VALIDATION AND OPTIMIZATION OF ANALOG CIRCUITS USING RANDOMIZED SEARCH ALGORITHMS

BY

SEYED NEMATOLLAH AHMADYAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

ABSTRACT

Analog circuits represent a large percentage of the chips used in mobile computing, communication devices, electric vehicles, and portable medical equipment today. Rapid scaling and shrinking chip geometrics introduce new challenging problems in verification, validation, and optimization of analog circuits. These problems include test generation and compression, runtime monitoring and analyzing the worst-case behaviors. State of the art techniques in Monte Carlo are unable to address these problems effectively. Consequently, designing an efficient and scalable CAD algorithm to address such problems is highly desirable.

In this thesis, we introduce Duplex, a methodology for search and optimization. Duplex supports optimizing nonconvex nonlinear functions and functionals. We use duplex to solve problems in analog validation and machine learning. Duplex uses random tree data structures. Duplex is based on partitioning and separating the problem space into multiple smaller spaces such as input, state and the function space. Duplex simultaneously controls, biases and monitors the growth of the random trees in the partitioned spaces. We have used the duplex framework to solve practical problems in analog and mixed signal validation like directed input stimuli generation, compressing analog stress tests, worst-case eye diagram analysis, performance optimization, machine learning, and monitoring runtime behaviors of analog circuits.

We used Duplex for validation and optimization of analog circuits. Duplex automatically generates input stimuli that expose bugs and improves coverage. Duplex automatically finds input corners that result in worst-case eye diagrams. Duplex simultaneously explores the parameter and performance spaces of analog circuits to optimize the circuit for best performance. We monitored the random trees and circuit execution against the specification properties described in formal languages. We formulated many challenging problems in the analog circuits, such as test compression and eye diagram analysis, as functional optimization problems. We use Duplex to solve these functional optimization problems.

We propose the Duplex algorithm as an optimization algorithm to posit the framework to other domains. Duplex can address nonlinear and functional optimization problems in continuous and discrete spaces such as design-space exploration and supervised and unsupervised machine learning.

The advantages of the duplex framework are efficiency, scalability and versatility. We consistently show orders of magnitude speedup improvements over the state of the art while objectively improving the quality of results. For generating input stimuli, duplex is the first technique that simultaneously does directed input stimulus generation and increases test coverage. We show over two orders of magnitude speedup over Monte Carlo simulations. For runtime monitoring, we check a large scalable circuit against a very expressive set of formal properties that were not possible to monitor before. For generating worst-case eye diagram, we show at least 20× speedup and better quality of results in comparison to the state of the art. Duplex is the first work to provide transient test compression for analog circuits. We compress stress tests up to 96%. We optimize analog circuits using Duplex and we show speedup and improved results with respect to the state of the art. We use Duplex to train supervised and unsupervised models and show improved accuracy in all cases.

ACKNOWLEDGMENTS

I would like to thank my adviser, Prof. Shobha Vasudevan, for her intellect, experience, and advice. Of all people, I owe her the most for the completion of this thesis.

I would like to thank my dissertation committee members, Prof. Rob Rutenbar, Prof. Sayan Mitra, Prof. Martin Wong, Prof. Xin Li, and Prof. Kesh Kesavadas, for their time, invaluable feedback and for agreeing to be part of my Ph.D. exam process.

I would like to also thank my co-authors and collaborators, Dr. Jayanand Asuk Kumar, Dr. Suriya Natarajan, Dr. Chenjie Gu, and Dr. Eli Chiprout, for their work, experience, and feedback.

I would like to thank my parents, who supported me all these years, my sisters, and friends Fardin Abdi, Hadi Hashemi, Mohammad Babaizadeh, Faraz Faghri, Mohammad Nourbakhsh, Jalal and Rasoul Etesami, and many others. Finally I would like to thank Samira Sheikhi, for her kindness, intellect, support and intriguing conversations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Design automation for analog circuits

Analog circuits represent a large percentage of the chips used in mobile computing, communication devices, electric vehicles, and portable medical equipment today [1]. This trend is projected to grow in the future [1], as analog IPs become ubiquitous in SoC designs. The analog IP market is expected to expand with the increasing demand for portable and wearable devices, smartphones, and low-power electronics [1]. Analog chips are increasingly being used in medical devices and electric vehicles [1], where safety and reliability are critical concerns.

