row of the resulting matrix $\boldsymbol{X_B}$ has the values of each training input vector as input to a given expression B_i.

Line 8 identifies the influence of each B_i, i.e. each row in $\boldsymbol{X_B}$, by conducting linear learning on the mapping from $\boldsymbol{X_B}$ to $\boldsymbol{y}$. Since it is possible (and likely) that the number of expressions exceeds the number of training samples, GDR is used because it can handle underdetermined linear systems. From GDR, each expression B_i will get a linear coefficient a_i.

Then, line 9 computes the *influence*, ι_i, of an expression B_i according to:

$$\iota_i = |a_i| * (\max_{1 \leq j \leq N}(B_i(\boldsymbol{x}_j)) - \min_{1 \leq j \leq N}(B_i(\boldsymbol{x}_j))) \tag{4.11}$$

Table 4.6: *Procedure ExtractUsefulExpressions()*

Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$, ι_{thr}
Outputs: $\boldsymbol{B_{useful}}$
1. $\boldsymbol{B}$ = {}; i = 1
2. for each input variable v = {x_1, x_2, ...}
3. for each operator op = {$unity()$, log_{10}, ...}
4. for each exponent exp = {-2, -1.5, ...}
5. define B_i as $op(v)^{exp}$
6. $\boldsymbol{B} = \boldsymbol{B} \cup B_i$; $i = i + 1$
7. $\boldsymbol{X_B}$ = simulate $\boldsymbol{X}$ on each B_i
8. $\boldsymbol{a}$ = GDR linear learning on $\boldsymbol{X_B} \mapsto \boldsymbol{y}$.
9. ι_i = compute influence of B_i according to (4.11); for each B_i
10. $\boldsymbol{B}$ = sort $\boldsymbol{B}$ in descending order of ι_i
11. $\boldsymbol{B_{useful}} = \emptyset$; ι_{tot} = 0; i = 1
12. while $\iota_{tot} < \iota_{thr}$:
13. $\boldsymbol{B_{useful}} = \boldsymbol{B_{useful}} \cup B_i$
14. $\iota_{tot} = \iota_{tot} + \iota_i$
15. $i = i + 1$
16. return $\boldsymbol{B_{useful}}$

where $\boldsymbol{x_j}$ is the j^{th} training sample. $\underset{1 \le j \le N}{max}(B_i(\boldsymbol{x}_j))$ is the largest value that B_i computes to across the training data, and $\underset{1 \le j \le N}{min}(B_i(\boldsymbol{x}_j))$ is the smallest value. Influence ι_i is essentially an absolute and normalized version of linear coefficient a_i.

Lines 10-16 use the ι_i information to do final selection of basis functions. First, line 10 sorts all the basis functions such that B_1 has highest influence, B_2 has second-highest influence, and so on. Line 11 initializes the loop that follows. Line 12 loops around until the total influence quota is hit, $iota_{thr}$. For example, $iota_{thr}$ = 0.95 means that the routine will keep the highest-influence expressions having 95% of total influence. To implement this aim, line 13 adds the next-most influencing expression, and lines 14-15 do bookkeeping.

Line 16 returns the final chosen expressions, $\boldsymbol{B_{useful}}$.

These $\boldsymbol{B_{useful}}$ get stored for use during the evolutionary run. During the run, whenever a sum of products expression is about to be randomly generated (as a basis function, or at a lower level in the CAFFEINE expression tree), then κ% of the time, only the useful expressions are considered. There has to be enough opportunity to try other expressions to avoid over-constraining the search, but the majority of search effort can be focused on known-promising expressions. We set κ = 80%.

Note that variable interactions can easily be generated via crossover and mutation operations on single-variable expressions. This strategy is reminiscent of MARS [Fri1991], which builds up complex multi-variable expressions from a foundation of single-variable expressions.

4.6.4 Always Include All Linear Basis Functions

The focus of the final scale-up technique is to enhance prediction ability. It was based on the following observations:

- Circuit problems with a larger number of input variables tend to have at least partially linear responses to some variables.
- GDR was very effective at pruning bases.
- A recent paper showed enhanced prediction ability by combining linear basis functions with a (non-CAFFEINE) nonlinear model [Fri2005].

So, we altered the search to always consider all linear basis functions (but not to evolve them). To be precise, when evaluating an individual, there is a step which does linear learning to find the best coefficients for the tree-based basis functions (and the offset). We altered that step to include more basis functions – one linear basis function for each input variable. This greatly increases the number of basis functions for linear learning, but not for the evolutionary search itself which only sees the number of GP trees.

This measure ensures that linear responses to variables are always considered. This biases the search towards more stable, understandable models, without having to ask the evolutionary algorithm to manage the extra bases.

This completes the description of the four enhancements to CAFFEINE which were designed to allow it to scale up to larger problems. We will now present some experimental results.

4.7 Scaling Up CAFFEINE: Results

4.7.1 Experimental Setup

In this section, the aim is to determine how well the scalability goals have been achieved with the improved CAFFEINE.

The tests are on three progressively larger circuits – the amplifiers shown in Figures 3.7, 3.16, and 3.21. The circuit regression problems have been set up with the parameters of Table 4.7. Four output performances are modeled for each circuit, with the intent to represent a cross-section of analyses and measures: A_V (gain), THD (total harmonic distortion), SR (slew rate), and OS (overshoot). The technology is 0.13 μm CMOS. The design variables are widths W, lengths L, multipliers M, capacitances C, and resistances R. The samples were taken using Latin hypercube sampling [Mck1979, Mck2000] on a uniform distribution in the hypercube having its center at a "good" design, and variable ranges $\pm 10\%$. The training and test data were split apart by sorting the samples according to the output value, allocating every 4^{th} sample to the test data, and the rest to training (i.e. 25% test data). This technique, inspired by "vertical slicing" in [Kor2007], guarantees that the test data will cover the whole range of possible output values.

The search strategy settings were as follows. For pre-evolution filtering: influence threshold ι_{thr} = 25%, bias to useful expressions κ = 80%. In GDR, pruning degree τ = 0.5. In CAFFEINE, all settings were like in section 4.5, except population size N_{pop} =

Table 4.7: *Parameters of Circuits for the CAFFEINE Scaling Experiments.*

# Variables	# Devices	# Train Samples	# Test Samples	Performances Modeled
24	10	129	32	A_V, THD, SR, OS
59	30	330	82	A_V, THD, SR, OS
129	50	1050	262	A_V, THD, SR, OS

100, and maximum number of generations $N_{gen,max}$ = 50. Far fewer generations are now needed to get to reasonable results because the pre-evolution filtering picks highly useful expressions, and the linear bases are always available.

4.7.2 Experimental Results

This section aims to see how well the scalability goals were achieved: on the above examples, run in a reasonable time (hours or minutes), have predictive models, and have interpretable models.

To assess the scalable CAFFEINE, we compare its models to a reference regression algorithm that has a good track record of predictive ability and of scalability: MARS [Fri1991]. To make the comparison as fair as possible, we used GDR for MARS' linear regression subroutine. A further motivation for MARS is that it was the most competitive to CAFFEINE in the experiments of section 4.5.

We first consider the interpretability of MARS-generated models versus CAFFEINE-generated models. We recognize that the judgement of interpretability is necessarily subjective, so here we aim to give the reader a feel. To do so, we must review MARS slightly further. Each MARS basis function is a product of "hockey stick" (HS) functions:

$$B_{MARS}(\boldsymbol{x}) = \prod_{i=1}^{N_{prod}} HS_{(i)}(x_{(i)}, t_i, q_i) \tag{4.12}$$

where $HS_{(i)}$ is the i^{th} HS function having either a $+$ or $-$ sign, and $x_{(i)}$, t_i, and q_i are the chosen input variable, split value, and power for $HS_{(i)}$, respectively. A HS function is:

$$HS_{\pm}(x,t,q) = \pm \begin{cases} 0 & if \;\; x < t \\ (x-t)^q & if \;\; x \geq t \end{cases} \tag{4.13}$$

To see how MARS basis functions look on real problems, we will use an arbitrarily chosen example OS, from the largest circuit (50T opamp). Table 4.8 shows the equation for just a *single* MARS basis function. As we can see, the hockey stick functions translate to very hard-to-interpret functions.

We saw that even a single basis function from MARS is extremely challenging to interpret. Table 4.9 shows the 50T opamp OS expression that CAFFEINE generated. The model is are not as interpretable as we have seen for smaller circuits, but *some* insights can be extracted. It is notable that of the 109 input variables, CAFFEINE pruned down to just use 17 variables, i.e. about 10% of the variables. The variables include widths

Table 4.8: *A single basis function in MARS-generated equation for OS of 50-transistor opamp*

$$\left\{\begin{matrix} 0 & if \;\; L_{M2} < 2.13*10^{-6} \\ (L_{M2}-t) & if \;\; L_{M2} \geq 2.13*10^{-6} \end{matrix}\right\} * \left\{\begin{matrix} 0 & if \;\; L_{M3} < 290.0 \\ (L_{M3}-t) & if \;\; L_{M3} \geq 290.0 \end{matrix}\right\} *$$

$$\left\{\begin{matrix} 0 & if \;\; m_{DP1M2} < 8.416*10^{-6} \\ (m_{DP1M2} - 8.416*10^{-6}) & if \;\; m_{DP1M2} \geq 8.416*10^{-6} \end{matrix}\right\}$$

Table 4.9: *Caffeine-generated equation of OS for the 50-transistor operational amplifier circuit. (**All** basis functions.)*

$$-780.8$$
$$+9.90*10^{8}*L_{MDP3} + 5.23*10^{8}*L_{MCM1} + 4.18*10^{8}*L_{M9}$$
$$-9.27*10^{8}*L_{MCMB2} - 4.24*10^{8}*L_{M4} - 4.20*10^{8}*L_{M13}$$
$$+11.46*m_{M2} + 7.11*m_{M17} - 8.83*m_{CM1M2}$$
$$+1.14*10^{8}*W_{MDP3} + 7.09*10^{7}*W_{MCM3} + 2.39*10^{7}*W_{M11}$$
$$-2.45*10^{7}*W_{M4}$$
$$-8.86*10^{6}*log_{10}(W_{MCM3})*m_{M10}^{3/2}*W_{MCM3}^{3/2}*(0.655*m_{CM5M1}^{2} + m_{CM2M1}^{5/2})$$

W, lengths L, and multipliers m. Most of the basis functions have a linear relation to OS. To decrease OS, there some L's which need to be decreased (e.g. L_{MDP3} and $5.23*10^{8}*L_{MCM1}$), while other L's need their values are increased (e.g. L_{MCMB2} and L_{M4}). Similarly, to decrease OS with m's, some m's need decreasing and others need increasing. And similarly for W's too. There is a single base with nonlinearity. It has interactions among the variables W_{MCM3}, m_{M10}, m_{CM5M1}, and m_{CM2M1}. It is very notable that of the 109 input variables, only 4 have significant interactions (in terms of affecting OS).

Table 4.10 summarizes the interpretability results for CAFFEINEversus MARS. In short, MARS models are definitely not interpretable, and CAFFEINE models (arguably) are, at least enough to extract some insights.

Table 4.10 also lists the CPU time that MARS and CAFFEINE each took to build each regression model. We see that the runtime is indeed reasonable, even for the largest problems. It is far faster than the original CAFFEINE on the smaller problems.

Table 4.10: *MARS and CAFFEINE build times and interpretability, for different problem sizes.*

# Variables	**Can interpret MARS model?**	**MARS build time (min)**	**Can interpret CAFFEINE model?**	**CAFFEINE build time (min)**
24	No	7	$\approx$ Yes	20
59	No	11	$\approx$ Yes	40
129	No	25	$\approx$ Yes	100

We have considered the interpretability and model construction times of MARS versus CAFFEINE. What about prediction ability? Table 4.11 presents the results of the regressors' prediction performance. We see that MARS and CAFFEINE have similar performance: in some cases CAFFEINE is slightly better, in other cases MARS is. We see that some problems are quite difficult to model (e.g. THD of the 10-device circuit), while other problems are quite easy (e.g. OS of the 50-device circuit).

In sum, this section has described techniques to scale up CAFFEINE to more input variables, and validated the new "scalable" CAFFEINE on 16 test problems.

Table 4.11: *Prediction (testing) error of MARS vs. CAFFEINE on larger circuit modeling problems.*

# Variables	# Devices	Output	MARS error (%)	CAFFEINE error (%)
24	10	A_V	3.52	2.95
24	10	THD	24.98	24.90
24	10	SR	0.18	0.42
24	10	OS	4.31	5.21
59	30	A_V	6.19	5.54
59	30	THD	3.53	6.85
59	30	SR	0.32	1.23
59	30	OS	6.25	6.06
109	50	A_V	3.42	3.28
109	50	THD	4.47	4.51
109	50	SR	0.90	0.92
109	50	OS	0.08	0.08

4.8 Other Applications

This section describes other problem types that CAFFEINE has been applied to, which include behavioral modeling, robustness modeling, and analytical performance tradeoff modeling.

4.8.1 Behaviorial Modeling

CAFFEINE has also used been applied to generate behavioral models of analog circuits, as an "Interpretable Behavioral Model Generator" (IBMG) [Mcc2005b]. There has been much progress in automated behavioral modeling and especially model order reduction (MOR) [Rut2007]. Despite this, manual design of models remains popular because humans can leverage their insights, and take responsibility as needed for the final model. CAFFEINE can *bridge* manual and automated design, by offering behavioral model "suggestions" to guide the modeling expert. These suggestions are resulting from evolving the models in CAFFEINE.

The problem description is as follows. We consider a p-input q-output nonlinear dynamic system, specifically a circuit, of the form:

$$\frac{dx}{dt} = f(x(t), u(t)) \tag{4.14}$$

$$y(t) = C * x(t) + D * u(t) \tag{4.15}$$

where $x(t)$ is the system's n-dimensional state (i.e. node voltages and branch currents in the circuit), $u(t)$ is the p inputs at time t, and $y(t)$ is the q outputs at time t. $f(x, u)$ is an arbitrary nonlinear function vector field that describes how the state changes. $y(t)$ is a linear function of $x(t)$ and $u(t)$.

The task is to create a more compact form of the given dynamical system, i.e. one with m states where $m = n$. The model must be interpretable behavioral expressions, i.e. easily readable functional forms that describe how the state changes. Finally, the approach must have error control by actually generating a set of models that trade off between error and complexity. The generator's inputs are $u(t)$ and $y(t)$, taken from a transient simulation using a standard SPICE simulator. With the aim of interpretability, $x(t)$ is not an input, even though it creates a more difficult learning problem. The expressions to be generated must take the form:

$$\frac{dz}{dt} = g(z(t), u(t)) \tag{4.16}$$

$$y(t) = E * z(t) + F * u(t) \tag{4.17}$$

where z is the system's state, and g, E, and F are the reduced-system equivalents of f, C, and D respectively. The initial system state is set to be $z(0) = \{0, 0, \ldots\}$. IBMG must "learn" the vector valued function $g(z, u)$ as well as E and F. Learning E and F is merely a set of linear learning problems (one for each output variable) once $z(t)$ for each t is known. Learning $g(z, u)$ is the major challenge, as each point $g \in G$ involves a choice of the number of basis functions, and the functional form of each of those basis functions (which takes the other basis functions and u as an input). Any possible composition of functions is allowed.

We could have formulated the problem more generally, i.e. y as a nonlinear function of x and u. But IBMG approximates nonlinear mappings via state variables that do not appear in $f()$, which relate x and u to y in a nonlinear fashion. In making this choice we simplify IBMG and also encourage re-use of expressions for outputs. Alternatively, we could have formulated the problem where IBMG is supplied the circuit's internal state information (a much easier problem). Instead we will force IBMG invent its own states and state transition equations.

To solve the problem, CAFFEINE was altered into "IBMG" by making it evolve the differential equations $g(z, u)$, which included discovering the state variables to use. E and F are found via solving a least-squares problem. Its setup parameters were the same as section 4.5.

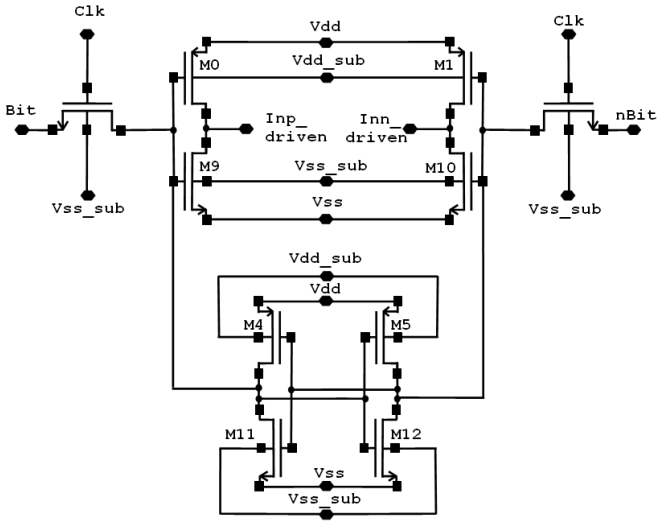


Figure 4.14: *Highly nonlinear latch circuit.*

We test IBMG on the strongly nonlinear latch circuit, shown in Figure 4.14. The technology is 0.18μm CMOS. Vdd=1.8V, Vdd_sub=1.8V, Vss=0.0V, and Vss_sub=0.0V. Figure 4.15 shows the circuit's input and output waveforms. IBMG's goal is to build a model that produces similar outputs given those same inputs. Vdd and Vss are also treated as inputs to IBMG. Each waveform had 2001 samples.

We ran IBMG to build models for the latch. Runtime was 72 hours (a compiled implementation would be about an order of magnitude faster). Figure 4.16 shows the best-performing result, which achieved an error of 1.31%. This is a fairly tight fit, especially given that IBMG did not use the circuit's internal state information and instead had to invent its own states and state transition equations. Examining the waveform, we see that the sharp nonlinear transitions are handled quite gracefully, though the model output jumps around somewhat at around 0.5 ns. The output is fairly smooth over time in part thanks to minimization of error of derivatives. Thus, IBMG has accomplished the error-minimization goal.

Figure 4.17 illustrates the outcome of IBMG's error-control strategy: a set of about 50 behavioral models that collectively trade off model complexity with error. Table 4.12 shows in detail a subset of the resulting models, at different levels of complexity and accuracy. Even the best model with 1.3% error is highly interpretable.

In short, IBMG is a variant of CAFFEINE, specifically designed for offering behavioral model "suggestions" to guide the modeling expert. This aim was confirmed by experimental results for a highly nonlinear latch circuit.

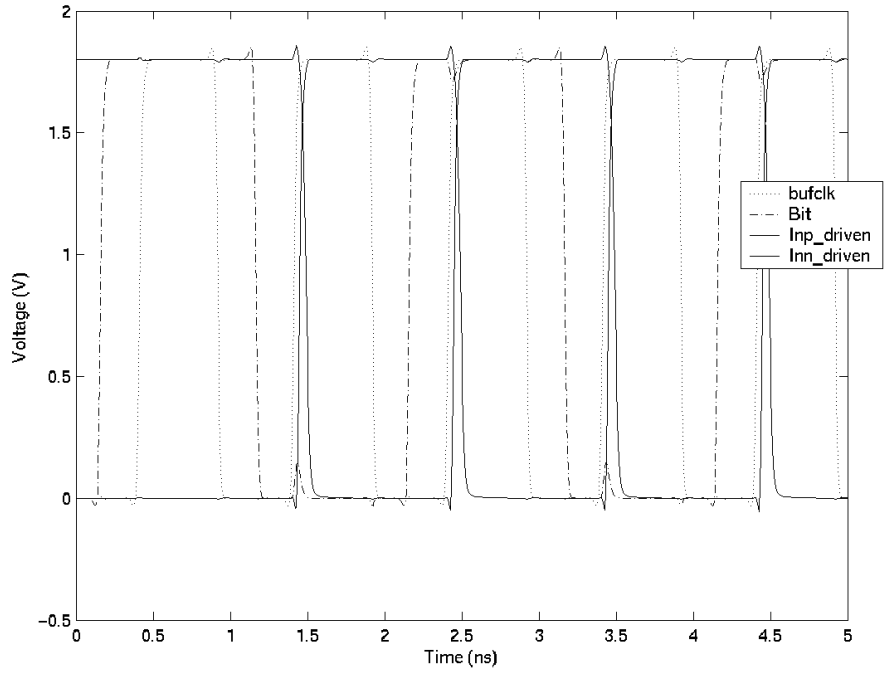


Figure 4.15: *Input and target output waveforms (nBit, not shown, is merely the inverse of Bit).*

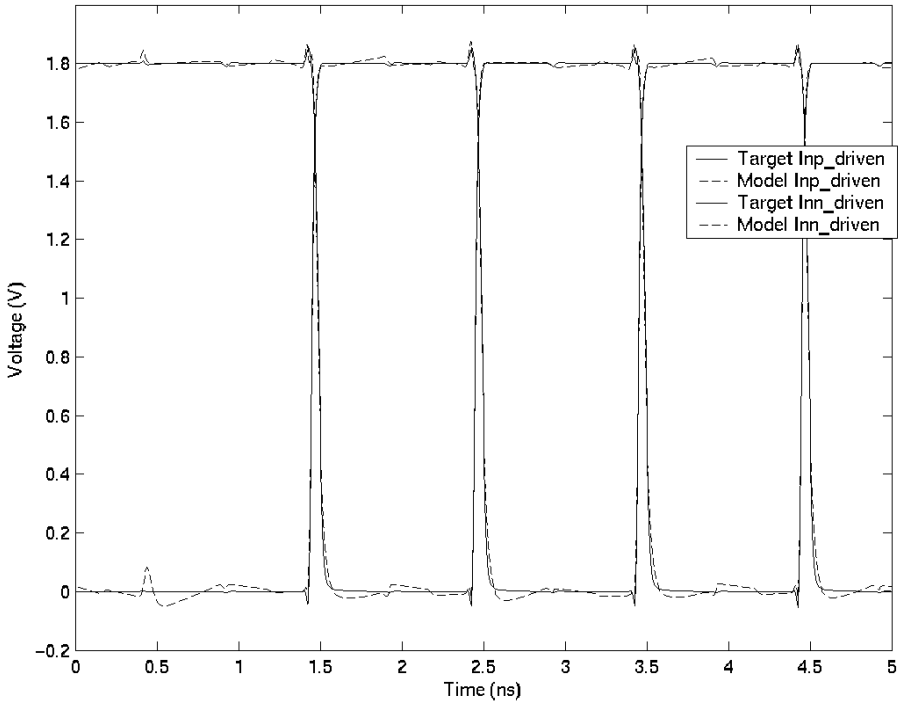


Figure 4.16: *Target output signals and model output signals.*

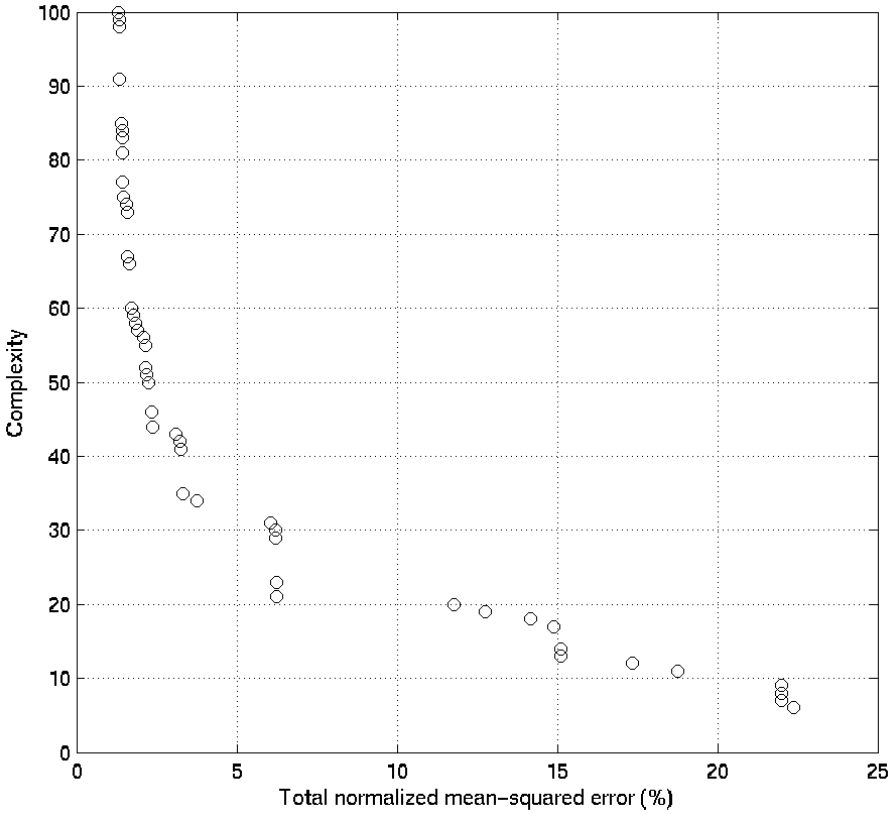


Figure 4.17: *Pareto front of complexity versus error, for CAFFEINE-evolved behavioral models.*

Table 4.12: *IBMG-generated behavioral models for the latch circuit.*

Train error	Expression
15.11%	$dx_1/dt = nBit$ $dx_2/dt = Bit * x_1$
6.25%	$dx_1/dt = -21.3 - 9.28 * 10^{-3} * bufclk * x_1 + 10^4 * nBit * bufclk$
3.32%	$dx_1/dt = 2.21e-2 - 3.72 * 10^{-2} * x_1 - 21.8 * Bit * nBit * bufclk$ $dx_2/dt = nBit * bufclk * x_1$ $dx_6/dt = x_1$
1.31%	$dx_1/dt = 78.2 + 1.06 * 10^{-3} * Bit * x_1 - 2.11 * 10^{-2} * bufclk * x_1$ $-4.85 * Bit * nBit * bufclk * x_{10}$ $dx_2/dt = nBit * bufclk * x_1$ $dx_3/dt = x_1$ $dx_4/dt = Bit * nBit * bufclk * x_1 * x_{10}$ $dx_6/dt = Bit * nBit * bufclk * x_1$ $dx_8/dt = Bit * nBit * bufclk$ $dx_9/dt = bufclk * x_1$ $dx_{10}/dt = 25.9 + 1.44 * 10^{-4} * Bit * x_1 - 1.89 * 10^{-3} * x_{10}$
E, F for 1.31%	$Inp\_driven = 7.62 * 10^{-3} + 8.85 * 10^{-7} * x_1$ $-2.98 * 10^{-8} * x_2 - 7.63 * 10^{-10} * x_3 + 3.02 * 10^{-11} * x_4$ $-7.77 * 10^{-7} * x_6 + 0.07 * x_8 + 7.43 * 10^{-8} * x_9$ $-1.05 * 10^{-5} * x_{10} - 2.32 * 10^{-12} * Bit - 7.77 * 10^{-7} * nBit$ $-2.60 * 10^{-7} * bufclk + 0.07 * Vdd - 0.05 * Vss$ $Inn\_driven = 0.42 - 3.91 * 10^{-7} * x_1 + 3.15 * 10^{-8} * x_2$ $-4.93 * 10^{-10} * x_3 - 2.32 * 10^{-12} * x_4 - 2.60 * 10^{-7} * x_6$ $-0.05 * x_8 - 8.82 * 10^{-9} * x_9 + 8.95 * 10^{-6} * x_{10}$ $-2.32 * 10^{-12} * Bit - 7.77 * 10^{-7} * nBit$ $-2.60 * 10^{-7} * bufclk + 0.07 * Vdd - 0.05 * Vss$

4.8.2 Robustness Modeling

In the application of statistical modeling, the designer's goal is to gain an understanding of how designable variables affect robustness measures such as yield or Cpk ("process capability") [Nist2006]. This application follows the same methodology as performance modeling, except in this case the Cpk is modeled from SPICE simulation data. The example circuit used is the 50-device amplifier of Figure 3.21.

Table 4.13 shows the CAFFEINE-generated equation for Cpk [Mcc2006b]. The testing error was 6.3%. Note that the technology variations are embedded in the numerical coefficients of the model. Cpk is not a function of these process parameters, only their aggregate effect on the design variables. Since the process parameters are not part of the model, this model is specific for the given technology.

In examination of the expression, we can learn a several things. First, only five variables are needed to hit the 6.3% test error: C_c, W_{dp2}, W_{dp1}, W_{dp2}, W_{mt4}, W_{mt1}. The variables comprise one compensation capacitor and four widths, and no lengths or multipliers. There are significant nonlinear interactions among the variables. An increase to W_{mt4} will increase Cpk, as will a decrease to W_{mt1}. Cpk is quite dependent on the square root of C_c. Cpk can also be increased by increasing W_{dp2} (big effect) or increasing W_{dp1} much smaller effect

Table 4.13: *Caffeine-generated equation of Cpk for 50-device amp.*

$+1231.4$
$+4.21 * 10^6 * W_{mt4}^2 / W_{mt1}$
$-0.0012/\sqrt{C_c}$
$-9.39 * 10^8 * W_{dp2}^2 * \sqrt{W_{dp1}} * min(0.104, 6.60 * 10^7 - 76.9/\sqrt{C_c})$
$+1.21 * 10^{12}/min(-4.96 * 10^6, 10^{10} - 2.48 * 10^5/(\sqrt{W_{dp2}} * C_c))$

4.8.3 Automated Sizing

While the original intent of CAFFEINE was to provide designers with insight about their circuit, CAFFEINE's predictive abilities were good enough to merit examining if its model building is fast enough to put into the loop of an automated circuit-sizing application. The paper [Mcc2006a] explored that opportunity, and achieved speedups which made CAFFEINE modeling fast enough for such an application. Note that the scalability enhancements in section 4.6 of the present chapter have improved CAFFEINE's speed sufficiently as well.

Details can be found in [Mcc2006a].

4.8.4 Analytical Performance Tradeoffs

In [Mcc2008b, Mcc2009], CAFFEINE was used to extract analytical models of the tradeoff among circuit performances. The approach is to use the performance values from a set of Pareto-optimal circuits. All but one of the performances is used as inputs to the

CAFFEINE models, and the remainining performance is used as the model output. Section 8.5 has details.

It can be concluded that CAFFEINE has many possible applications in circuits and elsewhere, because it is appropriate wherever regression tools are used, and where insight into the mapping is desirable.

4.9 Sensitivity To Search Algorithm

There is possibility to change the search representation, yet constrain the search to canonical form functions. The representation discussed so far is direct: a tree-based genotype maps to the "function" phenotype. In [Mcc2006a], a string-based genotype with neutral networks ("introns") was used. In [Mcc2006c], five different variants of search representations and algorithms were explored. They included a comparison of grammatical GP variants, where the genotype is either a tree [Whi1995] or a string of derivation rules [ONe2003]. No approach was markedly better. The salient result was that all of them could return useful interpretable models. Section 4.6 described other changes to the model-construction algorithm to improve its scalability.

This underscores the key contribution of this chapter: given SPICE simulation data, apply *some* competent search algorithm and representation to the space of canonical form functions, and one can get reasonable, interpretable, template-free circuit performance models. Of course, the choice of algorithm affects the model building time and ability to scale to more input variables or training samples, as section 4.6 discussed.

We refer the reader to [Mcc2006a, Mcc2006c] for more details.

4.10 Conclusion

This chapter has presented a tool to support analog circuit sizing by giving the designer insight into the mapping from design variables to performances.

CAFFEINE is a tool which for the first time can generate interpretable, symbolic models of nonlinear analog circuit performance characteristics as a function of the circuit's design variables, without *a priori* requiring a model template. The keys to CAFFEINE are: a flow which leverages SPICE simulation data, a means of extracting interpretable functions from the simulation data based on genetic programming search, and canonical-form constraints on the functions to ensure interpretability. Using multi-objective genetic programming, CAFFEINE generates, without an initial template, a set of models that collectively trade off between error and complexity.

In the first round of experiments, visual inspection of the models has demonstrated that the models are interpretable. The performance models were also shown to be significantly more compact than posynomials. The CAFFEINE models also had markedly better prediction ability than posynomials, projection-based polynomials, support vector machines, MARS splines, neural networks, and boosted neural networks. This indicates that CAFFEINE can be applied to under-the-hood applications too, such as circuit optimization that uses model-building within the optimization loop to determine new candidate

design points. CAFFEINE has also demonstrated promise in applications of robustness modeling and behavioral modeling.

This chapter has also described techniques to scale up CAFFEINE to handle many more input variables: subtree caching, gradient-directed regularization to prune during linear learning, pre-filtering single-variable expressions, and generously considering linear basis functions. In second-round experiments on problems with more than 100 input variables, CAFFEINE has achieved a prediction performance comparable to state-of-the-art blackbox techniques like MARS; and unlike MARS models the resulting CAFFEINE equations can be visually inspected and are not constrained to a predefined functional template.

This chapter has also described the application of CAFFEINE to other analog circuit problems such as behavioral modeling and robustness modeling. It can be concluded that CAFFEINE has many applications in circuits and elsewhere, because it is appropriate wherever regression tools are used, and where insight into the mapping is desirable.

The last three chapters have discussed background, design-aiding, and insight-aiding tools for the designer task of global varation-aware sizing. The next several chapters generalize beyond the sizing task, to also search for circuit *structure* or topology design.

Chapter 5

Circuit Topology Synthesis: Background

I do not know what I may appear to the world; but to myself I seem to have been only like a boy playing on the seashore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.
–Isaac Newton

5.1 Introduction

Recall from chapter 1 the basic flow for analog circuit design at a node in the hierarchy. We show it again in Figure 5.1. The first step in this flow is topology design or selection. The choice of analog circuit topology has a giant impact on the performance of the overall design. Designers often make the topology selection decision based on experience. While the choice of a topology is often thought of as a relatively quick decision compared to sizing and layout, the implications of the choice resonate throughout the rest of the design cycle. Even the best circuit optimizers can only produce as good a result as the chosen topology allows [Rut2002].

Unfortunately, a suboptimal topology choice can occur:

- The topology may not worsening effects due to Moore's law, such as larger statistical variations [Itrs2007].
- The topology may not handle new effects, previous undesigned-for effects like such as proximity [Dre2006].
- Functionality requirements may be qualitatively new to the designer.
- Or, the designer may unknowingly miss an advance in topology design.

In the combined steps of topology design/section and sizing (first two steps of Figure 5.1), the aim is to automatically determine the circuit components, interconnections, and suggested component dimensions to meet a set of circuit design goals. Goals can be

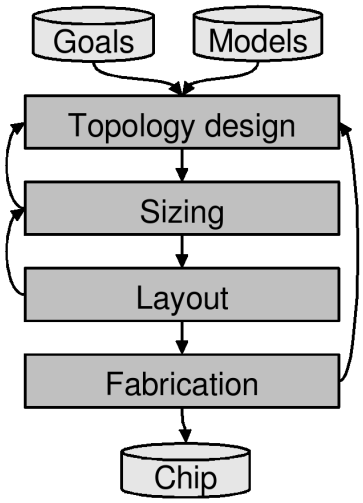


Figure 5.1: *Basic flow for analog circuit design (at a node in the hierarchy).*

performance constraints and / or objectives. If the goals are just constraints, the aim can be called a "specs-in, sized-topology out" flow.

Because of the importance and challenge of topology selection and design, techniques relating to it have been researched extensively in both the analog CAD literature [Rut2002, Rut2007], as well as the evolutionary computation literature (specifically, in genetic programming and evolvable hardware). Despite the extensive work, there is not yet an industrial tool for topology selection or for design.

Topology selection / design tools will be the focus of this chapter and subsequent chapters. This chapter is a review, and the other chapters present a set of techniques with industrially-oriented applicability.

The rest of this chapter is organized as follows. Section 5.2 examines different possible topology-sizing design flows (per sub-block), and their relation to techniques in industry and academia. Section 5.3 discusses which flows incorporate best into hierarchical design methodologies to handle system-level design. Section 5.4 presents requirements for a topology selection / design tool, with an eye towards industrial applicability. Because there has been much recent research to open-ended topology synthesis using GP, section 5.5 examines and explains why open-ended topology synthesis is so problematic. Section 5.6 concludes this review chapter.

Subsequent chapters present MOJITO and its derivatives. MOJITO is a topology selection / design tool that has industrially-acceptable inputs and outputs, accuracy, and runtime. Derivatives leverage MOJITO to enable topology-performance knowledge extraction, and accelerate design of novel topologies.

5.2 Topology-Centric Flows

This section examines different possible topology-sizing design flows, and the related literature for each flow.

5.2.1 Flow: Industrial Status Quo

The first flow of interest is the industrial status quo, as shown in Figure 5.2. Here, the designer starts by manually selecting an initial topology off-the-shelf, based on the input design goals (performance specifications and objectives). He then sizes that topology, using either an automatic optimizer e.g. [Cdn2005b] or manually.

If he hits the target design goals with that topology, he can declare the topology-sizing steps done, and proceed to layout. If not, he will select the next-most promising off-the-shelf topology, and size it. Once again, if goals are hit, he can stop, otherwise he will select another promising topology. He will continue trying this until he runs out of appropriate off-the-shelf topologies.

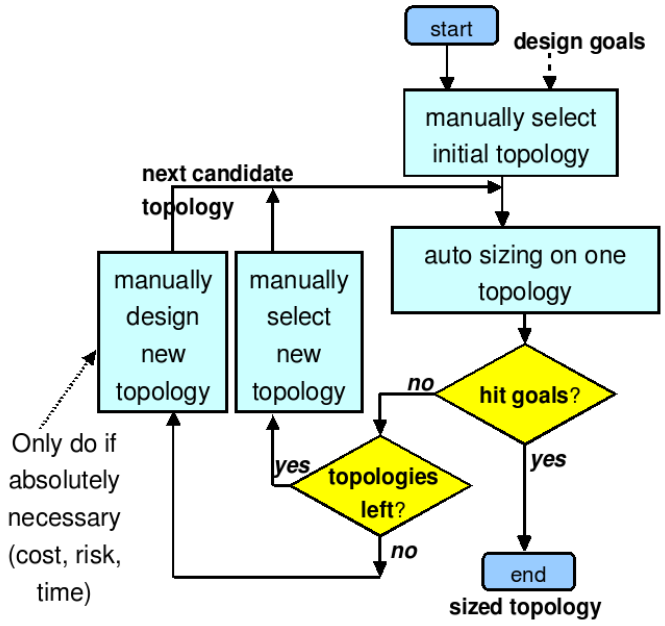


Figure 5.2: *Status quo industrial flow for topology selection/design and sizing.*

If no topology can meet specs, a decision must be made. If a new design is not absolutely needed, such as if performance goals can be loosened, the designer can declare the design "good enough" and move on to other work. There is strong desire to avoid designing a new topology because of the greatly increased risk that the design will not work, with significant cost ramifications due to the need to re-spin and increased in time-to-market. Because of these risks, the motivation for a new topology has to be strong. New topologies only come about if there is no other way, if the idea has possible orders of magnitude payoff such that it's worth the money to try, or if there is some way to make trying it zero risk.

However, sometimes those motivations for a novel topology design exist, so the designer will have to create a new topology. He will try to minimize risk and design time by basing the new topology on other analog circuit topologies and building blocks, using his knowledge and experience. He will continue working on it until he hits the target, or runs out of time or ideas.

The advantage of this flow is that the final topology is either fully trusted (because it either uses a known topology), or mostly trusted because the new topology is similar to previous topologies, with changes that an analog engineer reasoned his way through. Unfortunately, the topology selection time can be unpredictable, especially if there are several iterations of topology selection and circuit sizing. Novel topology design relies on the designer being able to be "inspired" which is hard to schedule, and there is still risk because until the new topology gets verified from manufacturing and test. Therefore the design time for handling new topologies is poor, and even holds risk of no invention happening. The flow is somewhat complicated, though it does feel natural to designers.

Can this status-quo flow be improved upon?

5.2.2 Flow: Automated Topology Selection

One way to improve upon the status quo industrial flow is to *automate* the topology selection process, as shown in Figure 5.3. The overall flow looks similar to the industrial flow of Figure 5.2, but the implementation is substantially different because of the extra input needed. In particular, the auto-selection step needs to have a topologies database (DB) as input, and possibly selection rules for the topologies database as well. (If the database does not have selection rules, it needs to compute them on-the-fly).

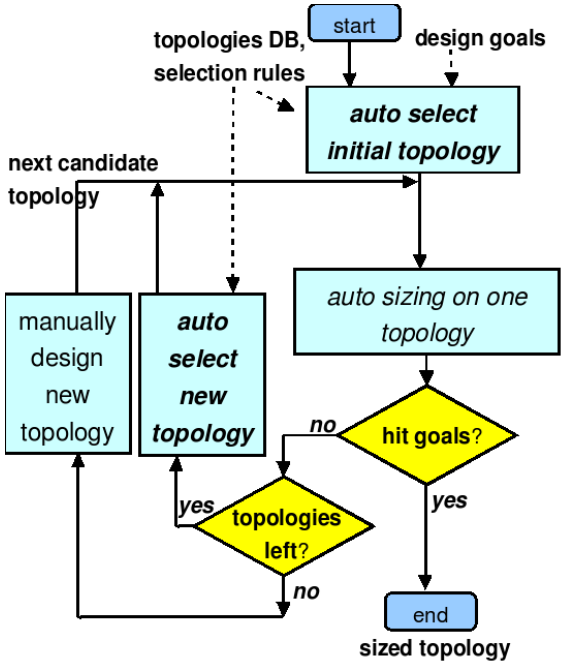


Figure 5.3: *Designer flow for topology selection/design and sizing, which incorporates automated topology selection.*

Several analog CAD systems in the literature have been proposed to follow this flow. Most notable are the rule-based "expert system" style approaches starting in the mid-1980's. These include BLADES [Tur1986], ISAID [Tou1990], OASYS [Har1992], and more [Ber1988, Fung1988, Koh1990, Sto1992, Ant1995, Ning1991, Swi1991, Hor1997].

For example, OASYS [Har1992] has a pre-specified decision tree [Bre1984] that chooses among 12 different topologies, depending on the 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. While many proposals were made, the approaches fell out of use as process generations inevitably changed due to Moore's Law [Itrs2007].

Approaches like AMGIE [Plas2002] aim to overcome the process node issue by automatically partitioning the space using SPICE in the loop, prior to the main automated sizing loop. The problem was that the approach did not support very many topologies, made potentially dangerous assumptions while partitioning, and had substantial computational effort for the partitioning step.

Each topology that the system might output has to be entered by the CAD developer beforehand. While this means that all the topologies are inherently trustworthy, it also means significant effort to enter the topologies into the CAD system. Another side effect is that the tool cannot help in generating novel circuitry, for example when no topology in the DB meets specifications.

A final variant, derived from [Her1998] takes an interesting twist on the topology "selection" problem. Rather than trying to intelligently select the topology, it merely iterates through a list of hundreds of possible topologies and optimizes each one until the target is hit. At first glance this sounds computationally expensive. However, it is not expensive in the proposed flow because each optimization is extremely *fast*, taking on the order of seconds. It is fast because the circuit has been pre-modeled as a convex optimization problem, so the optimization needs to merely run a convex solver on the pre-set equations [Boyd2004]. The problem is that convex performance models are needed *a priori*. In early work, these models were manually generated, which is obviously very tedious. More recent work [Dae2002, Dae2003, Agg2007] has shown how to automatically create convex posynomials from simulation data, but unfortunately they have poor prediction accuracy, as section 4.5 has examined.

5.2.3 Flow: Flat, Lightweight Multi-Topology Sizing

Rather than manually specifying each topology separately, a different approach is to define a *family* of topologies 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. We call this "multi-topology sizing" because the approaches simultaneously consider the topology choices and sizes/biases in a unified search space. The flow is shown in Figure 5.4. Such approaches include DARWIN [Kru1995] and MINLP [Mau1995] for opamps, and [Dob2003, Fra2000, Tang2006] for system-level designs.

A key advantage is that these approaches only require structural information about the circuit, which is independent of the process node. (Though in some system-level cases, behavioral models are also an input, to speed search.) Another advantage is that the topologies are trustworthy by construction.

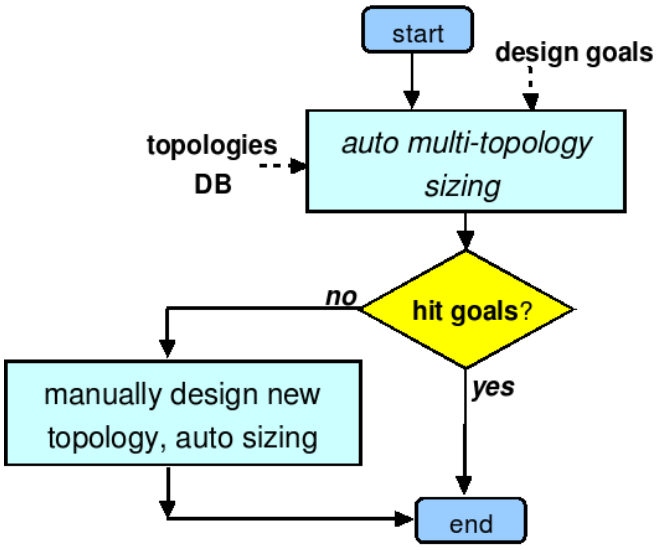


Figure 5.4: *Designer flow for flat, lightweight multi-topology sizing*

Unfortunately, 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. These limitations mean that DARWIN has just 24 possible opamp topologies, and MINLP just 64 possible opamp topologies. For this reason, give the search space of flat multi-topology sizing the "lightweight" label.

5.2.4 Flow: Open-Ended Topology Synthesis

Starting from the mid-1990s, a very different approach to determining a sized topology was taken in the evolutionary computation literature, in the subfields of genetic programming (GP) and evolvable hardware. The flow for the approach is shown in Figure 5.5.

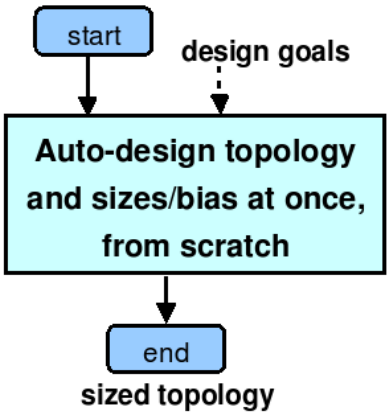


Figure 5.5: *(Ideal) designer flow when using open-ended topology synthesis.*

The flow looks amazingly simple at first glance, taking just the design goals as inputs, and producing a sized netlist. GP is given a set of devices (transistors, resistors, etc) 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. No topology information is input.

GP has a natural ability to handle search spaces with tree-like and graph-like structures (topologies). Therefore it seems to be a natural fit for searching the space of analog circuit topologies. Many researchers have explored this, including [Koza1997, Lohn1998, Koza1999, Gri2000, Zeb2000, Goh2001, Shi2002, Sri2002, Zeb2002, Koza2003, Ando2003, Koza2004b, Hu2004, Das2005, Cha2006, Mat2007, Sap2008].

The early approaches such as [Koza2003, Lohn1998, Shi2002]) were very open-ended, having few constraints. Unfortunately, they had prohibitive CPU effort. Even worse, and results which were not *trustworthy* because there was no apparent logic behind them. The trust issue was exacerbated because the results often looked strange. Such odd circuits can be found in early papers like [Koza1997], all the way to very recent papers like [Sap2008].

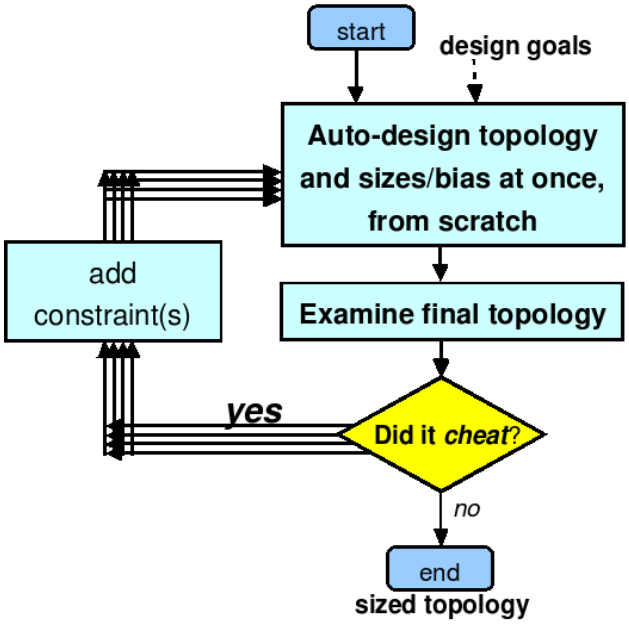


Figure 5.6: ***Actual*** *designer flow when using open-ended topology synthesis.*

Researchers who wanted to have industrially interesting circuits found themselves adding constraints, then re-running the system, and adding more constraints, and so on, in a seemingly non-stop loop. Some of the more recent efforts [Sri2002, Das2005, Mat2007]) added tighter constraints using domain knowledge to improve efficiency and trustworthiness, but there is still no guarantee of trustworthy results or even trackable novelty. Judging by the published results, the constraints added by this research seem to have restricted the synthesis system to very tiny circuits of just a few transistors. Furthermore, many circuit robustness issues were still ignored. The manual, painful, iterative

constraint-adding loops still exist. Therefore, while Figure 5.5 was the *ideal* flow for open-ended topology synthesis, Figure 5.6 is a more *realistic* flow that actually reflects the constraint-adding process.

As evidenced by the number of publications, the GP community has strong interest in open-ended topology synthesis. Interestingly, the community considers topology synthesis a success story for GP, *despite* the fact that it is not being used commercially after 10 years of intense research. Because of this interest, and these misconceptions, we explore open-ended topology synthesis' deeper issues in significantly more detail in section 5.5.

5.2.5 Flow: Hierarchical, Massively Multi-Topology Sizing

This flow aims to get a structurally-diverse search space like open-ended synthesis, yet return topologies that are trustworthy-by-construction. The flow for this approach is shown in Figure 5.7.