Verification and validation of the behavior of these complex and safety critical circuits is a daunting challenge. The scale and complexity of the circuits have increased significantly beyond those of the hand-crafted, isolated analog circuits of the past. However, given the scale and complexity of the analog components used in modern devices, traditional validation methodologies based on Monte Carlo simulations are inefficient, expensive, error-prone, and frequently misleading in predicting unknown behaviors, worst case corners and stressing weak components of the circuit. It is critical to automate the validation tasks for analog circuits to meet demands [2].

1.1.1 Fast and reliable circuits

Analog circuits are the drivers of the computer and electronics industry. Circuits are manufactured at a very large scale at a very low margin. In this competitive market, analog designers always strive to design reliable chips with maximum performance. It is crucial to vigorously test against design errors to ensure reliability, correctness, and safety.

The analog design process (Figure 1.1) is an iterative process that goes

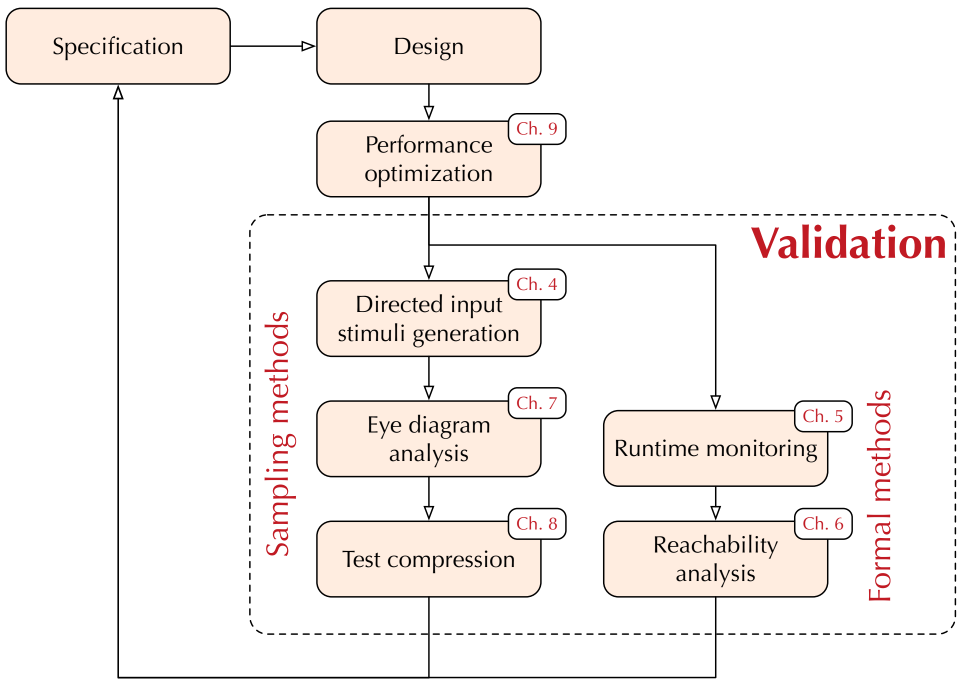


Figure 1.1: The overview of the problems solved in analog design flow.

through the cycle of **specification** → **design** → **optimization** → **validation**. Currently every task in analog validation is manual and ad-hoc. As a result, the analog design process is error-prune, expensive and very time-consuming. *Our mission is to provide automation in analog design flow to improve the quality and reliability of analog circuits.* In particular, we focus on optimization and validation of analog circuits.

Validation: We define the validation problem in analog circuits as ensuring the circuit is behaving correctly according to its specification. An analog circuit is a nonlinear system with input and output signals. For validating such regime, two categories of problems arise. Firstly, how to excite the input; how to generate meaningful input stimuli to test the circuit. We wish to generate tests that trigger failures, bugs, or worst-case responses in the circuits. Secondly, how to monitor the output response, check for violations, and ensure the safety of the circuit.

The goal of exciting the inputs to identify input stimuli and corners that can cause failure in the circuit. Failures can be viewed as having a failed state (such as signal cut-off, unstable output, failure to lock, etc.) or noisy output (jitter, weak signal, low noise margin, etc.). Then we can analyze the generated input stimuli to discover bugs in the design and improve the circuit.

Table 1.1: Keystone problems in analog validation and optimization

Type	Chapter	Problem
Validation	4	Directed input stimuli generation
Validation	5	Runtime monitoring and property checking
Validation	7	Worst-case eye diagram and signal-integrity analysis
Optimization	8	Test compression
Optimization	9	Performance optimization of analog circuits
Validation	6	Reachability analysis and safety verification

Finally, During test and after finding the test stimuli, we can optimize it such that the test would be executed faster on the circuit in order to save time (Chapter 8).

For ensuring the safety and correctness of the circuit, we can take either a *universal* or *existential* approach. In the *universal* approach, we have to prove the circuit is correct by verifying every possible execution meets the specification. One universal tool for verifying safety properties is *reachability analysis* (Chapter 6). On the other hand, the *existential* method looks for circuit traces that violate the specification and cause failures. In order to implement the existential technique we need techniques that actively search for violating stimuli (Chapter 4 and 7) and monitor the executions (Chapter 5). The analog circuits exhibit complex nonlinear dynamics and have a large scale with hundreds of dimensions. Any method for validating analog circuits has to be scalable and faithful to the nonlinearities of the circuit.

Optimization: The circuit's performance metrics include power, bandwidth, gain, etc. The goal of the optimization step is to improve the performance without changing the functionality of the circuit. We optimize the circuit's parameters, such as transistor width and length, to improve the performance (Chapter 9).

In this thesis, we address all of the above issues in analog validation and optimization by solving six keystone problems as shown in Table 1.1. To address these significant problems in analog EDA, we need a new scalable, efficient, versatile, mathematically sound and unified methodology to replace the old techniques.

1.2 Duplex methodology

We introduce Duplex, an optimization methodology to solve non-convex and functional optimization problems. We use Duplex to solve validation and optimization problems in analog. Our methodology is based on formulating analog problems as optimization problems, then using Duplex to optimize the objective function and determine the optimum solution. The optimum solutions depend on the problem and range from the input stimuli that will cause the circuit to fail, to the worst-case eye diagram, to the optimum parameters for optimizing the circuit.

Duplex can solve three types of optimization problems: i) state-based search, ii) non-convex optimization, and iii) functional optimization. Automated input stimuli generation is formulated as a state-based search problem. Performance optimization is formulated as a non-convex optimization problem. Finally, eye diagram analysis and test compression problems are formulated as functional optimization problems. The overview of the Duplex methodology is shown in Figure 1.2.

The Duplex idea is centered around growing random tree structure in the space model of the problem. Depending on the problem type, Duplex algorithm partitions the problem space into the input space, the output space, and the function space. We simultaneously search and grow multiple mirrored random trees in these spaces to meet the boundary conditions and optimize the objectives.

The principle of Duplex is different from other known optimization algorithms like simulated annealing [3], gradient descent [4], etc., used in optimization. Duplex simultaneously analyzes and partitions the problem space into smaller spaces such as input (feature), output (objective) and function spaces. Global decisions are made in the objective space and actions are taken locally in the input space. In every iteration, Duplex identifies the best strategy to get closer to the goal region (the optimum solution) and takes the appropriate action within the input space. The algorithm then passes the control back to the objective space, where the next such strategic decision is made. To the best of our knowledge, this is the first algorithm to simultaneously keep track of an objective/goal and use it to guide local steps.

Duplex maintains the full history of all the visited states in the space.

Duplex provides valuable feedback to the user by looking at and analyzing the grown tree in the state space. Duplex uses the random tree to determine the Pareto frontier and the optimum solutions. Furthermore, Duplex avoids getting stuck in the local minima. We utilize these properties of Duplex to generate a Pareto optimal set of the circuit's parameters in circuit optimization problem and analyze the distribution of worst-case samples in eye diagram analysis.