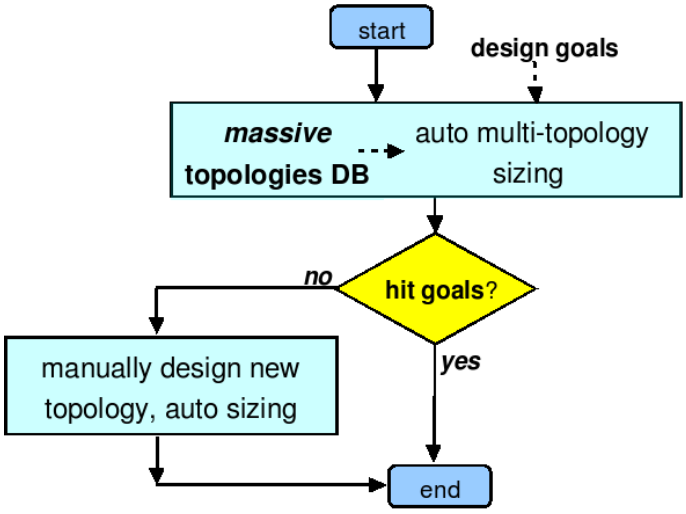


Figure 5.7: *Designer flow for massively multi-topology sizing.*

At first glance, the flow has strong similary to the flow of flat, lightweight multi-topology sizing (Figure 5.4): in both cases there is multi-topology sizing, which takes in design goals and a topologies DB as input. But the difference is in how the input topologies DB can be interpreted. If the library has a *sufficiently rich* number of topologies, it means that the designer does not have to intervene in a typical design problem. That is, the library is hidden, from the perspective of the designer. That is why the topologies DB is *within* the tool block in Figure 5.7, as opposed to being an explicit input from the designer. This is of interest to designers, because it means that it uses the same inputs and outputs as existing industrial automated sizers like [Cdn2005b]. Actually, there is even one less input – unlike the sizers, the topology does not need to be specified. The flow is simply "specs in, sized topology out".

The challenge is in how to specify a sufficiently rich library. After all, the "flat" topology libraries for opamps had < 100 topologies, with no clear way to grow much bigger.

The answer is in the use of hierarchy to specify the library. That is, the set of possible topologies is specified by a set of hierarchical analog building blocks. Some building blocks can instantiate (refine) into one of many building blocks. This framework can be viewed as a grammar, where different sentences in the grammar are different topologies, and the building block composition of the topologies follows the grammar's derivation rules.

Interestingly, one could do a multi-topology sizing run with many objectives, resulting in a results DB of Pareto-optimal sized topologies. Using this results DB, future queries for a sized topology, given performance specifications, are merely a cheap database lookup away. That is, it is "specs in, sized topology out, *immediately*."

One group uses this approach [Mar2008] for system-level (A/D) design. The algorithm starts with *just* abstract models having behavioral model performance descriptions. Over the run of the algorithm, it allows the abstract models to go through refinements, having transforms to well-known structural descriptions to give trusted topologies. This is, in effect, a grammar. An issue with this approach is that multiple resolutions of behavioral models must be specified, which makes defining the library of topologies more difficult.

An approach requiring less library setup effort takes in *just* structural descriptions, not behavioral. The papers [Mcc2007, Mcc2008a] showed how just 30 building blocks could expand into thousands of different topologies. Chapters 6 and 7 describe this approach in much greater detail.

5.2.6 Hierarchical, Massively Multi-Topology Sizing *With Novelty*

A problem with the previous flow (hierarchical, massively multi-topology sizing) is that if no topology can meet specifications, then the designer must manually determine a *novel* topology to meet specifications. Novel topology design relies on the designer being able to be "inspired" which is hard to schedule. Therefore the design time for handling new topologies is poor, and even holds risk of no invention happening.

A flow that can help to de-risk this is shown in Figure 5.8. This tool searches across 100% trusted topology space, and adds novelty *only* if there is a performance payoff. That is, only novel designs that actually give a payoff are rewarded. That is, it "innovates" as needed. It is especially useful if there is a mechanism to track novelty, to measure the degree of trust designers have in the topology.

5.2.7 Flows: Summary

The industrially interesting categories are hierarchical, massively multi-topology sizing, and its novelty extension because they achieve the trustworthiness of the analog CAD approaches, yet search through rich topology search spaces like the GP approach.

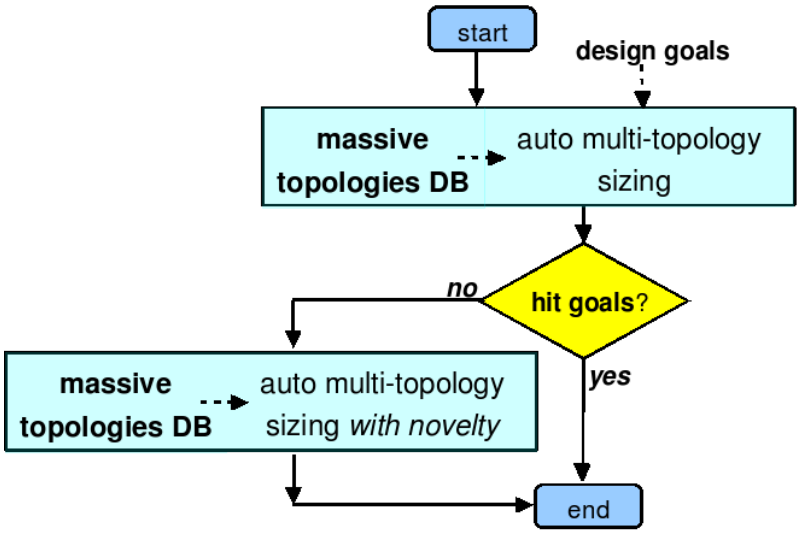


Figure 5.8: *Designer flow for massively multi-topology sizing with novelty.*

5.3 Reconciling System-Level Design

We desire for our target topology selection / design tool to handle complex system-level designs.

Recall from section 1.2.3 that system-level designs can easily be hierarchically decomposed into a set of sub-blocks, where each sub-block has known constraints and objectives. One can design a system-level circuit using a hierarchical design methodology to traverse the nodes (sub-blocks) in the hierarchy. Each node in the hierarchy is designed following the generic analog design flow of Figure 5.1. The choice of cell-level analog circuit topology can have a giant impact on the performance of a system.

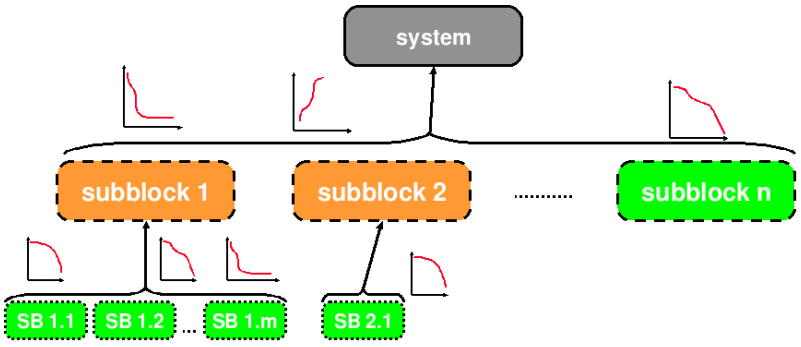


Figure 5.9: *The multi-objective bottom-up (MOBU) hierarchical design methodology (from section 1.2.3).*

An example is the multi-objective bottom-up (MOBU) hierarchical design methodology [Eec2005], which finds Pareto-optimal hypersurfaces at the lowest-level blocks with multi-objective optimization, then does multi-objective optimization at successively higher levels, as shown in Figure 5.9. The top-down constraint-driven methodology

[Cha1997] (TDCD) designs the top block first, resulting in specifications for each of its sub-block. Then, each sub-block is designed to meet those specifications, and so on. Before its top-down constraint-satisfaction step, TDCD needs models of feasibility of each sub-block. The Pareto-optimal surfaces of MOBU turn out to be a good way to generate such models.

Because these hierarchical methodologies are an effective way to approach system-level design, we desire for the topology selection / design tool to fit into them. MOBU needs a tool that generates a Pareto-optimal set of sized topologies, and TDCD needs a tool that performs constraint-satisfaction across several topologies *and* a way to generate a node's feasibility model (e.g. via generating a Pareto-optimal set).

Figure 5.10 gives two ways to handle constraint satisfaction, and Figure 5.11 gives two ways to handle multi-objective synthesis. The next paragraphs consider the relative merits of each approach within constraint satisfaction, and each approach within multi-objective synthesis.

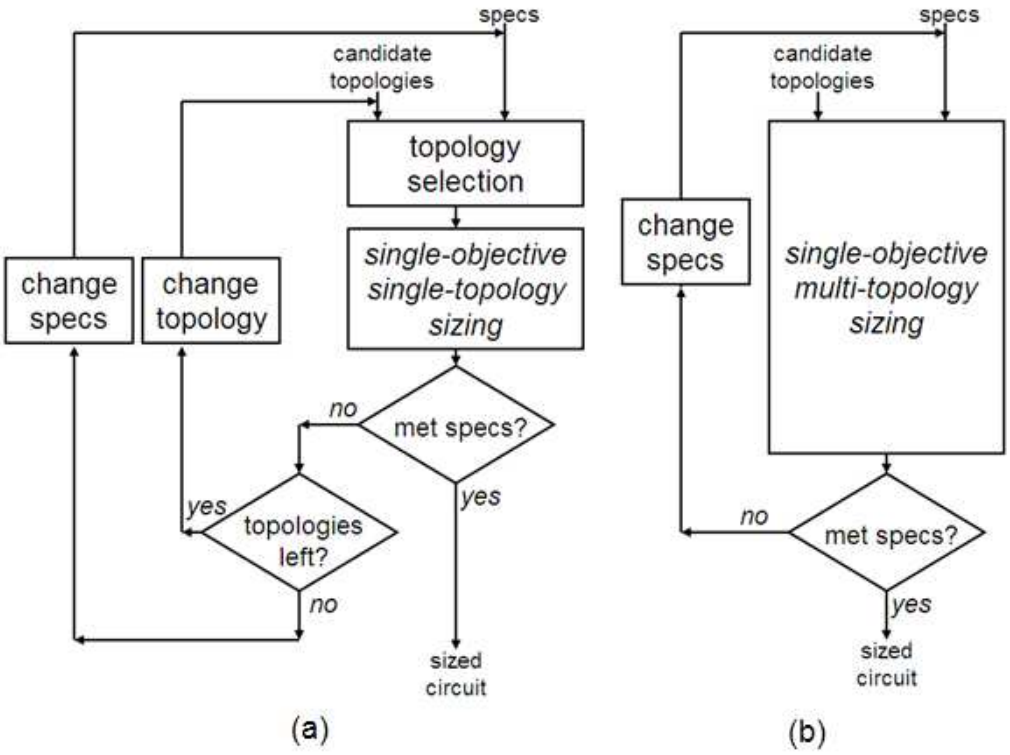


Figure 5.10: *Constraint-satisfaction flows for hierarchical design. (a) Uses just single-topology sizing, and (b) Uses multi-topology sizing.*

Figure 5.10 (a) shows a constraint-satisfaction approach that has two nested loops. The inner loop uses automatic topology selection in the "change topology" backtracking step. The outer loop changes specifications if needed (it is sometimes needed within the context of TDCD). This approach has two issues: two loops makes for a complicated flow.

Worse, the automated topology-selection step needs a topology rule-partitioning scheme, and those are problematic as section 5.2.2 described.

Figure 5.10 (b) is an improved way to do constraint satisfaction. In its flow, the topology-changing loop is replaced by multi-topology sizing. This overcomes both the issues: two loops, and topology rule-partitioning. Therefore, for constraint satisfaction, the flow of Figure 5.10 (b) is preferred to the flow of Figure 5.10 (a).

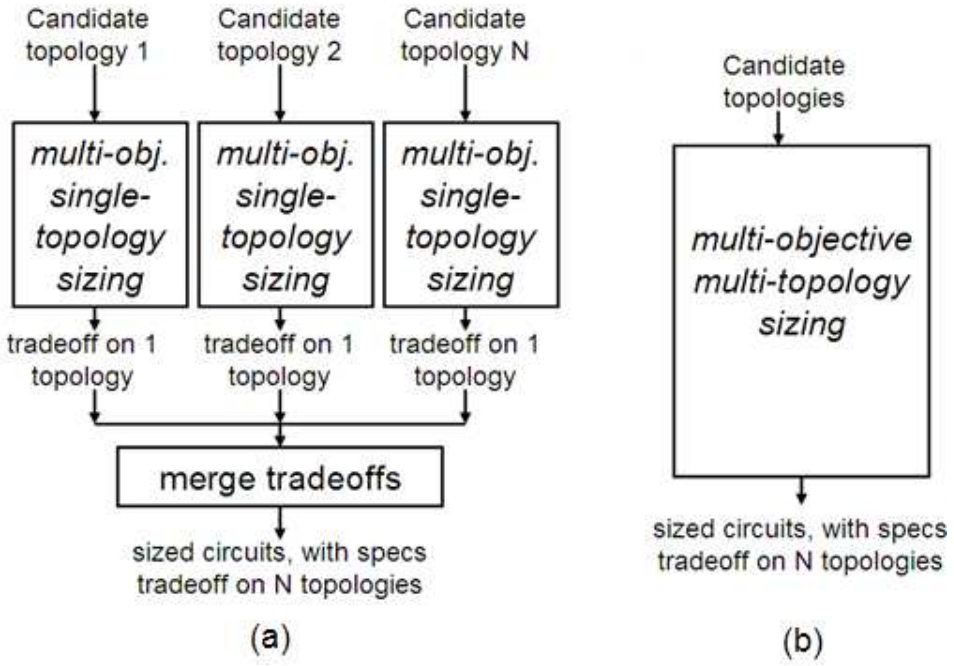


Figure 5.11: *Multi-objective flows for hierarchical design. (a) Uses just single-topology sizing, and (b) Uses multi-topology sizing.*

Figure 5.11 describes how multi-objective synthesis might be handled. In general, multi-objective sizing algorithms [Sme2003a] bypass the issue of needing specifications set *a priori* by optimizing on >1 objectives to generate Pareto-optimal performance tradeoffs as part of the search task. But so far, multiobjective sizers have only worked on one topology at a time. This means that to get an optimal tradeoff across multiple topologies, one needs one sizing run *per topology* before merging the topologies, as Figure 5.11 (a) shows. The Pareto-optimal sets across topologies are merged into a single set. Then, search at higher-level blocks implicitly performs topology selection of lower-level blocks [Eec2007]. Unsurprisingly, this enhanced-MOBU flow gave better results than using just a fixed topology approach to MOBU. But tradeoff-merging has limits: it would be extremely tedious and time-consuming to do a different sizing run for each of 100 or 1000 or more topologies.

Figure 5.11 (b) shows the ideal approach for multi-objective synthesis, which simultaneously considers a large number of possible topologies and returns a multi-topology tradeoff across specs, all in one sizing run with no re-loops needed. For multi-objective synthesis, Figure 5.11 (b) is preferred over Figure 5.11 (a).

In short, this section has described topology synthesis approaches that best fit the context of a hierarchical design methodology. The methodologies need either constraint satisfaction or multi-objective synthesis. In both cases, a *multi-topology* approach is the preferred flow. Specifically, Figure 5.10 (b) shows the preferred flow for constraint satisfaction, and Figure Figure 5.11 (b) shows the preferred flow for multi-objective synthesis. Since multi-objective optimization is typically more challenging than constraint satisfaction, and it is needed for MOBU and desirable to TDCD, then we will focus our multi-topology sizing efforts to be also multi-*objective*.

In summary, multi-topology synthesis can be readily incorporated into hierarchical design methodologies, and therefore handle complex system-level design.

5.4 Requirements for a Topology Selection / Design Tool

Now that we have reviewed several topology selection / design flows and related research, within the context of a hierarchical design methodology, we are well positioned to consider the requirements for a topology design / selection tool. To be appropriate for industrial use, a topology design / selection tool must have the following attributes.

If a topology that is *known* to be 100% trustworthy will meet the performance goals, then the tool should return that topology.

The tool should strive to keep the inputs and outputs as close as possible to existing industrial tools. The core aim is to use inputs and outputs that are acceptable for industrial optimizers like [Cdn2005b, Snps2005], but to add minimal extra I/O 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 [Raz2000, San2006]. And in fact, since we have already designed an example library (see chapter 6), the designers can use that. The tool provider would typically provide a default library for each circuit type.

The other inputs relate to computing performance: testbenches specify the circuit analysis and test harness to measure performance, objectives and constraints specify the aims of each performance measure, and simulator model files describe how transistors etc. behave for a particular semiconductor process node. Note that MOJITO only needs structural information; it does not need a special decision rule base, nor does it need abstract models with mappings to refined structures. This makes it straightforward to switch technologies, or even add new building blocks to the library.

The tool should output a Pareto-optimal set sized topologies, for selection by a designer or within a hierarchical methodology like MOBU or TDCD. By doing so, it can handle complex system-level circuits.

To avoid reinventing the wheel, and maximize trustworthiness, The tool should draw on as much prior structural design knowledge as possible, so long as that knowledge is convenient to the user. (It does not *have* to be convenient to the tool provider.)

To have SPICE accuracy and to be readily flexible to changes in process nodes, the tool should use SPICE for feedback, rather than specially designed performance estimators.

The tool should resort to adding novelty *only* if no existing known topology can meet performance goals. To add novelty otherwise would introduce unnecessary risk, such as the risk of a costly respin. If novelty is added, it should be easy to track where the novelty is added, and what the performance payoff is.

These requirements can be seen as a pragmatic fusion of requirements from knowledge-based CAD, optimization-based CAD, and evolutionary computation.

5.5 Open-Ended Synthesis and the Analog Problem Domain

This section elaborates on why open-ended topology synthesis is so problematic. It is primarily aimed at GP / AI researchers, to give some background and context so that the challenging nature of the structural synthesis problem becomes more clear.

5.5.1 Design "Implementation"

When GP researchers read about GP for analog topology synthesis, they are used to reading about "front-end design", as discussed in chapter 1. Specifically, the input is a set of circuit specifications (e.g. get gain > 60 dB, power consumption < 10mW), and the target output is a "netlist", which describes the synthesized circuit in terms of components, interconnections, and component dimensions.

That's actually just one step in a much broader flow, which is shown in Figure 5.1. Somehow, that netlist has to get into the real world, as a "chip" (VLSI circuit). The back-end flow is as follows. Once the netlist is determined, it is converted into a "layout", which is essentially a set of overlapping polygons at different layers, where specific layers / patterns represent specific types of components and interconnects. The layout is integrated into an overall system layout, which is sent to a billion-dollar fabrication facility. The system layout is used for creation of process "masks," which are a sort of physical filter on whether to dope / etch / etc. different parts of a silicon wafer. In modern process technologies, mask construction costs millions of dollars [Itrs2007]. Using the masks, many chips at once are fabricated on a wafer. The chips are sliced apart from each other, then packaged, and finally tested.

If a problem is detected after a step in this flow, then the process backtracks to the previous step. The most expensive step is creation of the process masks, so this is where it is most important to avoid backtracking. In a worst case, which still often happens in practice, a fabricated chip does not work at all. To fix it, one needs to go back to designer, and then fabricate and test again. This is known as a "respin." Obviously, respins are highly undesirabel because of mask costs. These days, respins also mean significant loss of profitability because of the delay in bringing the product to market.

Using a new analog topology *significantly* raises the chance of a respin due to lack of experience with that topology. This makes adoption of an analog structural synthesis tool a costly, risky proposition.

5.5.2 Analog Designer Perspective

Since the mid 1980's, analog designers have been presented with claims about "analog synthesis." Researchers have labeled "analog synthesis" to mean many things, including global parameter optimization, automated conversion from netlist to layout, and automated topology design (the version that GP targets). For a survey, see [Gie2002a].

Our focus here is automated topology design. Most analog designers would acknowledge that if such a technology actually worked, and well, it would considerably alter the nature of analog design. Their counterparts in digital design have already experienced such a revolution: the mid 1980's introduction of digital circuit logic synthesis.

Unlike digital synthesis, few claims of analog synthesis have held true. The analog synthesis techniques were typically too unscalable or brittle to be useful in industry. Of the dozens of various types of analog "synthesis" technologies reported over the last twenty years, just a few have found their way into industrial use, and that was only recently [Snps2005, Cdn2005a, Cdn2005b, Mun2008, Ext2008]. None of these do automated topology design (they do automated sizing/biasing and automated layout). Thus, when designers hear about a new structural synthesis technology, from GP or elsewhere, they are highly skeptical. In contrast, digital designers are very open to automated-design techniques.

How do the claims of GP look, from a designer's perspective?

For starters, they are not shocked, even when they see the work on reinvented patents [Koza2003]. With every other structural synthesis technology reported until now, *something* was missing, which limited its widespread industrial use. Despite their limited understanding of GP, designers have no real reason to treat or trust GP differently. They simply believe that something's missing for GP too.

They are right. When an analog designer digs more deeply into the GP methodology for automated topology design, he will find problems. Some are obvious (to an analog designer), and some are subtle.

5.5.3 Performance Estimation and Circuit Robustness

In analog synthesis, circuit robustness is strongly related to performance estimation. A performance estimator takes in a sized topology, and estimates its performances. To achieve a robust design, the synthesis engine must use performance estimation that is as accurate as possible.

The ideal performance estimator would predict with 100% accuracy how a design performs after layout, manufacturing, and testing without actually fabricating it. Moreover, it would run quickly enough to be invoked thousands or millions of times throughout optimization, to allow through automated exploration of designs. SPICE [Nag1973, Nag1975] is the most accurate and general estimator, but there are also faster, less general, less accurate ones.

5.5.3.1 Environmental conditions

The manufactured chip will need to work at the desired performance level, even as temperatures change, power supply changes, and load changes. These are conditions of the circuit's operating environment.

5.5.3.2 Manufacturing variations

When manufacturing a VLSI circuit, random variations get introduced into the implementation of the designs as an inherent effect of the fabrication process. The automated tool must model this and handle it.

The simplest model is so-called "Fast/Slow corners", which in effect try to capture the 3-sigma extremes in each type of transistor's operating speed due to manufacturing variations. This approach is popular for its simplicity and availability. However, corners do not model the problem well, because they are designer to bracket variations in *digital* circuit performance, not analog circuit performance. Section 2.2.5 elaborates.

The approach [Pel1989] is historically the most popular approach to modeling local process variations (mismatch). In its formulation, mismatch between two devices is proportional to their distance, and inversely proportional to their $sqrt(W * L)$. There is about one random variable per device. The random variables are typically normally-distributed, and may be correlated.

Some approaches build empirically-based statistical models to estimate a probability density function, such as [Pow1994]. These models almost always make assumptions that render them inaccurate, for example assuming that certain random variables are independent when they are not, or ignoring local statistical variations as in [Alp2003].

The backpropagation-of-variance approach [Dre2003] uses a more physical basis for randomness modeling and is quite accurate. However, an implication is that for every transistor, ≥ 8 random variables are introduced. Therefore, a medium-sized circuit could have hundreds or thousands of random variables. The random variables are typically normal, independent, and identically distributed (NIID) [Box2005].

5.5.3.3 Layout issues

"Layout parasitics" are effects that are not accounted for prior to layout [Lam1999]. An example layout parasitic is when the material between two wires acts like a circuit component (e.g. a capacitor) rather than acting like the "ideal" open circuit. A parasitic-annotated netlist can be extracted from a layout, but it will have 10x to 100x more devices. This could take significantly longer simulation time.

As a further aggravation, layout parasitics are subject to process variations themselves. This dramatically increases the number of process variables, and therefore the effect that process variation has on performance.

5.5.3.4 Other Effects

As process nodes change and devices shrink to follow Moore's Law, new effects crop up and others become worse. Depending when an open-ended synthesis approach is de-

ployed, and for what applications, it may have to consider some of the following effects: proximity effects [Dre2006], electromagnetic compatibility (EMC) [Paul1992], aging/reliability effects, and more. A thorough, up-to-date description of issues can be found at the latest ITRS, e.g. [Itrs2007].

5.5.3.5 SPICE Can Lie

SPICE [Nag1973, Nag1975] will sometimes output results that are dramatically different than true silicon operation. This can be due to problems in its device models, in convergence of the SPICE solver, or perhaps inadequate models of parasitics. SPICE transistor models seem to be in a continually inadequate state, with known deficiencies, such as discontinuities from one operating region to another. Part of the difficulty is that the model structure must work for several processes. Models typically require *hundreds* of parameters that ideally are easy to extract. Extracting parameters for *statistical* SPICE models is particularly problematic.

Because of these model deficiencies, designers do not fully trust SPICE, and have tactics to avoid known problems. For example, they consciously avoid transistor operating regions where the models are known to be inadequate.

5.5.4 Robustness of Manually-Designed Topologies

Manually-designed topologies are almost always designed with robustness in mind. This section highlights how a manually-designed topology implicitly has this robustness. From another perspective, this section discusses what other robustness issues must be considered when doing open-ended synthesis of a topology.

5.5.4.1 Robustness in Manual Topology Design

We now examine what analog designers do to make topologies more robust. We will refer to a well-known circuit shown in Figure 5.12 (copied from Figure 4.8 for convenience).

The effect of "local" ("mismatch") variations within a chip has traditionally been smaller than "global" variations, which are between chips and between runs; traditionally having values of 1-2$\%$ vs. 10-20$\%$, respectively[1]. The main tactic to deal with global variations is to design structures in which performance is a function of *ratios of sizings*, rather than absolute values. For example, in common-source gain stages, a load resistor would have variation of 10-20$\%$. So, designers use a PMOS load instead, matched up to an NMOS gain transistor, and gain is dependent on the ratios (e.g. in Figure 5.12, M5a is a resistive load for M3a) [Lak1994, Raz2000, San2006].

Differential design is another tactic to move away from "absolute" signal values, which are susceptible to global variation. In differential design, "mirrors of structures" are created, and the circuit operates on a *difference* between two voltage/current signals, rather a single signal. The OTA in Figure 5.12 is symmetrical about a vertical axis centered on M5 and M7. The output is a function of the difference between the positive and negative inputs, nin_p and nin_n.

[1]Though for modern geometries $\leq$ 90nm, local and global variations are the same order of magnitude.

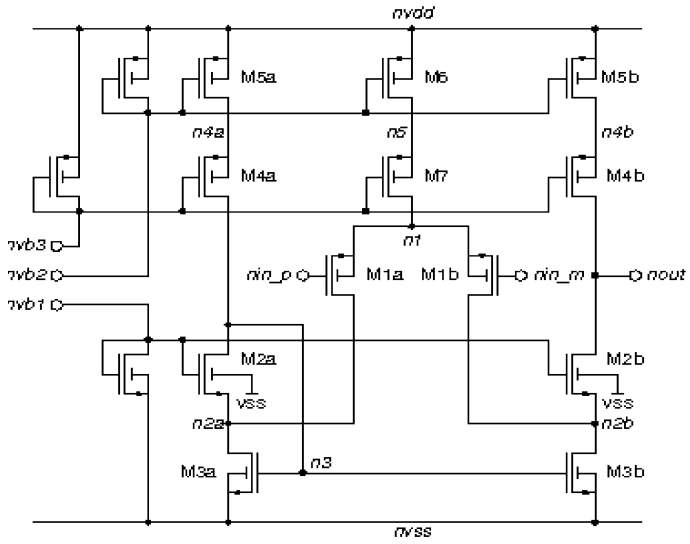


Figure 5.12: *CMOS high-speed OTA*

A precise current is expensive to generate; it's a much better idea to generate one or a few reference currents and copy them throughout the circuit with "current mirrors." The OTA in Figure 5.12 does this: the three transistors on the left are the "biasing" circuitry to generate currents, which are then copied throughout the circuit. Sometimes a single current can be shared, rather than trying to match two separate currents. The OTA's differential-pair devices (M1a and M1b) do this: instead of having different "tail" currents, M1a and M1b share the same current which goes through M6 and M7.

Negative feedback is a well-known general engineering technique for compromising some performance in the interest of precision. Analog circuits often use negative feedback, such as for improving the common-mode rejection ratio of a differential amplifier, or for reducing the variation of an amplifier's gain [Raz2000].

5.5.4.2 Trust and Re-Use

The topology is trusted because it (a) has been created and characterized by expert analog designer(s) based on logic and experience, and (b) has been fabricated and tested over many process generations. Topology re-use is widespread in industry because past success means more confidence that the topology will work. A new topology is typically a derivative of an existing topology, because similarity to known designs maintains trust. We will see that re-use plays a highly important role.

5.5.4.3 Device Operating Constraints

As discussed in section 2.2.9, topologies have device operating constraints (DOCs) based on the topology's design principles. Every transistor in the circuit has been designed with the assumption that it will be operating in a specific region. There is a good chance that the design assumptions do not hold in operating regions outside those constraints.

5.5.4.4 Clear Path To Layout

The designer knows that, for manually-designed topologies, there is a clear path to layout; to a large extent the designer has already anticipated the parasitics. Layout designers also have tactics to improve robustness, such as: folding transistors, guard rings, and careful routing to avoid cross-coupling between sensitive wires [Has2000, Lam1999]. Analog *layout* synthesis is another analog CAD subproblem [Rut2000]; it is difficult to model and solve well, as illustrated by continued research activity despite the availability of commercial tools like [Cdn2005a]. When layout parasitics are more pronounced, such as in RF design, there are ways to tighten the coupling between sizing and layout design [Van2001, Sme2003a, Zha2004, Bha2004].

To properly account for layout effects in synthesis, one possibility is to unite the front-end design space (topology and circuit sizes) with the back- end space (layout), and approach the whole problem at once, as in section 5.2 of [Koza2003]. Unfortunately, runtime was 50x slower than a GP run on an equivalent problem that did not consider layout. And, that work drastically simplified the layout synthesis problem – it did not even extract the parasitics from the layout before simulating the netlist.

5.5.4.5 Synthesis Exaggerates "Cheating" of Search Algorithms

We say a "cheat" occurs when design has good measured performances, but which upon inspection is useless (e.g. not physically realizable). An example is too many long, narrow transistors. The solution in that case is to add more constraints on width/length ratios. Each added constraint takes time to detect, correct, and re-run. There is dramatically more opportunity for structural synthesis to cheat compared to sizing optimization, because the open-ended topology design space is drastically larger, and there are more opportunities for SPICE to lie. Evolvable hardware research is filled with examples of odd designs; however, in non-reprogrammable or non-reconfigurable analog VLSI, one cannot embrace odd design results because of the high cost of fabrication.

5.5.5 An Updated Model of the Open-Ended Synthesis Problem

5.5.5.1 A Realistic Model

Most earlier GP structural open-ended synthesis work such as [Koza1999, Lohn1998, Zeb2002, Sri2002, Koza2003] did not have a very thorough model of the problem compared to analog CAD optimization, but some of it has been getting better recently. In [Koza2005], corners were added to account for environmental and (very roughly) manufacturing variations, and they employ testbenches directly from an industrial CAD vendor [Snps2005]. However, other recent research has not yet acknowledged the need for more robustness, e.g. [Das2005, Mat2007, Sap2008].

GP does not have DOCs, because it does not make assumptions about what region each transistor will operate in. GP actually has stronger performance measures in one regard: it also tries to match waveforms of behavior (i.e. minimize the difference between a target waveform and the candidate circuit's waveform).

Compared to analog CAD optimization work, after the trust issue, GP's biggest deficiency in problem modeling is its lack of a good model of manufacturing variations. The closest method presented is robust HFC [Hu2005a]. It did have Monte Carlo sampling, but the randomness model is not suitable for VLSI circuits.

Beyond analog CAD optimization, GP-evolved circuits must somehow get the same advantages as a manually-designed topology. Such circuits must get designer trust, including an explanation and formulae for behavior. Ultimately, successful fabrication and testing of GP-synthesized circuits is necessary. On the way, there are the hurdles of SPICE (mis)behavior, layout parasitics, search space cheats, and extra challenges from process variations.

5.5.5.2 Computational Challenges

Ultimately, the only way to accurately model manufacturing variations is via *simulation* on *accurate statistical models*. Let us examine the runtime of a typical structural open-ended synthesis run that uses brute force Monte Carlo sampling. Except for layout, we will temporarily ignore all the extra challenges wrought by a non-fixed topology.

Let us say: there are 8 corners (for environmental variations), 10 Monte Carlo samples (for manufacturing variations; 10 is optimistic), and assume a simulation time of 1 minute for a circuit at one corner and one sample on all testbenches on a 1 GHz machine. Parasitic-extracted layouts can easily lead to 10x longer simulation times. Larger designs and/or longer-than-transient analyses could easily take 6x, 60x, or even 600x longer to simulate.

It is typical for a GP run to explore 100 million designs for more challenging problems; 1 billion or even 10 billion would not be unreasonable [Koza2003]. But let us have 1,000 1-GHz machines in parallel.

Then, the total run time = 152 years! And it's even longer for tougher problems, where the simulation time is 6x-600x longer and the number of individuals is 10x-100x more.

One might ask if Moore's Law can ease this challenge.

5.5.5.3 Mooreware vs. Anti-Mooreware

GP is considered an example of "Mooreware" [Koza1999], where an algorithm becomes more effective with more computational power, and therefore with the march of Moore's Law [Moo1965] over time.

However, Moore's Law when attacking VLSI design problems is a double-edged sword. Each new technology generation also requires more modeling effort, and therefore more compute time! For example, the need for substrate noise modeling is growing; to model this takes 30 minutes on four modern processors [Soe2005], i.e. 120x more computational effort.

Thus, open-ended analog synthesis is an "Anti-Mooreware" problem: it gets more difficult as Moore's Law progresses. So, we cannot rely on the "Mooreware" aspect of GP to eventually be fast enough[1].

[1]While we cannot *rely* on the "Mooreware" aspect, GP *can* opportunustically take advantage of advances in computing that drive down the cost per flop, including multicore machines, massively multicore

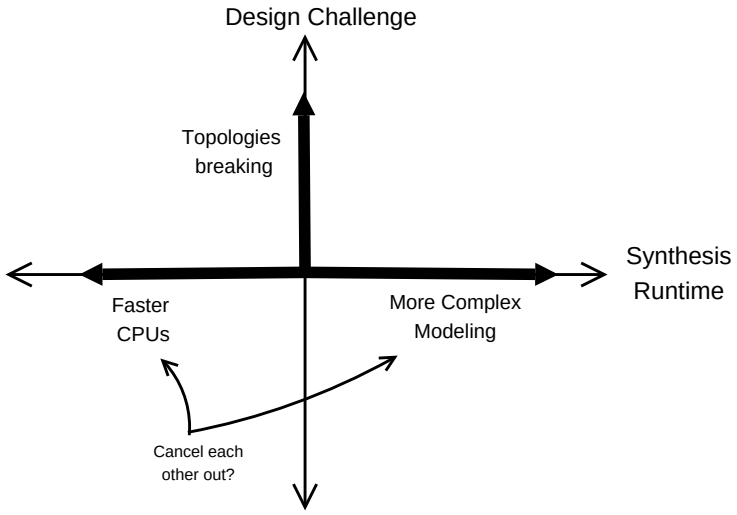


Figure 5.13: *Effects of Moore's Law on open-ended topoology synthesis.*

5.5.5.4 Moore's Law Breaks Topologies

Topologies are getting constrained in new ways due to Moore's Law. Here is an example. Supply voltages and threshold voltages are steadily decreasing, but threshold voltages cannot scale as quickly because of fundamental physical constants. At some point, "cascode" configurations, which stack two transistors on top of each other, are unusable (e.g. M4b and M5b in Figure 5.12 are in cascode). The alternatives are less ideal: folded cascodes mean larger power consumption, and extra stages mean slower speed and instability risk.

Figure 5.13 summarizes the effect of Moore's Law on open-ended topology synthesis.

5.6 Conclusion

This chapter examined different possible topology-sizing design flows (per sub-block), and their relation to techniques in industry and academia. Approaches came from both analog CAD and from genetic programming / evolvable hardware. This chapter discusses the pros and cons of each flow, and how they might handle system-level design via a hierarchical design methodology. Based on the analysis, the final recommended flows involved massively multi-topology sizing, used with either constraint satisfaction or multi-objective optimization.

After that, specific requirements for a topology selection / design tool were presented, with an eye towards industrial applicability.

Then, a special section was dedicated to explaining why open-ended topology synthesis is so problematic. In short, to get near the trustworthiness of other approaches, an open-ended approach would need 150 years on a 1000-node cluster of 1-GHz computers.

machines, cloud computing, and graphics processing units. GP can also opportunistically take advantage on the improvements in SPICE / simulation / analysis technology.

The key reason for this is that manually-designed analog circuits embed a lot of implicit knowledge about the problem, and it is very computationally expensive to automatically account for them.

The next chapters present MOJITO and its derivatives. MOJITO is a topology selection / design tool that has industrially-acceptable inputs and outputs, accuracy, and runtime. Derivatives leverage MOJITO to enable topology-performance knowledge extraction, and to accelerate design of novel topologies (MOJITO-N).

Chapter 6

Trustworthy Circuit Topology Synthesis: Design Flows and MOJITO Search Space

Maybe my caveman ancestors invented the wheel or something. I'm not sure.
–Brendan Fraser

6.1 Introduction

This chapter and the next present a design tool, called MOJITO, to aid the designer in the task of topology selection, design, and sizing. It does *trustworthy* multi-objective structural synthesis of analog circuits. MOJITO is composed of a search space (this chapter), and a search algorithm which traverses the space (next chapter). MOJITO defines a space of thousands of possible topologies via a hierarchically organized combination of designer-trusted analog building blocks, which can be found in analog circuit textbooks [Raz2000, San2006]. That is, they are *field-specific*, *pre-defined*, and *hierarchical* as discussed in chapter 1. Using these types of blocks overcomes several key issues: the issues of trust and runtime (issues in open-ended synthesis approaches), and a rich set of possible topologies (an issue in "flat" search-space CAD approaches).

6.1.1 Target Flow

MOJITO is a system for multi-objective and topology sizing [Mcc2007, Mcc2008a, Mcc2008c, Pal2008]. Its inputs and outputs are shown 6.1. MOJITO fits into the user flows for hierarchical, massively multi-topology sizing of Figures 5.7 and 5.8. In those figures, it implements the "auto multi-topology sizing block" as well as the "massive topologies DB" that feeds into the block. As hinted in the introductory chapter, MOJITO actually follows the generally applicable framework for GP in structural design, as described in Figure 6.2.

Before we proceed to describe the MOJITO search space, we first describe the payoff of using domain knowledge.

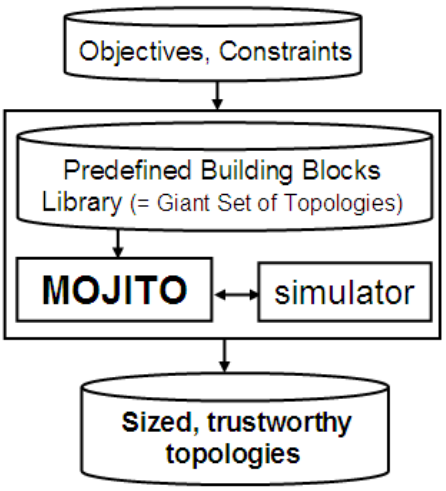


Figure 6.1: *MOJITO inputs and outputs.*

6.1.2 Background: The Power of Domain Knowledge

Domain knowledge, applied appropriately, can give dramatic improvement in the runtime, reduction in the search space size, or improvement in quality of results. For industrial applications, then speed and quality of results are of utmost importance. So, embedding domain knowledge can be well worth the up-front effort needed to capture the knowledge. We now give some illustrative examples from both evolutionary computation (EC) and analog CAD.

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. [Hor2003]
- permutation design via floating-point representations [Rot2006];
- avoiding "danglers" in circuit topology design e.g. [Koza2003]
- machine-code symbolic regression [Nor1994]
- machine-code digital logic design [Poli1999]
- avoiding the need for learning the linear weights in symbolic regression [Kei2004]
- thorough exploration of smaller building blocks, e.g. one variable at a time in symbolic regression [Kor2006]
- and many more.

In analog CAD, some examples on the payoff of domain knowledge include:

- 1,000,000x speedup when building behavioral models of circuits, by using knowledge of its connectivity [Phi1998]
- 1000x by exploiting sparsity in matrices [Lai2006]

- 100x space reduction via cheap-to-compute device operating constraints (DOCs) in circuits [Ding2005]
- 1,000,000x space reduction in [Ber2005] by reformulating the independent design variables of a design problem to more natural variables - i.e. operating point driven formulation [Leyn1998]
- and many more.
- in SPICE simulation, 10x-1000x or more speedup by selectively choosing when to do a full LU decomposition in corners analysis [Hu2008]

For non-trivial practical applications, using domain knowledge is therefore key. As we will see, this also applies to topology synthesis.

6.1.3 Reuse of Structural Domain Knowledge

In [Koza2004a], the authors 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, new circuit topologies and building blocks.

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 (as described above). For automotive design, GP would literally have to reinvent the wheel – and the piston, crankshaft, transmission, etc.

The authors of [Das2005] partially overcome this by supplying loosely grouped building blocks for possible use by the system. Unfortunately, this and past EAs have a tendency to exploit missing goals to return circuits with odd structures or behavior; this is a major issue because one must trust the topology sufficiently to commit millions of dollars to fabricate and test the design. Up-front constraints such as current-mode analysis [Sri2002] and Kirchoff's-Law constraints [Das2005] can be added, but plugging such "holes in goals" [Mcc2006d] is tedious and provide no guarantee that the circuit returned to the designer will be trustworthy enough to send for fabrication. Furthermore, the open-ended approach makes the EAs extremely computationally intensive, taking weeks or more of CPU time.

This chapter will show how the *appropriate* 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 6.2 illustrates the general approach to such problems. The first step is to determine an appropriate data structure to hold the knowledge, and then to take the effort to

put domain knowledge into that library. In our case, we use a parameterized grammar, then manually entered a library of analog circuit building blocks. The library is based on the large amount of structural knowledge that has accumulated in analog design over the decades [Raz2000, San2006]. With that data structure to guide the search, GP is run, subject to the constraints of the domain knowledge. In our case, since our domain knowledge is captured as a grammar, applying grammatically-constrained GP is appropriate. The results from the run can be used as guidance to add new blocks to the library, and the loop repeats.

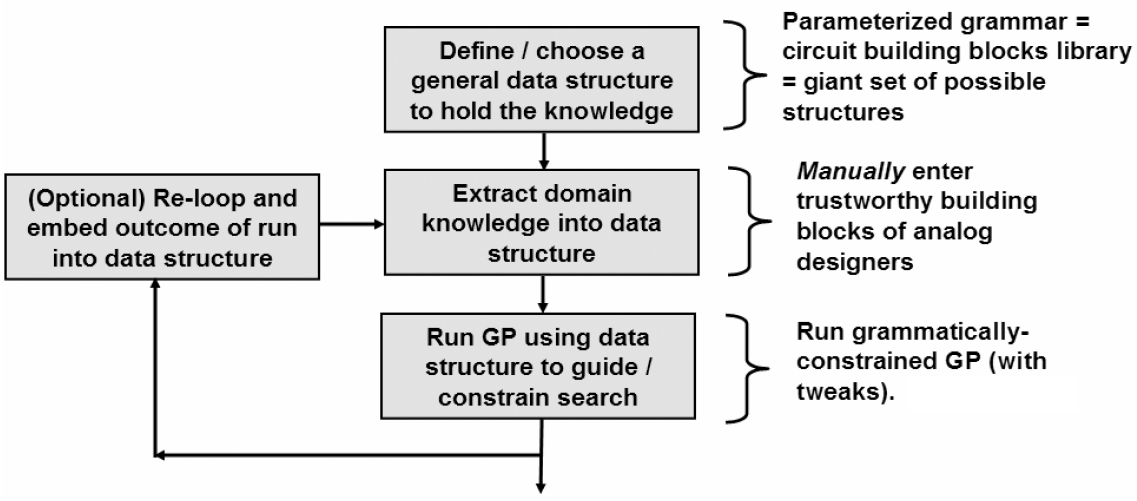


Figure 6.2: *A general framework to leverage domain-specific structural knowledge with GP. The instantiation of the framework for analog circuit topology synthesis, MOJITO, is described with the text on the right.*

6.2 Search Space Framework

This section describes the MOJITO topology space. This space 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 that has been thoughtfully designed for analog circuits. Any topology (sentence) drawn randomly from the grammar will be trustworthy *by construction*.

6.2.1 Search Space Framework I: Base

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 large library or to larger designs (as discussed in section 5.2.3). Analog circuit hierarchies can be represented by analog hardware description languages (HDLs) [Ash2002, Kun2004], by analog circuit database representations, and even by

grammars [Res1984, Tan1993]. With these options already existing in the analog domain, why not just use one of them? We actually could, but we choose to develop a notation that can address the following problem more directly. 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 [Kru1995, Mau1995] 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 [Hor2003] provided inspiration for our work. A generative representation transforms a genotype to a 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 "Block" types, which we now describe.

Let us define a "Block" as merely a circuit block at any level of the hierarchy. It has a fixed set of the 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 Block's embedded Blocks are a function of arguments above. To fully implement (netlist) a given Block, the Block only needs to be given values for its input arguments.

The block types are:

- **Atomic Block.** These are the leaf nodes in the building block hierarchy. Therefore, they do not contain any sub-blocks. It is only Atomic Blocks that appear on an implemented netlist. Figure 6.3 gives examples.
- **Compound Block.** A Compound Block holds a composition of sub-blocks. Sub-blocks can have internal connections among themselves and to the parent Compound Block's external ports. Figure 6.4 gives examples.
- **Flexible Block.** These hold several $alternative$ sub-blocks, where only one alternative is chosen during netlisting based on the value of the Flexible Block's $choice_i$ parameter. Each sub-block has its own choice of wiring as well. It is due to the different block possibilities in Flexible Blocks that enables a *library*. Figure 6.5 gives an example.

Despite having just the three simple types of blocks above, the blocks' combinations and interactions allow us capture of essential structural domain knowledge of analog circuits. The Flexible Blocks are what turn a Block 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 of

the top-level block.

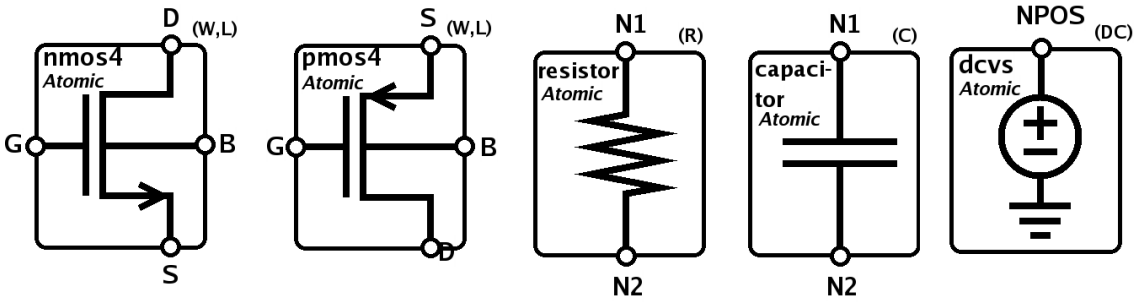


Figure 6.3: *Example Atomic Blocks: nmos4 transistor, pmos4 transistor, resistor, capacitor, dc-controlled voltage source (dcvs).*

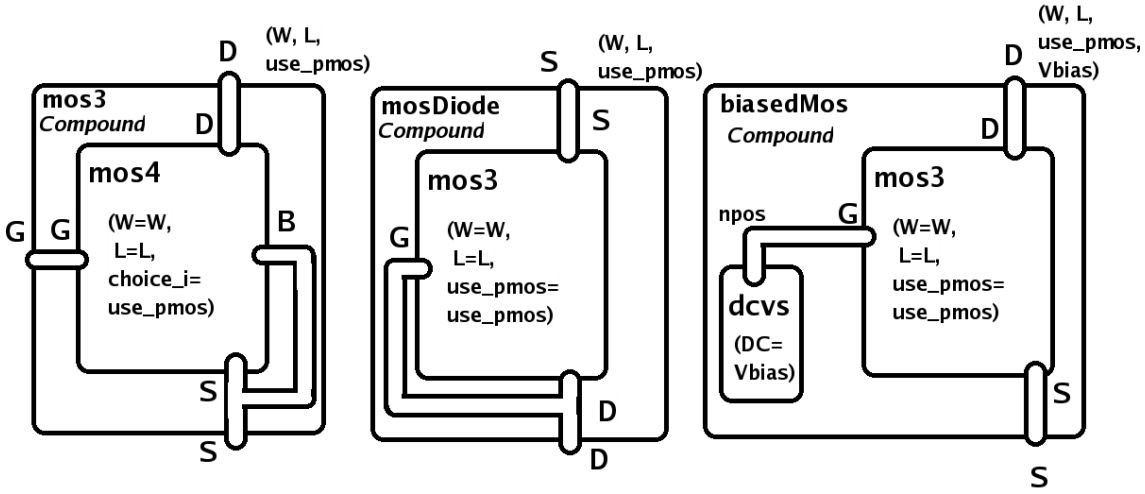


Figure 6.4: *Example Compound blocks: nmos3, mosDiode, biasedMos.*

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 from the top-level block through the hierarchy to the leaves. 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 its subblocks, which keeps the overall variable count lower than it might have been. For example, devices along a current branch share the same I parameter. This is crucial to the locality [Rot2006] of the space, where small changes to the genotype lead to expected small changes in the objective function. Locality is crucial to the ultimate success of the search algorithm.

In Figure 6.3, the *nmos4* block has four external ports: *G*, *D*, *S*, and *B*. It has two input parameters, *W* and *L*, as shown in the part's top right. The other Atomic Blocks are similar. Note how the *dcvs* block (DC-controlled voltage source) has only one external port; the other port ties directly to ground. This makes it convenient to parameterize the

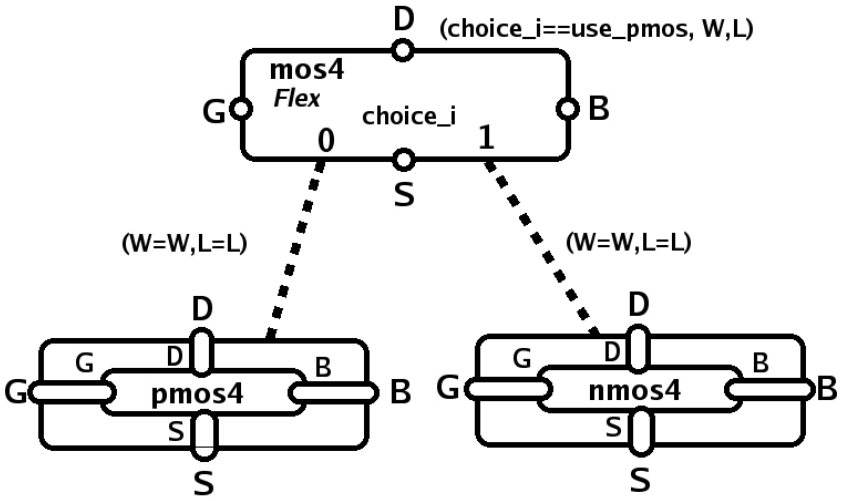


Figure 6.5: *Example Flex Block: mos4 turns the choice of NMOS vs. PMOS into a parameter* "choice_i".

biases of other blocks. The top-right of each block shows the block's input parameters: (W,L) for *nmos4*'s width and length, (W,L) for *pmos4*, (R) for *resistor*'s resistance, (C) for *capacitor*'s capacitance, and (DC) for *dcvs*'s voltage value. These values are used directly within the device's instantiation in the SPICE netlist.