1.2.1 Advantages of the Duplex methodology

Duplex algorithm provides the following advantages over traditional optimization methods such as gradient descent and Newton optimization:

- **Convergence guarantees toward global optimum:** Optimization algorithms traverse the state space locally. They walk in the state space linearly along the direction of the gradient. Therefore, they have no notion of the solution, or where it might lie. They are prone to getting stuck in local minima and saddle points where the gradient is zero. However, Duplex performs an open ended search only in the objective space. In the objective space, it uses the basic random tree search to find the globally optimal design or the *goal region*. In the state space, it decides which parameter needs to change to get closer to the goal region. This decision is made using a noisy gradient descent algorithm in combination with reinforcement learning that evaluates the history of previous changes to the parameters in the parameter tree based on a reward function. There is no open ended search in the parameter modification phase (local step) of the algorithm. Duplex grows multiple different branches toward the goal state. If one of those branches gets stuck in local minima, the algorithm can branch from another state and make forward progress toward the goal. Due to the probabilistic completeness property of random trees, Duplex does not get stuck in local minima. This is in contrast to random walk based methods like simulated annealing or gradient descent. The guidance in every step from the global search towards the local step decision helps in converging quickly to the optimal goal region.

- **Performance Efficiency:** A search based optimization algorithm like Duplex has a benefit over classical optimization algorithms in being able to keep track of the global picture (goal states) and react with local actions. Random trees are shown to consistently outperform random walk based search methods such as Monte Carlo simulations for search applications. The improvement in efficiency is partly due to the tree data structure maintained by the random tree algorithm during the simulation. While growing, it samples a new state in the goal region (desired solution set), and then determines which state is closest (in L_2-norm sense) to that sampled goal state among all of the previously visited states in the tree. It simulates a path between the closest state and the newly sampled state and adds the new state to the tree. This is in contrast to the memory-less sampling of points in the Monte Carlo based methods. The branching to any previously visited state makes convergence to the solution set quicker than memory-less methods due to the versatility in paths traversed.

- **Scalability:** A tradeoff in search based algorithms comes from the step where they search for their nearest neighbors. Searching for the nearest neighbors in the state space (which scales) can be very expensive. Duplex addresses this issue by separating the objective functions from the state space and limiting the search for nearest neighbors only to the objective functions. The space of objective functions is much smaller than the state space, helping the scale and efficiency of Duplex significantly.

1.3 Analog problems solved with Duplex

We achieve our mission by using Duplex methodology for verification, validation and optimization of analog circuits. We use the Duplex methodology to automate keystone problems in analog optimization and validation. Figure 1.2 shows the flowchart of analog design flow using the Duplex methodology. We solved select challenging problems in analog design flow.

Specifically, we use Duplex for state search, non-convex and functional optimization in order to address problems in directed input stimuli generation (Chapter 4), worst-case eye diagram analysis (Chapter 7) and test compres-

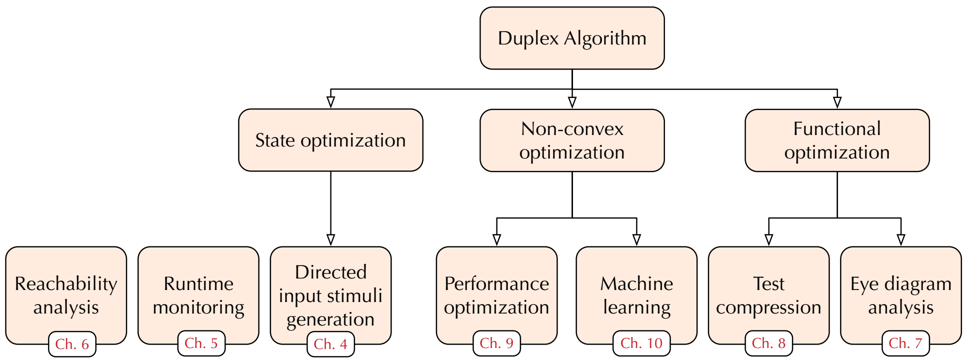


Figure 1.2: Duplex solves different types of optimization problems.

sion (Chapter 8). We use Duplex for design-space exploration of analog circuits to determine the optimum configuration and optimize the circuit's performance (Chapter 9). We monitor the growth of random trees to falsify logical properties (Chapter 5) or verify safety properties by reachability analysis (Chapter 6). Beyond analog, we use Duplex in machine learning to solve problems in classification and clustering where we use Duplex to optimize the loss/cost/error/energy function (Chapter 10).