Figure 6.4 shows three example Compound Blocks: *mos3*, *mosDiode*, and *biasedMos*. *mos3* only contains one sub-part, a *mos4*, but it hides the *mos4*'s*B* terminal by tying it to the *mos4*'s *S* terminal, reflecting the fact that designers commonly conceptually work with transistors having three terminals, not four. The *mosDiode* has a similar tying mechanism. The *biasedMos* reflects the common designer approach of deferring work on biasing circuitry to focus first on signal circuitry. Like Atomic Blocks, the top-right of each block shows that block's input parameters. For example, *mos3* takes W, L, use_pmos as inputs. Those inputs get propagated down to each sub-block via the notation inside each sub-block. For example, within *mos3*, its *mos4* sub-block gets its *W* parameter set to the *mos3*'s *W* value (W=W); the same for L (L=L); and the *mos4*'s parameter *choice_i* set to the *mos3*'s *use_pmos* value.

Figure 6.5 shows the *mos3* Flexible Block, which instantiates into an *nmos4* block if *choice_i* = 0, or a *pmos4* block if *choice_i* = 1. *mos3*'s *W* and *L* sizing parameters propagate directly to*nmos4*'s and *pmos4*'s *W* and *L* parameters in a unity mapping of "(W=W, L=L)". Of course, more complex mappings can be used. For example, higher-level blocks set the *is_pmos* variable depending on the block's context in the hierarchy, which will propagate to the bottom to use *nmos4* vs. *pmos4*.

We have shown how larger blocks can be 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 (op amp). Despite the simplicity of such blocks, the combination of block types, especially Flexible Blocks, means that a given block is its own *library of possible topologies*. A block's search space is merely combinations of the possible values that each parameter in the block can take. The example library for op

amps is presented as a whole in Figure 6.10, and will be elaborated in subsequent sections. The library can readily be specified in an analog HDL such as VHDL-AMS [Ash2002] or Verilog-AMS [Kun2004], a hierarchical circuit schematic editor, or a programming language (we use Python [Pyt2008]). The library for opamps only needs to be developed once, and this is it. This library already implicitly includes the libraries for all its sub-parts, such as cascode stages and current mirrors. To develop libraries for larger blocks, this library can be extended upwards.

6.2.2 Search Space Framework II: On "Small" Changes

We already briefly mentioned *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 [Rot2006]. 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 performance, there may still be a *dramatic* change in the netlist (phenotype).

We will now give an example. Figures 6.6 to 6.9 show schematics with only small conceptual differences, and similar behavior / performance. However, as we see in the figures, they have *very* different schematics / netlists / phenotypes. There are many other examples in analog circuit design, such as "folding" an input stage, flipping all NMOS transistors to PMOS and vice versa, and many more in analog textbooks like [Joh1997, Raz2000, San2006].

Past open-ended synthesis approach do not cover these at all, because their respective representations do not capture design intent, they just capture the structural information, i.e. one particular instance of a netlist. For example, a GP-synthesized tree does not have an explicit means to convert among the schematics of Figures 6.6 to 6.9 via small changes to individuals' genotypes. Past trustworthy synthesis approaches only cover *some* of the examples. For example, [Mau1995] could flip NMOS/PMOS transistors by changing a single bit in the genotype. However, each of the past trustworthy approaches only covered a small fraction of these possible transformations. For example, none of them captured the relation among Figures 6.6 to 6.9. A core reason is that the "flat" representation makes it difficult to do so; for example Figures 6.6 to 6.9 have *interactions* with NMOS/PMOS flipping that are hard to capture in a flat fashion.

This characteristic complicates the design of an appropriate search representation. We need to capture the domain knowledge such that small conceptual changes lead to small changes in performance, despite possibly dramatic changes in the netlist. That is, in the mappings of genotype → phenotype → performance: genotype → phenotype is sometimes *large* when genotype → performance → is *small* (due to small change in design intent).

The framework of Atomic, Compound, and Flexible Blocks can handle this challenge. It leverages the Flexible Block's *choice_i* parameter, which can be a *function* of one or more higher-level parameters, and choose between sub-blocks that are identical except how those sub-blocks are wired to their parent block.

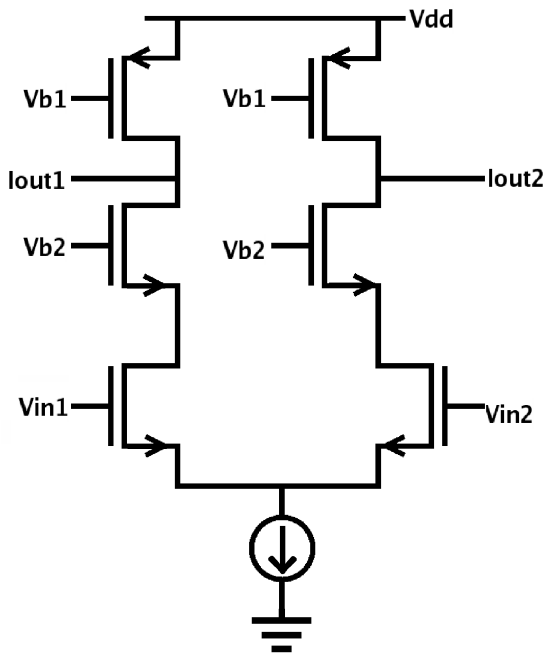


Figure 6.6: *Schematic:* $input\_is\_PMOS$ = *False,* $loadrail\_is\_vdd$ = *True.*

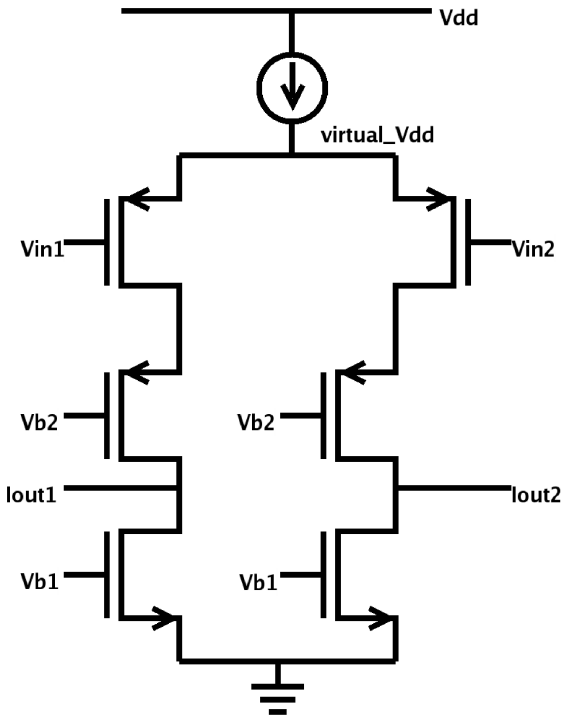


Figure 6.7: *Schematic:* $input\_is\_PMOS$ = *True,* $loadrail\_is\_vdd$ = *False.*

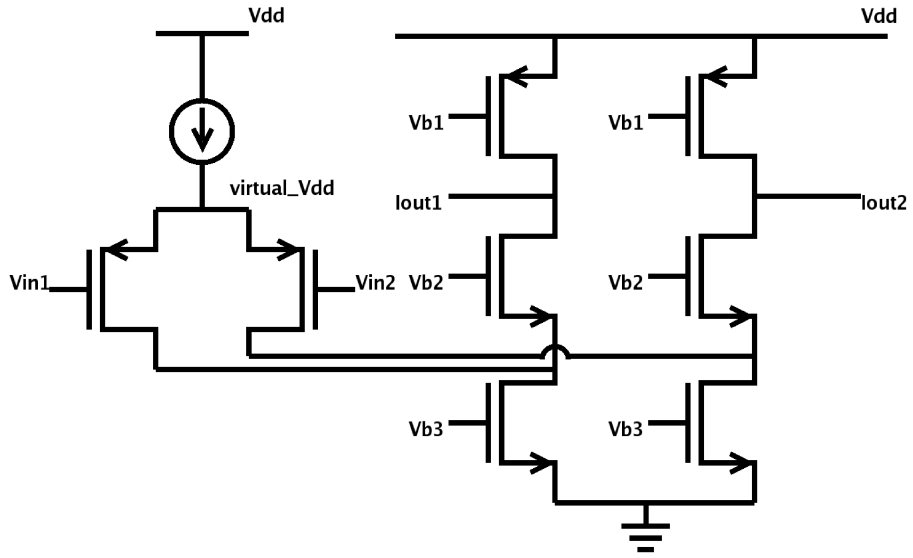


Figure 6.8: *Schematic:* $input\_is\_PMOS$ = *True,* $loadrail\_is\_vdd$ = *True.*

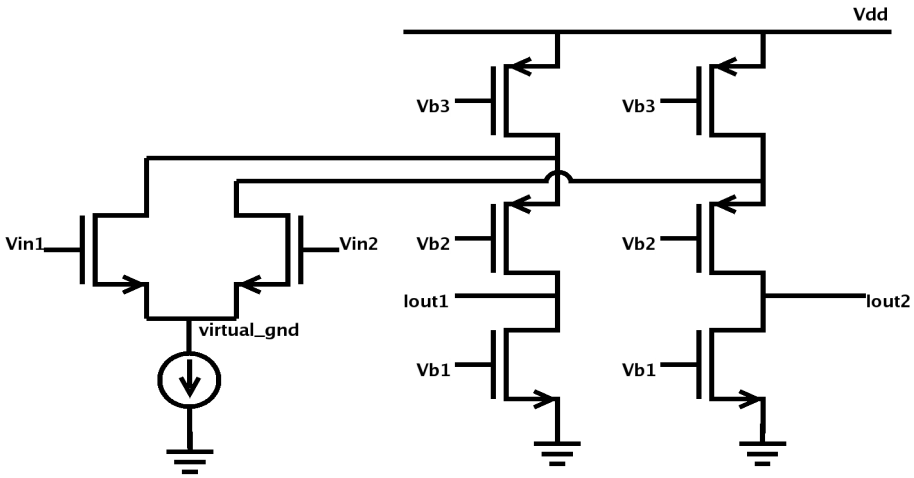


Figure 6.9: *Schematic:* $input\_is\_PMOS$ = *False,* $loadrail\_is\_vdd$ = *False.*

As an example, we now describe how the framework handles Figures 6.6 to 6.9. Near the top of the blocks 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*). Figures 6.6 to 6.9 show how the two top-level parameters of *input_is_pmos* and *loadrail_is_vdd* translate into four different schematics.

While all four schematics all have similar *conceptual* behavior, and similar building blocks, they have very different final topologies. The library has captured this: it is only a small change in a given circuit design to move from one design to another (changing the value of *loadrail_is_vdd* and/or *input_is_pmos*).

The MOJITO space has some relation to *generative* representation like L-systems [Lin1968] or GENRE [Hor2003] because creation of the phenotype (netlist) is performed by running an algorithm defined by the genotype (design point). However, whereas GENRE dramatically changes the structural search space during a run via operations on the grammar itself, MOJITO searches through a gramatically-constrained structural space. Furthermore, MOJITO diverts significant search effort to refining structures' parameters (device sizes and biases).

6.3 A Highly Searchable Op Amp Library

This section describes the example library that we developed for operational amplifiers. It is shown as a whole in Figure 6.10. $\approx$ 30 blocks combine to allow $\approx$ 3500 different topologies. It 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, and single-ended and differential inputs.

In the Figure, each box is a building block. The expansions due to Flexible Blocks are denoted by OR operators. Compound Blocks get expanded via AND operators. Blocks with "**:**" underneath get defined elsewhere in the diagram. Atomic Blocks comprise the leaf blocks.

The root node is the "ds amp vdd/gnd ports" block at the top left. "ds" means differential-input, single-ended output. It can expand into either a "ds amp1 vdd/gnd ports" block, or a "ds amp2 vdd/gnd ports" block, i.e. either a 1-stage or 2-stage ds amplifier. The "ds amp2 vdd/gnd ports" block holds a 1-stage ds amp (for the first stage), a 1-stage ss amp (for the second stage), and a capacitor (for feedback). All blocks with "vdd/gnd ports" means that the rails have been resolved to connect to power (vdd) and ground (gnd). They all have a *loadrail_is_vdd* parameter as described in section 6.2.2. Correspondingly, the block "ds amp1 vdd/gnd ports" can expand into a "ds amp1" in one of two ways: with its load rail connected to vdd, or with its load rail connected to gnd. The block "ss amp1 vdd/gnd ports" expands similarly.

The library includes: 3 current mirror choices, 2 level shifter choices (one choice is a wire); 2 choices of how to allocate vdd/gnd ports for a 1-stage amplifier and 4 for a 2-stage amplifier; 3 source-degeneration choices; 3 single-ended load choices; and more.

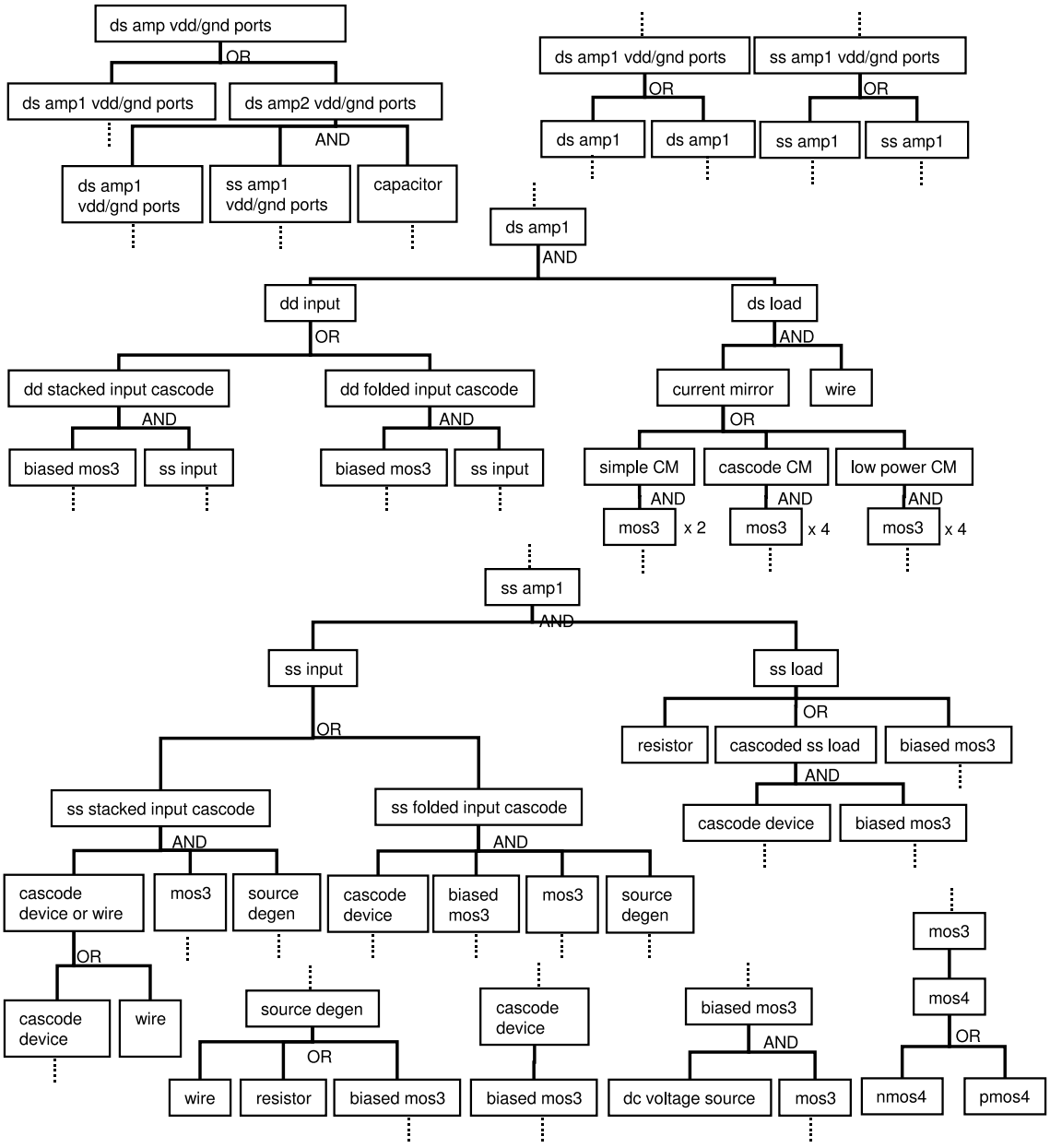


Figure 6.10: *MOJITO op-amp building blocks library. About 30 building blocks are hierarchically composed to form ≈ 3500 different topologies.*

Instantiation into NMOS vs. PMOS devices is deferred until the very leaf blocks, as a function of the parameters that flow through the hierarchy from the root node to leaf nodes. This flexibility allows for a large number of possible topologies, without having an excess number of building blocks in the library. It also means that many parameters are shared in the conversion from one block to subblocks, which preserves locality of the search space mapping, and keeps total variable count low.

The library can be readily modified for other problems by building up different blocks and/or using a different block as the root node. For example, search of a single-ended input, single-stage opamp would use "ss amp1" as the root node; or search for a digital cell might use a library that builds off "mos3" blocks.

This is an example library that has *field-specific*, *pre-defined*, and *hierarchical* building blocks, as discussed in chapter 1. It is this combination of traits which enables a rich search space of trustworthy-by-construction topologies.

6.4 Operating-Point Driven Formulation

MOJITO uses an operating-point driven formulation [Leyn1998] in which bias current (I) or voltage (V), and transistor length (L), are the independent variables, rather than transistor width (W) and L. Its advantages are that designable variables have less nonlinear coupling than a W/L (sizing) formulation; and that one can have device operating constraints (DOCs)(section 2.2.9) in which the DOCs can be measured by simple function calculations on design variable values without need for circuit simulation.

To implement it, we need to compute W from device biases (I's and V's), for each device of each candidate design. First- or second- order equations are too inaccurate, and SPICE in the loop, per device, is too slow. So we sampled 350,000 points in the $\{L, I_{ds}, V_{bs}, V_{ds}, V_{gs}\}$ space, SPICE simulated each point once on an NMOS and once on a PMOS BSIM3 model, then stored all the points in a lookup table (LUT). Therefore, during a MOJITO run we can directly compute W's from biases, accurately and with no extra simulations in the given technology process.

6.5 Worked Example

The search space is a library of circuit building blocks, and is equivalently a parameterized grammar. Therefore, a point in the search space is a circuit, and equivalently is a sentence. In EA terms, it is an individual. This section illustrates the different ways one can view an individual, with emphasis on the individual's structure.

Let us start with the view that is both concrete and intuitive: the schematic. This is the view that is most natural to analog designers. Figure 6.11 is an example individual shown in schematic form. From an analog designer perspective, it is a "PMOS-input Miller OTA" with these characteristics:

- It is a two-stage amp, using a simple capacitor for feedback / Miller compensation.

- The first amplification stage has differential input and single-ended output. The input devices are PMOS. There is no cascode-folding or source degeneration. A simple current mirror, attached to the ground rail, reflects the signal and serves as load.
- The second stage has single-ended input and output. The input device is NMOS. A single PMOS device serves as load. There is no source degeneration or cascoding.

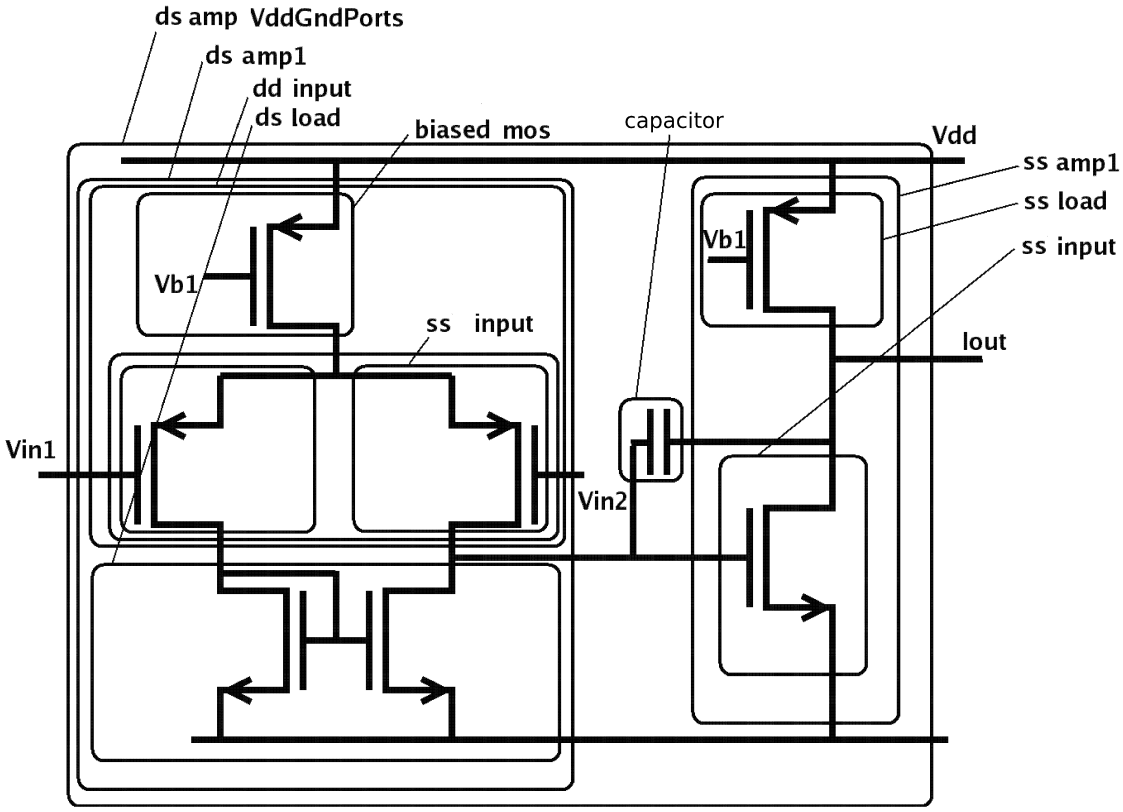


Figure 6.11: *An example individual ("PMOS-input Miller OTA") shown in schematic form, annotated with MOJITO Building Blocks.*

The schematic of Figure 6.11 is *annotated* to illustrate its relation to the building blocks library. Each box that outlines a set of devices is an instantiation of a building block. The whole design is the grammar's root node "ds amp VddGndPorts", as indicated in the schematic's top left corner. "ds amp VddGndPorts" is composed of three sub-blocks: "ds amp1", "capacitor", and "ss amp1", which are all labeled on the schematic. In analog design terms, the OTA is composed of: a differential-in, single-ended-out input stage ("ds amp1"), a single-ended-in, single-ended-out output stage ("ss amp 1"), and a Miller feedback capacitor.

The "capacitor" Block is a non-divisible Atomic Block, but the other two sub-blocks *do* subdivide further as stages 1 and 2 of the amp. The "ds amp1"subdivides into "dd input" and "ds load" blocks, and those keep subdividing until eventually they hit "nmos4" and "pmos4" Atomic Blocks. The "ss amp1" block on the right subdivides into an "ss load" and "ss input" block, which also keep subdividing until they eventually hit "nmos4"

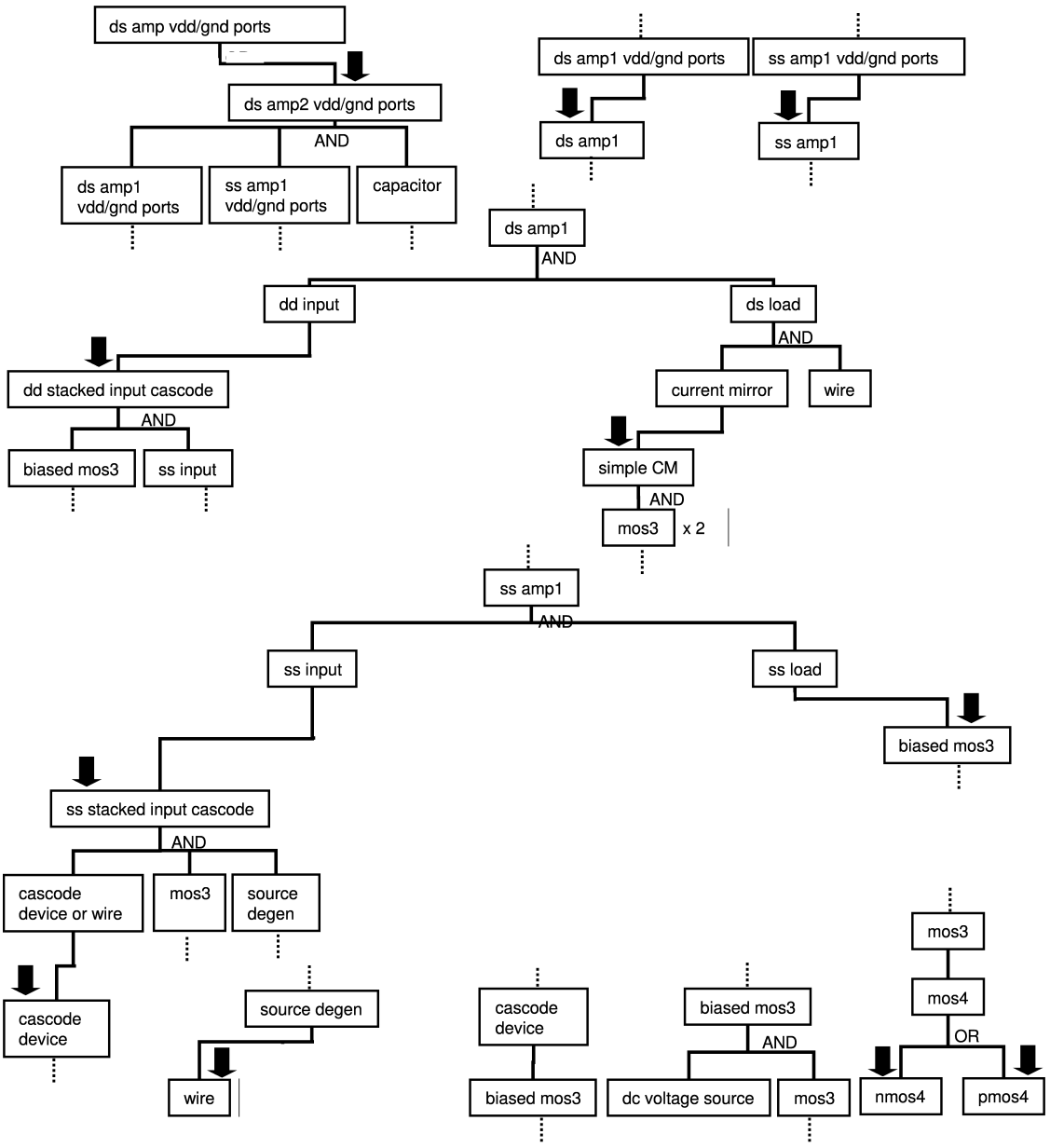


Figure 6.12: *Choices made in library for the schematic of Figure 6.11.*

and "pmos4" Atomic Blocks. (Note: For ease of understanding, not all intermediate subblocks are highlighted.)

We just described the subdivision of an individual from a schematic-centric view. One can also take a view that is centered around the library/grammar itself, as shown in Figure 6.12 for the same individual. This view contains a subset of the library of Figure 6.10: where at each Flexible Block's "OR" option, a single choice is made and the non-chosen options are not shown. Arrows indicate where choices were made.

Starting at the root node "ds amp vdd/gnd ports" in the top left, we see that the first choice was to "ds amp2 vdd/gnd ports" (and not to "ds amp1 vdd/gnd ports", which is not shown). That is, a two-stage OTA was chosen over a single-stage OTA.

Then, the "ds amp2 vdd/gnd ports" block expands into three sub-blocks: "ds amp1", "capacitor", and "ss amp1", just like the expansion shown in the schematic (Figure 6.11). "ds amp1 vdd/gnd ports" makes the choice "ds amp1", as shown in top middle. Its alternative was actually also "ds amp1", but the difference between the choices is how the *vdd* and *gnd* ports are allocated – specifically, whether the load goes to the *vdd* or the *gnd* rail, as discussed in section 6.2.2. We know from the schematic that the input stage's load goes to the *gnd* rail, therefore the stage's value *loadrail_is_vdd* will be False. Similarly, "ss amp1 vdd/gnd ports" takes one of the "ss amp1" choices, with its difference based on the allocation of *vdd* and *gnd* ports. In this case, *loadrail_is_vdd* will be True.

In the very center of Figure 6.12, the "ds amp1" block subdivides into the "dd input" and "ds load" blocks, akin to what we saw in the schematic's input stage (Figure 6.11 left). In middle-left, "dd input" takes the further choice "dd stacked input cascode" (vs. "dd folded input cascode"), which subdivides into "biased mos3" and "ss input" blocks. The "ds load" block subdivides into a "current mirror" and "wire" block. In middle-right, the "current mirror" takes the choice "simple CM" (vs. "cascode CM" or "low power CM"). This is the current mirror in the bottom left of the schematic in Figure 6.11. The "simple CM" subdivides into two "mos3" Blocks.

The individual's "ss amp1" block subdivides further, as shown in the bottom half of Figure 6.12, corresponding to the output stage of the schematic in Figure 6.11. "ss amp1" divides into "ss input" and "ss load". The specific "ss input" chosen is "ss stacked input cascode" (vs. "ss folded input cascode"), which itself is composed of "cascode device or wire", "mos 3", and "source degen".

The block subdivisions continue via "AND" combinations and "OR" arrow-choices, until finally Atomic Block leaf nodes are reached. Since this individual has both NMOS and PMOS devices, then the arrows for "nmos4" and "pmos4" are both shown in Figure 6.12.

An individual is represented within the synthesis engine's code in a vector-valued fashion. This is its genotype; all the other representations can be computed from the genotype. Specifically, the vector representation is an unordered mapping from variable names to corresponding chosen value. The variables are the variables needed to instantiate the root block, "ds amp vdd/gnd ports". Some variables are for topology choices (*choice_index*), and others are for setting specific device values (I's and V's which translate to W's and L's). Tables 6.1 and 6.2 gives the example individual's topology choice values and device-setting values, respectively.

Table 6.1: *Example Individual: Values for Topology Choice Variables.*

Variable Name	Value	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

Each parameter in Table 6.1 relates to one of the Flexible Block "OR" choices in the library of Figure 6.10. Eand each specific parameter *value* reflects the specific "OR" choice made as shown in Figure 6.12. The parameter *chosen_part_index* decides between one and two stages. A value of 1 means two stages were chosen, confirmed by the two-stage schematic in Figure 6.11. *stage1_loadrail_is_vdd* = 0 means that stage 1's loadrail is not set to *vdd*, but to *gnd* instead, as we already saw. *stage1_input_is_pmos* = 1 means that stage 1's input is PMOS, not NMOS, as we already saw. And so on. Note that some variables may be ignored, depending on values of other variables. For example, if *chosen_part_index*=0 to choose a one-stage topology, then all variables related to the second stage will be ignored. (Section 7.4 discusses this further.)

Table 6.2 gives example device-setting values. These are all parameters which do not affect the topology. Because we employ an operating-point driven formulation (section 6.4), the parameters are *I*'s and *V*'s, not *W*'s and *L*'s. As discussed in section 6.4, the *I*'s and *V*'s get translated into *W*'s and *L*'s at the level of NMOS4 and PMOS4 netlisting, using a lookup table.

A final view of an individual is the SPICE netlist. It is merely a text-based listing of each device's connections, type, and parameters in a line-by-line fashion. This is the form used as input to SPICE simulation, to estimate the individual's performance values.

6.6 Size of Search Space

This section examines how large a fully trustworthy (reuse-only, no novelty) space can become in terms of number of possible topologies, and what that means from the designer's perspective.

6.6.1 Size of Op Amp Space

We first ask: 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

Table 6.2: *Example Individual: Values for Device-Value Variables.*

Variable Name	Value	Variable Name	Value
feedback_C	1.1883e-11	*stage1_Ibias*	0.016297
stage1_Ibias2	0.015746	*stage1_Vds_internal*	1.4584
stage1_Vout	1.6830	*stage1_ampmos_L*	2.1885e-06
stage1_degen_fracDeg	0.86943	*stage1_folder_L*	3.5032e-05
stage1_folder_Vgs	1.2074	*stage1_fracAmp*	0.52026
stage1_fracVgnd	0.47741	*stage1_inputbias_L*	3.9621e-05
stage1_inputbias_Vgs	1.3419	*stage1_inputcascode_L*	2.7729e-05
stage1_inputcascode_Vgs	1.5280	*stage1_load_L*	1.9609e-05
stage1_load_cascode_L	4.0927e-05	*stage1_load_cascode_Vgs*	1.2094
stage1_load_fracIn	0.70442	*stage1_load_fracOut*	0.03599
stage2_Ibias	0.0087295	*stage2_Ibias2*	0.01206
stage2_ampmos_L	3.8062e-05	*stage2_ampmos_Vgs*	1.2756
stage2_ampmos_fracAmp	0.54704	*stage2_degen_fracDeg*	0.05280
stage2_inputbias_L	4.4781e-05	*stage2_inputbias_Vgs*	1.1022
stage2_inputcascode_L	2.2396e-05	*stage2_inputcascode_Vgs*	1.4486
stage2_load_L	9.7247e-06	*stage2_load_Vgs*	1.5154
stage2_load_fracLoad	0.19648	*stage2_loadcascode_L*	6.1258e-06
stage2_loadcascode_Vgs	0.92834		

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 do not overcount. Example: a two-transistor current mirror should have two choices (nmos vs. pmos), not four (NMOS vs. PMOS x 2).

Using the above method, the size of the MOJITO opamp library is 3528 topologies. Table 6.3 summarizes this and compares it to the other approaches.

Table 6.3 shows that MOJITO's flexible hierarchical nature increases the number of possible trustworthy op amp topologies to increase by 50x compared to [Kru1995, Mau1995]. All the approaches have an industrially palatable amount of effort for constructing the library: GP does not have a library, DARWIN and MINLP each have one big "flat" block, and MOJITO has about 30 small, hierarchically composed blocks.

Table 6.3: *Size of Op Amp Topology Spaces.*

Technique	# topologies	Trustworthy?
GP without reuse, e.g. [Koza2003]	billions	NO
DARWIN [Kru1995]	24	YES
MINLP [Mau1995]	64	YES
GP with reuse: MOJITO (this work)	**3528**	**YES**

Having such a rich set of options can qualitatively change the designer's perception of the process: rather than doing *selection* from a few dozen topologies, the tool is *synthe-*

sizing the optimal combination of building blocks from a huge set of possibilities. Since the library only needs to be defined once for a given problem type (e.g. op amp), the designer no longer needs to view it as an input, even if the process node changes. Figure 6.1 illustrates this, where the library is hidden *within* the tool, and the key input for the user is merely objectives / constraints. Once a MOJITO run is complete for a given process node, the database of Pareto-optimal sized topologies can be queried for the solution to a given set of specs in the same technology node. This therefore supports a designer workflow "specs-in sized-topology-out" with *immediate* turnaround.

6.6.2 How Rich Can A Search Space Be?

The second major question of this section 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* the development of new topologies and circuit 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 [Itrs2007]. Each design technique advance would increase the size of the space by at $\approx$ 2x, so if we merely took the top 10 advances in op amp design, we would increase the space by $\approx 2^{10} = 10^3$, 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 a total of $2 * 10^{17}$ trustworthy op amp topologies.[1]

- **Recursion.** Circuits' designs can recurse. For op amps, this is for instance 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 $(4 * 10^{34})^2 = 1.6 * 10^{69}$ trustworthy op amp topologies. 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 the 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

[1] This does not account for the subtleties discussed in section 6.6 so it may be slightly optimistic, but the point is clear: adding just a few more blocks can *dramatically* increase the number of possible topologies

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 blocks all with about the same number of topologies; then its size would be $(4.2 * 10^{553})^4 = 3.1 * 10^{2214}$ possible topologies. (For reference, if there are just 3528 topologies 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* block type, the more possible trustworthy topologies that the overall library contains. One can do system-level trustworthy topology design in spaces with 10^{113} topologies, 10^{832} topologies, or more, if: (a) If one can decompose their design into sub-problems (where each sub-problem has its own goals), (b) if one has a competent hierarchical design methodology, and (c) if the problem of "massively multi-topology" cell-level sizing design can be solved. We *can* do (a) because 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) [Raz2000, San2006]. We can do (b) because competent hierarchical design methodologies have been demonstrated. In fact, they have recently demonstrated the ability to choose among different candidate topologies [Eec2007]. This chapter and the next demonstrate (c). All conditions (a)-(c) are satisfied, so therefore system-level trustworthy topology design in spaces with 10^{832} topologies is achievable.

6.7 Conclusion

This chapter has presented a search space framework that is simultaneously trustworthy-by-construction, yet with a massive set of possible topologies. Such a rich set of options topology 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.

Whereas this chapter described the search space for topology design, the next chapter describes the search *algorithm* for topology design. Together, they comprise MOJITO [Mcc2007].

Chapter 7

Trustworthy Circuit Topology Synthesis: MOJITO Search Algorithm

Natural selection is a mechanism for generating an exceedingly high degree of improbability.
–Ronald A. Fisher

7.1 Introduction

This is the second of two chapters describing MOJITO, a design aid for analog circuit topology selection and design [Mcc2007, Mcc2008a, Mcc2008c, Pal2008]. While the previous chapter described the MOJITO search space, this chapter describes the MOJITO search algorithm which traverses the space, along with the MOJITO experimental results.

Even with the well-structured search space described in the previous chapter, there is a need for a highly competent search algorithm because the space is so large, and the performance estimation time for an individual can be on the order of minutes (using SPICE [Nag1973, Nag1975] to maintain industrial relevance, accuracy, and flexibility with technology nodes). We also need multi-objective results. Some degree of parallel computing is allowed – industrial setups for automated sizing typically have a cluster of 5-30 CPUs.

The MOJITO search method we implemented is a multi-objective evolutionary algorithm (EA) [Deb2002] that uses an age-layered population structure (ALPS) [Hor2006] to balance exploration vs. exploitation, with operators that make the search space a hybrid between vector-based and tree-based representations [Mic1998, Rot2006]. A scheme employing average ranking on Pareto fronts (ARF) [Cor2007] is used to handle a high number of objectives. Good initial topology sizings are quickly generated via multi-gate constraint satisfaction. Aggressive reuse of known designs brings orders-of-magnitude reduction in computational effort, and simultaneously resolves trust issues for the synthesized designs, for genetic programming applied to automated structural design.

MOJITO's combined multi-objective and multi-topology nature means that it generates a new type of database: Pareto-optimal sized circuits with many different topologies. Once this database is generated at a given process node, it can be reused for an immediate-

turnaround "specs-in, sized-topology-out" designer flow in the same process node. The database also opens up the opportunity for new circuit design insights via data-mining, as the next chapter will explore.

While the application here is analog circuit design, the methodology is general enough for many other problem domains, from biology to automotive design.

7.1.1 Problem Formulation

The algorithm's aim is formulated as a constrained multiobjective optimization problem:

$$\begin{array}{rll} minimize & f_i(\phi) & i = 1..N_f \\ s.t. & g_j(\phi) \leq 0 & j = 1..N_g \\ & h_k(\phi) = 0 & k = 1..N_h \\ & \phi \in \Phi & \end{array} \tag{7.1}$$

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 *non-dominated*, 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 $\{f_i(\phi_a) < f_i(\phi_b)\}\exists i$. We follow the dominance rules of NSGA-II [Deb2002] 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.

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

7.2 High-Level Algorithm

We use an evolutionary algorithm (EA) as the base of the search algorithm because EAs can readily incorporate our hybrid tree/vector representation [Mic1998], perform constrained multi-objective optimization [Deb2002], naturally support parallel processing [Can2000], and offer flexibility in overall algorithm design [Mic1998, Gol1989].

A key issue with most EAs is premature convergence. That is, the algorithm converges to a local region of the design space too early in the search without having explored the global space sufficiently, and returns sub-optimal results. This is certainly an issue in multi-topology optimization because some sub-blocks may get little chance to size properly before being filtered out via selection. We need to ensure an adequate supply of building blocks [Gol2002]. Possible tactics include massive populations [Koza1999, Koza2003], restarting e.g. [Aug2005], or diversity measures like crowding e.g. [Deb2002]; all tactics are painful or inadequate [Hor2006]. The algorithm could periodically inject randomly-drawn individuals, which contain new building blocks. However, in a typical EA, such new individuals would compete against the prior individuals which had

been evolving. Therefore the new individuals would almost always die off quickly. To give randomly-drawn individuals a chance, the technique of hierarchical fair competition (HFC) [Hu2005b] segregates individuals into layers of similar fitness, and restricts competition to within layers. Unfortunately, near-stagnation can occur at some fitness levels because the best individuals per level have no competition. Also, it is unclear what each layer's fitness range should be.

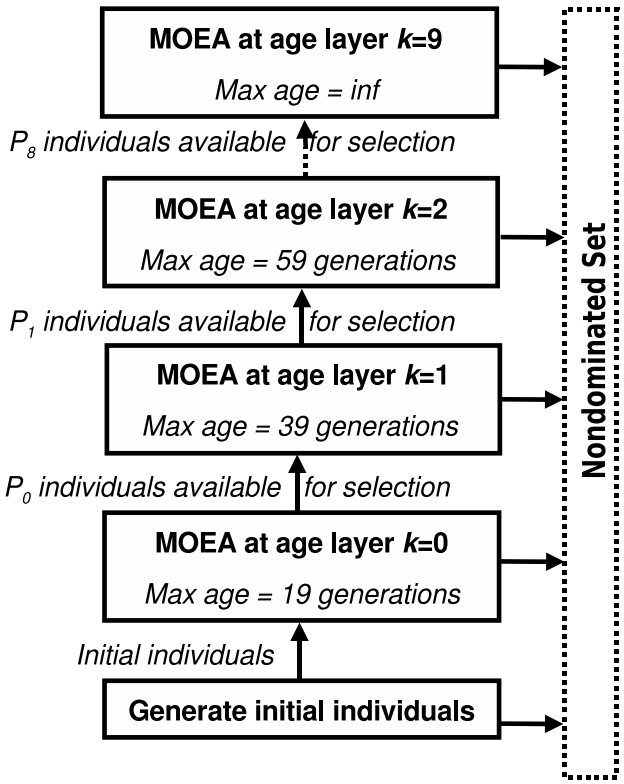


Figure 7.1: *Structure of* MojitoSynthesis()*, which uses a multi-objective age-layered population structure (ALPS).*

To overcome this issue, we use the age-layered population structure (ALPS) [Hor2006], which segregates by genetic age instead of fitness. The structure and behavior of *MojitoSynthesis()* are given in Figure 7.1 and Table 7.1, respectively. Each age layer P_k holds N_L individuals. By example, P_1 might allow individuals with age 0-19; P_2 allows age 0-39, and so on; the top level P_K allows age 0-∞. If an individual gets too old for an age layer, it gets removed from that layer.

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. 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. Every N_a ("age gap") generations (Table 7.1, line 3), a new age layer may be added (lines 4-5), and initial individuals enter layer *k*=0 as either random individuals or "tuned" random individuals (line 6; *InitialCircuit()* details are in section 7.5).

ALPS can be characterized as explicitly handling the exploration-vs-exploitation tradeoff within its search data structure, making it different than most algorithms which try to handle the tradeoff in behavior in time (dynamics) rather than in structure. Since notions like "getting stuck" and "premature convergence" are from the realm of exploration-vs-exploitation behavior in time, ALPS gains an unusual degree of reliability in global optimization. It just keeps "going and going," continually improving. Notably, the user does not have to worry about whether ALPS is stuck, just whether or not to use the current best design or to keep going. This is very important in circuit design, because the analog designer should not have to worry about issues in algorithm dynamics.

Table 7.1: *Procedure MojitoSynthesis()*

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

We designed a multi-objective version of ALPS (only a single-objective ALPS exists in the literature [Hor2006]). The approach is to have a multi-objective EA (MOEA) at each age layer k, running one generation at a time (line 9 of Table 7.1). Whereas a canonical MOEA would select at just layer k, here the MOEA selection also considers layer $k-1$'s individuals (line 8). An external archive holding the Pareto-optimal set Z is always maintained. Stopping conditions (line 2) can include a maximum number of individuals $N_{ind,max}$ or a maximum number of generations $N_{g,max}$.

Table 7.2: *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 7.2 shows the algorithm for the MOEA at one age layer, for one generation. Note how individuals from the lower layer are imported for selection. Key steps are MOEA-specific selection (line 1) and evolutionary operators (line 2).

A master-slave configuration is used for parallel processing, in which the high-level routine *MojitoSynthesis()* is run on the master, and slaves implement *InitialCircuit()*, *ApplyOperators()* for parent pairs, and *Evaluate()* for individuals.

The next two sections elaborate on selection and evolutionary operators, respectively.

7.3 Handling Multiple Objectives

This section describes the MOEA used in MOJITO for the first-round experiments of this chapter (section 7.6.1.1), and the second-round experiments of the next chapter (section 7.6.2) which overcame issues of the first round.

In the experiments of this chapter, we use NSGA-II [Deb2002], because it is relatively simple and reliable, is well-studied, and can readily incorporate constraints.

A key part in MOEAs is *selection* – how to choose the N_L selected parents P_{sel} from the candidate parents $P_k \cup P_{k-1}$.

NSGA-II Selection. NSGA-II performs selection in the following steps (just as section 4.4.3 described).

Its first step is to sort the candidate parents into *nondomination layers* F_i, $i = 1..N_{ND}$, where F_1 is the nondominated set, F_2 is what would be nondominated if F_1 was removed, F_3 is what would be nondominated if $F_1 \cup F_2$ was removed, etc. F contains all the candidates with no duplicates $F_1 \cup F_2 \cup \cdots F_{ND} = P_k \cup P_{k-1}$; $F_1 \cap F_2 \cap \cdots \cap F_{ND} = \emptyset$.

NSGA-II's second step is to fill up P_{sel}. It first adds all individuals from F_1, if they all fit, i.e. if if $|P_{sel}| + |F_1| \leq N_L$. If there is space left, it then adds all individuals from F_2 if they all fit. If there is space left, then adds all individuals from F_2 if they all fit. And so on.

For the third step, once the P_{sel}-filling step reaches an F_i where *not* all of F_i's individuals can fit, then a *subset* of F_i's individuals needs to be chosen. NSGA-II chooses this subset as the individuals with the highest distance from other F_i individuals in the performance space. This selection criteria is known as "crowding".

NSGA-II Selection Issues. The experiments of the next chapter have five objectives, which more closely resembles analog circuit design goals [Raz2000]. Unfortunately, most MOEAs including NSGA-II do poorly when there are more than two or three objectives [Cor2007].

To improve NSGA-II, we need to understand why it does poorly. The problem is that with many objectives, most or all of the population is nondominated, i.e. there is just one nondomination layer $F_1 = P_k \cup P_{k-1}$. Therefore NSGA-II uses crowding to filter down the nondominated individuals. Crowding biases towards the corners of the performance space that are the farthest apart; and not the center points which are close to all designs. That is, it focuses on designs that are excellent on one or two objectives, yet terrible at the rest. *For a high number of objectives, NSGA-II degenerates into near-random search.*

ARF Selection. To solve the NSGA-II selection issue, we use the technique of "Adaptive Ranking on Pareto Front" (ARF) [Cor2007]. Like in crowding, ARF gives preference to some individuals in the Pareto front compared to others. However, whereas crowding biases to corners of the performance space, ARF biases *to individuals that do relatively well on each objective*. We now elaborate.

The "Adaptive Rank" (AR) criterion is defined by:

$$AR(\phi) = \sum_{i=1}^{N_f} rank(f_i, \phi, Z) \tag{7.2}$$

where $rank(f_i, \phi, Z)$ is the rank of individual ϕ with reference to the Pareto-optimal set Z, for objective f_i. For a given objective, the best individual has a rank value of 1, the second-best has rank 2, etc.

ARF is implemented like NSGA-II in the first two selection steps, but it differs in the third step: if the Pareto front will fill up all of P_{sel}, then the AR criterion is used; otherwise the usual NSGA-II "crowding" criterion is used. That is, use AR only if: $|P_{sel}| + |F_1| \geq N_L$. Therefore ARF is only used to distinguish among Pareto-optimal individuals, not among the remaining (Pareto-dominated) individuals.

7.4 Search Operators

Each building block in the hierarchical blocks library has its own parameters, which fully describe how to implement the block and its sub-blocks. Thus, the search space for this circuit type (e.g. current mirror) is merely the possible values that each of the block's parameters can take, as illustrated in section 6.5. 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.

However, an individual has an intrinsic hierarchy, as illustrated in Figures 6.11 and 6.12. Since a vector-oriented view does not recognize this hierarchy, operations on merely vectors will 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. For example, a variable controlling a second-stage device's parameter has no effect if only the first stage is active. From the perspective of a search algorithm, this means that there are vast regions of neutrality [Sch1994, Huy1996]; or alternatively the representation is non-uniformly redundant and runs the risk of stealth mutations [Rot2006].
- 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 [Gol2002].

One might ask if this can be reconciled by applying a hierarchical design methodology [Cha1997, Eec2005]. The answer is "no" because *there are not goals on the sub-blocks*, only at the highest-level blocks (we could, however, still apply a hierarchical methodology to the results).

The search algorithm needs *both tree-based and vector-based views* of the space. Accordingly, we design novel EA *mutation and crossover operators that reconcile both views.* Because crossover (interpolation) needs two or more individuals to operate on, the population-based nature of EAs makes them a natural choice [Koza1992].

We now describe the individual operators.

7.4.1 Mutation Operator

This operator varies one or more parameters of the genotype, to change the structure and/or parameters of the phenotype. Continuous-valued parameters follow Cauchy mutation [Yao1999] which allows for both tuning and exploration. Integer-valued $choice\_i$ parameters follow a discrete uniform distribution. Other integer and discrete parameters follow discretized Cauchy mutations.

Mutation can affect either topology choice parameters and / or device-value parameters. A change in a device-value parameter might be, for example a change in *feedback_C* from 1.1883e-11 to 1.6814e-11 Farads. Such changes affect the devices' parameter settings in the SPICE netlist, but not the choice of devices or device connectivity.

A change in topology choice parameter means a change in the topology itself. Consider for a moment the topology from the example in section 6.5, which is reproduced in Figure 7.2. It has two stages (*choice_index*=1) and its stage 1 load attaches to *gnd*, not *vdd* (*loadrail_is_vdd* = 0).