1.3.1 Validating analog circuits by generating directed input stimuli

Directed input stimuli generation

Validating nonlinear analog circuits is a major challenge and an ongoing topic of intensive research. Simulation-based verification is performed using several test cases for the circuit. Each test case is a sequence of values that is applied to the circuit inputs. For each test case, the circuit is simulated by applying the corresponding sequence of values to the inputs. The behavior induced by these sequences is then analyzed. If an erroneous or illegal set of circuit states (*i.e.*, a "bad" region) is known, it is desirable to check whether there is a legal sequence of input values that takes the circuit from an initial state to the bad region. Simulation-based verification is non-exhaustive and therefore checks only a subset of all possible behaviors. The quality of simulation-based verification is determined by the choice of test cases.

We use Duplex for directed input stimuli generation for analog circuits. We use a learning-based approach to guide the Duplex algorithm towards a goal region. Duplex can search the state space for multiple objectives such as reaching a goal region and improving the coverage of the state space.

We present the first technique for directed input stimuli generation for analog circuits. Duplex can generate tests with multiple objectives such as reaching a goal region and improving the coverage of the state space. Duplex can find input stimuli that trigger failures two orders of magnitude faster than Monte Carlo.

Worst-case eye diagram generation In many RF applications and driver circuits the signal integrity is crucial for the performance of the circuit. Signal integrity is the major bottleneck to the system's performance in a high speed CMOS circuit. Eye diagrams [5] are the main diagnostic technique for evaluating the signal integrity. Important signal properties such as noise margin and jitter can be measured from the eye diagram using Monte Carlo transient simulations [6], statistical methods [7],[8], and analytical convolution-based techniques [9],[10].

The worst-case eye diagram is a geometric product of the distortion in the transient response of the circuit. Using Duplex, we optimize the distortion in circuit's response my controlling the circuit's inputs and finding the worst-case input.

We verify the signal integrity of analog circuits by analyzing the eye diagrams. We generate worst-case eye diagrams using Duplex. Duplex is $20\times$ *faster than Monte Carlo based simulation and provides reasoning why the circuits is under-performing as a feedback for the user.*

Optimizing input stimuli for compressing analog tests:

With the movement towards system-on-chip (SoC) ICs, the number and diversity of mixed-signal circuits on a die has increased significantly in the form of different high-speed IOs, sensors, power, and clocking circuitry. Among these, analog components are tested using specification-based functional tests with some design-for-test (DFT) features built in.

The steps in manufacturing test are broadly categorized into wafer/sort testing, packaged part class test (using functional and structural tests) which includes stress testing/burn-in, and system testing [11]. These steps need to be performed on every part that is shipped, resulting in a high volume of parts to be tested. To achieve fast product ramp to customers, the test time per part should be small, to the order of a few seconds. Although short test times can be achieved by increasing the test equipment, this is not a preferred choice due to the sharp increase in capital cost that accompanies it. Instead, it is cost effective to abbreviate each step in the testing of parts [11] so that the test time is reduced for each part. While the steps themselves cannot be eliminated due to the coverage they provide, reduction of time in each step is the best resort. Since every step usually provides some incremental coverage, reduction of time in each step is usually resorted to, rather than eliminating a step itself.