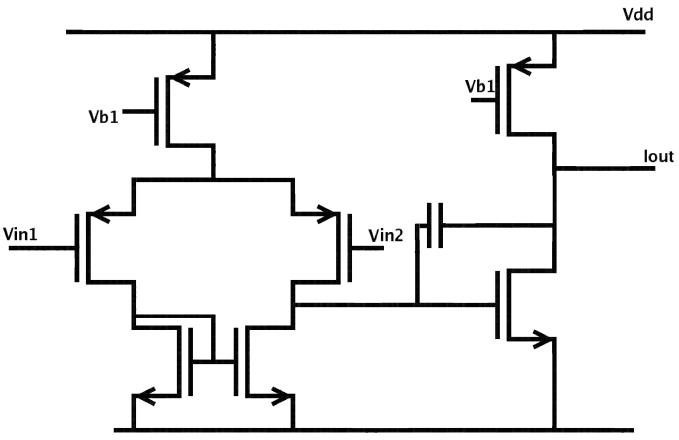


Figure 7.2: *Opamp where choice_index=1, stage1_input_is_pmos = 1, stage1_loadrail_is_vdd = 0.*

By mutating the variable *loadrail_is_vdd* from 0 to 1, the topology will change from Figure 7.2 to Figure 7.3. We see that the second stage does not change, but the first stage is now folded. It still has PMOS inputs, but the load is now attached to *vdd* rather than *gnd*. (More similar operations are illustrated in Figures 6.6 to 6.9.)

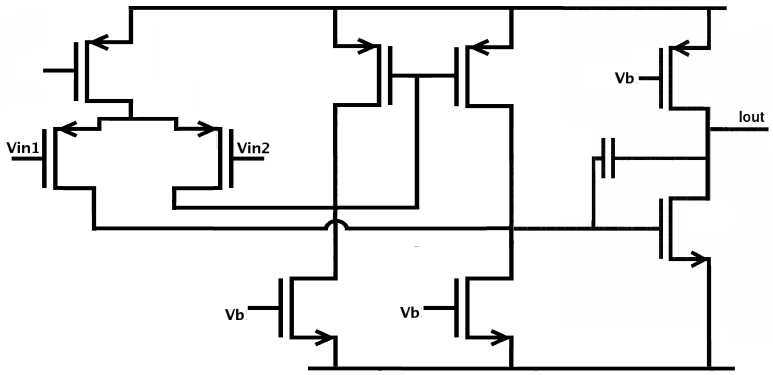


Figure 7.3: *Opamp where choice_index=1, stage1_input_is_pmos = 0, stage1_loadrail_is_vdd = 1. This is a single topology-parameter mutation away from Figure 7.2.*

Alternatively, if we mutate the variable (*choice_index* from 1 to 0, we are effectively asking for just one stage. The second stage disappears, but the first stage remains identical, as Figure 7.4 shows.

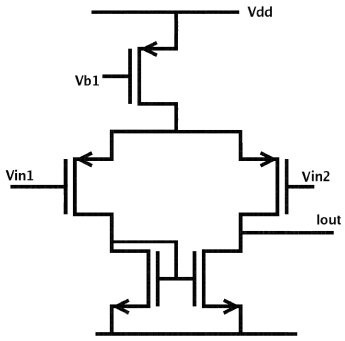


Figure 7.4: *Opamp where choice_index=0, stage1_input_is_pmos = 1, stage1_loadrail_is_vdd = 1. This is a single topology-parameter mutation away from Figure 7.2.*

Sometimes a change to a genotype will *not* affect the phenotype, because the parameters being changed are in building blocks that have been turned off by other parameter choices. For example, if a 1-stage opamp is selected, changes in parameters related to the second stage will have no effect, regardless of whether they are topology or device-sizing parameters. A change to genotype that does not affect phenotype is called a "neutral" operation [Sch1994, Huy1996]. Though such neutral operations ("stealth mutations") of the space has been shown to help exploration in some applications [Vas2000, Mcc2006a], results are mixed. It has been shown that in general, stealth mutations make the algorithm's performance more unpredictable [Rot2006]. We prefer predictability, and therefore rely on other means like ALPS to enhance exploration. Therefore, for MOJITO to avoid stealth mutations: in a mutation, the change is retained *only* if the phenotype (netlist) changes. Mutation attempts are *repeated* until a phenotype change occurs.

The number of parameters to vary is drawn from a discrete distribution which biases towards fewer variables, but sometimes allows significantly more variables to change. The biases are shown in Table 7.3.

Table 7.3: *Biases to Number of Variables to Mutate.*

Number of Variables to Mutate	1	2	3	4	5	6	7	8
Relative Bias	100	15	5	3	2	2	2	1

7.4.2 Crossover Operator

Crossover 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.

For example, we could randomly choose the "ds amp1 vdd/gnd ports" block in parent A, identify the parameters assocatied with that block in both parent A and parent B, then swap them. What this amounts to for the circuit is to swap the first stage between parent A and parent B. But the swapping can happen at any level, from current mirror to "ss input" (stacked vs. folding cascode) to source degeneration to others.

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).

7.5 Generation of Initial Individuals

This section describes how we generated initial individuals for our first round of experiments (vector-oriented random sampling), the issues it presented, and how those issues were overcome for the second round of experiments by using tree-oriented random sampling and multi-gate constraint satisfaction.

7.5.1 Initial Individuals: First Round Approach

In the first round of experiments, each initial individual was generated with uniform random sampling per variable (Table 7.4). Specifically: recall from section 6.2 that a design at the top block of the library hierarchy can be fully specified by a set of N_d variables (line 1). So, a vector $\boldsymbol{d}$ can define the design point ϕ (and correspondingly, $\Re^{N_d}$ can define Φ). A random ϕ is generated as follows: for each variable i (line 3), either draw a value from a continuous uniform random distribution $U([min, max])$ in lines 4-5, or from a discrete set of values with equal bias $U(\{val1, val2, \ldots\})$ in lines 6-7.

Table 7.4: *Procedure InitialCircuit() (First Implementation)*

Inputs: Φ
Outputs: $\phi \in \Phi$
1. $\boldsymbol{d}$ = topLevelVariables(Φ)
2. $\phi = \emptyset$
3. for i = 1 to N_d:
4. if d_i is continuous:
5. $\phi_i \sim U([d_{i,min}, d_{i,max}])$
6. else: #d_i is discrete:
7. $\phi_i \sim U(\{d_{i,1}, d_{i,2}, \ldots, d_{i,max}\})$
8. return ϕ

7.5.2 Initial Individuals: First Round Issues

This approach had issues: uneven sampling of topology types, and difficulty in maintaining diversity of topologies. For example, we observed that the algorithm was generating single-stage amplifiers just as often as two-stage amplifiers, despite the fact that there are many more possible two-stage topologies. This is because the approach of Table 7.4 views the space "flat", randomly picking a value for each of the topology choice parameters, with equal bias.

7.5.3 Initial Individuals: Second Round Approach

To fix this the first round issues, we instead give equal bias to each possible *topology*, which is akin to *uniform* sampling of sentences in a *grammar* [Iba1996].

When synthesis begins, a one-time computation of the number of possible topologies for each part is made, using the rules of section 6.6. The counts c are used as a bias on corresponding Flexible block $choice\_i$ values on the top-level part.

Table 7.5 gives the improved sampling procedure, called *RandomDrawCircuit()*. The key difference compared to Table 7.4 is the introduction of lines 4-5, where each choice variable i's value is chosen according to a discrete density function (ddf) having a probability $p_{i,j}$ for each possible value $d_{i,j}$; $p_{i,j} = c_{i,j} / \sum_{j=1}^{j_{max}} c_{i,j}$; $\sum_{j=1}^{j_{max}} p_{i,j} = 1$; $c_{i,j}$ is th: e number of sub-topologies if the j^{th} value is used for variable i.

7.5.4 Initial Individuals: Second Round Issues

With further runs, we found that most randomly generated higher-complexity amplifiers (e.g. folding topologies, 2-stage amplifiers) would die out within a few generations of being generated. While ALPS generated more topologies in later random injection phases, those would die out too.

Upon investigation, we found that the randomly-generated complex amplifiers' performances were much worse than simple ones due to poor initial sizing, and that they did not improve as quickly. This is because the more complex amplifiers have more sizing and biasing variables to set reasonably in order to reach a minimal performance bar.

Table 7.5: *Procedure RandomDrawCircuit()*

Inputs: Φ
Outputs: $\phi \in \Phi$
1. $\boldsymbol{d}$ = topLevelVariables(Φ)
2. $\phi = \emptyset$
3. for i = 1 to N_d:
4. if d_i is a choice parameter:
5. $\phi_i \sim ddf(\{p_{i,1}$ for $d_{i,1}, p_{i,2}$ for $d_{i,2}, \ldots\})$
6. else if d_i is continuous:
7. $\phi_i \sim U([d_{i,min}, d_{i,max}])$
8. else: #d_i is discrete:
9. $\phi_i \sim U(\{d_{i,1}, d_{i,2}, \ldots, d_{i,max}\})$
10. return ϕ

We also found that the first feasible topology found would overtake other topologies, further hurting diversity. This is because of NSGA-II's constraint handling: it lumps all constraints into one overall violation measure, and always prefers feasible individuals over infeasible individuals. It effectively does single-objective search until the feasible individual is found (killing some topology diversity then), and then emphasizes the first feasible individual excessively (because no other topology gets there quite as fast).

7.5.5 Initial Individuals: Third Round Approach

There are plenty of possible ways to resolve this, as there is plenty of literature (EA and otherwise) in reconciling multi-objective optimization and constraint handling. However, for this problem domain, there is an opportunity to not only give harder-to-optimize topologies more time to get decent performance, but to simultaneously cheaply optimize each randomly drawn topology:

- A guideline is to *defer competition among topologies until each topology is at least nearly feasible.* It is acceptable to have them compete once they are past feasible, because each topology will occupy its own niche in the performance space and will therefore be maintained within the multi-objective framework.
- This guideline is implemented by doing cheap constraint-satisfaction style optimization, on a per-topology basis. It is cheap because we deploy a series of constraint-satisfaction "gates". The first gates prunes poor designs with a computationally cheap test. Subsequent gates do more thorough pruning via successively more expensive tests. Upon exiting the final gate, the topology can be assured to be competitive with other topologies.

Table 7.6 describes the algorithm for this improved *InitialCircuit()* routine. We now describe it in more detail.

For the first gate (lines 2-5), we use *function* device operating constraints (DOCs) (section 6.4), which measure if current and voltage conditions are being met. Function

DOCs can be measured directly from the design point, because the design point *is* currents and voltage due to the operating point driven formulation (section 6.4). Since function DOCs can be measured directly from the design, no simulation is necessary and gate 1 is extremely fast.

The second gate (lines 6-9) uses DOCs computed from dc *simulation*. While slower than direct examination of an operating point, dc simulation is still much cheaper than other analyses like transient. It can be viewed as a more accurate version of function DOCs.

The third gate (lines 10-13) is based on performance constraints across all the circuit analyses (ac, dc, transient, etc). It is the most thorough, but the most expensive. Failing simulation DOCs means that the circuit is still not hitting the topology's implicit design intent, e.g. for certain transistors in saturation, and others in linear, etc.

In all three gates, the *mutateSizings()* operator is like section 7.4, except only device-value (non-topology) parameters get changed.

Table 7.6: *Procedure InitialCircuit() (Improved Implementation)*

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

7.5.6 Initial Individuals: Third Round Observations

In our experiments, we found that the first gate would take about 1000-3000 designs to pass (very cheap because it requires no simulation), the second gate would take 100-300 designs, and the third gate would take 300-1000 designs. Overall runtime for the procedure was typically less than 10 minutes on a single 2.5-GHz machine. (This compares favorably with other recent single-topology circuit sizers, e.g. [Ste2007]).

We achieved the aim of this subsection: for the sake of topology diversity, to reliably generate complex topologies which could compete against simple topologies for multi-objective search.

7.6 Experimental Results

7.6.1 General Experimental Setup

The complete search space had N_d = 50 variables which include both topology selection variables and sizing variables; there were 3528 possible topologies using the library of Figure 6.10.

MOJITO's library and search algorithm were implemented about 25000 lines of Python code [Pyt2008], which uses the libraries Pyro [Pyro2008] for parallel processing and Numeric v24.2 [Pyn2008] for matrix computations.

Table 7.7 gives futher parameter settings. The EA settings are "reasonable" values in the same range as those used in [Hor2006]. No tuning was done to optimize their performance.

Table 7.7: *Experimental setup parameters.*

Technology	0.18μm CMOS
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 function and simulation device operating constraints (DOCs)
Objectives	See specific experiment
EA settings	K = 10, N_L = 100, N_a = 20, $N_{ind,max}$ = 100,000 for chapter 7, N_I = 180,000 for chapter 8. (Parameter descriptions are in sections 7.3 - 7.5.)

7.6.1.1 Experiment: Hit Target Topologies?

In this section, we aim to validate MOJITO's ability to find targeted topologies.

7.6.1.2 Specific Experimental Setup

The objectives of the experiment were to maximize GBW, and to minimize power, while other performances needed to satisfy the constraints from Table 7.7.

Three runs were done, the only difference between them being the common-mode voltage $V_{cmm,in}$ at the input. We know that for V_{dd} = 1.8V and $V_{cmm,in}$ = 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.

7.6.1.3 Experimental Results

Each run took the equivalent of overnight on ten single-core 2.0 GHz Linux machines, covering about 100,000 individuals. Figure 7.5 illustrates the outcome of the experiments. It contains the combined results of the three optimization runs.

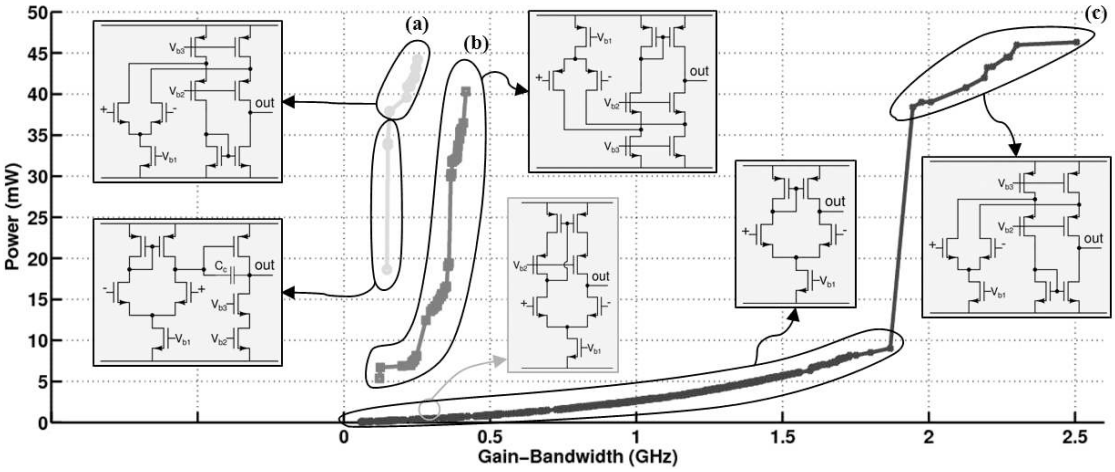


Figure 7.5: *Combined result plot for the 3 synthesis runs of the first experiment. Set (a) shows a Pareto front for $V_{cmm,in}$ = 1.5 V. Set (b) is for $V_{cmm,in}$ = 0.3 V. Set (c) is for $V_{cmm,in}$ = 0.9 V. Aim is to minimze power and maximize Gain-Bandwidth. Each point is a sized topology; each topology has many different sets of sizings. The expected topologies were found.*

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 plotted here, or actively searched, because they are not part of the Pareto-optimal set. 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 has two complementary options. The first is to use the topology-performance *knowledge extraction* tools introduced in the next chapter, and the second complementary is to use his expertise and experience in circuit analysis to manually dive deeper. An analog-design answer for the jump in power is: going from a

non-folded to a folded input means that one more current branch is needed, which has its own corresponding current needs (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*.

7.6.2 Experiment: Diversity?

The aim of this experiment is to verify that MOJITO can 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 the performance space.

7.6.2.1 Experimental Setup

In this experiment, one run was done using the settings and constraints of Table 7.7.

There were three objectives: maximize Gain-Bandwidth (GBW), maximize gain, and minimize area.

7.6.2.2 Experimental Results

The results are shown in Figure 7.6. We can see that MOJITO found rich and diverse structures as expected. MOJITO determined the following. Folded-cascode op amps gave high GBW, but with high area. Also, 2-stage amps give high gain, but at the cost of high area. The low-voltage current mirror load is a 1-stage amplifier with high gain. There are many other 1 stage topologies which give a broad performance tradeoff. These are all results that a circuit designer would expect.

Note that subsequent chapters will have further experiments validating MOJITO.

7.6.3 Comparison to Open-Ended Synthesis

For problems of comparative complexity, status quo GP (i.e. with no reuse) needed 100 million or more individuals [Koza2003, Koza2004a], and *those results were not trustworthy by construction*. Recall from section 5.5.5.2 that for status-quo GP to get a reasonable degree of trust would take 150 years on a 1,000-node 1-GHz cluster. 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 with MOJITO to comparable results.

The core reason for MOJITO's efficiency is that unlike GP, *every* topology in the MOJITO search space is trustworthy *by construction*. This is not a limiting factor for MOJITO, as the number of possible topologies is still massive.

Status quo GP makes the problem artificially hard, by ignoring all the domain knowledge that has been accumulated over the decades. As discussed before, almost every discipline having any degree of maturity has accumulated structural domain knowledge,

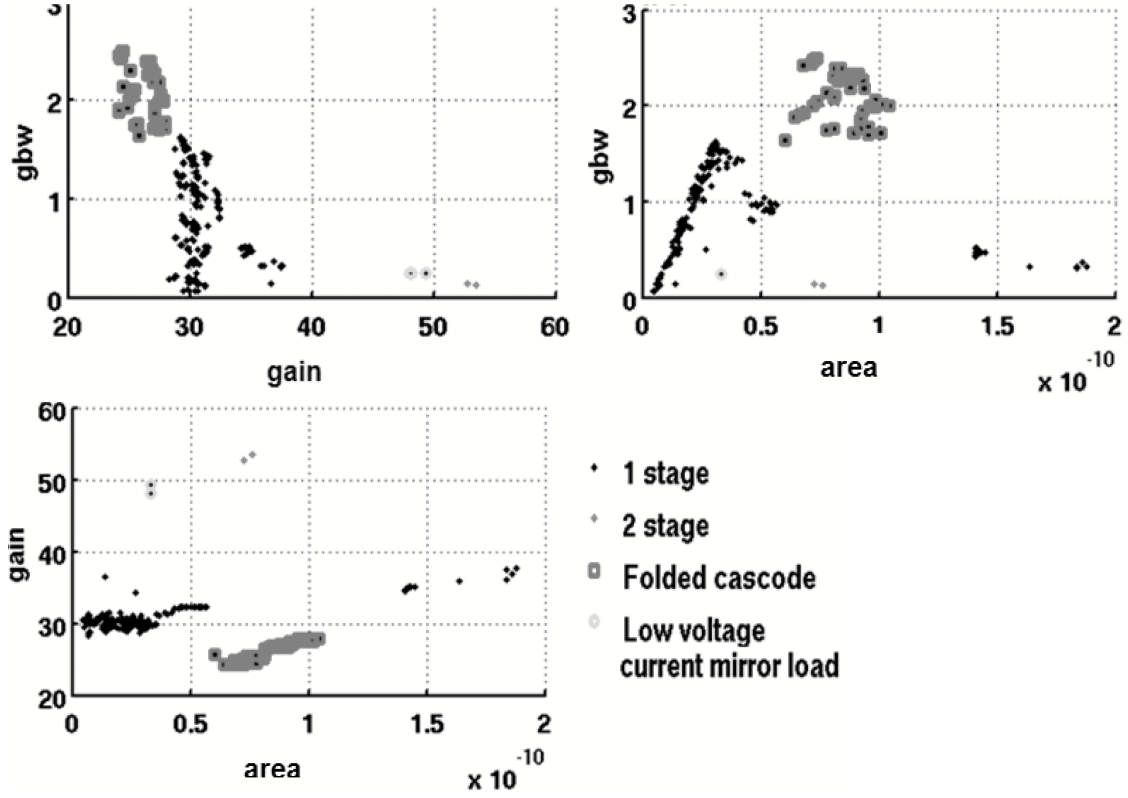


Figure 7.6: *Pareto front for a MOJITO run, for the second experiment, which has 3 objectives (maximize GBW, maximize gain, minimize area). Individuals are grouped according to some of their structural characteristics (e.g. 1-stage vs. 2-stage), to illustrate their diversity.*

that could at least be partially codified. By making the effort to codify structural domain knowledge of analog circuits into a searchable framework, MOJITO not only reduce the runtime by several orders of magnitude, it returns results that are guaranteed trustworthy. For the first time, structural analog synthesis can be done with industrially palatable speed and trust. This proves that there's a lot to be said for topology reuse!

7.7 Conclusion

This chapter has described the search algorithm as part of the design aid for topology selection and design, called MOJITO. The key technical challenges in such a tool were to generate topologies that are trustworthy by designers, in reasonable computer time.

This chapter has shown how aggressive *reuse* of known designs brings a vast reduction in computational effort in GP applied to automated structural design. By leveraging pre-specified hierarchical building blocks, MOJITO automatically designs 100% *trustworthy* topologies of industrially relevant complexity, with commercially reasonable computational effort. MOJITO's effectiveness has been demonstrated in two separate experiments,

showing how it hit the target designs as expected, from a library of thousands of possible topologies.

MOJITO also used state-of-the-art ideas in EA design. It has a hybridized tree/vector view of the search space, implemented as operators having those two perspectives. It has been guided by recent advances in theory of EA representations [Rot2006]. To avoid premature convergence and to minimize sensitivity to population size setting, it employs the age-layered population structure (ALPS) [Hor2006], and embedded modern MOEAs into each age layer of ALPS to make it multiobjective: the popular NSGA-II [Deb2002], and an ARF variant [Cor2007] to handle a higher number of objectives. Good initial topology sizings are quickly generated via multi-gate constraint satisfaction.

The next chapter takes MOJITO results to the next level: to help designers *extract further knowledge* from the results of a MOJITO run.

Chapter 8

Knowledge Extraction in Trustworthy Circuit Topology Synthesis

An investment in knowledge pays the best interest.
–Benjamin Franklin

8.1 Introduction

8.1.1 Abstract

Whereas the last two chapters have presented a *design aid* for the designer task of topology selection / design, this chapter presents *insight aids* for that task.

This chapter presents a methodology for analog designers to maintain their insights into the relationship among performance specifications, topology choice, and sizing variables, despite those insights being constantly challenged by changing process nodes and new specifications. The methodology is to take a data-mining perspective on a Pareto-optimal set of sized analog circuit topologies as generated with MOJITO [Mcc2007, Mcc2008a, Mcc2008c]. With this perspective, it does: extraction of a specs-to-topology decision tree; global nonlinear sensitivity analysis on topology and sizing variables; and determining analytical expressions of performance tradeoffs [Mcc2008b, Mcc2008d, Mcc2009]. These approaches are all complementary as they answer different designer questions. Once the knowledge is extracted, it can be readily distributed to help other designers, without needing further synthesis. Results are shown for operational amplifier design on a database containing thousands of Pareto-optimal designs across five objectives.

8.1.2 Background

Analog designers use their experience and intuition to choose, to the best of their knowledge, circuit topologies and to design new topologies. Unfortunately, the topology used may not be optimal, with possible adverse affects on the related product's performance, power, area, yield, and profitability. The suboptimal design may be because the design has to be done on a new or unfamiliar process node, the designer is too time-constrained

to be thorough, or simply because 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 the art of analog design takes decades to master [Wil1991]. That said, it still means that a suboptimal topology may be used. Hence, it is desirable to provide support for the designer in this critical step of topology selection and design, and ideally to catalyze the learning process. Prior CAD research has focused on automated topology selection and design (with nice successes [Rut2002, Rut2007]), but has had little emphasis on giving insight back to the user. In fact, by deferring control to automated tools, a designer's learning might slow. Even worse, the designer could end up poorly-equipped when problems arise. It is perhaps this last reason – losing control – that explains why most analog designers have been reluctant to adopt analog circuit optimizers.

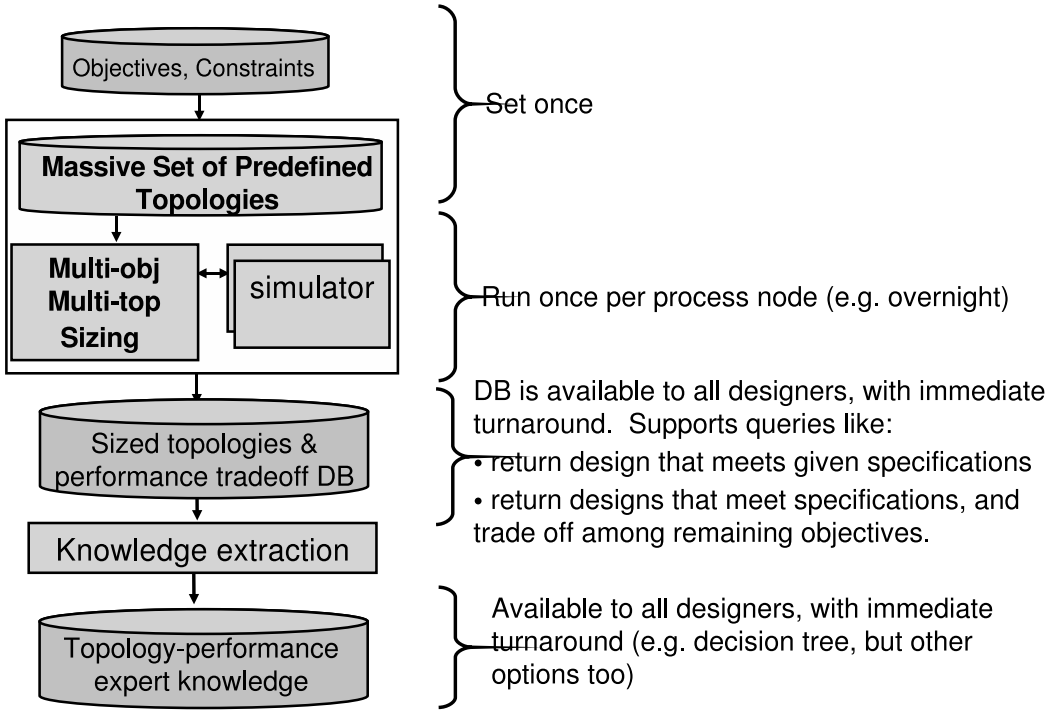


Figure 8.1: *Target flow. The extracted knowledge is readily available to all designers, without requiring them to invoke automated sizing.*

8.1.3 Aims

This chapter therefore addresses the following question: is there a means to help analog designers maintain and build expert topology-performance knowledge, in such a way that even the most reluctant designers might benefit? The starting point is MOJITO (described in chapters 6 and 7), which traverses thousands of circuit topologies to automatically generate a database of Pareto-optimal sized topologies. Data-mining is now applied to this database.

The contribution of this chapter is twofold:

- A data-mining perspective on the database to extract the following expert knowledge: (a) a specs-to-topology decision tree, (b) global nonlinear sensitivities on topology and sizing variables, and (c) analytical performance-tradeoff models. The techniques are complementary, as each addresses specific question types as summarized in Figure 8.2.
- A suggested flow in which even reluctant users can conveniently use the extracted knowledge (Figure 8.1). The database generation and knowledge extraction only needs to be done once per process node, e.g. by a single designer or a modeling group. The extracted knowledge can be stored in a document (e.g. pdf, html), and simply made available to other designers. Figure 8.1 also highlights how the sized topologies database (DB) is available to all designers, for immediate turnaround in specs-to-sized topology queries.

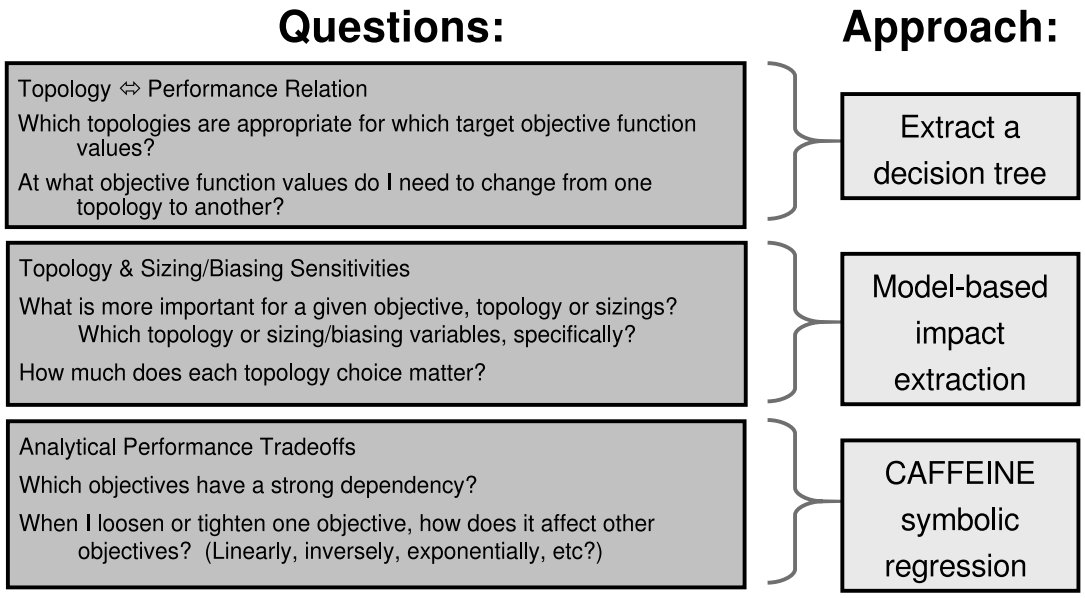


Figure 8.2: *Questions addressed by each knowledge extraction approach.*

This chapter is organized as follows. The knowledge extraction procedures will be explained using a reference database, generated as described in section 8.2. Section 8.3 describes how a specs-to-decision tree is extracted from the database. Section 8.4 describes the extraction of global nonlinear sensitivities, and Section 8.5 the extraction of analytical tradeoffs model. Section 8.6 concludes.

8.2 Generation of Database

This section describes the setup to generate the sized-topologies database.

We use MOJITO, which has been described in chapters 6 and 7 along with the details of the experimental settings used. The problem has five objectives: maximize GBW, minimize power, maximize gain, maximize dynamic range, and maximize slew rate. The

library is the opamp library described in chapter 6, which contains 3528 topologies. Algorithm settings, and the constraints on circuit performances, are in Table 7.7.

A single synthesis run was done on the five objectives, using parameters described in section 7.6.1. The run took approximately 12 hours on a Linux cluster having 30 cores of 2.5 GHz each (which is palatable for an industrial setting). 180 generations were covered, traversing 3528 possible topologies and their associated sizings. It returned a database of 1576 Pareto-optimal sized topologies.

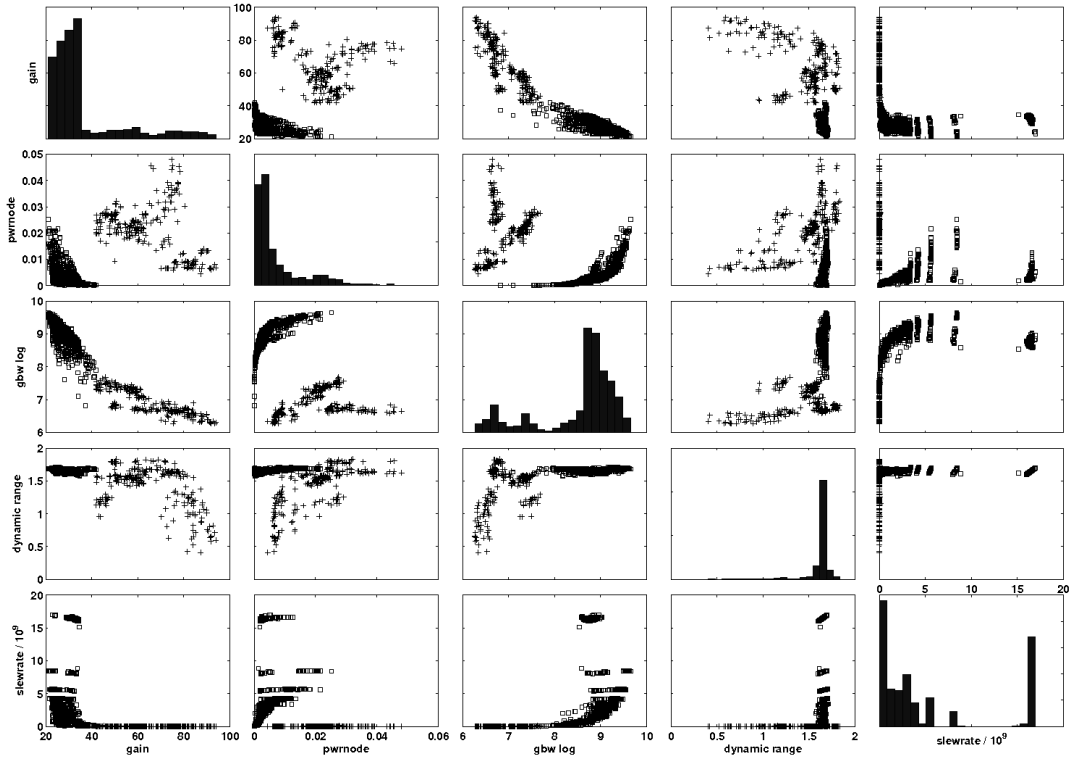


Figure 8.3: *Grid illustrating the Pareto front. The diagonal entries show histograms of performance; the rest show two-dimensional projections from the five objectives. The squares are 1-stage amplifiers, and the pluses are two-stage amplifiers.*

To become oriented with the raw results data, Figure 8.3 shows a grid of 2D scatterplots and histograms for the five performance objectives.

From the histograms, we can get a quick picture of the distribution and bounds of performances. For example, the $gain$ histogram in the top right corner shows that the majority of $gain$ values are in the 20-30 dB range, and but there are designs with gains in the range of 40, 50, 60, 70, 80, and even 90 dB. Or, about 90% of the topologies have a power ($pwrnode$ value) $<$ 30 mW, and a significant portion of those have power $<$ 10 mW.

From the scatterplots, we begin to understand the performance bounds and take note of trends. For example, let us examine the $gain$ vs. $pwrnode$ subplot (top row, second from left). Note how the one-stage topologies (squares) occupy the corner having lowest gain

and lowest power, whereas the two-stage topologies (+'s) spread out across the rest of the $gain$ vs. $pwrnode$ subplot. In fact, the two-stage topologies spread into two distinct clusters: one with 70-90 dB gain and 10-20 mW power, and another broader cluster having 40-80 dB gain and 20-50 mW power.

The tradeoffs are more sharp in other 2d-scatter subplots, such as $pwrnode$ vs. $gbwlog$ (second row, middle). $gain$ vs $gbwlog$ (top row, middle) is even more sharp. We also see that some performances onyl return very specific values, e.g. with $slewrate$.

But examining these scatterplots can only give us some degree of insight. Key questions remain, such how specific performance requirements guide the choice of specific topologies. This is where knowledge extraction can help.

8.3 Extraction of Specs-To-Topology Decision Tree

8.3.1 Introduction

We explain the motivation for decision trees with a simple example having two objectives: minimize power and maximize GBW. Figure 8.4 left shows the power-GBW tradeoff results from a MOJITO run of section 7.6.1.1. Each point is a different sized topology; the eight sized topologies (eight points) have two unique topologies, as indicated by the two ellipses grouping the points.

We can see by inspection that the division between the ellipses (topology choice) is best done at a power between 34 and 38 mW; here 37 mW is chosen. An algorithmic way to choose such a division is to sweep all possible values of power, and all possible values of GBW, and to choose the value and output that gives the best split. That single split to choose between the two topologies is embodied in the decision tree of Figure 8.4 right: if a power < 37 mW is chosen, the two-stage amplifier is chosen, otherwise a one-stage amplifier with folded-cascode inputs is chosen.

Making a topology decision based on inspecting the two-dimensional tradeoff is easy. But when there are more dimensions, such as the five dimensions of Figure 8.3, it becomes dramatically harder. Decision trees encapsulate and illustrate the relevant decisions in a high-dimensional space.

This section describes the automatic extraction of decision (CART) trees [Bre1984] that map from performance values to topology choice starting from the Pareto-optimal results DB of a previous MOJITO run. Decision trees have a double use: they can directly suggest a choice based on inputs, but they also expose the series of steps underlying the decision. CART trees are in widespread use, such as medicine: "In medical decision making (classification, diagnosing, etc.) there are many situations where decision must be made effectively and reliably. ... Decision trees are a reliable and effective decision making technique that provide high classification accuracy with a simple representation of gathered knowledge" [Bre1984].

Decision trees have not gone unnoticed in analog CAD either. They have been proposed as the centerpiece of topology-choosing "expert systems", such as in the OPASYN system [Koh1990]. Unfortunately, these trees had to be manually constructed at that time which took weeks to months of effort, and were based on rules of thumb that became

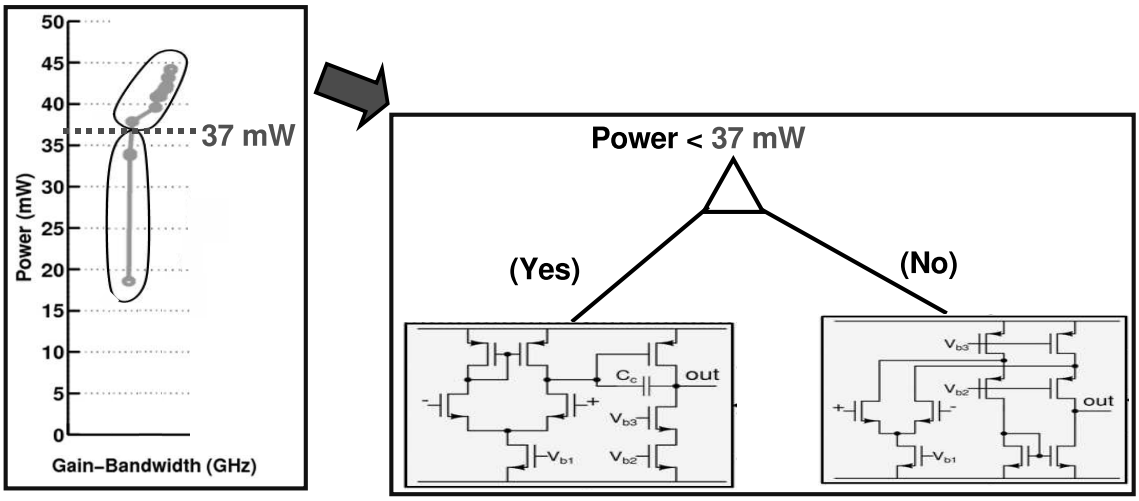


Figure 8.4: *Left: results of a two-objective run (minimize power, maximize GBW) which gave two topology choices indicated by the ellipses. Right: corresponding a decision tree to guide the topology choice based on the power input specifications*

.

obsolete as soon as the process node changed. In contrast, this chapter constructs the specs-to-topology decision tree *automatically* from data. This is only possible now, because a prerequisite to get the data was a competent multi-topology multi-objective sizer that could output a diverse set of topologies. MOJITO is the first of this new class of sizers [Mcc2007, Mcc2008a, Mcc2008c].

8.3.2 Decision Tree: Problem Formulation and Approach

We formulate specs-to-topology decision tree induction as a classification problem from a Pareto-optimal set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_j^*, \ldots, \phi_{N_Z}^*\}$ resulting from a MOJITO run. Within Z, there are N_T unique topologies ($N_T \leq N_Z$) with corresponding class labels $\Upsilon = \{1, 2, \ldots, N_T\}$. 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. $\widehat{\upsilon_j} = \omega(\boldsymbol{f_j})$. 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 $\upsilon_i \in \Upsilon$. This is similar to the regression form of CART trees (equation 3.6).

Tree construction using the CART algorithm [Bre1984] 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 the 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 $\{split\_variable, split\_value\}$ tuples in the data, the algorithm chooses the tuple with the highest information gain according to the chosen split criterion. That split creates a left and right child, where the left child is assigned the data points and region

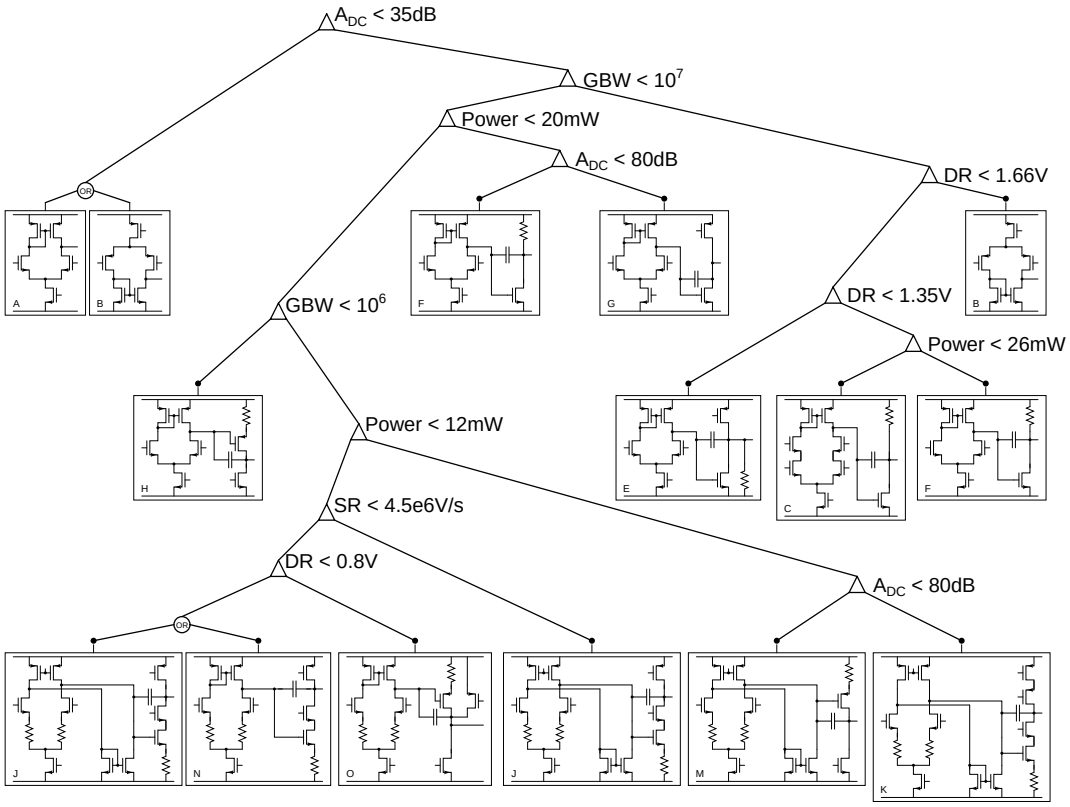


Figure 8.5: *A decision tree for going from specifications to topology. Unlike past approaches, this was automatically generated. The technology is 0.18 μm CMOS, V_{dd} = 1.8 V.*

.

meeting $split\_variable \leq split\_value$, and the right child is assigned the remaining data points and region. The algorithm recurses, splitting each leaf node until the number of points in leaf node $\leq$ a pre-set maximum. The final set of regions is defined by the leaf nodes' regions only.

The tree-constructing algorithm implementation was implemented in Matlab [Mat2008]. The "gini" splitting criterion was used; it selects the $\{variable, value\}$ that splits off the most data points [Bre1984]. The max number of points per leaf node was 10, so that a compact tree would be generated for the example of section 8.2.

8.3.3 Decision Tree: Experimental Results

Figure 8.5 shows the decision tree that was automatically generated from a MOJITO results DB. It provides insight into what topologies are appropriate for which performance ranges, and actually even gives a suggested topology (one of the leaf nodes), given a set of input specifications (which guide traversal from the top of tree to a leaf node).

8.3.4 Decision Tree: Discussion

We see that the low-frequency gain (A_{DC}) is the first variable selected on, and following through the tree, we see that all specifications play a role for selecting some topologies: gain-bandwidth (GBW), power, slew rate (SR), and dynamic range (DR). When the specifications require low gain, the tree suggests single-stage topologies; and two-stage topologies are suggested when a higher gain is required. In cases where a very large gain is required with a limited power budget, a two-stage amplifier with large degrees of cascoding is suggested. If power is less of an issue, one can also use a non-cascoded two-stage amplifier. Since only Pareto-optimal individuals are used to generate the tree, the choice for the more power-efficient variant implies lower performance for one or more other metrics (in this case e.g. dynamic range). Also reassuring is that while there were thousands of *possible* topologies, just 15 were returned. This is in line with many analog designers' expectation that just a couple dozen opamp topologies serve most purposes. The challenge, of course, is to find out which topologies those are, and for what specifications they are appropriate.

It is important to remember that the tree is a classifier at its core, so one must avoid reading *too* much into it, such as the meaning of the exact values of the performance split values. In many cases the split value could increase or decrease by a few percent with no effect on the classification. There are CART extensions to capture the sensitivities to the split values, but this is at a cost of additional complexity in the reported tree. This sensitivity does not seem to have affected the deployment of CART trees into other industries such as medicine. As long as the user understands the sensitivity, no additional action is probably needed. Another extension is to let the user give preference to choosing certain split variables first, which may result in interesting alternative trees [Bre1984]. We leave both to future work.

An additional benefit of tree extraction is based on there being more than 2-3 objectives, which means that the raw data is difficult to visualize, as we find when inspecting Figure 8.3. The tree gives an alternate perspective among the 5 objectives, highlighting which topologies cover which performance regions.

8.4 Global Nonlinear Sensitivity Analysis

8.4.1 Introduction

The aim here is to address questions such as: "how much does each topology choice matter? Should I be changing the topology or the device sizes? *Which* block or variables should I change?" There may even be more specific questions, such as "How much does cascoding affect gain?"

8.4.2 Sensitivity Analysis: Problem Formulation and Approach

Our approach to handle such questions is to perform global nonlinear sensitivity analysis. We need to be global – across the range of variables – because we have thousands of training points, and one cannot do small perturbations on integer-valued design variables such

as topology-choice variables. We cannot assume linear because not being local means a Taylor approximation does not apply; small ad-hoc tests showed that linear models fit poorly; and topology-choice variables are *categorical* (variables which have discrete values and no relation among the discrete values, such as topology choice variables).

The sensitivity extraction flow we follow for each performance metric y is:

1. Given: a MOJITO-generated Pareto-optimal set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_{N_Z}^*\}$. Let $\boldsymbol{X} = \{\boldsymbol{x_j}\}$, $j = 1..N_Z$, where $\boldsymbol{x_j}$ is an N_d-dimensional design *vector* corresponding to design point ϕ_j. Let $\boldsymbol{y} = \{y_j\}$ where y_j is the corresponding scalar performance value of ϕ_j for the target objective (e.g. GBW)
2. Build a nonlinear regression model ψ that maps $\boldsymbol{X}$ to $\boldsymbol{y}$, i.e. $\hat{y} = \psi(\boldsymbol{x})$
3. From ψ, compute *global* sensitivities $\boldsymbol{\zeta} = \{\zeta_i\}, i = 1..N_d$ (Table 8.1)
4. Return $\boldsymbol{\zeta}$

Steps 2 and 3 have specific challenges. Step 2, regressor construction, needs to handle numerical *and* categorical input variables. This prevents usage of polynomials, splines / piecewise polynomials, support vector machines, kriging, and neural networks. CAFFEINE ([Mcc2005a], also see chapter 4) handles categorical variables, but it would run very slowly on 50 input variables and 1500 training samples. However, a relatively recent technology achieves the effect of regression on CART trees by boosting them: stochastic gradient boosting (SGB) [Fri2002], which section 3.3 described in detail. SGB has acceptable scaling and prediction properties, so we employ it here.

Step 3 above needs to compute, from the model, sensitivities of the output variable to each of the model's input variables. In the computation, it needs the variables to have global coverage (not local), allow for nonlinearity (versus assuming linear), allow for categorical and non-categorical variables, and, ideally, make no assumptions about the nature of the data distributions (i.e. be nonparametric).

The proposed solution defines the *global nonlinear sensitivity impact* for a variable d_i as: the relative error that a scrambled input variable d_i will give in predicting the output, compared to the errors across the other variables $d_j, j = 1..d, j \neq i$ when those variables are are scrambled.

Table 8.1 gives the algorithm *ModelSensitivities()* that uses this concept to extract impacts (inspired by chapter 10 of [Has2001]). For each variable (line 1), it does repeated scrambling (lines 3-4) and keeps track of the resulting model error (lines 5-6). It normalizes the results (line 6-7) and returns. N_{scr} is number of scrambles; $nmse$ is normalized root mean-squared error (equation 3.7).

8.4.3 Sensitivity Analysis: Results and Discussion

With this proposed flow, we extracted the global nonlinear sensitivities for each performance. SGB and CART were coded in about 500 lines of Python [Pyt2008]. The SGB parameters were: learning rate α=0.10, minimum tree depth ι_{min} = 2, maximum tree

Table 8.1: *Procedure ModelSensitivities()*

Inputs: $\boldsymbol{X}$, $\boldsymbol{y}$, N_d, ψ, N_{scr}
Outputs: $\boldsymbol{\zeta}$
1. for i = 1 to N_d:
2. $\quad \zeta_i = 0$
3. $\quad$ Repeat N_{scr} times:
4. $\quad\quad \boldsymbol{X_{scr}} = \boldsymbol{X}$ except randomly permute row i (for d_i)
5. $\quad\quad \boldsymbol{y_{scr}}$ = $\psi(\boldsymbol{X_{scr}})$ # simulate model
6. $\quad\quad \zeta_i = \zeta_i + nmse(\boldsymbol{y}, \boldsymbol{y_{scr}})$
7. $\quad \zeta_{sum} = \sum_{i=1}^{d} \zeta_i$
8. $\quad \zeta_i = \frac{\zeta_i}{\zeta_{sum}}$, $i = 1..N_d$
9. $\quad$ Return $\boldsymbol{\zeta} = \zeta_i$, $i = 1..N_d$

depth ι_{max} = 7, target training erro rϵ_{targ} = 5%. Number of scrambles N_{scr} = 500. The build time for the SGB on modeling the objective GBW was about 15 s on a 2.0 GHz Linux machine, returning a model containing 282 CARTs. The impact extraction from the model took about 25 s.

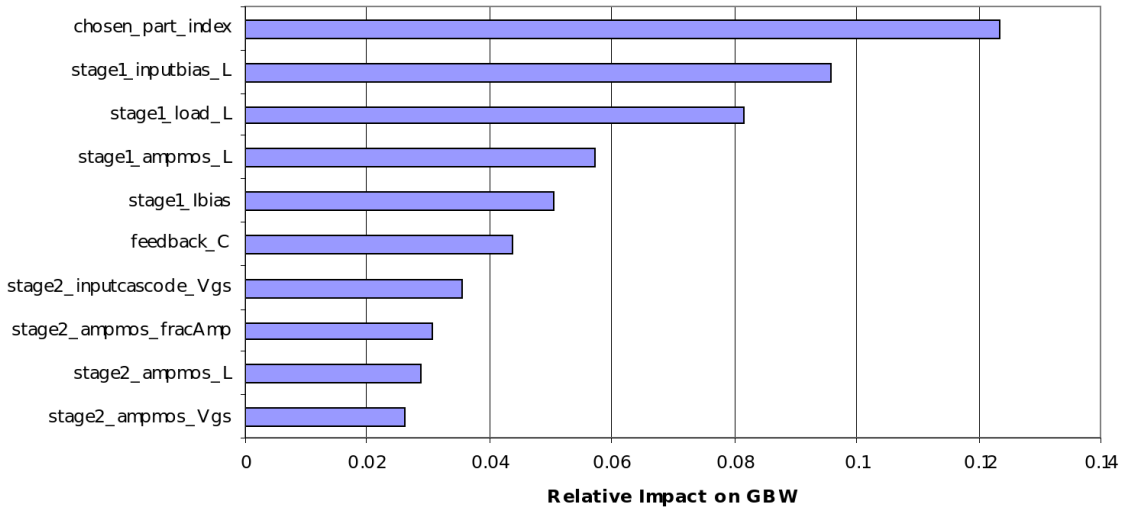


Figure 8.6: *Global nonlinear sensitivity of GBW with respect to topology, sizing, and biasing variables, for the 10 most important variables*

Figure 8.6 illustrates the ten variables that impact the GBW most. We see that the most important variable is $chosen\_part\_index$, which selects one vs. two stages. The variables that are commonly associated with the GBW of opamps – the bias current of the first stage and the size of the compensation capacitance – also show up. Interestingly, the figure also indicates a large influence of the length of the transistors in the first stage (input, folding and load). This can be readily explained: these lengths directly influence the impedance on the internal nodes, and hence the location of the non-dominant pole. The phase margin requirement (> 65°) translates into the requirement that this non-dominant pole frequency

is sufficiently higher than the GBW ($\approx$ 2x) [San2006]. It is also interesting to see that for the value of the GBW, only one topology parameter made it into the top ten variables; sizing parameters comprise the other nine. This means that once one vs. two stages is chosen, changing the right sizing variables will make the biggest difference to GBW. Of course, the most sensitive variables can be different for different performance metrics, and the designer must consider all metrics.

8.5 Extraction of Analytical Performance Tradeoffs

8.5.1 Introduction

Designers often manually manipulate equations that relate performance tradeoffs [Raz2000]. Equations facilitate understanding because a direct relationship is expressed explicitly. Furthermore, the model can be easily written, which makes it easy to manipulate by hand or in a math program like Mathematica. The problem is that hand-derived analytical expressions based on 1^{st}- or 2^{nd}- order approximations may have little relation to the process technology, which leads to errors of 20% or more.

Some recent work has hinted towards automation. [Sme2003a] did a single-topology multi-objective optimization run, then generated a blackbox model of the performance tradeoffs. Unfortunately the model is blackbox (giving no insight), and a single topology does not adequately describe the capabilities of the circuit type (e.g. opamp) for a process technology. The authors of [Vog2003] conducted a thorough manual search of A/D converter technical publications to get Pareto front data across many topologies, then created a whitebox model for the performance tradeoffs. This, of course, was highly time-consuming, becomes obsolete with any new process technology. Furthermore, the models themselves were restricted to a fixed template.

Related work in *symbolic modeling* also hints at the motivations here. Recall from chapter 4 that symbolic modeling uses simulation data to generate interpretable mathematical expressions for circuit applications, typically relating the circuit performances to the design variables. Like symbolic analysis, the applications of symbolic modeling include knowledge acquisition and educational / training purposes, design space exploration, repetitive formula evaluation, and more. A tool that can help a designer improve his understanding of a circuit is highly valuable, because it leads to better decision-making in circuit sizing, layout, verification, and topology design, regardless of the degree of automation [Mcc2005a]. Symbolic model extraction tools like CAFFEINE [Mcc2005a] have demonstrated how useful insights can be found.

Accordingly, the aims of this section are to (a) automatically extract analytical performance-tradeoff equations that are (b) in agreement with process technology, (c) span a huge set of possible topologies, (d) without being restricted to a predefined functional template, and (e) can be automatically generated with each new process.

8.5.2 Analytical Performance Tradeoffs: Approach

To meet the above aims, we propose the following approach:

1. Given: a MOJITO-generated Pareto-optimal set $Z = \{\phi_1^*, \phi_2^*, \ldots, \phi_j^*, \ldots, \phi_{N_Z}^*\}$. Let $\boldsymbol{Y} = \{\boldsymbol{y_j}\}$, $j = 1..N_Z$, where $\boldsymbol{y_j}$ = $\boldsymbol{f_j}(\phi_j)$ = $\{f_1(\phi_j^*), f_2(\phi_j^*), \ldots, f_{N_f}(\phi_j^*)\}$. $\boldsymbol{Y} \in \Re^{N_f x N_Z}$.

2. Choose a target performance objective (e.g. GBW) having index I; $I \in \{1, 2, \ldots, i, \ldots, N_f\}$.

3. Let $\boldsymbol{X_{-I}}$ be all rows in $\boldsymbol{Y}$ except I. Let $\boldsymbol{y}$ be the I^{th} row of $\boldsymbol{Y}$. Therefore $\boldsymbol{y}$ has the data of the target objective, and $\boldsymbol{X_{-I}}$ has the data of the remaining objectives.

4. Run CAFFEINE to return a Pareto-optimal set of analytical functions, M. Each function maps $\boldsymbol{X_{-I}}$ to $\boldsymbol{y}$. The Pareto-optimal set trades off model error vs. model complexity.

5. Return the resulting functions M to the user for inspection / analysis.

CAFFEINE (described in chapter 4) automatically generates a set of template-free analytical whitebox models, where each model is on the tradeoff of model complexity vs. modeling error [Mcc2005a]. The CAFFEINE settings used were the same as in section 4.5. The runtime was about 10 minutes a 2.5 GHz Linux machine.

8.5.3 Analytical Performance Tradeoffs: Results and Discussion

Table 8.2 shows the results for equation for GBW as a function of the other performances. We expected the gain to be strongly related to the GBW, and it turns out that a simple linear relation (first model in Table 8.2) between the two will get < 9% training error. That is, a linear relation with gain will explain all but 9% of the variation of GBW. But for a better fit, i.e. to explain the variation with better accuracy, more complex nonlinear relations are needed. The next level is an inverse relationship of GBW with $\sqrt{gain}$. Slew rate must also be included for reasonable performances-tradeoff model (last model in Table 8.2). Interestingly, dynamic range and power are not needed to get within 4.1% training error. Cross-examination with the scatterplots (Figure 8.3) confirms that the strongest tradeoffs are indeed among gain, GBW, and slew rate.

Table 8.2: *Whitebox models to capture performance tradeoff.*

Train error	$log(GBW)$ **Expression**
8.7 %	$10.28 - 0.049 * gain$
7.3 %	$5.65 + 86.5/gain + (2.92e\text{-}11) * slewrate$
6.8 %	$5.72 + 80.2/gain + (4.75e\text{-}6) * \sqrt{slewrate}$
5.7 %	$7.30 + 47.76/gain - (3.430e\text{+}3)/\sqrt{slewrate}$
4.1 %	$4.48 + 24.9/\sqrt{gain} - (8.60e\text{+}6)/(gain^2 * \sqrt{slewrate})$

8.6 Conclusion

This chapter has presented a methodology to help designers maintain or obtain their expert insights in the topology-sizing-performance relationship, which is a challenge due to changing process nodes, circuit design advances, and more.

The approach is to take a data-mining perspective on a Pareto-optimal set of sized analog circuit topologies as generated with the MOJITO tool, and apply specific knowledge extraction techniques to specific question types. They are: extract a specs-to-topology decision tree (via CART [Bre1984]); do global nonlinear sensitivity analysis on topology and sizing variables (via SGB [Fri2002] and a variable-scrambling heuristic inspired by [Has2001]); and generate analytical whitebox models to capture tradeoffs among performances (via CAFFEINE [Mcc2005a]). These techniques are all complementary, as they answer different designer questions outlined in Figure 8.2[1].

Once extracted, the knowledge for a circuit type on a process node can readily be distributed to other designers, without need for more synthesis (see Figure 8.1).

Results have been shown for operational amplifier design on a database containing thousands of Pareto-optimal designs across five objectives. As a final note, we must emphasize once again that these techniques are meant to augment the designer experience, not to replace it. The designer is key.

This is the final chapter on design and insight aids for topology selection and design... without considering process variations. The next chapter extends MOJITO to consider *process variations* for both aiding topology design *and* getting insight.

[1]Since no tool can answer *all* questions, some unanswered questions will undoubtedly remain. But being an expert, the designer will undoubtedly find ways to answer all questions that matter.

Chapter 9

Variation-Aware Circuit Topology Synthesis and Knowledge Extraction

Divide each difficulty into as many parts as is feasible and necessary to resolve it.
–Rene Descartes

9.1 Introduction

This chapter aims to simultaneously address problems which previous chapters attacked separately. In chapter 3, SANGRIA did variation-aware design (sizing). In chapters 6-7, MOJITO did topology synthesis (nominally). This chapter does both: variation-aware topology synthesis. By combining the two features at once, we bias the MOJITO search towards topologies that are robust, resulting in the MOJITO-R tool [Mcc2008f]. It also does knowledge extraction on the variation-aware synthesis results, using tools from chapter 8.

The rest of this chapter is organized as follows. Section 9.2 gives the problem specification. Section 9.3 gives background on variation-aware topology synthesis. Section 9.5 describes MOJITO-R, the proposed approach. Section 9.6 gives experimental results which include knowledge extraction. Finally, section 9.7 concludes.

9.2 Problem Specification

This thesis' variation-aware sizing work focused on a single objective (yield or Cpk), and the topology synthesis work of chapters 6 and 7 had multiple performance objectives (but not yield / Cpk). An interesting thing happens when one treats performances *and* yield as objectives: *each* given design candidate has its own Pareto-optimal set which trade off performance(s) with yield. For example, let us have gain as one objective (to maximize) and yield as the other. Then, an unattainably large gain value (constraint threshold) will give a yield = 0.0%. But yield will rise as we loosen the gain constraint (0.0% < yield < 100%), and eventually the gain constraint will always be met (yield = 100%). This whole tradeoff of gain vs. yield is possible from a single design. We can generalize to have more

than one performance objective too. From the perspective of a product manager, yield-performance tradeoffs are highly useful, as some slightly arbitrary specifications can be loosened in order to enhance yield. [1]

This chapter treats both performances and yield as objectives. Therefore, each design candidate will have its tradeoff curve, and all design candidates' tradeoffs are merged to form the overall tradeoff.

The algorithm's aim is formulated as a constrained multiobjective optimization problem:

$$\begin{array}{rll} minimize & f_i(\phi) & i = 1..N_f \\ s.t. & g_j(\phi) \leq 0 & j = 1..N_g \\ & h_k(\phi) = 0 & k = 1..N_h \\ & \phi \in \Phi & \end{array} \tag{9.1}$$

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. Note that this set Z may contain multiple topologies.

f_1 is the objective to maximize yield, and $f_2, f_3, \cdots, f_{N_f}$ are the circuit performances (λ's in the notation of chapter 2) which are being maximized / minimized (e.g. maximize gain, minimize power, etc.). The remaining performances are treated as constraints. As discussed, the value of yield for a given ϕ is dependent on the values of $f_2, f_3, \cdots, f_{N_f}$.

9.3 Background

This section is a brief review of the literature in robust circuit topology synthesis, from the perspective of analog CAD, and of genetic programming (GP).

9.3.1 Robust Topology Synthessis in Analog CAD

First, we review work in the analog CAD literature. To our knowledge, there is no prior research which simultaneously does topology search and considers process variations, let alone using SPICE in the loop. So, we resort to examining the work in variation-aware automated sizing, as covered in depth in chapter 2, because some of the techniques there may generalize to non-vector search spaces.

The global commercial optimizers (e.g. [Snps2005, Cdn2005b]) as well as some academic research [Phe1999] use FF/SS model corners which, as has been mentioned in chapter 2, is less accurate. On-the-fly automated corner discovery like [Och1999] is too slow .

Plain Monte Carlo sampling with SPICE is also too slow, unless a faster performance estimator is used [Deb1998], which is then inaccurate.

[1] An elaboration of this concept is "yield binning" in which a single design's manufactured chips are binned into several performance categories, e.g. CPUs meeting clock speeds of >3.0 GHz, >2.5 GHz, >2.0 GHz, >1.5 GHz, and throwaway. To maximize revenue, each die is put into the highest-performance bin possible, where it has the highest price point.

One could use safety margins [Phe2000], but this means overdesign, and the degree to set safety margins is unclear. One could build regression models [Sch2001], but in general structural synthesis search spaces, the notion of input vector is poorly defined. That said, in section 8.4 we have shown how the MOJITO tree search space can be viewed as a vector-valued space. Unfortunately, for variation-aware synthesis we must also consider process variables as input dimensions, which causes the number of input variables to be too large to build sufficiently predictive models. One could do nominal synthesis to generate a tradeoff of designs, and after that take small steps in a manufacturing-aware space [Sme2003b], but that only works if there is tight correlation between nominal and robust designs.

9.3.2 Robust Topology Synthesis in Genetic Programming

We also review robust synthesis techniques from genetic programming (GP).

The technique of [Tel1997] could be applied to robust design: in a tournament selection scheme, only take as many Monte Carlo samples as needed to have confidence that one design is better than another. The problem is, two nearly-identical individuals have a "racing" effect where each individual needs to get fully evaluated in order to differentiate from its neighbor. This means that there are have expensive competitions between two relatively unfit individuals. And, in general, all attempts to *refine* designs will be computationally expensive.

Another approach is robust hierarchical fair competition (robust HFC) [Hu2005a]. Recall that HFC segregates individuals by layers of different *fitnesses*. *Robust* HFC, evaluates all individuals at all layers nominally, *except* it takes 10 Monte Carlo samples for individuals on the top, highest-fitness layer. It has issues as follows. First, there may be significant distance from a design region which does well on nominal evaluations only, to a design region which does well when accounting for statistical variations. In robust HFC, all the individuals entering the top layer start in the good-at-nominal region. They must each make that journey from the good-at-nominal region to the good-at-robust region, which will be a computationally expensive journey requiring many steps if the distance between the design regions is large. The effect is even worse than that. Because HFC segregates by fitness layers, the first reasonably good individuals to enter the top layer will get refined. The vast majority of new individuals that enter the layer afterwards will do poorly because they are in the good-at-nominal region, and will die out when competing against the older individuals. The effect is that robust HFC could easily get stuck in a local optimum, despite its fitness-layering scheme to avoid exactly that.

9.4 Towards a Solution

The next section will present our approach, MOJITO-R, which overcomes the issues of robust topology synthesis that we see in analog CAD and in GP. In some ways MOJITO-R is similar to robust HFC, in that a distinction is made between the degree of Monte Carlo evaluation in different population layers. But it is different in order to overcome the issues of robust HFC:

- First is the robust HFC issue of having a sharp jump from nominal to robust evaluations at the very top layer, which hinders the ability to globally explore robust design regions. MOJITO-R overcomes this by having a *smooth* transition from nominal to fully-evaluated Monte Carlo designs: the lowest population layer has just nominal evaluations, but subsequent higher layers have a gently increasing number of Monte Carlo samples, until the top layer which has the full number of samples. This is an instantiation of the "structural homotopy" concept introduced in chapter 3 for the SANGRIA algortihm.
- The second robust HFC issue, that nominal individuals entering the top layer almost always lose out to existing top-layer individuals, is naturally handled by the combination of the structural homotopy framework and segregating individuals by *age*, not fitness (ALPS). The structural homotopy framework means that there is no dramatic change in the objective function, so up-and-coming individuals are not assaulted by dramatically-better individuals that were already in the layer. The ALPS framework helps further: stagnant individuals will get removed from each age layer once they become too *old* for the age layer (we saw this effect happening often in SANGRIA's experimental results of section 3.6).

  The next section will describe MOJITO-R in more detail.

9.5 Proposed Approach: MOJITO-R

MOJITO-R [Mcc2008f] can be viewed as a combination of MOJITO [Mcc2007] and SANGRIA [Mcc2008e]: it uses the search space and core multi-objective algorithm of MOJITO, and the "structural homotopy" idea of SANGRIA to handle variation issues. MOJITO-R returns a Pareto-optimal set of sized topologies, which trade off yield and performances. There is an opportunity to extract knowledge from this Pareto-optimal set, which we also explore.

The high-level structure of MOJITO-R is shown in Figure 9.1. The high-level algorithm and sub-algorithms are identical to that of MOJITO in section 3.5.2, except:

- Evaluations at higher levels get progressively tightened as they progress towards the top level (i.e. structural homotopy).
- For each design (sized topology), a yield-performances tradeoff is generated by (1) sweeping through all combinations of specifications, and computing yield for each vector of performance values, then (2) applying nondominated filtering on the resulting combined yield-performance vectors.
- At every generation, the final nondominated set is updated by merging the tradeoffs of the fully-evaluated individuals (top two layers), then applying further nondominated filtering.

To demonstrate the generality of structural homotopy, for MOJITO-R we set the prescribed evaluations for each age layer slightly differently than in SANGRIA. In SANGRIA, there were two axes on which to tighten the objective function: (1) ac/dc vs. ac/dc/transient/other, and (2) nominal vs. corners. In MOJITO-R, there is just one axis: number of Monte Carlo

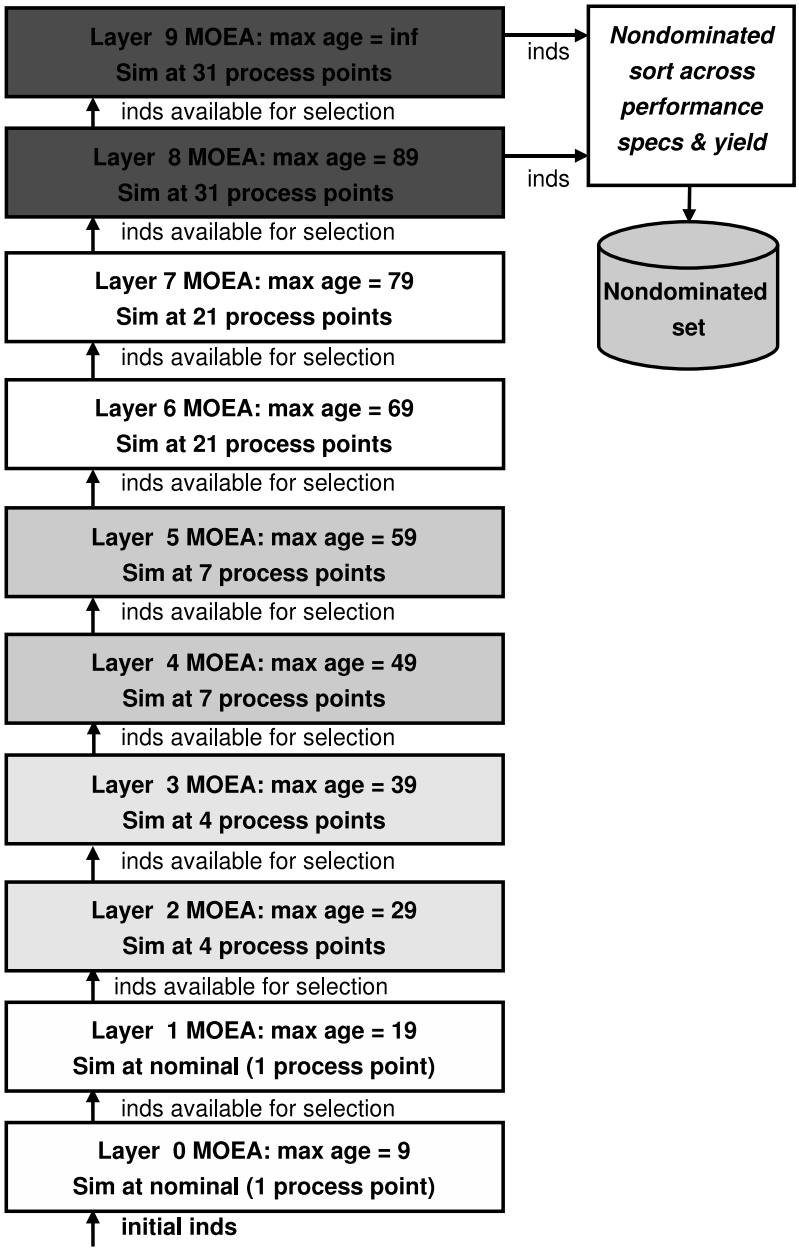


Figure 9.1: *MOJITO-R combines SANGRIA's structural homotopy strategy with MOJITO's multi-objective search across the topology space.*

samples (process points). Each layer in MOJITO-R simulates at *all* testbenches: ac, dc, transient, and other. The breakdown of process points is: nominal vs. 4 process points vs. 7 process points vs. 21 process points vs. 30 process points[1].

This generality is important for real-world use of SANGRIA and MOJITO-R, because it is not always clear *who* would determine which simulations are done at the lower levels, and which simulations are done on all layers. The choice could be the user, or the tool provider, or both. For a reasonable tradeoff between usability and control, the default could be set by the tool provider, but the users who would like to have the extra control would have access to it. And in fact it can be easily auto-determined for certain approaches, such as MOJITO-R's approach where each layer has merely a different number of "corners" across *all* testbenches.

We can readily calculate 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 come 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 + 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: in our experience it takes on average 500 simulations to generate a decent initial individual (using the algorithm in Table 7.6). 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 evaluation-units, MOJITO-R to 500/10 (init. gen.) + 128 (baseline) = 178 evaluation-units, and therefore **MOJITO-R is only 3.0 times slower than MOJITO**. For comparison: a brute-force Monte Carlo implementation in which all individuals are evaluated on 30 Monte Carlo samples is 30 times slower than MOJITO, and 10 times slower than MOJITO-R.

9.6 Experiments

This section describes experimental results from running MOJITO-R, including knowledge extraction, on operational amplifiers.

9.6.1 Generation of Database

This subsection describes the setup to generate the sized-topologies database using MOJITO-R, having tradeoffs across both performances and yield. We use the same experimental settings as section 7.6.1, with some values updated as in section 9.5.

[1]These numbers were chosen using the following reasoning. 30 process points gives reasonable accuracy for the context of a yield optimizer, and is the value we also used in SANGRIA (chapter 3). 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.

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 (which is palatable for an industrial setting). 242 generations were covered, traversing 3528 possible topologies and their associated sizings. 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 100% yield (estimated).

9.6.1.1 Performances on Whole Pareto Front

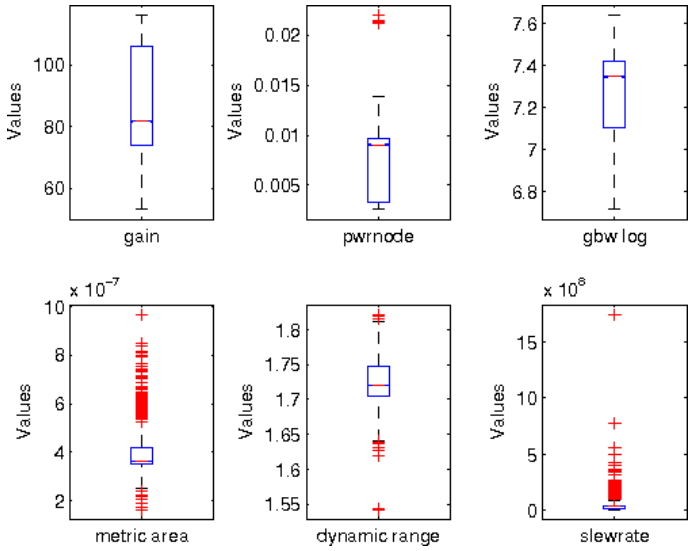


Figure 9.2: *Box plots summarizing the performance distributions across whole Pareto front.*

To begin with, we will get a rough feel for the distribution of the performances of the Pareto front. Figure 9.2 gives a box plot for each metric. We see that gain has a range from about 55 dB to about 115 dB, with the bulk of designs having values between 75 dB and 110 dB. For power, some designs get as low as 2 mW, most designs are between 2.5 mW and 10 mW, and some designs consume more than 20 mW. The minimum GBW is around $10^{6.7}$ = 5.0 MHz, and the maximum is about $10^{7.6}$ = 39.8 MHz. The distributions for area, dynamic range, and slew rate are also shown.

Let us now examine the database more directly by looking at each raw point's performance, in 2-D scatterplots which also give insight into overall tradeoffs between two performances *and* yield, at once. Figure 9.3 illustrates. Each diagonal coordinate gives a histogram for a performance metric (or yield), and other entries in the grid give a 2-D scatterplot of performance / metric values.

We can note many interesting things. The histogram for gain indicates a bimodal distribution of gain values, with one peak around 55 dB and another around 110 dB. These two clusters of data appear in other plots where gain is on one axis; for example in the

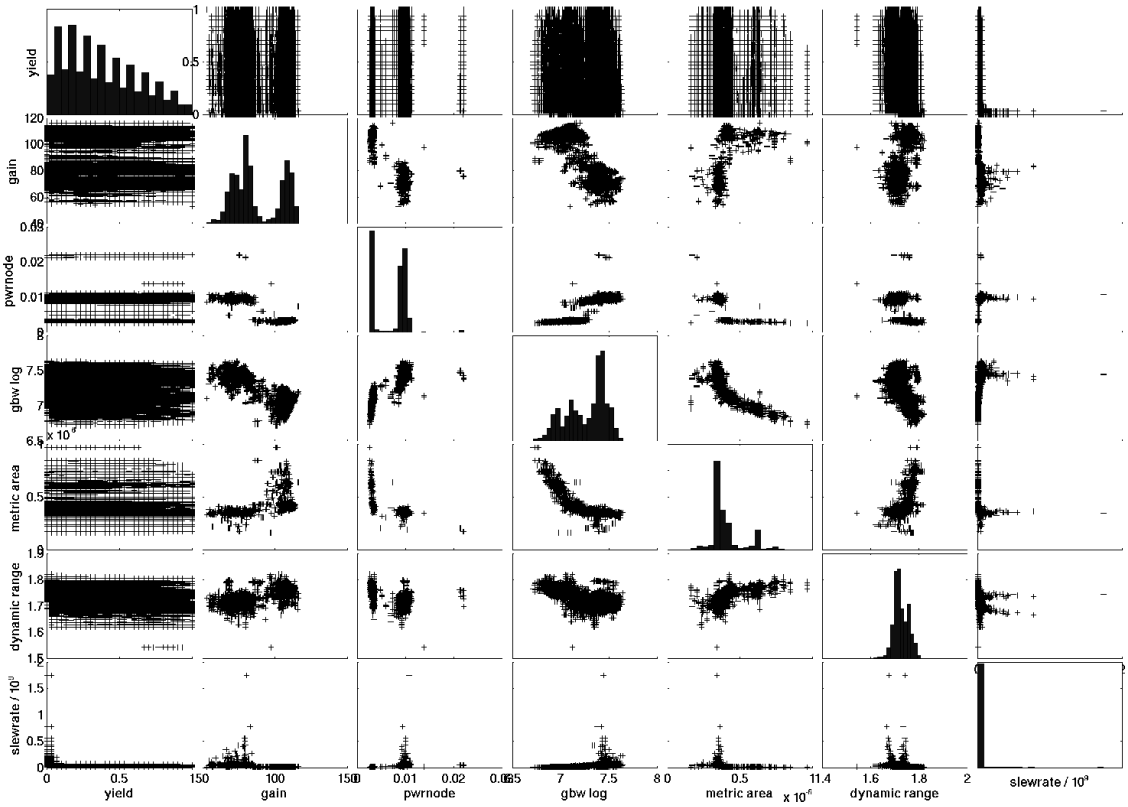


Figure 9.3: *2-D scatterplots and histograms of the whole Pareto front. Each "+" is a different nondominated entry in the Pareto front. There are 78,643 entries in total.*

gain vs. power plot we see one distinctly higher-power group, and one distinctly lower-power group, plus some outliers. We cannot ignore the outliers as they are also points on the Pareto front, and are therefore "best" in their own specific way. Interestingly, it is the higher-gain cluster which has lower power, indicating that there is not a strong tradeoff between gain and power. On the other hand, the plot of GBW vs. gain indicates a strong tradeoff: one can have either high GBW or high gain, but not both. There is also a strong tradeoff between area and gain: higher gains typically take more area. The tradeoff is softer for lower-gain circuits, in which going to substantially lower gain values does not buy much more area. We also see that there is a strong tradeoff between power and area, especially for the lowest-power circuits. Dynamic range and power do not exhibit strong tradeoffs; and in general dynamic range does not have strong tradeoffs with any metric.

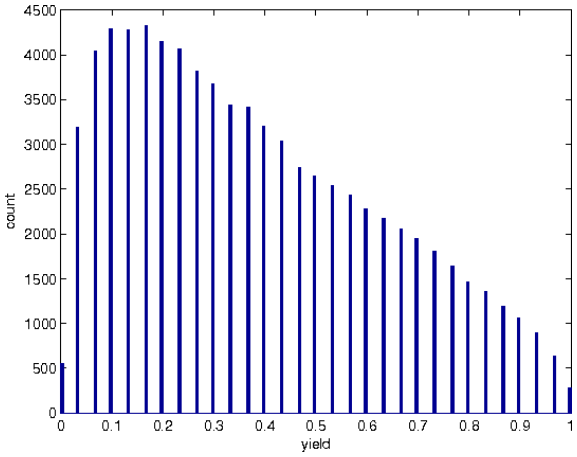


Figure 9.4: *Histogram of the yield values for whole Pareto front.*

Having the yield dimension affects what all these insights mean, so it is important to consider yield versus the other specifications. First, we examine the yield histogram, which is the upper left of Figure 9.3, but which is also zoomed into more detail in Figure 9.4. Note that all points are not 100% yield, therefore the performance tradeoffs we just examined mean that in some cases to hit the extremes of performance, power, and area shown, one will have to settle for a yield of <100%. Also, we see that the histogram peaks at a yield of 10% to 20%, and tapers off to both sides; and the bulk of points have a yield of less than $\approx$ 50%. So the 2-D scatterplot tradeoffs of Figure 9.3 are actually on many points with $\ll$ 100% yield. Note that some points even have 0% yield, in which each point barely does *not* meet the performance specifications.

Let us re-examine the 2-D scatterplots of Figure 9.3, and in particular the subplots 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 what that means is that each performance value on its own is achievable no matter what the yield requirement is, though there of course will be tradeoffs with other performance values. One notable exception is slew rate vs. yield (bottom left): it shows that the only way to achieve significantly higher values of slew rate is to have yields of <10%.

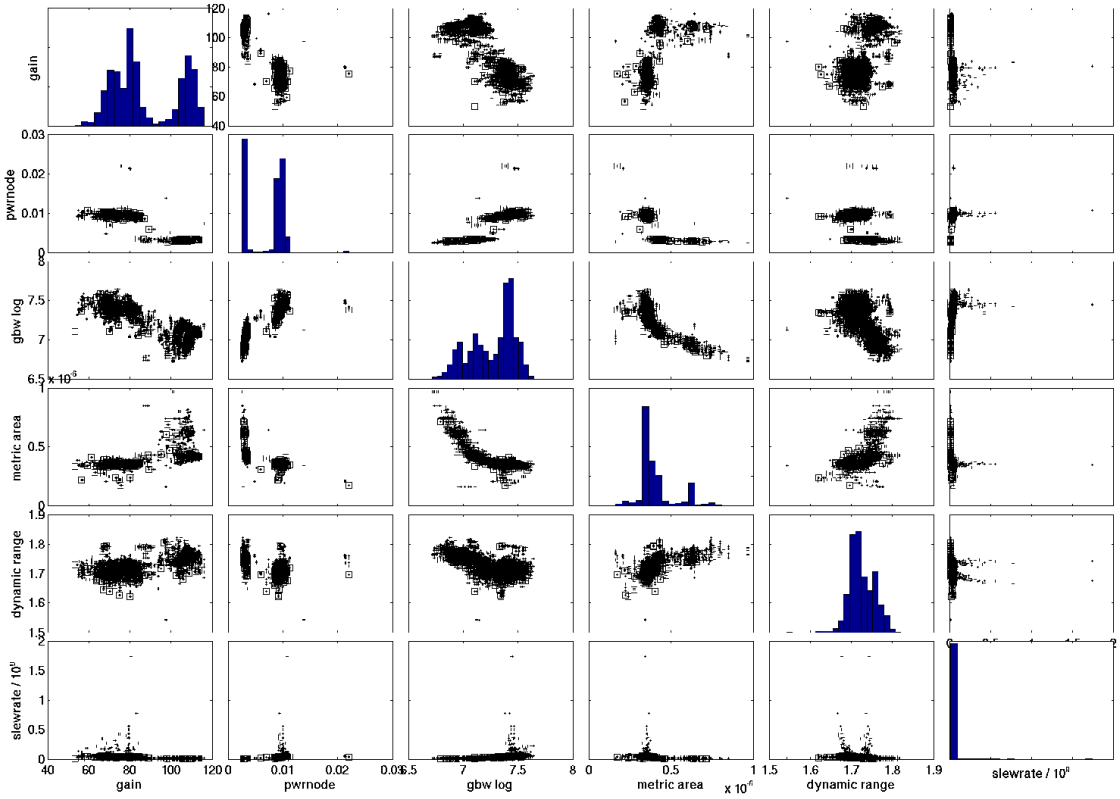


Figure 9.5: *On the whole Pareto front, with the 100%-yield points highlighted by squares.*

We might consider interpreting the 2-D scatterplots on non-yield dimensions according to the following intuition: wherever the tradeoff is strong, the most aggressive edge of the curve has the 0%-yield points, and the least aggressive edge of the curve has the 100%-yield points. In fact, we can test this insight. Figure 9.5 illustrates the whole Pareto front, and highlights the 100%-yield points with squares. We see that boxes are not just on the edges of clusters in the 2-D plots, but throughout each 2-D plot including some of the outliers. However, one must remember that the tradeoffs are not just in two performance metrics but in six; so in six dimensions the 100%-yield points are on the least-aggressive edge and the 0%-yield points are on the most aggressive edge. Put in another way, the performances tradeoff curve represented by the 100%-yield points are less aggressive than the performances tradeoff curve represented by the 0%-yield points.

9.6.1.2 Topologies on the Whole Pareto Front

In the Pareto front, there were nine different topologies. They are illustrated in Figure 9.6. 982 sized versions of the topologies expanded into 78,643 Pareto-optimal points. 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.

Some topologies turned out to cover much more of the Pareto front than others. Specifically, 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. Table 9.1 gives a complete count per topology in the whole Pareto front (second column).

Table 9.1: *Topology Count in Pareto front*

Topology Label	**Number of Instances in Whole Pareto Front**	**Number of Instances in 100%-Yield Pareto 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

9.6.1.3 100%-Yield Pareto Front

We have briefly discussed the 100%-yield points, but this section examines them more closely. Aiming for 100% (estimated) yield is common in analog circuit design; usually designers just view this as solving at all corners. Correspondingly, we can filter down the points of the Pareto-optimal front down to just the 100%-yield designs. 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 9.5 showed the 100%-yield designs highlighted among the whole Pareto front, Figure 9.7 shows just the 284 100%-yield designs. There are a few notable details. First, the plot is far simpler since the 284 points are only 0.36% of all the points. Second, the general trends are largely the same, including the clusters. Third, the performance values are less aggressive; most notable is the slew rate with a maximum value that is about three times smaller compared to when <100%-yield designs are allowed. Some tradeoffs are more refined, such as the tradeoff between area and GBW. The slew rate tradeoffs (for lower-performing slew rates) are now more visible; we see that slew rate has a loose tradeoff with gain and with dynamic range, and a relatively tight tradeoff with power. But it has little tradeoff with GBW and area: higher slew rates turn out to be correlated with larger GBW and smaller area values.

The third column of Table 9.1 gives the count for each topology. Note that some topologies never achieved 100% yield. This is interesting because it means that when

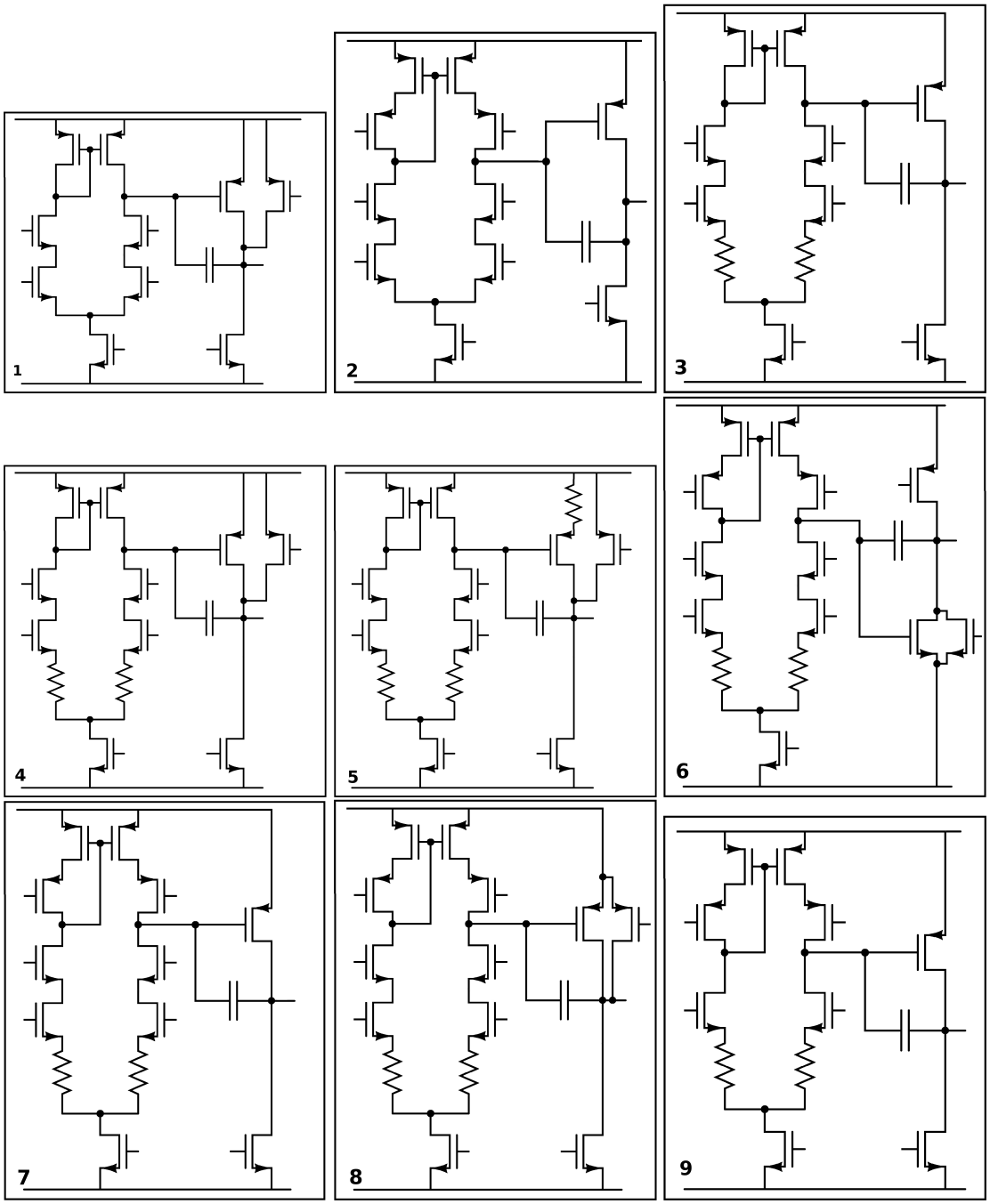


Figure 9.6: *Topologies in the MOJITO-R run's Pareto front.*

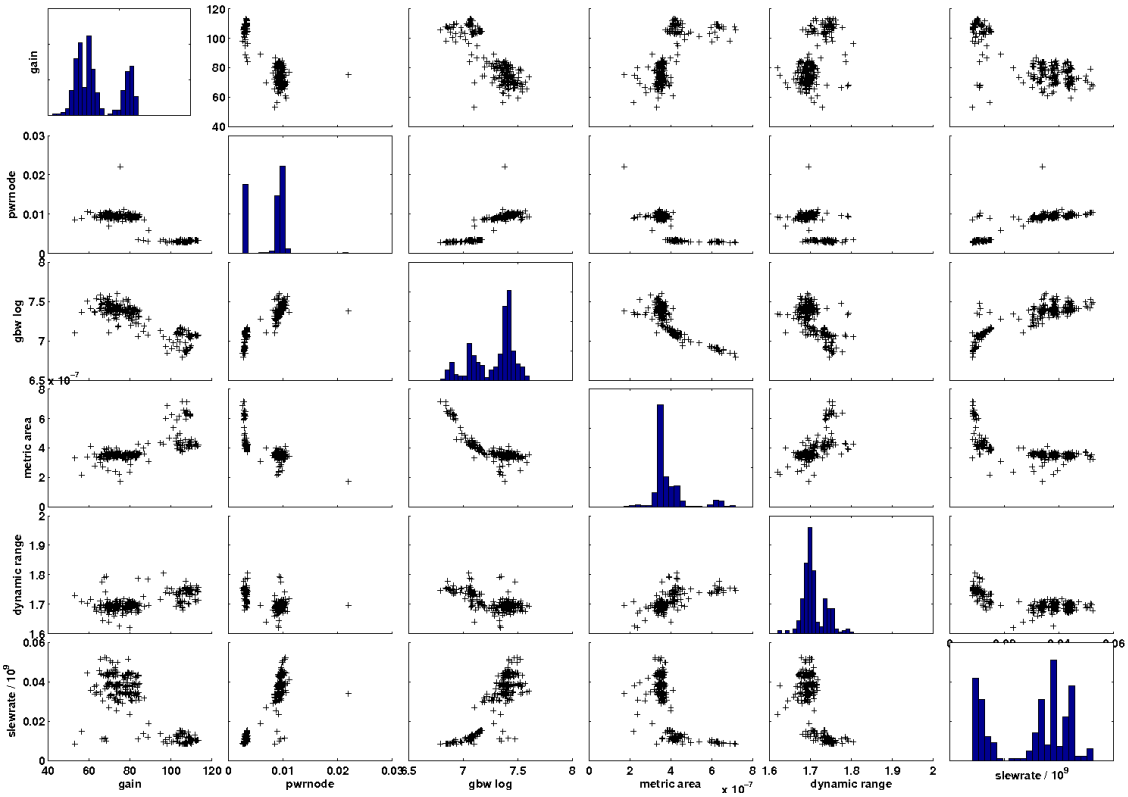


Figure 9.7: *The 100%-yield points in the Pareto front.*

yield matters, fewer topologies are needed. Also note that the topologies with the most 100%-yield entries (third column) are the ones with the most any-yield entries (second column).

9.6.1.4 Per-Topology Highlights on 100%-Yield Pareto Front

So far in this section, we have examined performances and topologies separately. Since it is highly useful to learn how they relate, we now examine the topology-performance relation by highlighting a specific topology in the 2-D scatterplots. To that extent, Figures 9.8, 9.9, 9.10, 9.11, and 9.12 give the highlights for topologies 1, 4, 5, 7, and 9, respectively. In each respective plot, the squares are the highlight topology and the pluses are the other topologies.

As hinted before, the topologies break into two clusters of performance. Topology 7 defines the cluster with the higher gain, lower power, lower GBW, higher area, higher dynamic range, and higher slew rate (Figure 9.11). Topologies 1, 4, 5, and 9 define the other cluster (other figures). The different clusters are most notable in the power dimension, where there are effectively two different possible powers: 5 mW (for topology 7) or 10 mW (for the rest). We can ask how topology 7's schematic is different from the other topologies; we see from Figure 9.6 that unlike topologies 1, 4, and 5 its input-stage current mirror is a low-voltage variant, whereas the other topologies have a simple

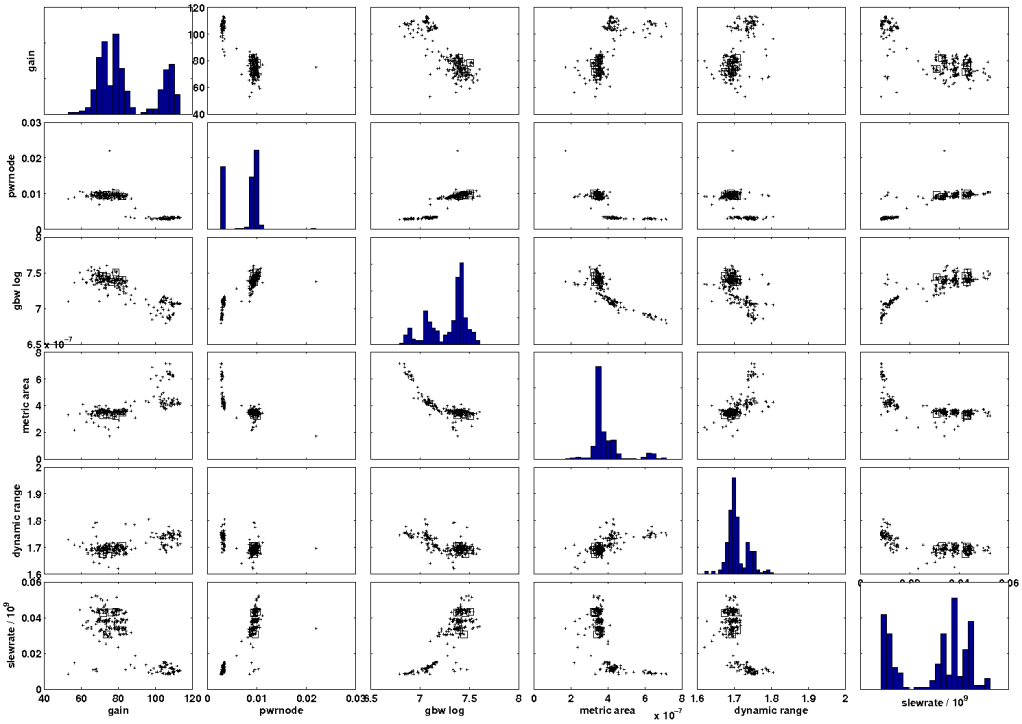


Figure 9.8: *The squares highlight topology 1 in the 100%-yield Pareto front.*

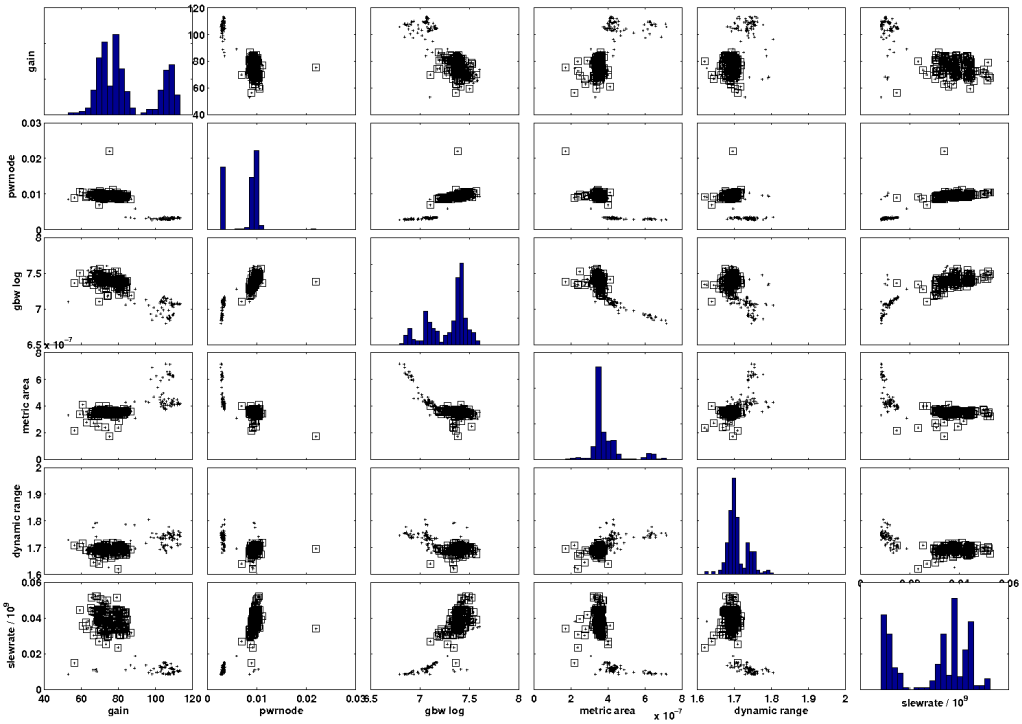


Figure 9.9: *The squares highlight topology 4 in the 100%-yield Pareto front.*

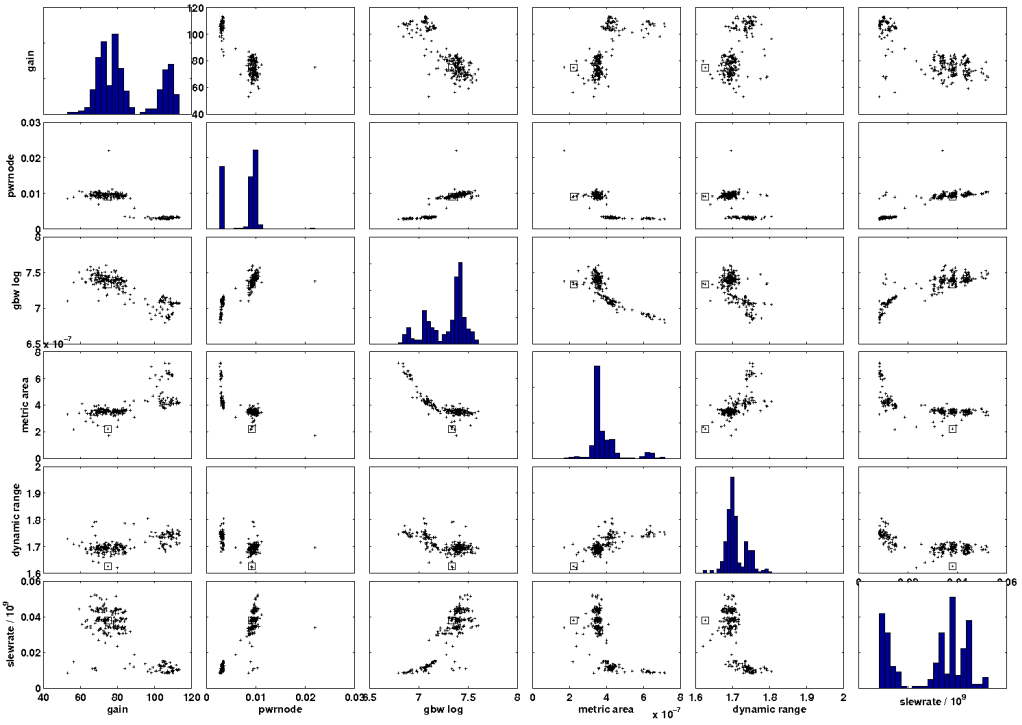


Figure 9.10: *The squares highlight topology 5 in the 100%-yield Pareto front.*

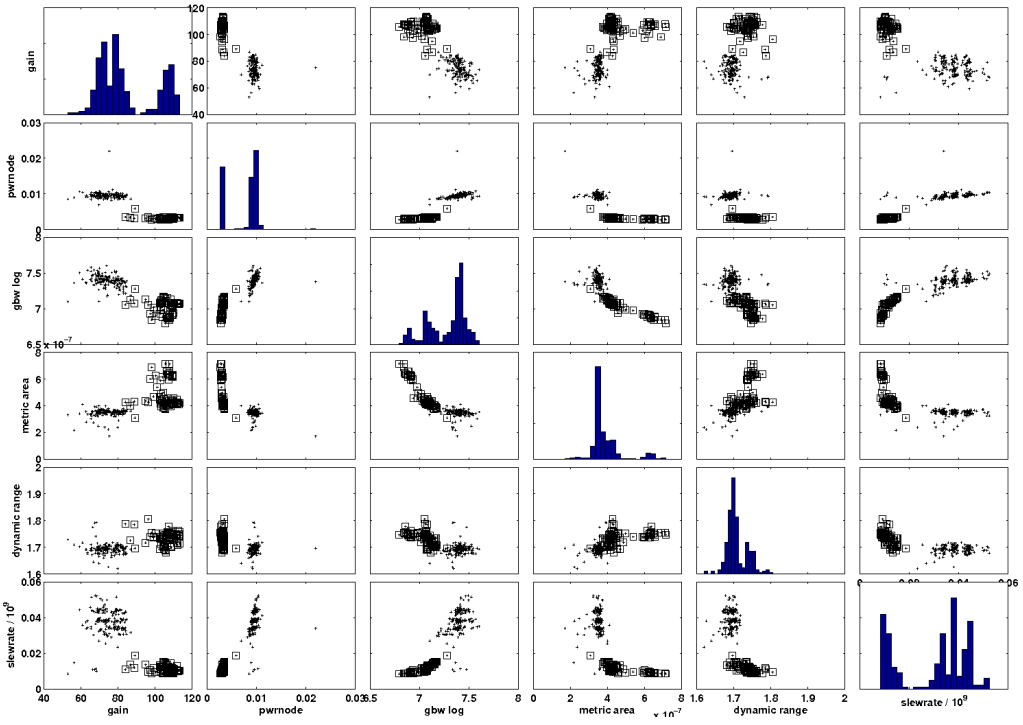


Figure 9.11: *The squares highlight topology 7 in the 100%-yield Pareto front.*

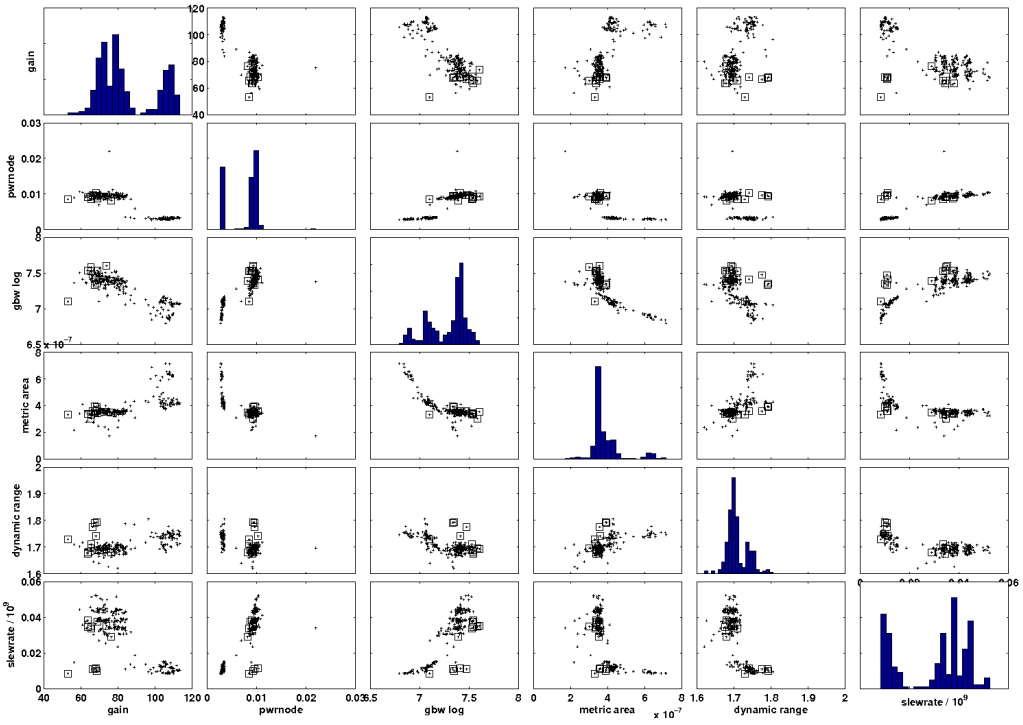


Figure 9.12: *The squares highlight topology 9 in the 100%-yield Pareto front.*

current mirror. Unlike topology 9, it has cascode inputs. So, it is the only topology with a combination of input-stage low-voltage current mirror with cascoding.

Topologies 1, 4, and 5 occupy the most similar performance niche. Topology 4 comprises the bulk of the Pareto front, as evidenced by its high count compared to topologies 1 and 5. Topology 1 is similar to topology 4; the only difference is that topology 1 does not have source degeneration on the inputs. Topology 5 is also very similar; the only difference is that topology 5 has a source degeneration at the output.

Topology 9 occupies a couple niches of the performance space as part of the performance cluster dominated by topology 4. Topology 9 gives the best GBW and dynamic range values of any topology, at relatively low areas.

9.6.1.5 Decision Tree Extraction on 100%-Yield Pareto Front

Recall the knowledge extraction technique of section 8.3, in which decision trees are used to summarize the performance choices between different topologies. The trees give insights that viewing raw data cannot give, especially when there are more than two or three performance metrics (and here we have six). So here, we extract a decision tree from the 100%-yield data following the flow of section 8.3. The tree has performances as input, and topology "class" as output.

The tree extraction took < 5 s. Figure 9.13 shows the resulting tree. There is some pruning in order to give a concise description of the key performance decisions leading to the topology choices. We see that, at the top node, a power requirement of of < 6.4

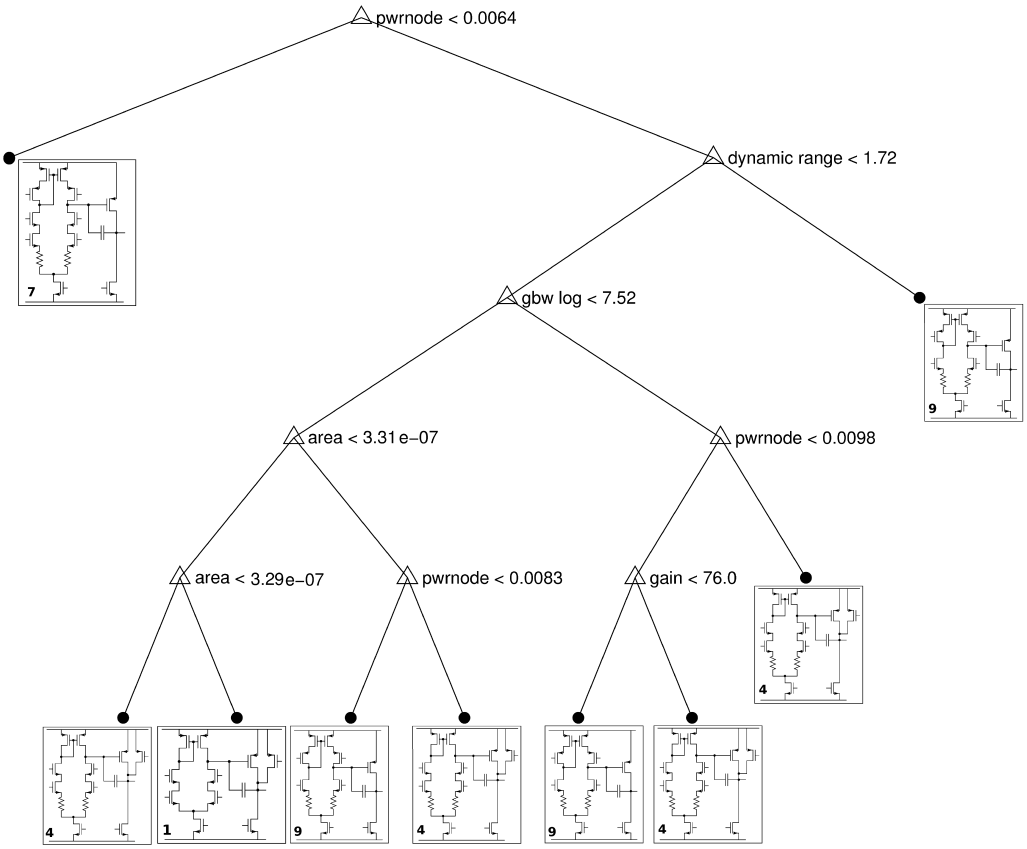


Figure 9.13: *Decision tree for topology choices in the 100%-yield data; minimum number of entries per leaf = 5.*

mW will lead to selection of topology 7. Otherwise, further decisions are needed. We see that topology 7 has a low-voltage current mirror load in its first tage, and its second stage is very simple. The next decision is to examine dynamic range, and if it must be ≥ 1.72 then topology 9 is chosen. There are a series of decisions to usually distinguish between topology 9 and topology 4, and in one case to select the use of topology 1.

The decision tree of Figure 9.13 has only four of the five topologies. To include the fifth topology we turn off the pruning, to get Figure 9.14. This decision tree includes topology 5, which is at the bottom left corner of the tree, and goes through the most performance decisions to get to. What this means is that topology 5 occupies a tiny region of the performances tradeoff space; which is in line with Table 9.1 in which topology 5 has only a single entry.

In summary, once again we have seen how decision trees can provide a complementary perspective into the relationship between topology and performances – here with the twist that all the sized topologies are process-variation-aware.

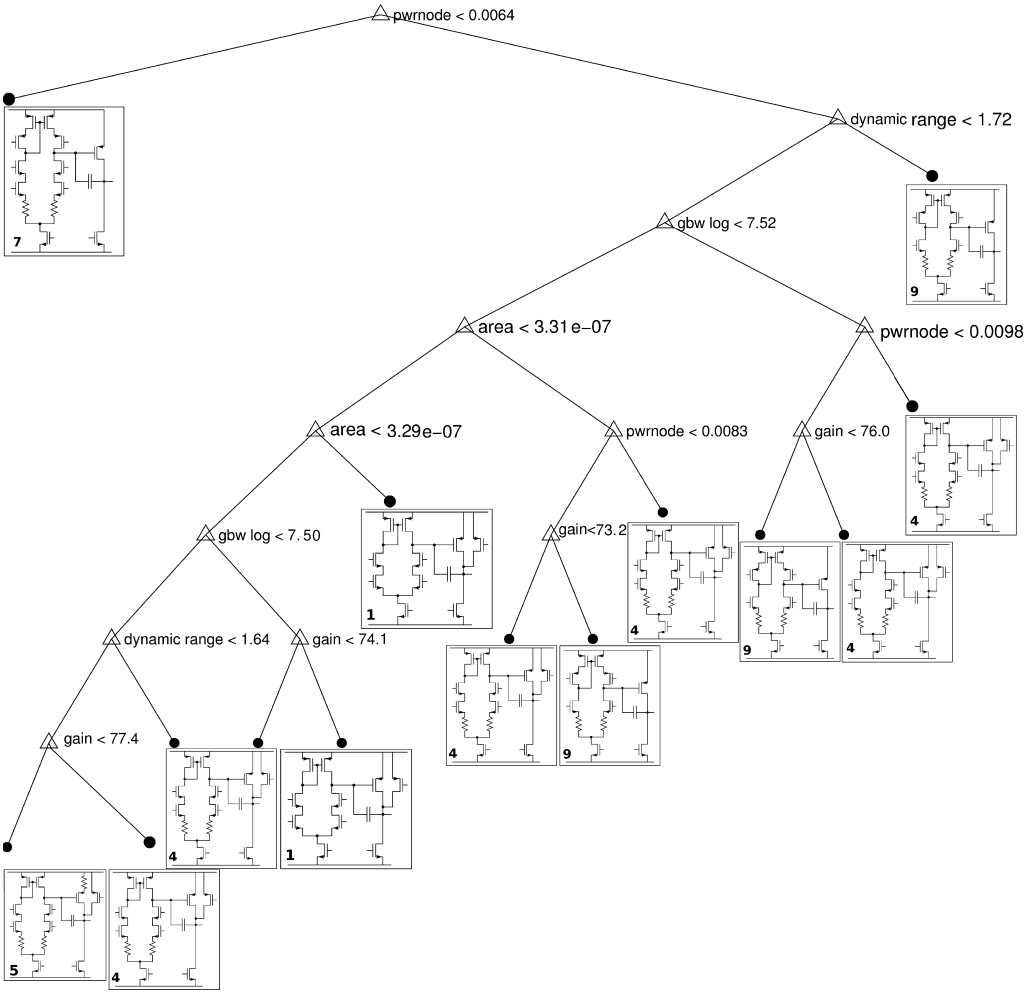


Figure 9.14: *Decision tree for topology choices in the 100%-yield data; minimum number of entries per leaf = 1.*

9.7 Conclusion

This chapter has described MOJITO-R [Mcc2008f], a system which combines SANGRIA's structural homotopy approach to yield optimization with MOJITO's trustworthy, multi-objective approach to structural topology synthesis. The result is an algorithm and system that generates a set of sized topologies which give tradeoffs across several performances *and* yield. We ran MOJITO-R across thousands of possible topologies (plus associated sizings) to generate a tradeoff database holding 78,643 Pareto-optimal points composed of 982 sized topologies, of which 284 have 100% yield. We then examined this database through visualization, and through decision-tree knowledge extraction to learn about performance tradeoffs, topologies which proved useful, and the relationship between topologies and performance.

In the series of successively more-general designer tasks, this thesis covers one more beyond MOJITO-R: variation-aware topology selection and design of *novel* topologies

and functionality. It is embodied in two tools, MOJITO-N and ISCLEs. These tools are discussed in the next chapter.

Chapter 10

Novel Variation-Aware Trustworthy Circuit Topology Synthesis

No great discovery was ever made without a bold guess.
–Isaac Newton

10.1 Introduction

This is an exploratory chapter which makes tentative first steps towards flows for analog circuit design in which the tool helps in designing novel circuit functionality and / or novel topologies.

Previous chapters showed how much flexibility there is in choosing fully-trustworthy topologies, using the MOJITO flow. But sometimes there may be a need for truly novel functionality, or novel topologies. 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 the idea has possible orders of magnitude payoff such 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, to have the computer generate new topologies.

This chapter describes two approaches to novelty:

- **MOJITO-N** [Mcc2008a] finds novel topologies for classes of circuits that typically have existing reference designs (i.e. non-novel functionality). MOJITO-N builds off MOJITO, and adds novelty to known topologies only when there is payoff.
- **ISCLEs** [Gao2008a, Gao2008b] finds novel topologies for classes of circuits without recourse to existing reference designs (i.e. novel functionality). ISCLEs uses the machine learning technique of boosting, which does importance sampling of "weak learners" to create an overall circuit ensemble. ISCLEs has promise to be *naturally* robust to process variations, and to scale with shrinking process geometries.

The rest of this chapter is organized as follows. Section 10.2 gives background. Section 10.3 describes the MOJITO-N approach and results. Section 10.4 describes the ISCLEs approach and results. Section 10.5 concludes.

10.2 Background

Table 10.1 shows synthesis approaches, by capability [Gao2008a]. The approach includes prior approaches, the MOJITO approach of chapters 6-7, and the two new approaches of this chapter, MOJITO-N and ISCLEs).

First, we give some definitions corresponding to the table's columns. "Novel functionality" means that the approach can be set to a new problem just by changing testbenches, which allows for new types of analog circuit functionality. "Novel structures" means that the approach may invent new structures. "Reasonable CPU effort" is for the context of industrial use by a tool user (semiconductor company). "Trustworthy" means that the results are either designer-trusted by construction, or the new structural novelty is easily identifiable by a designer. "Topology variety" means that the set of possible topologies is sufficiently rich that it contains appropriate solution(s) to the target functionality, including problem variants with different objectives and constraint settings.

Table 10.1: *Topology Synthesis Approaches*

Approach	Novel functionality?	Novel structures?	Trustworthy?	Topology variety?	Reasonable CPU effort?
Open-ended	yes	yes	no	yes	no
Open-ended with tighter constraints	yes	yes	no	yes	borderline
Flat pre-specified BBs	no	no	yes	no	yes
Hierarchical pre-specified BBs: MOJITO	no	no	yes	yes	yes
Hierarchical pre-specified BBs plus novelty: MOJITO-N	**no**	**yes**	**yes**	**yes**	**yes**
Boost pre-specified BBs: ISCLEs	**yes**	**yes**	**yes**	**yes**	**yes**

The early approaches of row 1 ([Koza2003], [Lohn1998], [Shi2002]) were very open-ended, having few constraints. Unfortunately, they had prohibitive CPU effort, and results which were not only untrustworthy, but the results often looked strange. More recent efforts of row 2 ([Sri2002], [Das2005], [Mat2007]) added tighter constraints using domain knowledge to improve efficiency and trustworthiness, but there is still no guarantee of trustworthy results or trackable novelty. Early CAD research of row 3 ([Kru1995], [Mau1995]) focused on searching through sets of known topologies, which gave both speed and trustworthy results. Unfortunately, the building block (BB) combinations were specified "flat", which severely limited the number of possible topologies; and there was no clear way to generalize the approaches to more problem types. More recent work - MOJITO [Mcc2007](of chapters 6-7) merges ideas from both fields: it searches through combinations of hierarchically-organized designer-specified analog building blocks, thus

giving a large set of topologies that can be readily applied to common analog design problems which are trustworthy by construction.

The final two rows contain the goals of this chapter. MOJITO-N [Mcc2008a] allows for more open-ended structural novelty, but tracking the novelty explicitly and only rewarding novel individuals that actually improve performance. But since MOJITO-N is constrained to problems that analog designers have attacked, it does not address problems with novel functionality. This gives rise to another opportunity: to determine topologies that can be novel in both functionality and topology, yet trustworthy, and in reasonable CPU effort. This is where ISCLEs [Gao2008a, Gao2008b] comes in.

The rest of this chapter is organized into two key sections. Section 10.3 describes MOJITO-N and its experimental results, and section 10.4 describes ISCLEs and its experimental results. Section 10.5 concludes.

10.3 MOJITO-N Algorithm and Results

10.3.1 Target Workflow for Designers

MOJITO-N [Mcc2008a] is a system for multi-objective and topology sizing, that adds novelty as needed. MOJITO-N fits (with MOJITO) into the designer flow of Figure 10.1 as follows.

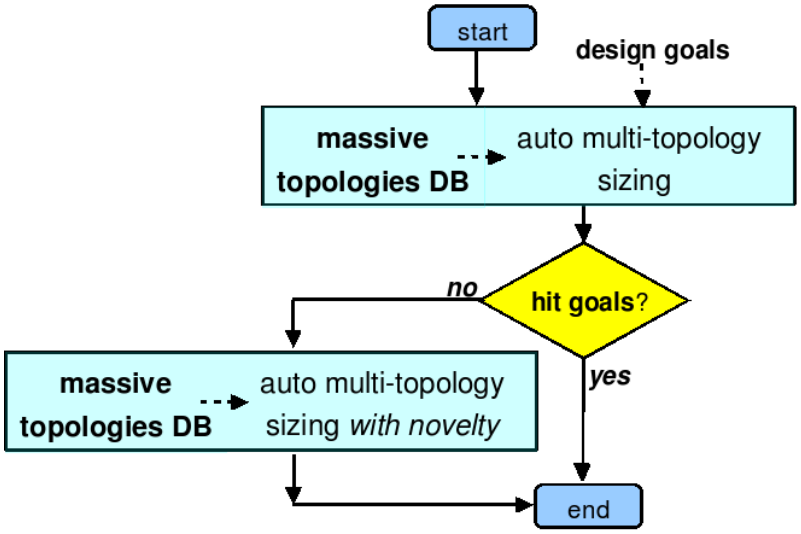


Figure 10.1: *Designer flow for massively multi-topology sizing with novelty. MOJITO is used for auto multi-topology sizing, and MOJITO-N is used for auto multi-topology sizing* with novelty.

When the designer starts the design process, he enters his design goals (performance constraints and objectives) into MOJITO. Because MOJITO has a massive DB of possible topologies, the designer does not need to be concerned with defining them. He runs MOJITO to do the "auto multi-topology sizing." In fact, MOJITO may not even need running if a prior MOJITO results DB exists, which is likely because a multi-objective

results DB for a circuit type only needs to be generated *once* on given a process node. Using a prior DB allows an immediate-turnaround "specs-in, sized-topology out" flow.

If the best topology from MOJITO hits the target goals, then the process is done. If the goals were not hit, then "auto multi-topology sizing *with novelty*" must be invoked, i.e. MOJITO-N. This will help the designer to explore novel topology designs, to augment the manual effort for creative novel topology design.

10.3.2 MOJITO-N Aims

The specifications for the MOJITO-N system, beyond (the non-novelty) MOJITO, are:

- If a topology that is known to be 100% trustworthy will meet the goals, then the tool should return that.
- Only if no existing known topology can meet the 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.

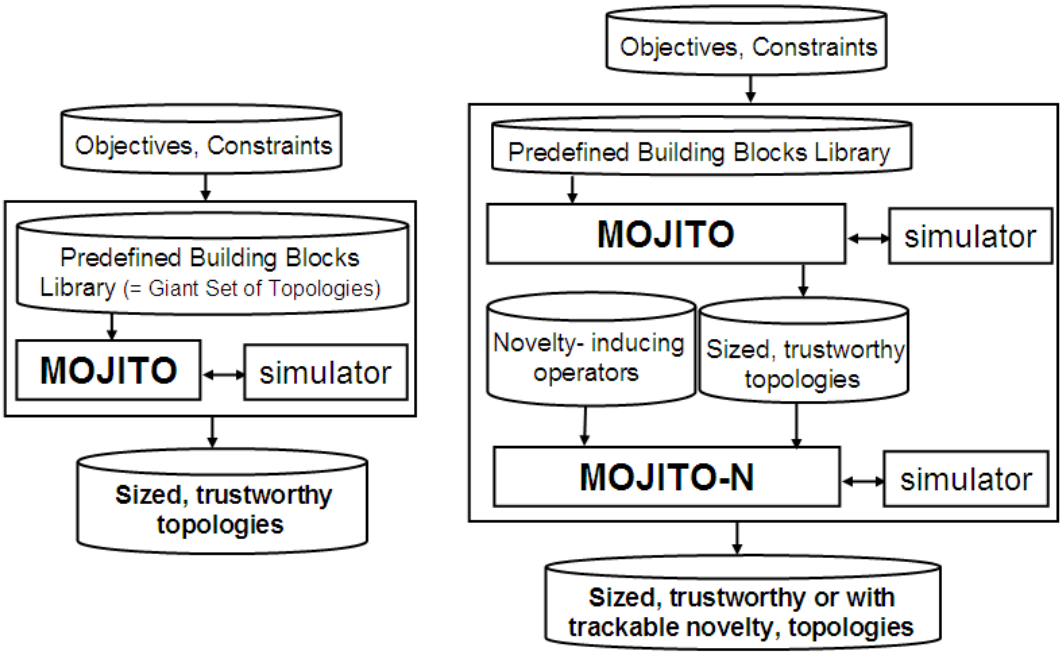


Figure 10.2: *Left: MOJITO inputs and outputs. Right: MOJITO-N inputs and outputs, in relation to MOJITO.*

10.3.3 MOJITO-N Inputs and Outputs

MOJITO's inputs and outputs are shown in Figure 10.2 left, while those for MOJITO-N are shown in Figure 10.2 right. Note how a MOJITO results DB is used as input to the MOJITO-N tool. This makes a big difference in computational effort, because it means

that MOJITO-N only needs to expend effort searching for, and tuning, novel building blocks, not in topologies that only have pre-specified building blocks.

10.3.4 MOJITO-N Algorithm

We have described the target designer workflow for which MOJITO-N is intended, then its specifications and inputs and outputs. So, we can now describe MOJITO-N.

MOJITO-N is composed of the following elements.

- Use MOJITO [Mcc2007] as the algorithmic baseline.
- Use trustworthy designs as the structural starting points. In fact, do a long 100% trustworthy run first (using MOJITO); then add novelty in a follow-on run.
- Create novel designs in the following fashion: 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 [Ryan2005].
- Have a multi-objective framework to manage *trustworthiness tradeoffs*: Let $trust = -1.0 * novelty$, where $novelty$ = number of times that a novel part is used, and a *novel part* is one that has had random structural mutations from a pre-existing part. Then, the trust measure is added explicitly as another objective (to maximize) in the multi-objective search algorithm. In this framework, only novel designs that *help* will be returned. Put another way, 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). The only changes to the MOJITO pseudocode are to measure novelty, and to support the novelty-inducing EA search operators.
- 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, for novel designs to have a fighting chance, MOJITO-N only creates novel designs for the *easiest-competition* age layer, i.e. 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. So that novel individuals only compete against novel individuals, all of layer 0 is populated with novel individuals. (Note: many other possible schemes exist here too, but a key enabler is the ALPS structure).

10.3.5 MOJITO-N Results

This section describes preliminary results for MOJITO-N [Mcc2008a].

The experimental setup was the same as for the non-novelty MOJITO of chapter 7.6.1, except for the following differences. The 100% trustworthy results from the MOJITO run were used as the inputs to the MOJITO-N run, in line with the flow of Figure 10.2 right.

MOJITO-N was run for 15 more generations (15 * 10 * 100 = 15000 more individuals) on one 2.0 GHz CPU, which took about 25 hours on a single 2.0 GHz CPU. The novelty-mutating operators were: add two-port series, add two-port parallel, add three-port parallel, and add transistor with channel in series. The two-port parts available for adding were: capacitors, resistors, NMOS/PMOS diodes, and biased NMOS/PMOS devices (a biased MOS is merely a transistor with a pre-set voltage bias). One more search objective was added: minimize novelty.

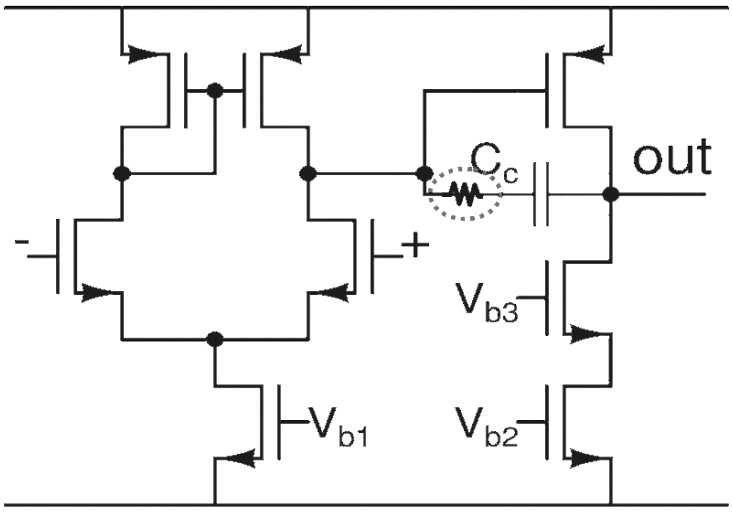


Figure 10.3: *Circuit which MOJITO-N successfully re-invented. The circled resistor in the feedback path was not in the library; MOJITO-N added it. This is a well-known design technique.*

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 C_c had been mutated to include a resistor in series. Figure 10.3 illustrates. This is actually a well-known design technique that one can find in many analog design textbooks [Lak1994, Raz2000]: it is a "zero-compensation" resistor which increases the effective gain from feedback; it does not help the feedforward gain as much because the feedforward path does not get its gain amplified.

10.4 ISCLEs Algorithm And Results

10.4.1 Motivations

ISCLEs has two motivations, one from analog circuit design and one from the general goal of structural synthesis:

- **Analog Area Problem.** As Moore's Law [Moo1965][Itrs2007] captures, the minimum size of the transistors in integrated circuits has been decreasing at an exponen-

tial rate for several decades. For digital design, the incentive to shrink geometries is high: it means simultaneously smaller area, higher speed, and lower power. However, scaling is less beneficial to analog circuits because mismatch, which limits performance of many analog circuits, worsens as geometries shrink [Itrs2007]. To cope with mismatch, analog designers increase device area [Pel1989], use many circuit-level techniques like feedback and differential design [San2006][Joh1997][Raz2000], and more recently, shift functionality to digital, and apply calibration and compensation [Fay2006]. But these only partially scale with Moore's Law because large analog-sized transistors must form the core signal path. As a result, the analog portion of mixed-signal chips risks dominating the area [Rut2004]. There is a further concern: these approaches all start with a circuit that performs well functionally, then adapt, tune or average out the variations caused by mismatch. While this is reasonable, some fear that analog will hit a brick wall when there is simply too much process variations and time-dependent degradation to tune around [Gie2005b]. An illustration is for the gate oxide layer: at three atoms thick [Itrs2007], one or a few atoms out of place can significantly affect the performance. In this chapter, we ask: is it possible to design analog circuits that are simultaneously (a) naturally robust to variations without needing tuning, yet (b) scale with Moore's Law, i.e. use the smallest-possible transistors?

- **Trustworthy Novel Structural Synthesis.** As described in Table 10.1, there is no analog circuit structural synthesis approach that can return trustworthy (or trackable-trustworthy) circuits on novel functionality.

Towards finding a solution, we point out two opportunities:

- **Tiny Digital Transistors.** Moore's Law is not only a hindrance; it may also be a help. IC transistor geometries have exponentially shrunk so much that each individual minimally-sized transistor has become virtually free. This means that in design, as predicted decades ago, we can *waste transistors* [Mead1980]. However, this only holds if the transistors are near-minimal for the process. Digital circuits obey this, but currently not analog – as discussed, designers have kept analog circuits larger as a key way to reduce the effects of process variation-induced mismatch.

- **Machine Learning Ensembles.** Recent advances in the machine learning field point to a new way for designing analog machines – learning ensembles [Pol2006, Fri2002, Has2000].

In his 2008 keynote at the Design Automation Conference, Intel CTO Justin Rattner described how designing circuits for analog functionality could (and should) be viewed as a computational problem, paving the way for development of wholly novel approaches to circuits to realize such functionality [Rat2008]. ISCLEs follows such a paradigm: it starts with the view of analog functionality as a computational problem, then maps it into a specific computational problem: *regression*. Machine learning is a recent and powerful approach to regression, so ISCLEs leverages a machine learning approach called boosting. Specifically, ISCLEs stacks together dozens or hundreds of minimally-sized

circuit topologies using the machine learning framework of boosting, to get analog functionality. ISCLEs returns novel topologies that are trustworthy by construction, robust to mismatch, and show promise in maintaining an area footprint that scales with Moore's Law. The stochastic boosting approach that ISCLEs employs makes it mathematically equivalent to *importance sampling* [Hes2003], on circuits. This is where ISCLEs earns its label: Importance Sampled Circuit Learning Ensembles.

The rest of this chapter is organized as follows. Section 10.4.2 reviews machine learning and importance sampling, and proceeds to describe the ISCLEs boosting algorithm developed specifically for circuit assembly. Section 10.4.3 describes the ISCLEs-specific library of digital-sized circuit blocks. Section 10.4.4 briefly describes the multi-topology sizing approach. Section 10.4.5 describes experiments in the design of a sinusoidal function generator and an A/D converter. Section 10.5 concludes for both MOJITO-N and for ISCLEs.

10.4.2 Machine Learning and ISCLEs

This section starts with a discussion on machine learning, and how its evolution as a field can be emulated in circuit design (Figure 10.4). Then, ISCLEs is described in detail.

Two major sub-problems in machine learning [Has2001] are regression and classification; the key challenge for each is to find an input-output mapping that predicts well on unseen data. For decades, the prevailing approach was to come up with a single model (bottom left of Figure 10.4). Single models almost always had overfitting issues, performing well on training data but generalize poorly to unseen data. That is, good training error but poor testing error.

However, a new approach has emerged in the last decade: *ensembles* of models [Pol2006], which combine the output of *many* learners (bottom right of Figure 10.4). Ensembles inherently overfit less because the errors made by sub-learners are averaged out over the ensemble (assuming the sub-learners' outputs are not correlated). In "bagging", each sub-learner learns the full input-output mapping. Alternatively, a series of "weak learners" (WLs) can be "boosted" into a "strong learner" that captures the overall mapping [Fre1997]. Weak learning is much easier to do than strong learning of one model: each learner only needs to do better than random, rather than fully capture the mapping. The outer boosting algorithm takes care of combining the many WLs together in order to get the target mapping. Boosting does importance sampling [Hes2003] in the model space, therefore earning the label Importance Sampled Learning Ensembles (ISLEs)[Fri2003]. Section 3.3 discusses the SGB approach to boosting.

In analog circuit design and in analog synthesis, almost all existing approaches do the equivalent of designing a single "strong" circuit realizing the target functionality (top left of Figure 10.4).

In contrast, ISCLEs take a cue from progress in machine learning. Whereas ISLEs are on regressors, ISCLEs is on *circuits*. An ISCLEs circuit is an *ensemble* of "weak" circuits which are boosted to collectively realize the target functionality (top right of Figure 10.4) [Gao2008b]. Crucially, these weak circuits each have small area (via near minimally-sized transistors) so that total area footprint is not prohibitive. The overall architecture of

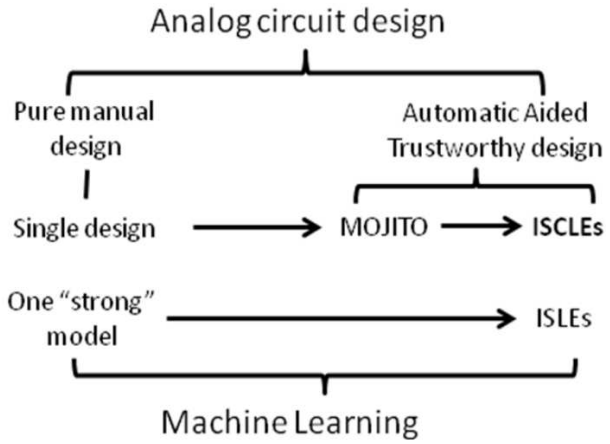


Figure 10.4: *ISCLEs shows how analog circuit design can shift from one "strong" model to ensembles of "weak" models, just as machine learning has.*

an ISCLEs circuit is shown in Figure 10.5, and each WL is one of the possible topologies in Figure 10.6.

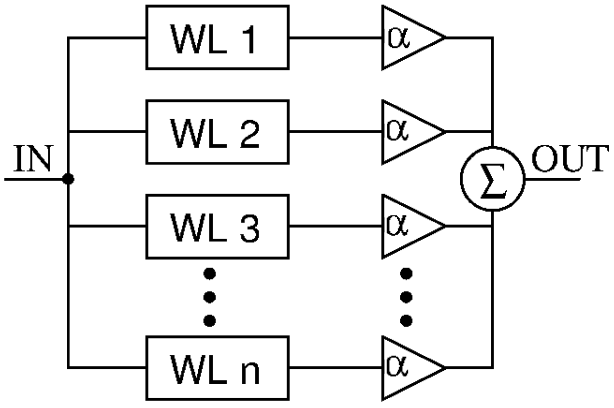


Figure 10.5: *An ISCLEs circuit combines many weak learner circuits (WLs) to achieve a target analog functionality. (Offset voltage is not shown.)*

Each topology is trustworthy by construction because the overall architecture, and each WL is trustworthy. The overall architecture merely does weighted addition of the WLs' outputs plus an offset voltage. Each WL topology is also trustworthy, as all the possible weak-learner topologies are in the set of hierarchically-organized designer-specified building blocks (these are described further in section 10.4.3).

Table 10.2 describes the high-level ISCLEs algorithm [Gao2008b]. The key input is an overall target waveform $y_{overall,target}$, and the output is an ensemble EL_{chosen} to realize

the waveform. At each ISCLEs boosting iteration, a weak learner WL_{cand} topology and sizing is chosen, and if it improves overall correlation, $r_{current}$, then it is added to the final ensemble with a weighting factor α, and the target waveform $y_{current,target}$ gets updated. The call to "find-weak-learned" is *precisely* a call to MOJITO search, where MOJITO searches over the weak learners library. Over time, the target waveform shrinks, zooming in on the hardest-to-capture parts of the mapping - i.e. ISCLEs does importance sampling of circuit learning ensembles. The loop repeats until the stopping criteria is hit, at which point the ensemble is returned. The whole process is automatic.

The main boosting parameter is α (learning rate), which governs the learning rate, and, by the definition of how the ensemble is constructed, is also the weighting factor for each WL in the ensemble. We set α to 0.10, meaning that on each iteration, 10% of the newest WL's output is used to update the overall target waveform. This setting strikes a compromise between the risk of overfitting (higher α), and slower convergence (lower α). Target correlation r_{target} is set to 0.95.

Table 10.2: *ISCLEs Algorithm*

Inputs: $\boldsymbol{y_{overall,target}}$, r_{target}, α
Outputs: EL_{chosen}
$\boldsymbol{y_{current,target}}$ = $\boldsymbol{y_{overall,target}}$
$r_{current}$ = 0.0
EL_{chosen} = $\emptyset$
while $r_{current} < r_{target}$: #for each boosting iteration
    WL_{cand} = find-weak-learner($\boldsymbol{y_{current,target}}$)
    EL_{cand} = $EL_{chosen} + \alpha * WL_{cand}$ #(as a circuit ensemble)
    $\boldsymbol{y_{cand}}$ = simulate(EL_{cand})
    r_{cand} = correlation($\boldsymbol{y_{overall,target}}$, $\boldsymbol{y_{cand}}$)
    if ($r_{cand} > r_{current}$): #improved
        $r_{current} = r_{cand}$
        $EL_{chosen} = EL_{cand}$
        $\boldsymbol{y_{current,target}}$ = $\boldsymbol{y_{overall,target}}$ - $\boldsymbol{y_{cand}}$
return EL_{chosen}

10.4.3 ISCLEs Weak Learners

This section describes the ISCLEs library of weak learner (WL) topologies, and their parameterization. Note that this library is a *different* library than the opamp library which was used in the other MOJITO experiments.

10.4.3.1 Weak Learners Library

A central challenge in this work was to design a competent library of possible WLs. Some applications may only need a simple inverter, and others may need more complex topologies. We designed three WLs: an inverter, an inverter with I-V amplifier, and

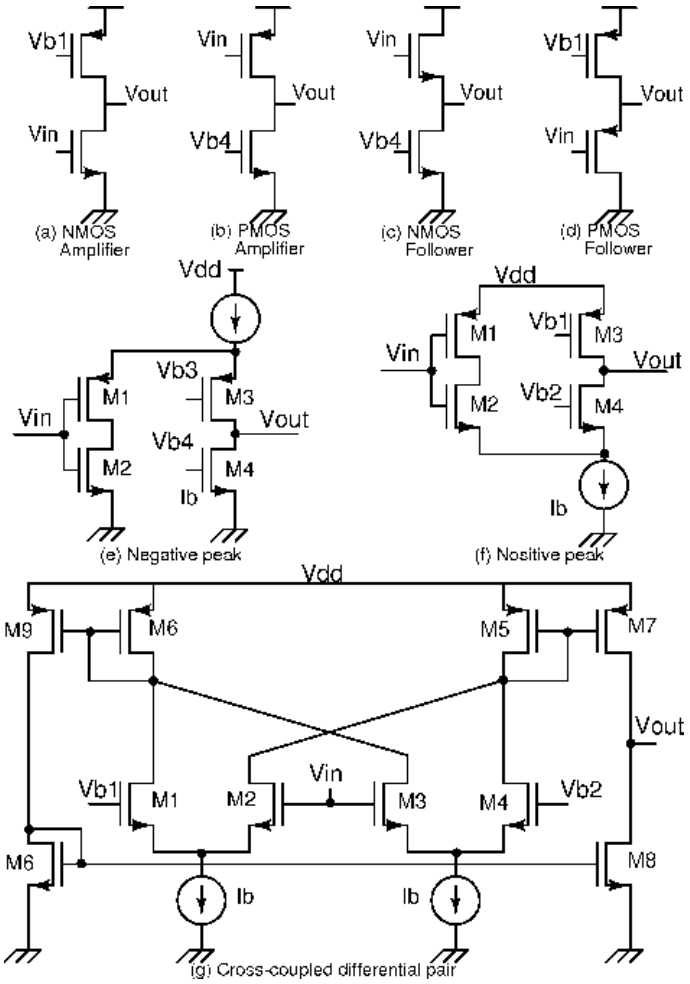


Figure 10.6: *Weak learners library.*

an inverter cross-coupled differential pair. Together, these form the library of possible topologies that MOJITO searches through. We now describe each WL.

1. **Inverter Learner.** This is the simplest WL. A top-level inverter can instantiate into one of four possible sub-blocks shown in Figure 10.6 (a) to (d).

2. **Inverter With I-V Amplifier.** Instantiations of this WL are in Figure 10.6 (e) and (f). Its core idea uses the fact that the current flow in an inverter is not a monotonic function of the input voltage. While the input sweeps from 0 to V_{dd}, the current will increase because the NMOS is gradually turned on, but after a certain threshold point, the PMOS switches off and the current will reduce to 0 again. This will form a current peak, and its position and width are determined by the sizing of the two transistors. If the aspect ratio of the NMOS is increased, the peak position will be lower, and vice versa. We then use an I-V amplifier to convert this current peak

into a voltage peak; and by sizing it, we have controllable voltage peak waveforms. The peak's minimum width is limited by the finite gain and sensitivity of the I-V amplifier. A peak simulation result is shown in Figure 10.7, which shows how different waveforms between any transition point and the higher transition point are realizable by using different transistor sizes.

3. **Cross-Coupled Differential Amplifier.** This WL circuit, shown in Figure 10.6 (g), is composed of a cross-coupled differential pair and several current mirrors. The input signal is connected to one of the input pins of each differential pair. The other input pins are connected to different bias voltages V_{b1} and V_{b2}. These two bias voltages set two fixed threshold points [Joh1997]. The size of input transistor pairs controls the threshold points, such that the output transfer curve will be similar to Figure 10.7.

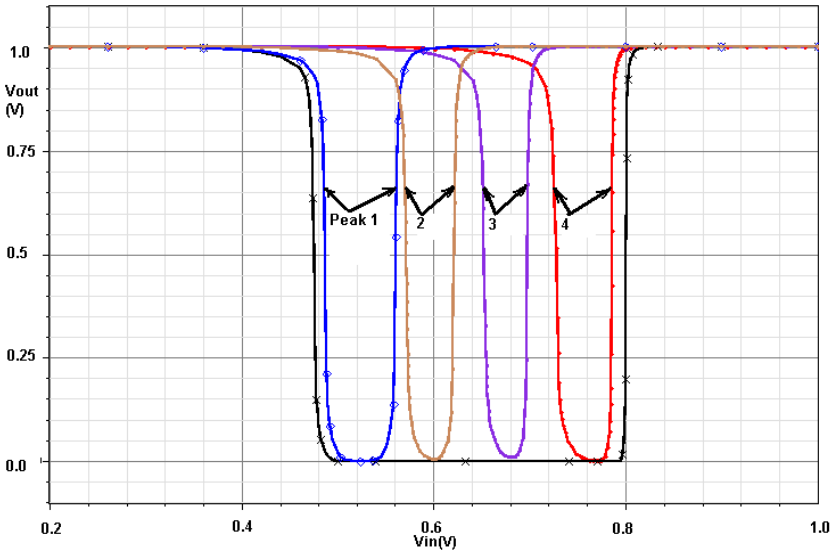


Figure 10.7: *Negative voltage peaks (generated by an inverter with I-V amplifier using parameter sweep simulations).*

10.4.3.2 Weak Learner Parameters

Table 10.3 enumerates the parameters for all WLs. Note that the maximum device size is just 20 times the minimum feature size, which forces the building blocks to be as small as digital circuits, i.e. to enable *analog circuits scaling*.

10.4.4 Multi-Topology Sizing

Each weak learner (WL) is found with MOJITO [Mcc2007](chapter 7) searching the possible topologies and sizings. MOJITO views the search space as a parameterized

Table 10.3: *Parameters for Weak Learners Topology Choice and Sizing*

Weak Learner	Parameter Names	Parameter Range
Inverter Learner	W_{in}, W_{load}, L_{in}, L_{load}	$[W_{min}, 20*W_{min}]$, $[L_{min}, 20*L_{min}]$
Inverter with I-V	W_1, W_2, W_3, W_4, W_5, L_1, L_2, L_3, L_4, L_5	$[W_{min}, 20*W_{min}]$, $[L_{min}, 20*L_{min}]$
Cross-Coupled Differential Pair	$W_1, W_2, W_3, W_4, W_5, W_6, W_7, W_8$, L_1, L_2, L_3, L_4, L_5, L_6, L_7, L_8	$[W_{min}, 20*W_{min}]$, $[L_{min}, 20*L_{min}]$
Topology choice	$choice\_index$	1,2,3,4,5,6,7

grammar, then finds the optimal "sentences" with grammatical genetic programming [Koza1992], [Whi1995]. MOJITO's objective is to maximize the correlation between the current target waveform(s) (as specified by the boosting loop) and its candidate circuit's waveform(s). By optimizing on correlation rather than squared error, MOJITO's problem is easier because correlation ignores the difference in offset between waveforms; the outer boosting loop takes care of this with an offset voltage. [Kei2004b] uses a similar trick for regression. MOJITO's constraints are device size constraints and DOCs (see section 2.2.9). MOJITO returns the highest-correlation individual to ISCLEs, which will be incorporated into the ensemble if it improves the overall ensemble.

MOJITO was configured to maximize search efficiency yet avoid getting stuck, using the following setup. At a given WL target, the population size was set to 10, and 50 generations were run. If the resulting circuit reduced the ensemble's overall error, then that WL was considered complete, and added to the ensemble. But if the overall error did not improve, then the population size was doubled and MOJITO was re-run. This is a strategy similar to [Aug2005]. In practice, we found that no doubling occurred in early iterations, but a few rounds of doubling occurred in later iterations. All other MOJITO settings were the same as in chapter 7.

10.4.5 ISCLEs Experimental Results

We applied ISCLEs to two different kinds of problems: a sinusoidal function generator, and a 3-bit flash A/D converter. The circuit simulator was HSPICE^{TM} [Snps2008a], using a 0.18μm TSMC CMOS process technology; other settings were provided in section 10.4.2. The runtime for each was under 8 hours on a Linux machine with a single-core 2.0 GHz Intel processor.

10.4.5.1 Sinusoidal Function Generator

In this example, ISCLEs is applied to generate a DC-in DC-out sinusoidal function generator. Specifically, the aim is to minimize the squared error difference between target DC response and the synthesized circuit's DC response, for several different input DC values.

Figure 10.8 shows the result of 40 boosting iterations, resulting in an ensemble of 40 WLs. Sub-figures (a) to (e) show the ensemble's output response waveform (lower

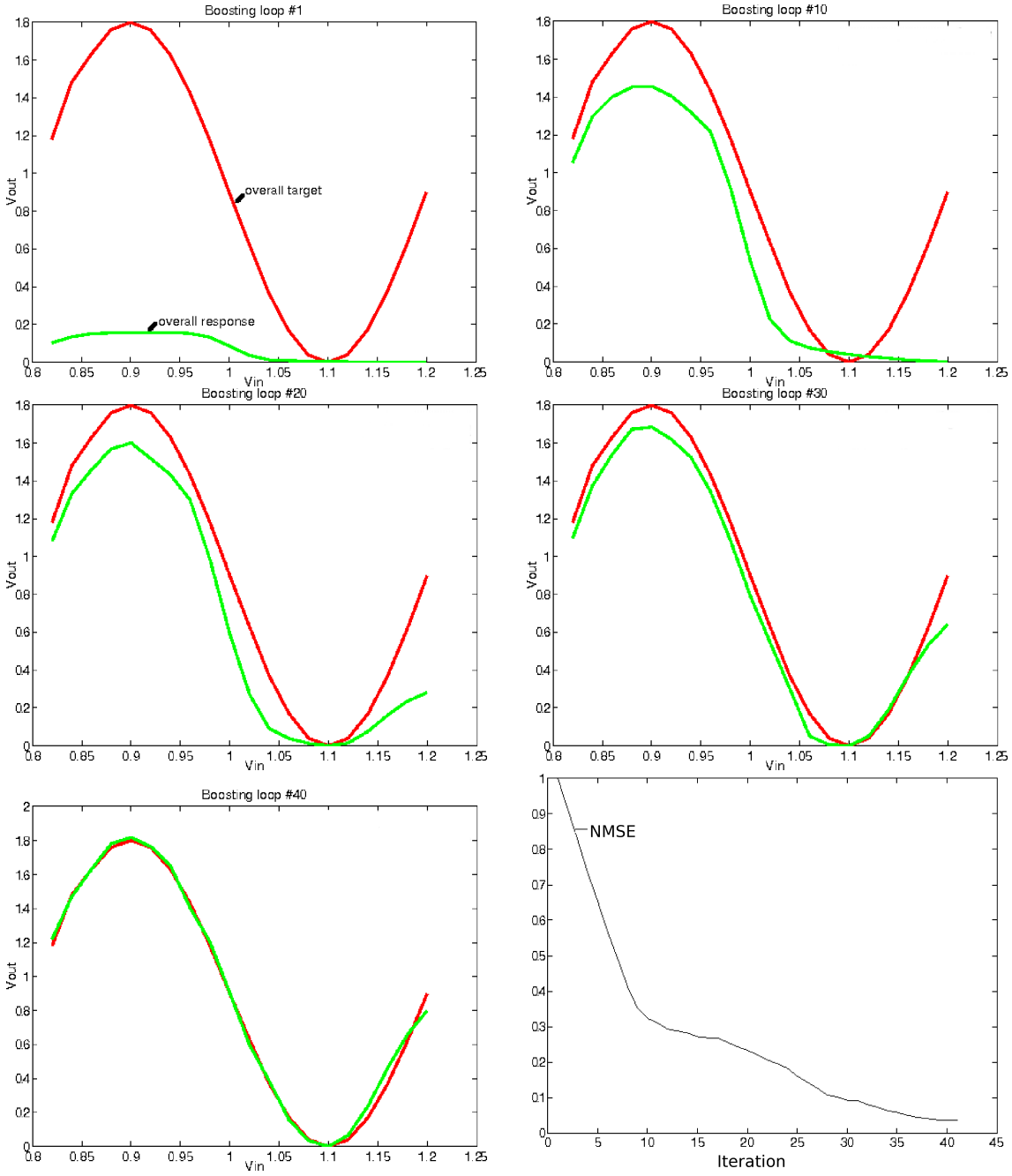


Figure 10.8: *ISCLEs sinusoidal waveform learning response; bottom right is convergence of nmse vs. boosting iteration.*

waveform) and the target waveform (upper waveform), where the input voltage is the x-axis and the output voltage is the y-axis. We see that as the boosting iterations progress, the ensemble's output response waveform converges to match the target waveform. Figure

10.8(f) shows the evaluation parameter NMSE (normalized root mean squared error) vs. boosting iteration[1]. We see that after 40 iterations, only a few percent error remains between the target and the ensemble circuit's responses. This example demonstrates that the core idea of ISCLEs – applying boosting to structural circuit synthesis – is promising.

Further validation that ISCLEs is behaving as expected, i.e. boosting circuits, is found by comparing Figure 10.8 (ISCLEs - boosting circuits) to Figure 3.3 (SGB - boosting decision trees). We see that despite one boosting approach being on circuits, and the other on decision trees, their convergence to solve the sinusoidal mapping has *nearly identical behavior*.

10.4.5.2 3-bit "Flash" A/D Converter

The aim of this example is to target A/D conversion; we specifically aim for a 3-bit flash architecture. Flash A/Ds are quite sensitive to process variations, due to the matching property of the resistor ladder and the comparator [San2006]. We approach this problem by designing one bit at a time. For each bit, the aim is to minimize the squared error difference between target DC response and synthesized circuit's DC response, for several different input DC values. Runtime for all three bits was two days.

While the bits could have been synthesized separately, we added some dependence so that the responses would be more in sync. First, the least significant bit (LSB) was synthesized. Then, the 2nd LSB was synthesized, using the output of the LSB as one of its inputs, as a "hint" to help it guide its output signal and more readily be synchronized with the LSB on the clock transitions. Then, the most significant bit (MSB) was synthesized, using the output of the LSB as one of its inputs, for the same reasons as the 2nd LSB.

Just like in the sinusoidal mapping, the overall boosting loop's objective was to minimize NMSE between the target signal and the candidate design's signal; and the inner MOJITO loop's objective was to maximize correlation between those signals.

Figure 10.9 shows the results. The top row is for the LSB, middle row is 2nd LSB, and bottom row is the MSB. For each row (bit), the left figure shows the output vs. input DC voltage, for both the target and the synthesized output response; and the right figure shows the convergence of NMSE vs. the boosting iteration. We observe that all the waveforms of the three output bits match target waveform within certain error margin. The LSB has the most complex input/output mapping, but ISCLES still achieved 13% NMSE, with 131 weak learners. To our knowledge, no prior ("strong learner") analog synthesis approaches have ever successfully synthesized a DC-DC mapping as complex as this. The 2nd LSB reached 9% NMSE with 126 weak learners. The MSB also reached 9% NMSE with 145 weak learners. Note that for the actual implementation, the bits' outputs are usually passed through an inverter that would rail the outputs to the high or low voltage value (i.e. Vdd and gnd), thus making the DC-DC mapping even tighter yet.

[1]NMSE is used in the outer, boosting loop, whereas correlation is used to guide the inner MOJITO search loop

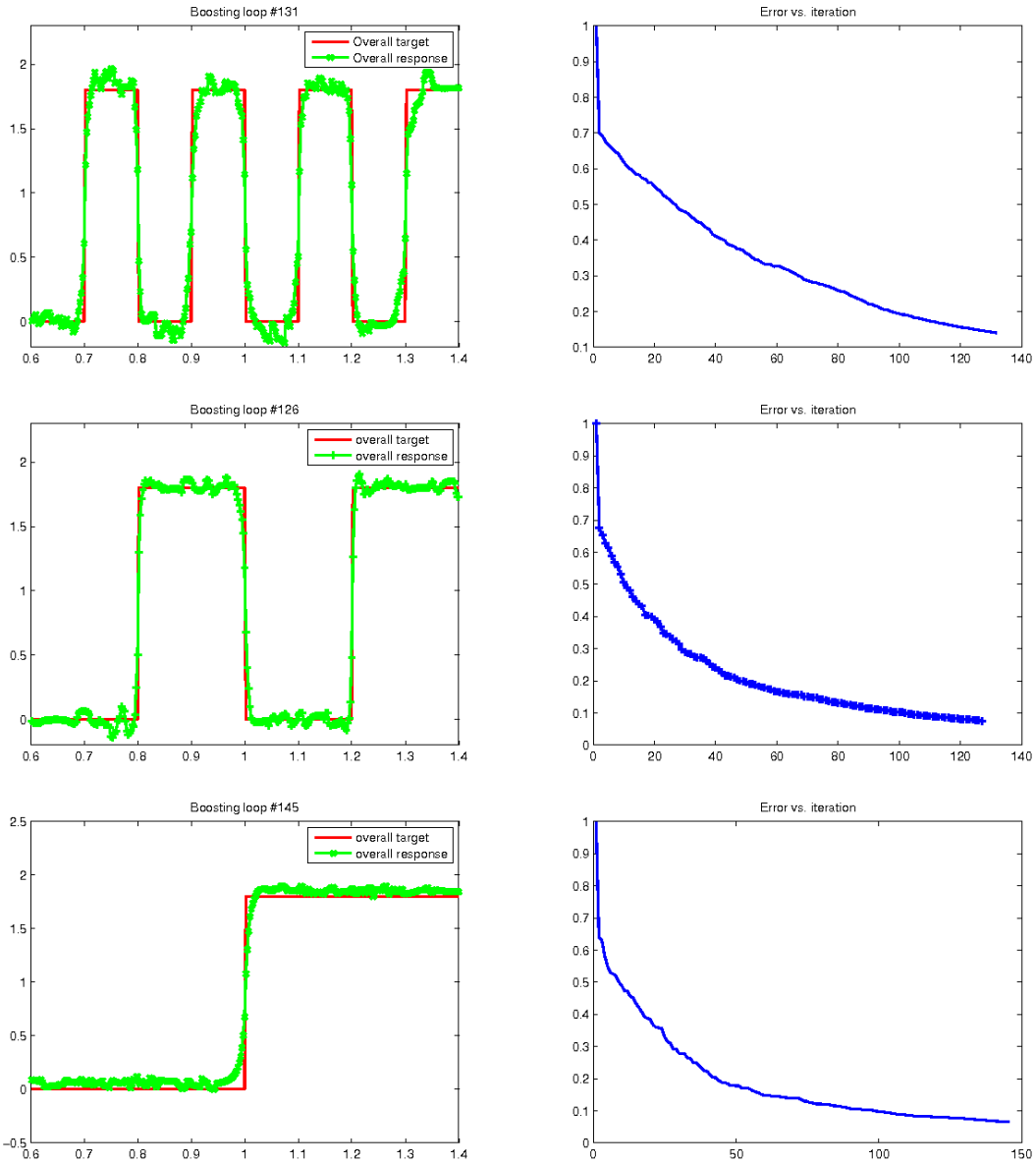


Figure 10.9: *ISCLEs 3-bit A/D converter outputs (left column) and corresponding convergence plots (right column). The top row is for the MSB, middle row is for the 2nd LSB, and bottom row is for the LSB.*

10.4.5.3 A/D Converter Simulation With Process Variations

Recall that the key challenge of using (near) minimally-sized transistors for analog circuits was to reduce sensitivity to process variations. So, we investigate the effect of variations here, with the hypothesis that the importance-sampling nature of ISCLEs might have some natural resilience to process variations. So, in order to test the tolerance of an ISCLEs circuit to process variation, we inject a differnt randomly-drawn variation in *each* transistor's model parameter V_{th} (threshold voltage) into the already-synthesized A/D circuits, and measure the response.

Figure 10.10 shows, for 4 random samples, the A/D's LSB (railed) simulations with A_{v_t} = 5mVμm. The overall response changes only slightly from sample to sample; that is, our ISCLEs-synthesized circuits have graceful tolerance to process variations. We acknowledge that it is likely safer to account for variations more directly by incorporating process variations into the boosting loop iself; we leave that to future work. These initial results are in any case promising.

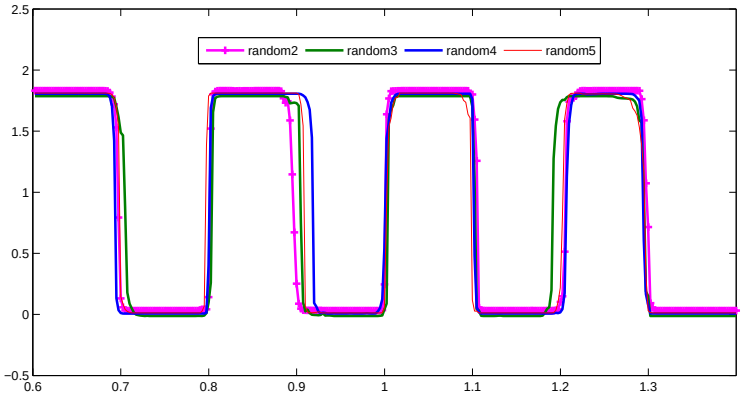


Figure 10.10: *3-bit ADC's LSB with smaller process variation injection.*

10.4.5.4 ISCLEs Scaling Potential

This section uses the A/D results to explore the potential of ISCLEs for scaling analog with Moore's Law. The ENOB of this 3-bit ADC @100kHz is 2.78. The estimated active chip area is 14e-9 m^2 (in 180nm CMOS) and 10e-9 m^2 (in 90nm CMOS). Then we predict the area of a conventional A/D [Pla2003], which should tolerate resistor matching (1%) and V_T variation (A_{v_t} = 5mVμm), and achieve similar ENOB. By rough estimation this chip should be larger than 1e-9 m^2 (in 180 nm CMOS) and 0.8e-9 m^2 (in 90 nm CMOS). According to the ITRS roadmap [Itrs2007], A_{VT} will stop shrinking, but the analog area will not shrink anymore. (The chip will still slowly get smaller because of the shrinking of the digital part). With ISCLEs, mixed-signal chips *can* continue shrinking because the "analog" side can use minimally-sized transistors. By extrapolating as shown in Figure 10.11, we see that when the technology shrinks to lower than 18nm, ISCLEs will become beneficial in chip area compared to conventional approaches. We believe,

however, that ISCLEs may be useful even sooner if additional measures are taken: (1) the MOJITO weak-learner synthesis within the boosting loop makes area minimization to be an additional objective, rather than ignoring it, and (2) ISCLEs is combined with other variation-handling schemes such as calibration or compensation [Fay2006, Li2007c].

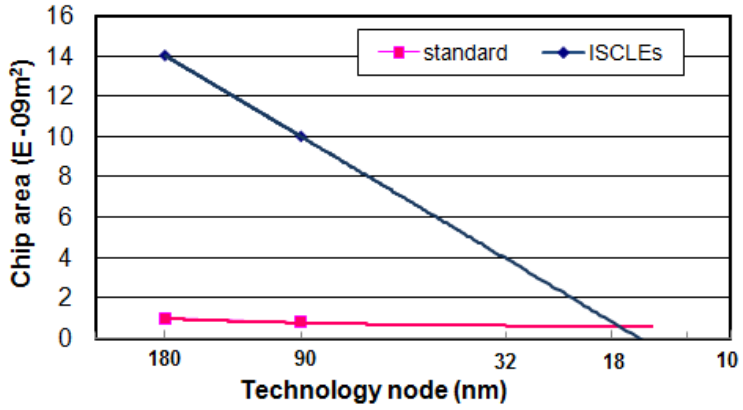


Figure 10.11: *Prediction of the chip area for 3-bit ADC using a standard design approach, versus using the ISCLEs approach.*

10.5 Conclusion

This chapter has presented MOJITO-N [Mcc2008a] and ISCLEs [Gao2008a, Gao2008b], as initial explorations into automation-aided novel analog topology design. An underlying theme for both MOJITO-N and ISCLEs was to remain as trustworthy as possible, or to track the design's novelty (trustworthiness).

MOJITO-N is an extension of MOJITO which adds novelty to the trustworthy designs. It returns a Pareto-optimal set of circuits that trade off novelty with performance, and does it in commercially reasonable computational effort. The novelty is fully trackable, so all changes can be readily understood. As part of the algorithm's verification, MOJITO-N successfully re-invented a known design of industrially relevant complexity.

ISCLEs is a promising technique to synthesize novel functionality, trustworthy, variation-robust analog circuits that may scale with Moore's Law, using reasonable CPU effort. ISCLEs extends the machine learning method of *boosting* [Fri2002] to circuit design. Boosting's "weak learners" (WLs) are designer-trusted topologies that are sized and chosen by MOJITO multi-topology sizing. The overall boosting ensemble ties together all the WL circuits with a weighted adder circuit. We designed a library of trusted WL topology choices for MOJITO to search. ISCLEs' effectiveness was demonstrated on two problems: a sinusoidal function generator, and a 3-bit A/D converter learning. By demonstrating promising resilience to process variations yet using minimally-sized devices, ISCLEs has promise as a way for analog circuits to scale with process technology, unlike traditional analog implementations.

Chapter 11

Conclusion

If one takes care of the means, the end will take care of itself.
–Ghandi

11.1 General Contributions

Over the years, analog CAD has progressed from SPICE simulators, to schematic and layout editors, to recent nominal global optimizers and local variation-aware optimizers. This thesis has proposed and demonstrated a roadmap for the next several steps of tool evolution (Figure 11.1). The roadmap starts with global variation-aware optimization, then progresses to *trustworthy* structural synthesis, and then adds variation-awareness and novelty to the structural synthesis.

Along the way, this thesis has also demonstrated techniques to use automation for *knowledge extraction* so that the designer can maintain and build his insight. Techniques include the extraction of template-free symbolic performance models, topology decision trees, and nonlinear impact analysis of topology vs. sizing/biasing variables. The techniques developed apply not only to analog CAD, but to a whole host of other AI problems related to the robust design of complex systems.

11.2 Specific Contributions

Besides the general contributions mentioned, this thesis has made the following specific contributions:

- **Comprehensive reviews** of variation-aware circuit sizing approaches (chapter 2), analog structural synthesis (chapter 5), and variation-aware analog structural synthesis (chapter 9), with a unified perspective across the CAD and EA communities.
- **SANGRIA**: [Mcc2008e] For the first time, a variation-aware circuit sizing algorithm has been presented which has reliable *global* convergence, having full SPICE accuracy yet reasonable runtime. SANGRIA is enabled by an algorithm structure that simultaneously explores on loosened statistical goals, exploits on tightened statistical goals, and the levels in between, i.e. it has "structural homotopy". To make the most

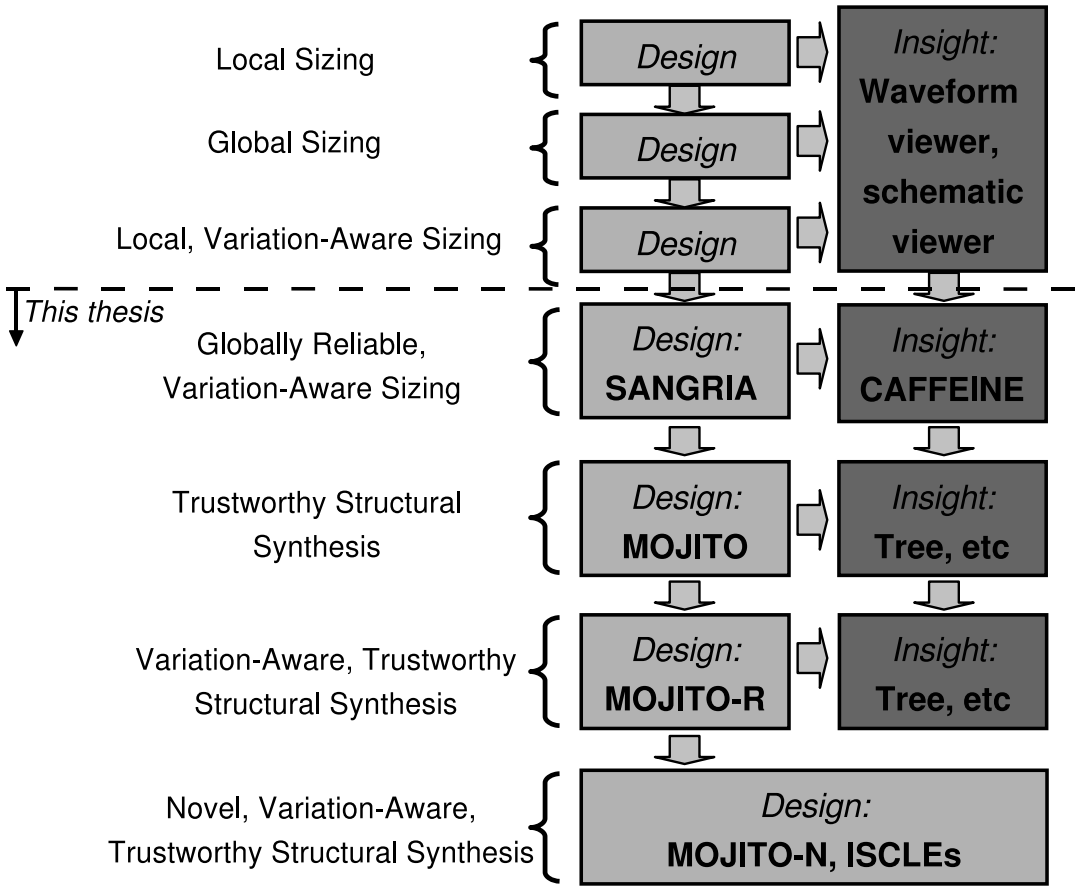


Figure 11.1: *Proposed roadmap in evolution of front-end design automation tools (first seen in Figure 1.12).*

use of valuable simulation data, SANGRIA uses model-building optimization with state-of-the-art nonlinear regression (SGB ensembles).

- **CAFFEINE**: [Mcc2005a, Mcc2005b, Mcc2005c] For the first time, a means has been presented to generate scalable, template-free, symbolic performance models with near-SPICE accuracy; and a first step to a related means to generate interpretable, nonlinear behavioral models. CAFFEINE is enabled by searching through a grammar-defined space of canonical-form functions to relate SPICE-simulation input/output data.
- **MOJITO**: [Mcc2007] For the first time, a multi-objective analog topology synthesis approach has been presented. MOJITO has industrially palatable accuracy, setup requirements, runtime, generality, and results. MOJITO is enabled by:
  - A framework to define a structural synthesis search space that is (a) trustworthy, (b) flexible, (c) specified by structural information only, and (d) leverages readily-transferable building-block domain knowledge.
  - An example amplifier topology space using the framework, which is qualitatively richer because it is 50x larger than past trustworthy structural-only spaces.

  - A structural synthesis EA that has reliable convergence, and handles many objectives

- **Topology-performance knowledge extraction**: [Mcc2008b, Mcc2008d] This is a tool which for the first time, allows automated extraction of expert knowledge relating structures, performances, and sizes has been presented. It is enabled by a data-mining perspective on MOJITO synthesis results. Specifically, techniques have been presented for automatically extracting: (a) a decision tree for navigating from specs to topology, (b) global nonlinear sensitivities on topology and sizing variables, and (c) analytical performance-tradeoff models across the whole set of topologies.
- **MOJITO-R**: [Mcc2008e] This variation-aware extension to MOJITO was presented, which embeds structural homotopy into MOJITO in order to automatically generate tradeoffs across performances *and* yield.
- **MOJITO-N**: [Mcc2008a] This is an open-ended novel-topology search extension to MOJITO which this thesis presented. MOJITO has novelty-inducing search operators, and the resulting topologies are managed via *trustworthiness tradeoffs* in a designer-trackable fashion.
- **ISCLEs**: [Gao2008a, Gao2008b] This method has been developed to synthesize circuit topologies which are novel in both functionality and topology, yet trustworthy, and within reasonable CPU effort. ISCLEs uses machine-learning "boosting" to construct an aggregate of digitally-sized circuits. By demonstrating promising resilience to process variations yet using minimally-sized devices, ISCLEs has promise as a way to implement analog circuits in a way that scales better with shrinking process technology.
- **Experimental results** have been presented for each of the above techniques to demonstrate their validity and applicability.

11.3 Future Work

This thesis opens up fertile ground for future work, so this section discusses the opportunities from several angles: industrial deployment and circuit design (section 11.3.1), CAD research (section 11.3.2), and AI research (section 11.3.3).

11.3.1 Industrial Deployment and Circuit Design

Future possible work from an industrial and design context is in deployment and usage of the tools inspired by the roadmap of Figure 11.1. This is possible because all the research maintained an eye on the specifications needed for industrial deployment, such as SPICE accuracy and scalability to larger circuits. There are several opportunities for major CAD vendors, CAD startups, semiconductor CAD groups, universities, and even fabs. We now describe the specific opportunities.

CAD Tools. SANGRIA would be deployed as a global yield optimization CAD tool, by CAD vendors or CAD groups, for direct use by design engineers. It would be comple-

mentary to a local yield "tuner". Some designers may use a nominal variant of SANGRIA for global nominal sizing, or a multi-objective version for yield-aware performance exploration (akin to MOJITO-R). CAFFEINE would be deployed as a CAD tool directly to design engineers; as well as to modeling groups found within fabs, semiconductor companies, and universities. ISCLEs could also be deployed as a CAD tool for synthesizing specific circuit types, especially under extreme process variations. Finally, there is a CAD tool opportunity for browsing IP results of MOJITO/MOJITO-R runs, and for extracting insight from those results (but not for running MOJITO directly, as we will explain next).

Jellybean IP. MOJITO and MOJITO-R have runtimes that are too long for a design engineer on a typical design, so they are not suited to be run directly within the analog design flow. However, since only one run is needed per process node (per circuit type), they fit a "jellybean IP" flow nicely. Specifically, with each process node, (a) a fab, vendor or CAD group would run MOJITO on each circuit type, then (b) make the results database IP easily accessible to design engineers through an IP-browsing tool. This database would support queries based on specifications, to immediately return a sized topology. It would also have a means to browse the performance tradeoffs, e.g. with parallel coordinates. And of course, it would also output specs-to-topology decision trees, variable impacts, and analytical performance tradeoffs. In short: while *running* MOJITO / MOJITO-R is not suitable for a typical design flow, using the resulting IP *is*.

There are several ways to view how browsers for analog jellybean IP might fit in. They can be viewed as an extension to analog environment library managers (e.g. in [Cdn2008g]), as an analog version of digital standard cell libraries supplied by digital synthesis vendors, or as an analog IP library supplied by analog IP vendors.

Topology Development. MOJITO, MOJITO-N, and ISCLEs fit usage by topology design groups in semiconductor/IP companies and universities, with homebrew versions of the code. They will likely *not* get deployed as topology-development tools by CAD vendors, because the market is too small to justify the effort. Instead, we believe that the following flow will be particularly fruitful:

- The designer conceives of a new building block.
- He enters new block into the MOJITO building blocks library, and runs MOJITO.
- Finally, he inspects the run results to determine what performance ranges the block is useful for, and what other building blocks it combines best with. He can browse the raw data or use the knowledge-extraction techniques.

This flow allows the designer to focus on the core creative aspect of analog design, and leaves the rest to MOJITO. In fact, we have already found this flow to be useful in EMC-aware current mirror design [Loe2009].

MOJITO and MOJITO-N topology design research will prove most interesting where the greatest circuit design issues, challenges and opportunities lie. Design issues found in ITRS [Itrs2007] and elsewhere include worsening process variation, parasitics, proximity effects, packaging issues, high analog area, and more. Design challenges include achieving extreme low power, extreme high frequency, extreme low noise, and more. Design

opportunities include exploiting new design techniques (e.g. by incorporating recently published inventions into the MOJITO blocks library), qualitatively new device behavior (e.g. digital transistors that switch fast enough to have analog functionality; e.g. memristors); and qualitatively new target circuit behavior.

Fabricating and testing a new design will be a turning point in using AI for topology development [Mcc2006d], because it will demonstrate that a palatable flow *does* exist. Since masks on modern geometries are extremely expensive ($millions), the most pragmatic way to test a new MOJITO-aided topology design is on an older (cheaper) process, as part of a multi-project wafer (MPW) to further divide the cost.

11.3.2 Possible Future CAD Research

This section describes possible directions for future CAD research, building on the work in this thesis.

System-Level Design. SANGRIA could be adapted for multi-objective sizing, returning tradeoffs between performances and yield (akin to MOJITO-R). It could also be extended to system-level design. For example, the tradeoffs could be used in a multi-objective bottom up approach [Eec2005, Eec2007], or to do bottom-up computation of feasibility regions, which would be followed by top-down design as outlined in [Gie2005]. A related approach is in [Rut2007].

MOJITO / MOJITO-R can be extended to system-level design in a similar multi-objective fashion, as outlined in [Mcc2007]. The resulting performance tradeoffs will likely be far better than tradeoffs from merely optimization, because such as large number of topologies is considered. There is also an opportunity to unify the system's hierarchical decomposition with MOJITO's hierarchical building blocks, and exploit this unified view. Specifically: (a) the MOJITO library would get extended upwards beyond op amps into ADCs and DACs, and higher; (b) performance goals are attached to the nodes in the hierarchy where they can be measured, including transistor DOCs, op amp {gain, GBW, etc.}, and ADC {ENOB, power, etc.}. Finally, (c) a hierarchical design methodology such as [Eec2005] is applied. Interestingly, [Mar2008] can be viewed as one instantiation of this idea, in which abstraction via behavioral modeling becomes a critical element.

Reconciling Layout. High-frequency, low-power, and low-noise design require consideration of layout parasitics. SANGRIA and MOJITO-R could be extended to handle this in one of many ways. One possible way is as follows. *Layout-aware* performance evaluation would (a) convert the netlist to layout via layout synthesis, (b) from the layout, extract a parasitic-annotated netlist, and (c) simulate the extracted netlist. The paper [Van2001] used a similar flow for layout-aware optimization. We must note that that modern simulator options (e.g. the "+parasitic" option in [Cdn2008d]) means simulation of extracted netlists is no longer excessively expensive. For further speed, the ALPS framework could be exploited, by only performing layout-aware evaluation on higher age layers, and not on the lower exploratory layers.

More Circuit Issues, Related Design Tactics. SANGRIA and MOJITO-R handled environmental, global statistical process variation, and local statistical process variation. Layout parasitics are one more issue to handle, but there also upcoming issues: proximity [Dre2006], electromagnetic compatibility (EMC) [Paul1992], aging/reliability effects, and more. There are also more tactics to handle variation, such as post-manufacturing calibration [Fay2006, Li2007c], body biasing, and more.

Further Novel Synthesis. The work done in ISCLEs and MOJITO-N were only initial tentative steps. There is tremendous opportunity to explore results and ideas for each of these more deeply, with corresponding payoff. (And this is only really possible now, as a prerequisite to both was development of MOJITO.)

Further Knowledge Extraction. The idea of using CAD for knowledge extraction has been around since symbolic analysis [San1989], but there have been few examples besides symbolic analysis. This thesis demonstrated several new techniques where knowledge extraction could give useful insights to the designer. Knowledge extraction is fertile territory for further exploration, especially because there continue to be new developments in the fields of visualization and human-computer interaction.

CAFFEINE Applications. CAFFEINE could be extended to model order reduction (MOR). Recall from section 4.8.1 that CAFFEINE was applied to behavioral modeling *without* supplying state variables. However, in many industrially-relevant behavioral modeling / simulation problems, the state variables *are* available; which makes the problem approachable by MOR techniques (see [Rut2007] for a survey). Many MOR techniques use regression at their core, e.g. piecewise polynomials [Dong2008]. Since CAFFEINE can generate compact nonlinear models, it should apply nicely here.

CAFFEINE could also be applied to MOS modeling, where the aim is to write SPICE-simulatable models describing transistor dynamics, which simulate quickly yet accurately capture the silicon measurements. Once again, this fits CAFFEINE's ability to generate compact nonlinear models. There is also potential interplay between CAFFEINE and LVR techniques [Sin2007, Li2008b], which may be useful for modeling process variation performance models having $\approx$ 1000's of input parameters.

Speed / Scalability / Usability Improvements. There is always room to further improve the speed and scalability of SANGRIA, CAFFEINE, MOJITO, and ISCLEs. However, we remind the reader that this thesis targeted industrial design flows, which had specific speed and scalability requirements for each problem, and the approaches hit those targets. If one improves these further, there ought to be a clear reason. One possibility: a 10x or 100x speedup could mean qualitatively new ways to use the tool. For example, if MOJITO-R were sped up to return good results overnight on one machine, then it could be directly usable by designers. It would also be highly interesting to find out how big the MOJITO library could get, e.g. by systematically adding more building blocks from a variety of sources.

Developing the MOJITO library was somewhat tedious; though fortunately the result can be directly transferred to different process nodes. There are ways to improve library

development. First, we could exploit the platform of a standard schematic editor (e.g. [Cdn2008f]) to have a GUI-based approach to block / hierarchy design. Second, because the blocks have currents and voltages as independent design variables, there should be a way to avoid needing to explicitly define them, which will simplify their definition. Finally, for propagating parameters upwards in the hierarchy, it would be beneficial to allow grouping of parameters.

11.3.3 Possible Future AI/General Research

This section describes possible directions for future research in AI and other fields, building on the work in this thesis. In many cases this is the application of developments in this thesis to fields beyond analog CAD, and in remaining case it involves combining with other promising techniques.

Global Optimization. The combination of ALPS with MBO/SGB could turn out to be highly effective for large-scale global optimization problems in general. Structural homotopy would apply to many problems in stochastic optimization and robust design.

Regression & Symbolic Regression. CAFFEINE was highly competitive with state-of-the-art blackbox regressors on our problem domain. It could likely be competitive to the other regressors in in a variety of other domains, such as financial modeling or chemical sensor fusion. The canonical functional form (as a grammar, or otherwise) could be combined with other state-of-the-art GP symbolic regression techniques to improve their speed or scalability.

Structural Synthesis & Insight. This thesis showed the payoff of using *field-specific*, *pre-defined*, and *hierarchical* building blocks (BBs), as a means of exploiting a field's knowledge base to quickly synthesize trustworthy structures (within MOJITO). The idea is general, and applies to any field where (a) there is a significant body of knowledge which can be captured as hierarchical building blocks, and (b) a design's quality can be measured. This includes software design, automotive design, mathematical theorems, biological reverse engineering, and music design.

The BBs technique / MOJITO can also be combined with a hierarchical design methodology. In the spirit of MOJITO-N, new building blocks can be discovered and incorporated into the library. This might prove to be particularly helpful for fledgling field, as a means to accelerate the field's knowledge base. We might also consider *automatically* re-incorporating new building blocks back into the library, akin to Run Transferable Libraries [Ryan2005].

We can also extract decision trees, relative impacts, and analytical performance tradeoffs. For example, an automotive decision tree would input specifications like top speed, cost, and mileage, and return a suggested car design. An impacts plot might report engine displacement and aerodynamic drag as the largest impacts on top speed. The analytical performance tradeoff might relate cost with top speed and mileage.

11.4 Final Remarks

This concluding chapter reviewed the general and specific contributions of this thesis, then described possible future work in terms of industrial deployment / circuit design, possible CAD research, and possible AI research.

As described in the introduction, the progress of the last half-century can be characterized by the exponential progress in information technology (IT). An IT-enhanced future promises to be bright if such progress continues [Bai2006, Kur2005]. But potential showstoppers include (a) process variations hindering development of computational substrate, and (b) an apparent "complexity wall" in the automated synthesis of structures.

This thesis aimed to play a role in addressing both issues, via (a) a suite of techniques to handle variation-aware circuit design problems, and (b) an approach for structural synthesis that breaks through the complexity wall by reusing field-specific, hierarchical, building blocks.

The future is bright.

List of Publications

List of Publications during PhD

Articles in Journals

- Trent McConaghy and Georges G.E. Gielen, "Template-Free Symbolic Performance Modeling of Analog Circuits Via Canonical Form Functions and Genetic Programming", in *IEEE Transactions on Computer-Aided Design* (submitted 2008)

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Trustworthy synthesis of analog circuit topologies via hierarchical domain-specific building blocks", in *IEEE Transactions on Evolutionary Computation*, (submitted 2008)

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Automated extraction of expert domain knowledge from genetic programming synthesis results", in *IEEE Transactions on Evolutionary Computation*, (submitted 2008)

- Trent McConaghy, and Georges G.E. Gielen , "Globally reliable, variation-aware sizing of analog integrated circuits via response surface modeling and structural homotopy", in *IEEE Transactions on Computer-Aided Design*, (submitted 2008)

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Trustworthy, variation-aware structural synthesis of analog circuits via structural homotopy", in *IEEE Transactions on Computer-Aided Design*, (submitted 2008)

Book Chapters

- Trent McConaghy and Georges G.E. Gielen, "Genetic programming in industrial analog CAD: Applications and challenges", in *Genetic Programming Theory and Practice III*, Edited by Tina Yu and Bill Worzel, Chapter 19, Springer, 2006.

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Genetic programming with reuse of known designs", in *Genetic Programming Theory and Practice V*, Edited by R. Riolo and B. Worzel, Chapter 10, Springer, 2008

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Automated extraction of expert domain knowledge from genetic programming synthesis results", in *Genetic Programming Theory and Practice VI,* Edited by R. Riolo and B. Worzel, Springer, 2009

Articles in Refereed Conference Proceedings

- Trent McConaghy, Tom Eeckelaert, and Georges G.E. Gielen, "CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming", in *Proceedings of the Design Automation and Test in Europe (DATE)*, pp. 1070–1075, March 7-11, 2005.

- Tom Eeckelaert, Trent McConaghy, and Georges G.E. Gielen, "Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces", in *Proceedings of the Design Automation and Test in Europe (DATE)*, March 7-11, 2005.

- Trent McConaghy, and Georges G.E. Gielen, "IBMG: Interpretable behavioral model generator for nonlinear analog circuits via canonical form functions and genetic programming", in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, May 23-26, 2005.

- Trent McConaghy, and Georges G.E. Gielen, "Analysis of simulation-driven numerical performance modeling techniques for application to analog circuit optimization", in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, May 23-26, 2005.

- Georges G.E. Gielen, Trent McConaghy, and Tom Eeckelaert, "Performance space modeling for hierarchical synthesis of analog integrated circuits", in *Proceedings of the Design Automation Conference (DAC)*, pp. 881–886, June 13-17, 2005.

- Trent McConaghy, Tom Eeckelaert, and Georges G.E. Gielen, "CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming", in *Proceedings of the European Solid-State Circuits Conference (ESSCIRC)*, September 12-15, 2005.

- Trent McConaghy, and Georges G.E. Gielen, "Automation in mixed-signal design: challenges and solutions in the wake of the nano era", in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 461–463

- Trent McConaghy, and Georges G.E. Gielen, "Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions", in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 855–862, 2006

- Trent McConaghy, and Georges G.E. Gielen, "Double-strength CAFFEINE: Fast template-free symbolic modeling of analog circuits via implicit canonical form

functions and explicit Introns", in *Proceedings of the Design Automation and Test in Europe (DATE)*, pp. 269–274, 2006

- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies", in *Proceedings of the Design Automation Conference (DAC)*, pp. 944–947, 2007
- Peng Gao, Trent McConaghy, and Georges G.E. Gielen, "ISCLEs: Importance sampled circuit learning ensembles for trustworthy analog circuit topology synthesis", in *Proceedings of the International Conference on Evolvable Systems (ICES)*, Prague, CZ, September 2008
- Trent McConaghy, Pieter Palmers, Georges G.E. Gielen, and Michiel Steyaert, "Automated extraction of expert knowledge in analog topology selection and sizing", in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, November 2008
- Peng Gao, Trent McConaghy, and Georges G.E. Gielen, "ISCLEs: Importance sampled circuit learning ensembles for robust analog IC design", in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, November 2008
- Johan Loeckx, Thomas Deman, Trent McConaghy, and Georges G. E. Gielen, "A novel EMI-immune Current Mirror Topology obtained by Genetic Evolution", in *Proceedings of the Conference in Electro Magnetic Compatibility (EMC '09)*, Zurich, 2009 (accepted)
- Pieter Palmers, Trent McConaghy, Michiel Steyaert, and Georges G. E. Gielen, "Massively Multi-Topology Sizing of Analog Integrated Circuits", in *Proceedings of the Design Automation and Test in Europe (DATE)*, (submitted 2008 for DATE 2009)

U.S. Patents Pending

- Pending No. 20070208548 Trent McConaghy, "Modeling of systems using canonical form functions and symbolic regression"
- Pending No. 20080022251 Trent McConaghy et al, "Interactive schematic for use in analog, mixed-signal, and custom digital integrated circuit design"
- Pending No. 20080022232 Trent McConaghy et al, "Data-mining-based knowledge extraction and visualization of analog / mixed-signal / custom digital circuit design flow"
- Pending No. 20080022239 Trent McConaghy et al, "System and method for determining and visualizing tradeoffs between yield and performance in electrical circuit designs"

- Pending No. (recently submitted) Trent McConaghy et al, "Pruning-based variation-aware design"

- Pending No. (recently submitted) Trent McConaghy et al, "Model building optimization"

- Pending No. (recently submitted) Trent McConaghy et al, "Global statistical optimization, characterization, and design"

- Pending No. (recently submitted) Trent McConaghy et al, "On-the-fly improvement of statistical estimates in statistical design, with corresponding visual feedback"

Tutorials and Invited Talks

- Georges Gielen and Trent McConaghy, "Analog CAD research at KU Leuven: Challenges, developments, and outlook", Philips Semiconductor, Eindhoven, The Netherlands, 2005

- Trent McConaghy, "How evolutionary algorithms provide a fresh perspective on analog circuit designs", MIT Computer Science and Artificial Intelligence Lab, Boston, USA, 2006

- Trent McConaghy, "New approaches to hierarchical design methodologies, for design of complex analog circuits", University of Venice ECLT, Venice, Italy, 2007

- Trent McConaghy, "The payoff of domain knowledge, with application to industrially relevant analog circuit design", ProtoLife Inc., Venice, Italy, 2007

- Trent McConaghy, "Statistical knowledge extraction in analog circuit design", Part of "Robust AMS Design" Tutorial, Design Automation Conference, Anaheim, CA, June, 2008

List of Publications Prior to PhD

Articles in Refereed Conference Proceedings

- Trent McConaghy and Henry Leung, "Genetic programming with least squares for fast, precise modeling of polynomial time series". In John R. Koza editor, *Late Breaking Papers at the Genetic Programming Conference*, University of Wisconsin, Madison, WI, USA, 1998

- Trent McConaghy, Henry Leung, and Vinay Varadan, "Functional reconstruction of dynamical systems from time series using genetic programming", in *Proceedings of the 26th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, volume 3, pp. 2031–2034, Nagoya, 2000

- Jianming Liang, Trent McConaghy, Alexander Kochlan, Tuan Pham, and Glen Hertz, "Intelligent systems for analog circuit design automation: A survey", in *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida (USA), 2001.

Articles in Journals

- Trent McConaghy, Henry Leung, Éloi Bosse, and Vinay Varadan, "Classification of audio radar signals using radial basis function neural networks", in *IEEE Transactions Instrumentation and Measurement*, vol.52, no.6, pp. 1771–1779, Dec 2003.

U.S. Patents and Patents Pending

- US Patent No. 6,807,652 Trent McConaghy et al, "Method of robust semiconductor circuit products design using rational robust optimization"

- US Patent No. 6,859,914 Trent McConaghy et al, "Smooth operators in optimization of circuit structures"

- US Patent No. 6,910,192 Trent McConaghy et al, "Method of robust technology design using rational robust optimization"

- US Patent No. 6,968,517 Trent McConaghy et al, "Method of interactive optimization in circuit design"

- US Patent No. 7,089,163 Trent McConaghy et al, "Smooth operators in optimization of structures"

- Pending No. 20030079188 Trent McConaghy et al, "Method of multi-topology optimization"

- Pending No. 20030131323 Trent McConaghy et al, "Method of schematic-level AMS topology optimization using direct representations"

- Pending No. 20040148578 Trent McConaghy et al, "Method and system for design selection by interactive visualization"

- Pending No. 20040153294 Trent McConaghy et al, "Top-down multi-objective design methodology"

- Pending No. 20050216324 Trent McConaghy et al, "System and method for constructing a schedule that better achieves one or more business goals"

Invited Talks

- Trent McConaghy, “Efficient optimization for industrial analog circuit design”, ESAT-MICAS, Katholieke Universiteit Leuven, Leuven, Belgium, 2003
- Trent McConaghy, “Industrial-scale analog synthesis”, Jet Propulsion Laboratory, Pasadena, USA, 2003
- Trent McConaghy, “Efficient optimization for industrial analog circuit design”, University of Cincinatti, Cincinatti, USA, 2003
- Trent McConaghy, “Efficient optimization for industrial analog circuit design”, State University New York, Cincinatti, USA, 2003
- Trent McConaghy, “A multi-objective hierarchical methodology for synthesis of large scale electronic designs”, 3rd NASA/DoD Workshop on Evolvable Hardware, Long Beach, CA, 2001

Magazine Feature Articles

- Trent McConaghy, “Using Synthesis in the analog design flow”, *EDN*, Jan 24, 2002
- Trent McConaghy, “Intelligent systems solutions for analog synthesis”, *Integrated Communications Design*, Penwell, January, 2002
- Trent McConaghy, “Forget digital, think analog”, *Electronic News*, Oct 28, 2002

Bibliography

[Agg2007] V. Aggarwal, U.-M. O'Reilly (2007). Simulation-based reusable posynomial models for MOS transistor parameters In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 69–74.

[Alp2003] G. Alpaydin, S. Balkir, and G. Dundar (2003). An evolutionary approach to automatic synthesis of high-performance analog integrated circuits. *IEEE Transactions on Evolutionary Computation*, 7(3), pp. 240–252.

[Amp2002] N. Ampazis, and S.J. Perantonis (2002). Two highly efficient second order algorithms for training feedforward networks. In *IEEE Transactions on Neural Networks*, 13(5), Sept. 2002, pp. 1064–1074.

[Ang1996] P.J. Angeline (1996). Two self-adaptive crossover operators for genetic programming. In *Advances in Genetic Programming 2*, MIT Press, chapter 5, pp. 90–115.

[Ando2003] S. Ando, M. Ishizuka, and H. Iba (2003). Evolving analog circuits by variable length chromosomes. In *Advances in evolutionary computing*, A. Ghosh and S. Tsutsui, Eds. New York: Springer, pp. 643–662.

[Ant1995] B.A.A. Antao, and A.J. Brodersen. (1995). ARCHGEN: Automated synthesis of analog systems. In *IEEE Transactions on Very Large Scale Integrated Circuits*, 3(2), pp. 231–244.

[Ant1994] K. Antreich, H. Graeb, and C. Wieser (1994). Circuit analysis and optimization driven by worst-case distances. In *IEEE Transactions on Computer-Aided Design*, 13(1), pp. 57–71.

[Ant2000] K. Antreich, J. Eckmueller, H.E. Graeb, M. Pronath, F. Schenkel, R. Schwencker, and S. Zizala (2000). WiCkeD: Analog circuit synthesis incorporating mismatch. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*.

[Ash2002] P.J. Ashenden, G.D. Peterson, and D.A. Teegarden (2002). *The System Designer's Guide to VHDL-AMS*. Morgan Kaufmann.

[Au2003] W.-H. Au, K.C.C. Chan, and X. Yao (2003). A novel evolutionary data mining algorithm with applications to churn prediction. *IEEE Transactions on Evolutionary Computation*, 7(6), Dec. 2003, pp. 532–545.

[Aug2005] A. Auger and N. Hansen (2005). A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pp. 1769–1776.

[Bai2006] W.S. Bainbridge, M.C. Roco, editors (2006). *Managing Nano-Bio-Info-Cogno Innovations: Converging Technologies in Society*. Springer, ISBN: 1402041063.

[Bec2007] Y. Becker, P. Fei, and A.M. Lester. (2007). Stock selection: An innovative application of genetic programming methodology. In R.L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12. Springer, Ann Arbor.

[Ber1988] E. Berkcan, M. d'Abreu, and W. Laughton (1988). Analog compilation based on successive decompositions. In *Proceedings of the Design Automation Conference (DAC)*, pp. 369–375.

[Ber2003] F. De Bernardinis, M.I. Jordan, and A. L. Sangiovanni-Vincentelli (2003). Support vector machines for analog circuit performance representation. In *Proceedings of the Design Automation Conference (DAC)*, pp. 964-969.

[Ber2005] F. De Bernardinis, P. Nuzzo, and A.L. Sangiovanni-Vincentelli (2005). Mixed signal design space exploration through analog platforms. In *Proceedings of the Design Automation Conference (DAC)*, pp. 875–880.

[Bha2004] S. Bhattacharya, N. Jangkrajarng, R. Hartono, and R. Shi (2004). Correct-by-construction layout-centric retargeting of large analog designs. In *Proceedings of the Design Automation Conference (DAC)*.

[Box2005] G.E.P. Box, W.G. Hunter, and J.S. Hunter (2005). *Statistics for Experimenters: Design, Innovation, and Discovery, 2nd Edition*. Wiley-Interscience. ISBN: 0471718130.

[Boyd2004] S. Boyd and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press, ISBN: 0521833787.

[Bro2004] E. Brooks and J. Grossman (2004). Improved algorithms speed it up for codes. In *Science and Technology Review (S & TR)*, Lawrence Livermore Laboratory, May 2004, pp. 16-19.

[Bre1984] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone (1984). *Classification and Regression Trees*. Chapman & Hall.

[Bre2001] L. Breiman (2001). Random Forests. In *J. Machine Learning* 45(1), pp. 5–32.

[Can2000] E. Cantu-Paz, and D.E. Goldberg (2000). Parallel genetic algorithms: theory and practice. In *Computer Methods in Applied Mechanics and Engineering*, Elsevier.

[Cas2005] F. Castillo, A. Kordon, J. Sweeney, and W. Zirk (2005). Using genetic programming in industrial statistical model building. In U.-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 3, pp. 31–48. Springer, Ann Arbor.

[Cdn2005a] Cadence Design Systems, Inc. (2005). Product page: Virtuoso NeoCell. http://www.cadence.com, last accessed 2005.

[Cdn2005b] Cadence Design Systems, Inc. (2005). Product page: Virtuoso NeoCircuit. http://www.cadence.com, last accessed 2005.

[Cdn2005c] Cadence Design Systems, Inc. (2005). Product page: Neocircuit DFM (yield optimization). http://www.cadence.com, last accessed 2005.

[Cdn2008d] Cadence Design Systems, Inc. (2008). Product page: Virtuoso Spectre circuit simulator. http://www.cadence.com/products/custom_ic/spectre/index.aspx, last accessed June 19, 2008.

[Cdn2008e] Cadence Design Systems, Inc. (2008). Product page: Virtuoso AMS Designer simulator http://www.cadence.com/products/custom_ic/ams_designer/index.aspx, last accessed June 19, 2008.

[Cdn2008f] Cadence Design Systems, Inc. (2008). Product page: Virtuoso schematic editor. http://www.cadence.com/products/custom_ic/vschematic/index.asp, last accessed June 19, 2008.

[Cdn2008g] Cadence Design Systems, Inc. (2008). Product page: Virtuoso Analog Design Environment. http://www.cadence.com/products/custom_ic/analog_design/index.aspx, last accessed June 19, 2008.

[Cdn2008h] Cadence Design Systems, Inc. (2008). Product page: Virtuoso XL Layout Editor. w2.cadence.com/products/custom_ic/vxlayout/index.aspx, last accessed November 1, 2008.

[Cdn2008i] Cadence Design Systems, Inc. (2008). Product page: Assura Layout vs. Schematic Checker. http://w2.cadence.com/products/dfm/assura_lvs/index.aspx, last accessed November 1, 2008.

[Cdn2008j] Cadence Design Systems, Inc. (2008). Product page: Assura Parasitic Extraction. http://w2.cadence.com/products/dfm/assura_rcx/index.aspx, last accessed November 1, 2008.

[Cha1997] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, E. Malavasi, A.L. Sangiovanni-Vincentelli, I. Vassiliou (1997). *A Top Down, Constraint Driven Design Methodology for Analog Integrated Circuits*. Kluwer. ISBN: 0792397940.

[Cha2006] S.-J. Chang, H.-S. Hou, and Y.-K. Su (2006). Automated passive filter synthesis using a novel tree representation and genetic programming. *IEEE Trans. Evolutionary Computation* 10 (1), Feb. 2006, pp. 93–100.

[Cor2007] D. Corne, and J. Knowles (2007). Techniques for highly multiobjective optimization: Some nondominated points are better than others. In Thierens, Dirk and et al., editors, *GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 773–780.

[Cov2007] T.M. Cover, and J.A. Thomas (2006). *Elements of information theory, 2nd edition.* Wiley-Interscience, New York, ISBN: 0-471-24195-4.

[Dae2002] W. Daems, G.G.E. Gielen, and W.M.C. Sansen (2002). An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits *Proceedings of the Design Automation Conference (DAC)*, pp. 431–436.

[Dae2003] W. Daems, G.G.E. Gielen, W.M.C. Sansen (2003). Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE Transactions Computer-Aided Design* 22(5), May 2003, pp. 517–534.

[Dae2005] W. Dames, B. De Smedt, E. Lauwers, E. Yannis, and W. Verhaegen (2005). *Method and apparatus for designing electronic circuits.* Patent application. US Number US20050257178, International Number PCT/US2005/016091, filed 05.05.2005, pub. no. WO/2005/114503, pub. date 01.12.2005.

[Dan1963] G.B. Dantzig (1963). *Linear programming and extensions*. Princeton University Press, Princeton, N.J.

[Das2005] T.R. Dastidar, P.P. Chakrabarti, and P. Ray. (2005). A synthesis system for analog circuits based on evolutionary search and topological reuse. *IEEE Transactions on Evolutionary Computation*, 9(2), pp. 211–224.

[Deb1998] G. Debyser, and G.G.E. Gielen (1998). Efficient analog circuit synthesis with simultaneous yield and robustness optimization. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 308–311.

[Deb2002] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation*, 6(2), pp. 182–197.

[Dem2005] I. Dempsey, M. O'Neill, and A. Brabazon (2005). Meta-grammar constant creation with grammatical evolution by grammatical evolution. In H.-G. Beyer, et al., editors, *GECCO 2005: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 2, pp. 1689–1696, Washington DC, USA. ACM Press.

[Ding2005] M. Ding, and R. Vemuri (2005). A Two-Level Modeling Approach to Analog Circuit Performance Macromodeling. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 1088–1089.

[Ding2005b] M. Ding, and R. Vemuri (2005). A combined feasibility and performance macromodel for analog circuits. In *Proceedings of the Design Automation Conference (DAC)*, pp. 63–68.

[Ding2006] M. Ding (2006). Regression based analog performance macromodeling: techniques and applications. PhD dissertation, *Dept. Computer Science and Engineering, University of Cincinnati*.

[Dob2003] A. Doboli, and R. Vemuri (2003). Exploration-based high-level synthesis of linear analog systems operating at low/medium frequencies. *IEEE Transactions on Computer-Aided Design*, 22(11), pp. 1556-1568.

[Dong2008] N. Dong, and J. Roychowdhury (2005). Nonlinear model order reduction using piecewise polynomial representations. In *IEEE Trans. Computer-Aided Design*, 27(2), Feb. 2008, pp. 249–264.

[Dre1999] P. Drennan, and C. McAndrew (1999). A comprehensive MOSFET mismatch model. In *Proc International Electron Devices Meeting (IEDM)*.

[Dre2003] P.C. Drennan, and C.C. McAndrew (2003). Understanding MOSFET mismatch for analog design. In *IEEE Journal of Solid-State Circuits*, 38(3), March 2003, pp. 450–456.

[Dre2006] P.G. Drennan, M.L. Kniffin, and D.R. Locascio (2006). Implications of proximity effects for analog design. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*.

[Edac2006] Electronic Design Automation Consortium (EDAC) Market Statistics Service (2006). 2006 Annual Summary Report. http://www.edac.org/downloads/mss_2006/MSS_2006_AnnualSummary Report_20070328.pdf, last accessed June 19, 2008.

[Eec2005] T. Eeckelaert, T. McConaghy, and G.G.E. Gielen. (2005). Efficient multi-objective synthesis of analog circuits using hierarchical pareto-otimal performance hypersurfaces. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 1070–1075.

[Eec2007] T. Eeckelaert, R. Schoofs, G.G.E. Gielen, and M. Steyaert (2007). An efficient methodology for hierarchical synthesis of mixed-signal systems with fully integrated building block topology selection. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 81–86.

[Efr1979] B. Efron (1979). Bootstrap methods: another look at the jacknife. In *The Annals of Statistics*, 7(1), pp. 1–26.

[Ext2008] Extreme Design Automation (DA), Inc. (2008). Product page: Road Suite (yield optimization). http://extreme-da.com/Road_Suite.html, last accessed June 19, 2008

[Fay2006] A. Fayed, and M. Ismail (2006). *Adaptive Techniques for Mixed Signal System on Chip*. Springer, Dordrecht, The Netherlands. ISBN: 0387321543.

[Feng2006] Z. Feng, and P. Li (2006). Performance-oriented statistical parameter reduction of parameterized systems via reduced rank regression. In *Proceedings of the International Conference on Computer Aided Design*, pp. 868–875.

[Fer1991a] F. Fernandez, A. Rodriguez-Vazquez, and J. Huertas (1991). A tool for symbolic analysis of analog integrated circuits including pole/zero extraction. *Proceedings European Conference on Circuit Theory and Design (ECCTD)*, pp. 752–761.

[Fer1991b] F. Fernandez, A. Rodriguez-Vazquez, and J. Huertas (1991). Interactive AC modeling and characterization of analog circuits via symbolic analysis. *Kluwer Journal on Analog Integrated Circuits and Signal Processing*, Vol. 1, November 1991, pp. 183-208.

[Flo1993] H. Floberg, and S. Mattison (1993). Computer aided symbolic analysis of large analog integrated circuits. *Alta Frequenza - Rivista di Elettronica* 5(6), November-December 1993, pp. 24–28.

[Fung1988] A.H. Fung, D.J. Chen, Y.N. Li, and B.J. Sheu (1988). Knowledge-based analog circuit synthesis with flexible architecture In *Computer Design: Proceedings of the International Conference on VLSI Computers and Processors*, pp. 48–51.

[Fra2000] K. Francken, P. Vancorenland, and G.G.E. Gielen (2000). DAISY: A simulation-based high-level synthesis tool for delta-sigma modulators In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 188–192

[Fre1997] Y. Freund, and R.E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Science*, 55(1), pp. 119–200.

[Fri1991] J.H. Friedman (1991). Multivariate adaptive regression splines. *Annals of Statistics*, 19, March 1991, pp. 1–141.

[Fri2002] J.H. Friedman (2002). Stochastic gradient boosting. *Journal of Computational Statistics & Data Analysis*, 38(4), pp. 367–378.

[Fri2003] J.H. Friedman, and B. Popescu (2003). Importance sampled learning ensembles. Technical Report, *Stanford University Department of Statistics*

[Fri2004] J.H. Friedman, and B.E. Popescu (2004). Gradient directed regularization for linear regression and classification. Technical Report, *Stanford University Department of Statistics*, Feb. 2004

[Fri2005] J. H. Friedman, and B.E. Popescu (2005). , Predictive Learning via Rule Ensembles Technical Report, *Stanford University Department of Statistics*, Feb. 2005

[Gao2008a] P. Gao, T. McConaghy, and G.G.E. Gielen (2008). ISCLEs: Importance sampled circuit learning ensembles for trustworthy analog circuit topology synthesis In *Proceedings of the International Conference on Evolvable Systems (ICES)*, Prague, CZ, September 2008.

[Gao2008b] P. Gao, T. McConaghy, and G.G.E. Gielen (2008b). ISCLEs: Importance sampled circuit learning ensembles for robust analog IC design In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, November 2008.

[Gie1989] G.G.E. Gielen, H. Walscharts, and W.M.C. Sansen (1989). ISAAC: A symbolic simulator for analog integrated circuits. *IEEE Journal of Solid-State Circuits* 24(6), Dec. 1989, pp. 1587-1597.

[Gie1990] G.G.E. Gielen, and W.M.C Sansen, editors (1990). *Proceedings of the International Summer Course on Systematic Analogue Design* Leuven, 2 volumes, June 1990.

[Gie2002a] G.G.E. Gielen, and R.A. Rutenbar (2002). Computer-aided design of analog and mixed-signal integrated circuits. In R.A. Rutenbar, G.G.E. Gielen, and B.A.A. Antao, editors, *Computer Aided Design of Analog Integrated Circuits and Systems*, chapter 1, pp. 3–30. IEEE Press, Piscataway, NJ.

[Gie2002b] G.G.E. Gielen (2002). Techniques and applications of symbolic analysis for analog integrated circuits: A tutorial overview. In R.A. Rutenbar, G.G.E. Gielen, and B.A.A. Antao, editors, *Computer Aided Design of Analog Integrated Circuits and Systems*, pp. 245–261. IEEE Press, Piscataway, NJ.

[Gie2005] G.G.E. Gielen, T. McConaghy, and T. Eeckelaert (2005). Performance space modeling for hierarchical synthesis of analog circuits. In *Proceedings of the Design Automation Conference (DAC)*, pp. 1070–1075.

[Gie2005b] G.G.E. Gielen, W. Dehaene, P. Christie, D. Draxelmayr, E. Janssens, K. Maex, and T. Vucurevich (2005). Analog and digital circuit design in 65 nm CMOS: End of the road? In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 36–42.

[Goh2001] C. Goh, Y. Li (2001). GA automated design and synthesis of analog circuits with practical constraints. In *The Congress on Evolutionary Computation 1*, pp. 170–177.

[Gra2001] H.E. Graeb, S. Zizala, J. Eckmueller, and K. Antreich (2001). The sizing rules method for analog integrated circuit design. In *International Conference on Computer-Aided Design*, pp. 343–349.

[Gra2007] H.E. Graeb (2007), *Analog design centering and sizing.* Springer, ISBN-10: 1402060033.

[Gri2000] J. B. Grimbleby (2000). Automatic analogue circuit synthesis using genetic algorithms. *Proceedings of IEEE Circuits, Systems, and Devices* 147(6), Dec. 2000, pp. 319–323.

[Gol1989] D. E. Goldberg (1989). *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley. ISBN: 0201157675.

[Gol2002] D.E. Goldberg (2002). *The design of innovation: lessons from and for competent genetic algorithms*. Springer, ISBN:1402070985.

[Han2001] N. Hansen, A. Ostermeier (2001). Completely derandomized self-adaptation in evolution strategies In *Evolutionary Computation*, 9(2), pp. 159–195.

[Har1992] R. Harjani, R.A. Rutenbar, and L.R. Carley. (1992). OASYS: A framework for analog circuit synthesis. In *IEEE Transactions on Computer-Aided Design*, 8(12), pp. 1247–1266.

[Has1989] M. Hassoun, and P. Lin (1989). A new network approach to symbolic simulation of large-scale networks. *Proceedings International Symposium on Circuits and Systems (ISCAS)*, pp. 806-809.

[Has2000] A. Hastings (2000). *The Art of Analog Layout*. Prentice-Hall.

[Has2001] T. Hastie, R. Tibshirani, and J.H. Friedman (2001). *The elements of statistical learning*. Springer. ISBN: 0387952845.

[Her1998] M. del Mar Hershenson, S.P. Boyd, T.H. Lee (1998). GPCAD: a tool for CMOS op-amp synthesis. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 296–303.

[Hes2003] T. Hesterberg (2003). *Advances in importance sampling*. PhD dissertation, *Stanford University Department of Statistics*, 1998, revised 2003.

[Hong2003] X. Hong, P.M. Sharkey, K. Warwick (2003). A robust nonlinear identification algorithm using PRESS statistic and forward regression. In *IEEE Transactions Neural Networks* 14(2), March 2003, pp. 454–458.

[Hoo1961] R. Hooke, T.A. Jeeves (1961). Direct search solution of numerical and statistical problems. In *Journal of the ACM*, Vol. 8, April 1961, pp. 212–229.

[Hor1970] A.E. Horel, and R.W. Kennard (1970). Ridge regression: biased estimation for nonorthogonal problems. In *Technometrics*, 12, pp. 56–67.

[Hor2003] G.S. Hornby (2003). *Generative representations for evolutionary design automation*. PhD dissertation, Brandeis University, Dept. of Computer Science, Boston, MA, USA.

[Hor2006] G.S. Hornby (2006). ALPS: The age-layered population structure for reducing the problem of premature convergence. In M. Keijzer, et al., editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pp. 815–822, Seattle, Washington, USA. ACM Press.

[Hor1997] N.C. Horta, and J.E. Franca (1997). Algorithm-driven synthesis of data conversion architectures In *IEEE Transactions on Computer-Aided Design*, 10(16), Oct. 1997, pp. 1116–1135

[Hu2004] J. Hu, and E. Goodman (2004). Robust and efficient genetic algorithms with hierarchical niching and sustainable evolutionary computation model. In *GECCO 2004: Proceedings of the 6th annual conference on genetic and evolutionary computation*, ACM Press.

[Hu2005a] J. Hu, and E. Goodman (2005). Topological synthesis of robust dynamic systems by sustainable genetic programming. In U-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 9. Kluwer, Ann Arbor.

[Hu2005b] J. Hu, E.K. Goodman, K. Seo, Z. Fan, and R. Rosenberg (2005). The hierarchical fair competition framework for sustainable evolutionary algorithms. *Evolutionary Computation*, 13(2), summer 2005, pp. 241–277.

[Hu2008] B. Hu, C.-J.R. Shi (2008). Simulation of closely related dynamic nonlinear systems with application to process / voltage / temperature corner analysis In *IEEE Transactions on Computer-Aided Design*, 27(3), May 2008, pp. 883–892.

[Huy1996] M.A. Huynen, P. Stadler, and W. Fontana (1996). Smoothness within ruggedness: The role of neutrality in adaptation. *National Academy of Sciences USA*, 93, pp. 397–401.

[Iba1996] H. Iba (1996). Random tree generation for genetic programming. In Voigt, Hans-Michael, Ebeling, Werner, Rechenberg, Ingo, and Schwefel, Hans-Paul, editors, *Parallel Problem Solving from Nature IV, Proceedings of the International Conference on Evolutionary Computation*, volume 1141 of *LNCS*, pp. 144–153, Berlin, Germany. Springer Verlag.

[Itrs2007] International Technology Roadmap for Semiconductors (ITRS) (2007). http://public.itrs.net, last accessed April 2008.

[Jar1968] N. Jardine, R. Sibson (1968). The construction of hierarchic and non-hierarchic classifications. In *The Computer Journal*, 11:177.

[Joh1997] D. Johns, and K. Martin (1997). *Analog Integrated Circuit Design.* Wiley. ISBN: 0471144487.

[Jon1998] D.R. Jones, M. Schonlau, and W.J. Welch (1998). Efficient global optimization of expensive black-box functions. In *Journal of Global Optimization*, Vol. 13, pp. 455–492.

[Kam2000] J. Kampe (2000). A new approach for the structural synthesis of analog subsystems. In *International Workshop on Symbolic Methods and Applications in Circuit Design*, pp. 33–38.

[Kei1999] M. Keijzer, and V. Babovic (1999). Dimensionally aware genetic programming. In *GECCO 1999: Proceedings of the conference on Genetic and evolutionary computation*, vol. 2, July 1999.

[Kei2001] M. Keijzer (2001). Scientific discovery using genetic programming. PhD dissertation, *Water & Environment and the Dept. for Mathematical Modelling, T.U. Denmark.*

[Kei2003] M. Keijzer (2003). Improving symbolic regression with interval arithmetic and linear scaling. In *Proceedings of the European Conference on Genetic Programming (EuroGP)*, LNCS 2610, 2003, pp. 71–83.

[Kei2004] M. Keijzer (2004). Alternatives in subtree caching for genetic programming. In *Proceedings of the European Conference on Genetic Programming (EuroGP)*, pp. 328–337.

[Kei2004b] M. Keijzer (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3), pp. 259–269.

[Kie2004] T. Kiely, and G.G.E. Gielen (2004). Performance modeling of analog integrated circuits using least-squares support vector machines. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, 2004, pp. 448-453.

[Kir2004] E. Kirshenbaum, and H.J. Suermondt (2004). Using genetic programming to obtain a closed-form approximation to a recursive function. In *GECCO 2004: Proceedings of the 6th annual conference on genetic and evolutionary computation*, ACM Press, pp. 543–556.

[Koh1990] H.Y. Koh, C.H. Séquin, and P.R. Gray (1990). OPASYN: A compiler for CMOS operational amplifiers. *IEEE Transactions on Computer-Aided Design*, vol. 9, pp. 113–125.

[Kol2003] T.G. Kolda, R.M. Lewis, V. Torczon (2003). Optimization by direct search: new perspectives on some classical and modern methods. *SIAM Review*, 45(3), pp. 385–482.

[Kon1988] A. Konczykowska, and M. Bon (1988). Automated design software for switched-capacitor ICs with symbolic simulator SCYMBAL. *Proceedings Design Automation Conference (DAC)*, pp. 363–368.

[Kor2006] A. Kordon, F. Castillo, G. Smits, and M. Kotanchek (2006). Application issues of genetic programming in industry. In T.Yu et al., editors, *Genetic Programming Theory and Practice III*, volume 4 of *Genetic and Evolutionary Computation*, chapter 18, Springer, pp. 241-258.

[Kor2007] M.F. Korns (2007). Large-scale, time-constrained symbolic regression. In R.L. Rick, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 16. Springer.

[Koza1992] J.R. Koza (1992). *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA.

[Koza1994] J.R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, USA.

[Koza1997] J.R. Koza et al (1997). Automated synthesis of analog integrated circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation* 1(2), July 1997, pp. 109–128.

[Koza1999] J.R. Koza, D. Andre, F.H. Bennett III, and M. Keane (1999). *Genetic programming 3: Darwinian invention and problem solving*. Morgan Kaufman.

[Koza2003] J.R. Koza, M.A. Keane, M.J. Streeter, W. Mydlowec, J. Yu, and G. Lanza (2003). *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Publishers.

[Koza2004a] J.R. Koza, M.J. Streeter, and M.A. Keane (2004). Automated synthesis by means of genetic programming of complex structures incorporating reuse, hierarchies, development, and parameterized topologies. In R.L. Riolo, and B.

Worzel *Genetic Programming Theory and Practice I*, chapter 14, pp. 221–237. Kluwer.

[Koza2004b] J.R. Koza, L.W. Jones, M.A. Keane, and M.J. Streeter (2004). Routine high-return human-competitive evolvable hardware. In R.S. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. Lohn, and A. Stoica, editors, *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, pp. 3–17, Seattle. IEEE Press.

[Koza2004c] J.R. Koza (2004). Tutorial on automated invention using genetic programming. In *Proc. American Association of Artificial Intelligence (AAAI)*, San Jose, July 25, 2004; also available at http://www.genetic-programming.com/aaai2004tutorial.pdf, last accessed June 19, 2008.

[Koza2005] J.R. Koza, L.W. Jones, M.A. Keane, and M.J. Streeter (2005). Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In U-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 8. Kluwer, Ann Arbor.

[Kru1995] W. Kruiskamp, and D. Leenaerts (1995). DARWIN: CMOS opamp synthesis by means of a genetic algorithm. In *Proceedings of the Design Automation Conference (DAC)*.

[Kun2004] K. Kundert, and O. Zinke. (2004). *The Designer's Guide to Verilog-AMS*. Kluwer.

[Kur2005] R. Kurzweil (2005). *The Singularity Is Near: When Humans Transcend Biology.* Viking Adult, ISBN: 0670033847.

[Lai2006] X. Lai, and J. Roychowdhury (2006). Macromodeling oscillators using krylov-subspace methods. In *Proceedings of the Asia And South Pacific Design Automation Conference (ASP-DAC)*.

[Lak1994] K.R. Laker, and W.M.C. Sansen (1994). *Design of analog integrated circuits and systems.* McGraw-Hill. ISBN: 007036060.

[Lam1999] K. Lampaert, G.G.E. Gielen, and W.M.C. Sansen (1999). *Analog Layout Generation for Performance and Manufacturability.* Kluwer Academic Publishers.

[Lan1987] P. Langley, H.A. Simon, G.L. Bradshaw, And J.M. Zytkow (1987). *Scientific Discovery, Computational Explorations of the Creative Process.* The MIT Press. ISBN:0-262-62052-9

[Leyn1998] F. Leyn, G.G.E. Gielen, and W.M.C. Sansen (1998). An efficient dc root solving algorithm with guaranteed convergence for analog integrated CMOS

circuits. In *International Conference on Computer-Aided Design*, pp. 304–307.

[Li2006] X. Li, J. Le, and L.T. Pileggi (2006). Statistical performance modeling and optimization, Section 3.3. In *Foundations and Trends in Electronic Design Automation*, 1(4), pp. 368–384.

[Li2007a] X. Li, J. Le, P. Gopalakrishnan, and L. Pileggi (2007). Asymptotic probability extraction for nonnormal performance distributions. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(1), pp. 16–37, Jan. 2007.

[Li2007b] X. Li, P. Gopalakrishnan, Y. Xu, and L. Pileggi (2007). Robust analog/RF circuit design with projection-based performance modeling. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(1) 1, pp. 2–15, Jan. 2007.

[Li2007c] X. Li, B. Taylor, Y.-T. Chien, and L.T. Pileggi (2007). Adaptive post-silicon tuning for analog circuits: concept, analysis and optimization. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 450–457.

[Li2008] X. Li, Y. Zhan, L.T. Pileggi (2008). Quadratic statistical MAX approximation for parametric yield Estimation of analog/RF integrated circuits. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(5), pp. 831–843, May 2008.

[Li2008b] X. Li, and Y. Cao (2008). Projection-based piecewise-linear response surface modeling for strongly nonlinear VLSI performance variations. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*, pp. 108–113.

[Li2008c] X. Li, and H. Liu (2008). Statistical regression for efficient high-dimensional modeling of analog and mixed-signal performance variations. In *Proceedings of the Design Automation Conference (DAC)*, pp. 38–43.

[Lin1968] A. Lindenmayer (1968). Mathematical models for cellular interaction in development I. Filaments with one-sided inputs In *Journal of Theoretical Biology*, 18, pp. 280-289

[Liu2002] H. Liu, A. Singhee, R.A. Rutenbar, and L.R. Carley (2002). Remembrance of circuits past: Macromodeling by data mining in large analog design spaces. In *Proceedings of the Design Automation Conference (DAC)*, pp. 437-442.

[Lohn1998] J.D. Lohn, and S.P. Colombano (1998). Automated analog circuit synthesis using a linear representation. In *Proceedings of the Second International Conference on Evolvable Systems: From Biology To Hardware (ICES)*, pp. 125–133. Springer-Verlag.

[Loe2009] J. Loeckx, T. Deman, T. McConaghy, and G.G.E. Gielen (2009, to appear). A novel EMI-immune current mirror topology obtained by genetic evolution. In *Proceedings of the Conference in Electro Magnetic Compatibility (EMC '09)*, Zurich, 2009

[Lohn2005] J. Lohn, G. Hornby, and D. Linden (2005). Evolutionary antenna design for a NASA spacecraft. In U.-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 3, pp. 31–48. Springer, Ann Arbor.

[Mag2008] Magma Design Automation, Inc. (2008) Product page: Titan Chip Finishing http://www.magma-da.com/products-solutions/customdesign/ titanchipfinishing.aspx, last accessed June 19, 2008.

[Man1991] S. Manetti (1991). New approaches to automatic symbolic analysis of electric circuits. *IEE Proceedings part G*, February 1991, pp. 22–28.

[Man2003] A. Manthe, Z. Li, and C.-J. Richard Shi (2003). Symbolic analysis of analog circuits with hard nonlinearity. In *Proceedings of the Design Automation Conference (DAC)*, pp. 542–545.

[Mat2007] C. Mattiussi, D. Floreano (2007). Analog genetic encoding for the evolution of circuits and networks. In *IEEE Transactions on Evolutionary Computation*, 11(5), pp. 596–607.

[Mau1995] P.C. Maulik, L.R. Carley, and R.A. Rutenbar (1995). Integer programming based topology selection of cell level analog circuits. *IEEE Transactions on Computer-Aided Design*, 14(4), pp. 401–412.

[Mar2006] E. Martens, and G.G.E. Gielen (2006). Top-down heterogeneous synthesis of analog and mixed-signal systems. In *Design Automation and Test Europe (DATE)*, pp. 275–280.

[Mar2008] E. Martens and G.G.E. Gielen (2008). *High-level modeling and synthesis of analog integrated systems.* Springer. ISBN: 9781402068010.

[Mat2008] The Mathworks, Inc. (2008). "classregtree()" routine in Matlab 7.5. *http://www.mathworks.com*, last accessed April 23, 2008

[Mca2003] C.C. McAndrew (2003). Statistical modeling for circuit simulation. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*.

[Mcc2005a] T. McConaghy, T. Eeckelaert, and G.G.E. Gielen (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 Conference (DATE)*, March 2005, pp. 1082–1087.

[Mcc2005b] T. McConaghy, and G.G.E. Gielen (2005). IBMG: Interpretable behavioral model generator for nonlinear analog circuits via canonical form functions and genetic programming. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, May 2005.

[Mcc2005c] T. McConaghy, and G.G.E. Gielen (2005). Analysis of simulation-driven numerical performance modeling techniques for application to analog circuit optimization. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, May 2005.

[Mcc2005d] T. McConaghy, T. Eeckelaert, and G.G.E. Gielen (2005). CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming. In *Proceedings of the European Solid-State Circuits Conference (ESSCIRC)*, September 12-15, 2005.

[Mcc2006a] T. McConaghy, and G.G.E. Gielen (2006). Double-strength CAFFEINE: Fast template-free symbolic modeling of analog circuits via implicit canonical form functions and explicit introns. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, March 2006, pp. 269–274.

[Mcc2006b] T. McConaghy, and G.G.E. Gielen (2006). Automation in mixed-signal design: Challenges and solutions in the wake of the nano era. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 461–463.

[Mcc2006c] T. McConaghy, and G.G.E. Gielen (2006). Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, July 2006, pp. 855-862.

[Mcc2006d] T. McConaghy, and G.G.E. Gielen (2006). Genetic programming in industrial analog CAD: Applications and challenges. In T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice III*, volume 9 of *Genetic Programming*, chapter 19, pp. 291–306. Springer, Ann Arbor.

[Mcc2007] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2007). Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies. In *Proceedings of the Design Automation Conference (DAC)*.

[Mcc2008a] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Genetic programming with reuse of known designs. In R.L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice V*, volume 6 of *Genetic and Evolutionary Computation*, chapter 10, pp. 161–186. Springer, Ann Arbor.

[Mcc2008b] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Automated extraction of expert knowledge in analog topology selection and sizing.

In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, San Jose, November 2008.

[Mcc2008c] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Trustworthy synthesis of analog circuit topologies via hierarchical domain-specific building blocks In *IEEE Transactions on Evolutionary Computation*, (submitted 2008).

[Mcc2008d] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Automated extraction of expert domain knowledge from genetic programming synthesis results. In *IEEE Transactions on Evolutionary Computation*, (submitted 2008).

[Mcc2008e] T. McConaghy, and G.G.E. Gielen (2008). Globally reliable, variation-aware sizing of analog integrated circuits via response surface modeling and structural homotopy. In *IEEE Transactions on Computer-Aided Design*, (submitted 2008).

[Mcc2008f] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Trustworthy, variation-aware structural synthesis of analog circuits via structural homotopy. In *IEEE Transactions on Computer-Aided Design*, (submitted 2008).

[Mcc2008g] T. McConaghy, and G.G.E. Gielen (2008). Template-free symbolic performance modeling of analog circuits via canonical form functions and genetic programming. In *IEEE Transactions on Computer-Aided Design*, (submitted 2008).

[Mcc2009] T. McConaghy, P. Palmers, G.G.E. Gielen, and M. Steyaert (2008). Automated extraction of expert domain knowledge from genetic programming synthesis results. In *Genetic Programming Theory and Practice VI*, Edited by R. Riolo and B. Worzel, Springer, 2009 (to appear).

[Mck1979] M.D. McKay, W.J. Conover, and R.J. Beckman (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. In *Technometrics*, 21, pp. 239–245.

[Mck2000] M.D. McKay, R.J. Beckman, and W.J. Conover (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. In *Technometrics*, 41(1), pp. 55–61.

[Mck1999] B. McKay, M.J. Willis, D.P. Searson, and G.A. Montague (1999). Nonlinear continuum regression using genetic programming. In *GECCO 1999: Proceedings of the conference on Genetic and evolutionary computation*, vol. 2, July 1999, pp. 1106–1111.

[Mead1980] C. Mead, and L. Conway (1980). *Introduction to VLSI systems.* Addison Wesley. ISBN: 0201043580.

[Medea2005] MEDEA+ EDA Roadmap, Version 5 (2005). http:///www.medeaplus.org

[Meh1998] V. Mehotra, S. Nassif, D. Boning, and J. Chung (1998). Modeling the effects of manufacturing variation on high-speed microprocessor performance. In *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, pp. 767–770.

[Men2008a] Mentor Graphics, Inc. (2008) Product page: Eldo circuit simulator. http://www.mentor.com/eldo/overview.html, last accessed June 19, 2008.

[Men2008b] Mentor Graphics, Inc. (2008) Product page: Calibre nmDRC. http://www.mentor.com/products/ic_nanometer_design/bl_phy_design/calibre_drc/, last accessed June 19, 2008.

[Men2008c] Mentor Graphics, Inc. (2008) Product page: Calibre nmLVS. http://www.mentor.com/products/ic_nanometer_design/bl_phy_design/calibre_lvs/, last accessed June 19, 2008.

[Men2008d] Mentor Graphics, Inc. (2008) Product page: Calibre xRC. http://www.mentor.com/products/ic_nanometer_design/bl_phy_design/calibre_xrc/, last accessed June 19, 2008.

[Men2008e] Mentor Graphics, Inc. (2008) Product page: Design Architect IC (analog / custom design environment) http://www.mentor.com/products/ic_nanometer_design/custom_design_simulation/design_architect_ic/, last accessed November 1, 2008.

[Men2008f] Mentor Graphics, Inc. (2008) Product page: IC Station (layout editor) http://www.mentor.com/products/ic_nanometer_design/events/custom_ic_layout.cfm, last accessed November 1, 2008.

[Men2008g] Mentor Graphics, Inc. (2008) Product page: Eldo Circuit Simulator. www.mentor.com/eldo/overview.html, last accessed November 1, 2008.

[Mic1998] Z. Michalewicz (1998). *Genetic algorithms + data structures = evolution programs*. Springer. ISBN: 978-3-540-60676-5.

[Mon2004] D.C. Montgomery (2004). *Design and analysis of experiments*, 6^{th} edition John Wiley & Sons, NY, NY, USA. ISBN: 047148735X.

[Moo1965] G.E. Moore (1965). Cramming more components onto integrated circuits. In *Electronics Magazine*, 38(8), April 19, 1965.

[Mun2008] MunEDA, Inc. (2008). Product page: YOP - Yield Optimization. http://www.muneda.com/Products_Yield-Optimization, last accessed June 19, 2008.

[Nag1973] L.W. Nagel and D.O. Pederson (1973). SPICE (Simulation Program with Integrated Circuit Emphasis). *University of California, Berkeley, Department of Electrical Engineering*, Memorandum No. ERL-M382, April 1973.

[Nag1975] L.W. Nagel (1975). SPICE2: A Computer Program to Simulate Semiconductor Circuits. *University of California, Berkeley, Department of Electrical Engineering*, Memorandum No. ERL-M520, May 1975.

[Nagy2006] Z.K. Nagy, and R.D. Braatz (2006). Distributional uncertainty analysis using power series and polynomial chaos expansions. In *Journal of Process Control* Vol. 17, Elsevier, pp. 229-240.

[Nas2001] S.R. Nassif (2001). Modeling and analysis of manufacturing variations. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*, 2001, pp. 223–228.

[Ning1991] Z. Ning, A.J. Mouthaan, and H. Wallinga (1991). SEAS: A simulated evolution approach for analog circuit synthesis. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*, pp. 5.2-1-4.

[Nist2006] National Institute of Standards and Technology (NIST) (2006). What is process capability? In *NIST / SEMATECH e-Handbook of Statistical Methods*, sec 6.1.6. Online at: http://www.itl.nist.gov/div898/handbook/pmc/section1/pmc16.htm, last accessed July 18, 2006.

[Noc1999] J. Nocedal, S. Wright (1999). *Numerical optimization*. Springer. ISBN: 0387987932.

[Nor1994] P. Nordin (1994). A compiling genetic programming system that directly manipulates the machine code. In K.E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 14, pp. 311–331. MIT Press.

[Och1999] E.S. Ochotta, T. Mukherjee, R.A. Rutenbar, and L.R. Carley (1999). Practical synthesis of high performance analog circuits. Kluwer, Norwell, MA, USA. ISBN: 0-7923-8237-4.

[ONe2003] M. O'Neill, and C. Ryan (2003). *Grammatical evolution: evolutionary automatic programming in an arbitrary language*. Kluwer Academic Publishers, Norwell, MA, USA. ISBN: 1402074441.

[Pal2008] P. Palmers, T. McConaghy, M. Steyaert, and G.G.E. Gielen (2008). Massively multi-topology sizing of analog integrated circuits. In *Proceedings of the Design Automation and Test in Europe (DATE)*, (submitted in 2008 for DATE 2009).

[Pan2004] L. Panait, and S. Luke (2004). Alternative bloat control methods. In *GECCO 2004: Proceedings of the 6th annual conference on genetic and evolutionary computation*, ACM Press, LNCS 3103, pp. 630–641.

[Paul1992] C.R. Paul (1992). *Introduction to electromagnetic compatibility*. Wiley Series in Microwave and Optical Engineering. ISBN: 0471549274.

[Pel1989] M.J. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers (1989). Matching properties of MOS transistors for precision analog design. In *IEEE Journal of Solid-State Circuits*, 24(5), pp. 1433–1439, October 1989.

[Phe1999] R. Phelps, M. Krasnicki, R.A. Rutenbar, L.R. Carley, and J.R. Hellums (1999). ANACONDA: Robust synthesis of analog circuits via stochastic pattern search In *Proceedings of the Custom Integrated Circuits Conference (CICC)*

[Phe2000] R. Phelps, M. Krasnicki, R.A. Rutenbar, L.R. Carley, and J.R. Hellums (2000). ANACONDA: Simulation-based synthesis of analog circuits via stochastic pattern search. In *IEEE Transactions on Computer-Aided Design*, pp. 703–717, June 2000.

[Phi1998] J.R. Phillips (1998). Model reduction of time-varying linear systems using approximate multipoint krylov-subspace projectors. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 96–102.

[Pla2003] R. v.d. Plassche (2003). *CMOS integrated analog-to-digital and digital-to-analog converters, 2nd ed.* Kluwer.ISBN: 1402075006.

[Plas2002] G. van der Plas, G.G.E. Gielen, W.M.C. Sansen (2002). *A computer-aided design and synthesis environment for analog integrated circuits*. Springer, ISBN:0792376978.

[Poli1999] R. Poli, and W.B. Langdon. (1999). Sub-machine-code genetic programming. In L. Spector, W.B. Langdon, U.-M. O'Reilly, and P.J. Angeline, editors, *Advances in Genetic Programming 3*, chapter 13, pp. 301–323. MIT Press, Cambridge, MA, USA.

[Poli1999b] R. Poli, J. Page, and W.B. Langdon (1999). Smooth uniform crossover, sub-machine code GP and demes. In *GECCO 1999: Proceedings of the conference on Genetic and evolutionary computation*, vol. 2, July 1999, pp. 1162–1169.

[Poli2008] R. Poli, W.B. Langdon, N.F. McPhee. (2008). *A field guide to genetic programming*. Lulu.com, ISBN: 978-1-4092-0073-4.

[Pol2006] R. Polikar (2006). Ensemble based systems in decision making. In *IEEE Circuits and Systems Magazine*, 3^{rd} quarter 2006.

[Pow1994] J.A. Power, B. Donnellan, A. Mathewson, and W.A. Lane (1994). Relating statistical MOSFET model parameters to IC manufacturing process fluctuations enabling realistic worst-case design. In *IEEE Transactions on Semiconductor Manufacturing*, 7(3), pp. 306–318, August 1994.

[Pow2006] M.J.D. Powell (2006). The NEWUOA software for unconstrained optimization without derivatives In G. Di Pillo and M. Roma, eds., *Large Scale Nonlinear Optimization*, Springer, Netherlands, 2006, pp. 255.

[Pyt2008] Python programming language (2008). *http://www.python.org*, last accessed April 23, 2008

[Pyro2008] PYRO - python remote objects (2008). *http://pyro.sourceforge.net*, last accessed April 23, 2008

[Pyn2008] Python Numeric version 24.2 (2008). *http://pypi.python.org/pypi/Numeric/24.2*, last accessed April 23, 2008

[Rat2008] J. Rattner (2008). EDA for digital, programmable, multi-radios. Keynote address at *Design Automation Conference (DAC)*, June 10, 2008, Anaheim, CA, USA.

[Raz2000] B. Razavi (2000). *Design of Analog CMOS Integrated Circuits*. McGraw-Hill.

[Res1984] A.L. Ressler (1984). *A circuit grammar for operational amplifier design*. PhD dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA.

[Rey2008] R.G. Reynolds, M.Z. Ali, and P. Franzel (2008). Simulating the evolution of an ancient urban center. In R.L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice V*, volume 6 of *Genetic and Evolutionary Computation*, chapter 15, pp. 263–278. Springer, Ann Arbor.

[Rot2006] F. Rothlauf (2006). *Representations for genetic and evolutionary algorithms*. Springer-Verlag, 2^{nd} edition. ISBN: 3-540-25059-X.

[Rus2003] S.J. Russel, P. Norvig (2003). *Artificial Intelligence: A Modern Approach* (2nd edition) Prentice Hall, Upper Saddle River, NJ, ISBN: 0-13-790395-2.

[Rut2000] R.A. Rutenbar, and J.M. Cohn (2000). Layout tools for analog ICs and mixed-signal SOCs: A survey. In *Proceedings of the ACM International Symposium on Physical Design*, pp. 76–83.

[Rut2002] R.A. Rutenbar, G.G.E. Gielen, and B.A.A. Antao, editors (2002). *Computer aided design of analog integrated circuits and systems*. IEEE Press, Piscataway, NJ, 2002. pp. 3–30

[Rut2004] R.A. Rutenbar, T. Bonaccio, T. Meng, E. Perea, R. Pitts, Sodini, and J. Weiser (2004). Panel: Will Moore's Law rule in the land of analog? In *Proceedings of the Design Automation Conference (DAC)*, pp. 633–633.

[Rut2007] R.A. Rutenbar, G.G.E. Gielen, and J. Roychowdhury (2007). Hierarchical modeling, optimization and synthesis for system-level analog and RF designs. In *Proceedings of the IEEE*, 95(3), March 2007, pp. 640-669.

[Ryan2005] C. Ryan, M. Keijzer, and M. Cattolico. Favorable biasing of function sets using run transferable libraries. In U.-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 7, pp. 103–120.

[San1989] W. Sansen, G.G.E. Gielen, and H. Walscharts (1989). A symbolic simulator for analog circuits. *Proceedings International Solid-State Circuits Conference (ISSCC)*, pp. 204-205.

[San2006] W.M.C. Sansen (2006). *Analog design essentials*. Springer. ISBN: 0387257462.

[Sap2008] Y. Sapargaliyev, and T.G. Kalganova (2008). Unconstrained evolution of analogue computational "QR" circuit with oscillating length representation. In *Proceedings of the International Conference on Evolvable Systems (ICES)*, G.S. Hornby et al., eds, LNCS 5216, pp. 1–10.

[Sas2002] M.J. Sasena (2002). *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations.* PhD dissertation, University of Michigan.

[Sch2002] R.E. Schapire (2002). The boosting approach to machine learning: An overview. *MSRI Workshop on Nonlinear Estimation and Classification*

[Sch2001] F. Schenkel, M. Pronath, S. Zizala, R. Schwencker, H. Graeb, and K. Antreich (2001). Mismatch analysis and direct yield optimization by spec-wise linearization and feasibility-guided search. In *Proceedings of the Design Automation Conference (DAC)*, pp. 858–863.

[Sch1994] P. Schuster, W. Fontana, P.F. Stadler, and I. Hofacker (1994). From sequences to shapes and back: A case study in RNA secondary structures. In *Proceedings Royal Society (London) B*, 255, pp. 279–284.

[Sed1988] S. Seda, M. Degrauwe, and W. Fichtner (1988). A symbolic analysis tool for analog circuit design automation *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pp. 488-491.

[Sed1992] S. Seda, M. Degrauwe, and W. Fichtner (1992). Lazy-expansion symboli expression symboli expression approximation in SYNAP. *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pp. 310–317.

[Shi2002] H. Shibata, S. Mori, and N. Fujii (2002). Automated design of analog circuits using cell-based structure. In *Proceedings of the Nasa/DoD Conference on Evolvable Hardware*.

[Shyu1984] J.-B. Shyu, G.C. Temes, and F. Krummenacher (1984). Random error effects in matched MOS capacitors and current sources. In *IEEE Journal of Solid-State Circuits*, 19(6), December 1984, pp. 948–956.

[Si22008] Silicon Integration Initiative (Si2) (2008). OpenAccess on OpenEDA. http://www.si2.org/openeda.si2.org/, last accessed June 19, 2008.

[Sia2008] Semiconductor Industry Association (2008). STATS: Global billings report history (3-month moving averages) 1976 - April 2008. http://www.sia-online.org/pre_stat.cfm?ID=287, last accessed June 19, 2008.

[Sin2007] A. Singhee, and R.A. Rutenbar (2007). Beyond low-order statistical response surfaces: latent variable regression for efficient, highly nonlinear fitting. In *Proceedings of the Design Automation Conference (DAC)*.

[Som1993] R. Sommer, E. Hennig, G. Droge, and E.-H. Horneber (1993). Equation-based symbolic approximation by matrix reduction with quantitative error prediction *Alta Frequenza - Rivista di Elettronica* 5(6), November-December 1993, pp. pp. 29-37.

[Sme2003a] B. De Smedt, and G.G.E. Gielen (2003). WATSON: Design space boundary exploration and model generation for analog and RF/IC design. *IEEE Transactions on Computer-Aided Design*, 22(2), pp. 213–223.

[Sme2003b] B. De Smedt, and G.G.E. Gielen (2003). HOLMES: Capturing the yield-optimized design space boundaries of analog and RF integrated circuits. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*.

[Smi2005] G. Smits, and M. Kotanchek (2005). Pareto-front exploitation in symbolic regression. In U.-M. O'Reilly, T. Yu, R.L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 17, pp. 283–300. Springer, Ann Arbor.

[Snps2005] Synopsys, Inc. (2005). Product page: Circuit Explorer. http://www.synopsys.com, last accessed 2005.

[Snps2008a] Synopsys, Inc. (2008). Product page: HSPICE circuit simulator. http://www.synopsys.com/products/mixedsignal/hspice/hspice.html, last accessed June 19, 2008.

[Snps2008b] Synopsys, Inc. (2008). Product page: HSPICE RF circuit simulator. http://www.synopsys.com/products/mixedsignal/hspice/adv_circuit_ds.pdf, last accessed June 19, 2008.

[Snps2008c] Synopsys, Inc. (2008). Product page: WaveView Analyzer (waveform viewer). http://www.synopsys.com/products/mixedsignal/acqs1/waveview_analyzer.html, last accessed June 19, 2008.

[Snps2008d] Synopsys, Inc. (2008). Product page: Galaxy Custom Designer (analog / custom design environment). http://www.synopsys.com/products/solutions/galaxy/custom_design/customdesign.html, last accessed November 1, 2008.

[Snps2008e] Synopsys, Inc. (2008). Product page: Galaxy Custom Designer Layout Editor. http://www.synopsys.com/products/solutions/galaxy/custom_design/cusdesignLE_ds.html, last accessed November 1, 2008.

[Snps2008f] Synopsys, Inc. (2008). Product page: Galaxy Custom Designer Schematic Editor. http://www.synopsys.com/products/solutions/galaxy/custom_design/cusdesignSE_ds.html, last accessed November 1, 2008.

[Snps2008g] Synopsys, Inc. (2008). Product page: Hercules Physical Verification Suite (PVS) http://www.synopsys.com/products/hercules/, last accessed November 1, 2008.

[Snps2008h] Synopsys, Inc. (2008). Product page: Star-RCXT (parasitic extraction) http://www.synopsys.com/products/starxct/star_rcxt_ds.html, last accessed November 1, 2008.

[Soe2005] C. Soens, P. Wambacq, G. Van Der Plas, and S. Donnay (2005). Simulation methodology for analysis of substrate noise impact on analog / RF circuits including interconnect resistance. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*.

[Sou2002] T. Soule, and R.B. Heckendom (2002). An analysis of the causes of code growth in genetic programming. In *Genetic Programming and Evolvable Machines*, 3(3), September 2002, pp. 283–309.

[Spe1994] G. Spencer (1994). Automatic generation of programs for crawling and walking. In *Advances in Genetic Programming*, MIT Press, Chapter 15, pp. 335–353.

[Spe1995] L. Spector (1995). Evolving control structures with automatically defined macros (1995). In E. V. Siegel and J. R. Koza editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pp. 99-105, MIT, Cambridge, MA, USA.

[Spe2004] L. Spector (2004). *Automatic quantum computer programming: A genetic programming approach*, volume 7 of *Genetic Programming*. Kluwer Academic Publishers, Boston/Dordrecht/New York/London. ISBN: 1-4020-7894-3.

[Spe2005] L. Spector, J. Klein, and M. Keijzer (2005). The Push3 execution stack and the evolution of control. In H.-G. Beyer, et al., editors, *GECCO 2005: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 2, pp. 1689–1696, Washington DC, USA. ACM Press.

[Sri2002] T. Sripramong, and C. Toumazou (2002). The invention of CMOS amplifiers using genetic programming and current-flow analysis. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 21(11), pp. 1237-1252.

[Ste2003] G. Stehr, M. Pronath, F. Schenkel, H.E. Graeb, and K. Antreich (2003). Initial sizing of analog integrated circuits by centering within topology-given implicit specification. In *Proceedings of International Conference on Computer-Aided Design*, pp. 241–246.

[Ste2007] G. Stehr, H. Graeb, and K. Antreich (2007). Analog performance space exploration by normal-boundary intersection and fourier-motzkin elimination. *IEEE Trans. Computer-Aided Design*, 26(10), Oct. 2007, pp. 1733 - 1748.

[Sto1992] J. Stoffels, and C. van Reevwijk (1992). AMPDES: A program for the synthesis of high-performance amplifiers In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, pp. 474–479.

[Suy2002] J.A.K. Suykens, J.Vandewalle (2002). Least squares support vector machines. World Scientific Publishing Company, Singapore, 2002.

[Swi1991] K. Swings, S. Donnay, and W.M.C. Sansen (1991). HECTOR: A hierarchical topology-construction program for analog circuits based on a declarative approach to circuit modeling. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*.

[Tan1993] T. Tanaka. (1993). Parsing electronic circuits in a logic grammar. *IEEE Transactions Knowledge and Data Engineering*, 5(2), pp. 225–239.

[Tang2006] H. Tang and A. Doboli (2006). High-level synthesis of delta-sigma modulator topologies optimized for complexity, sensitivity, and power consumption. *IEEE Transactions on Computer-Aided Design*, 25(3), pp. 597–607.

[Tel1997] A. Teller, and D. Andre (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pp. 321–328, Stanford University, CA, USA. Morgan Kaufmann.

[Tib1997] R. Tibshirani (1997). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58(1), pp. 267–288.

[Tic1999] A.B. Tickle, F. Maire, G. Bologna, R. Andrews, and J. Diederich (1998). Lessons from past, current issues, and future research directions in extracting the knowledge embedded in artificial neural networks. In *Neural Hybrid Systems*, pp. 226–239.

[Tiw2006] S.K. Tiwary, P.K. Tiwary, R.A. Rutenbar (2006) Generation of yield-aware Pareto surfaces for hierarchical circuit design space exploration. In *Proceedings of the Design Automation Conference (DAC)*, pp. 31–36

[Top2001] A. Topchy, and W.F. Punch (2001). Faster genetic programming based on local gradient search of numeric leaf values. In L. Spector et al, eds., *GECCO 2001: Proceedings of the 2nd annual conference on Genetic and evolutionary computation*, Morgan Kaufmann Publishers, 155–162.

[Tou1990] C. Toumazou, C.A. Makris, and C.M. Berrah (1990). ISAID: A methodology for automated analog IC design. In *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, volume 1, pp. 531–555.

[Tur1986] F.M. E1-Turky, and R.A. Nordin (1986). BLADES: An expert system for analog circuit design. In *Proceedings of the International Conference on Circuits and Systems (ISCAS)*, pp. 552–555.

[Van2001] P. Vancorenland, G. Van der Plas, M. Steyaert, G.G.E. Gielen, and W.M.C. Sansen (2001). A layout-aware synthesis methodology for RF circuits. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, Nov. 2001, p. 358

[Vas2000] V.K. Vassilev and J.F. Miller (2000). The advantages of landscape neutrality in digital circuit evolution. In *Proceedings of the Third International Conference on Evolvable Systems (ICES)*, pp. 252–263. Springer-Verlag.

[Vog2003] M. Vogels, and G.G.E. Gielen (2003). Architectural selection of A/D converters. In *Proceedings of the Design Automation Conference (DAC)*, pp. 974–977.

[Wam1995] P. Wambacq, F.V. Fernandez, G.G.E. Gielen, W.M.C. Sansen, and A. Rodriguez-Vazquez (1995). Efficient symbolic computation of approximate small-signal characteristics. *IEEE Journal of Solid-State Circuits*, 30(3), March 1995, pp. 327–330.

[Whi1995] P.A. Whigham (1995). Grammatically-based genetic programming. In J.P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pp. 33–41, Tahoe City, California, USA.

[Wie1989] G. Wierzba et al. (1989). SSPICE - A symbolic SPICE software for linear active circuits. *Proceedings Midwest Symposium on Circuits and Systems.*

[Wil1991] J. Williams, editor (1991). *Analog design: Art, science, and personalities*. Newnes Press.

[Wil1927] E.B. Wilson (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association 22*, pp. 209-212. See also http://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval.

[Wol2003] G. Wolfe, and R.Vemuri (2003). Extraction and use of neural network models in automated synthesis of operational amplifiers. In *IEEE Transactions on Computer-Aided Design*, 22(2), Feb. 2003, pp. 198–212.

[Wol2004] G. Wolfe, and R.Vemuri (2004). Adaptive sampling and modeling of analog circuit performance parameters with pseudo-cubic splines. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*.

[Xu2005] Y. Xu, K.-L. Hsiung, X. Li, I. Nausieda, S. Boyd, and L. Pileggi (2005). OPERA: Optimization with ellipsoidal uncertainty for robust analog IC design. In *Proceedings of the Design Automation Conference (DAC)*, PP. 632–637

[Yang2005] J. Yang, S.X.-D. Tan, Z. Qi, and M. Gawecki (2005). Hierarchical symbolic piecewise-linear circuit analysis. In *Proceedings of the Workshop on Behavioral Modeling and Simulation (BMAS)*.

[Yao1999] X. Yao, Y. Liu, and G. Lin (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2), July 1999, pp. 82–102.

[Yu2007a] T. Yu, D. Wilkinson, and A. Castellini (2007). Applying genetic programming to reservoir history matching problem. In R.L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12. Springer, Ann Arbor.

[Yu2007b] G. Yu, P. Li (2007b). Yield-aware analog integrated circuit optimization using geostatistics motivated performance modeling. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 464–469.

[Yur1994] D. Yuret (1994). From genetic algorithms to efficient optimization. Master's Dissertation, *MIT AI Laboratory*.

[Zeb2000] R. Zebulum, M. Vellasco, and M. Pacheco (2000). Variable length representation in evolutionary electronics. *Evolutionary Computation*, 8(1), pp. 93–120.

[Zeb2002] R. Zebulum, M. Pacheco, and M. Vellasco (2002). *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. CRC Press.

[Zha2004] G. Zhang, E.A. Dengi, R.A. Rohrer, R.A. Rutenbar, and L.R. Carley (2004). A synthesis flow toward fast parasitic closure for radio-frequency integrated circuits. In *Proceedings of the Design Automation Conference (DAC)*, pp. 155–158.

Biography

Trent McConaghy was Research Assistant in the MICAS lab at Katholieke Universiteit Leuven in Belgium from 2004-2008. He co-founded Solido Design Automation Inc. and is currently its Chief Scientific Officer. Before that, 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 machine learning research for the Canadian Department of National Defense. He has a Bachelor's in Engineering (with great distinction) and a Bachelor's of Computer Science (with great distinction) from the University of Saskatchewan, Saskatoon, Canada. He received its 2001 Outstanding Young Alumni Award for significant accomplishments since graduation. He has more than 40 peer-reviewed technical papers, patents granted / pending, and magazine articles. He has been an invited speaker at many labs, universities, and conferences ranging from JPL to MIT. 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, GECCO, etc. His research interests include statistical machine learning and intelligent systems, with transistor-level CAD applications such as statistically-aware design, analog topology design, automated sizing, knowledge extraction, and symbolic modeling. Funding for the reported research results is acknowledged from IWT/Medea+ Uppermost, Solido Design Automation and FWO Flanders.