Reducing the cost of production test has been a topic of intense research in analog testing [11]. There are three approaches to reduce test time: i) optimal ordering of the tests, where the most failed tests are strategically placed first in order to reduce the total test time [12, 13], ii) selecting the subset of the tests to achieve the same coverage [14, 15], iii) automated development of better and more efficient tests that provide more coverage [16]-[17] and, iv) reducing the communication time by compressing tests on-chip [18, 19]. To the best of our knowledge, no previous work in analog domain has addressed the problem of reducing *each individual test's time.*

We formulate the test compression problem as an optimization problem. We use Duplex for optimizing the test's execution time in order to compress the test. *We use Duplex algorithm for compressing stress tests for nonlinear analog circuits. We compress tests for an opamp, VCO and a charge-pump PLL circuit. We show that we can consistently achieve on average* 93% *reduction in test length for multiple functional and burn-in stress tests for the op-amp.*

1.3.2 Ensuring correctness and safety of analog circuits

Formal verification of analog circuits is a lofty, but highly desirable goal. Some strides have been taken in analog verification research. However, since many of these techniques linearize and discretize analog circuit behavior, their practical applicability remains limited. A major challenge in formal analog verification is proving the safety properties of the system. Safety is an indication that the system's operation would always remain inside the safe regions within the state space.

Runtime monitoring of Duplex execution In industry, the traditional method to verify analog designs was manual, by the designer of the circuit. Correspondingly, their verification graduated to Monte Carlo [20] simulations. Formal methods have as yet not penetrated the practical analog designer's environment. Formal methods typically make linearization or discretization assumptions to tackle the issue of scale. Analog designers find these assumptions limiting, as compared to the simulation semantics that closely model the circuit's physical behavior.

A way to bridge the gap between the popular circuit simulation techniques and formal analysis is through runtime monitoring of formally specified properties. Such a *dynamic verification* strategy handles the nonlinear, continuous behavior, while introducing formal reasoning tools. Although it does not provide guarantees of correctness, it can falsify (disprove) properties along traces. The simulation traces along which a property fails can assist the debugging process in these circuits. We use a runtime monitoring algorithm to check the tree data structure in the Duplex algorithm. Our algorithm can detect whenever the specification property is violated in the circuit's response.

We propose a runtime monitoring algorithm to incrementally monitor the execution of analog circuits using Duplex algorithm. We propose an analog specification language to describe the specification properties. We use our algorithm to monitor complex behaviors of tunnel diode circuit and a PLL circuit.

Reachability analysis of analog circuits

Reachability analysis is a solution to the safety verification problem. Reachability analysis focuses on computing the *reachable set* of the system. The reachable set is the union of all possible trajectories generated by the system from every initial state for all input signals. To prove safety, we must show that the reachable set of the system does not intersect with any unsafe set. Generally, computing the reachable set of the nonlinear analog circuit is computationally undecidable [21]. Over the last decade, many researchers have been investigating the reachability problem [22]-[23]. A common problem in previous works toward reachability analysis is memory explosion due to the inefficiency of the data structure involved in modeling the state space [24]. Importantly, these methods do not directly handle nonlinear systems, but use linearization or interval arithmetic to model nonlinearities. Both these modeling techniques result in introduction of large and often unrealistic approximation errors.

Although computation of the exact reachable set is undecidable [21], it is possible to prove the safety of a system by computing an over-approximation of the reachable set [25]. Therefore, in a safe system, there is a feasible trajectory from the initial set of states to an erroneous or undesirable set of states (specified by the user). If the over-approximated reachable set is safe, we can conclude that the exact reachable set is safe as well. However, if the over-approximated reachable set intersects with the unsafe regions, we cannot determine the safety of the system. Over-approximation introduces its own errors to the analysis. Hence, minimizing the approximation error while maintaining computational efficiency is a challenge.

We propose a technique for computing a reachable set of nonlinear systems in near real-time. We compute the reachable set of the Van der Pol oscillator by iteratively removing the unreachable regions from the state space.

1.3.3 Optimizing the performance of analog circuits

In the traditional analog/RF IC design flow, designers would manually calculate optimal assignments to a circuit's parameters to ensure that the design meets the performance specification requirements [3, 26, 27]. In modern designs, analog and mixed signal ICs are ubiquitous due to their desirable flexibility in power, performance, etc. This coupled with shrinking transistor sizes, circuit complexity and new challenges in fabrication processes has made manual calculations infeasible [26, 28, 29].

Recent pioneering research has developed automatic optimization algorithms for analog design [28]-[30]. Despite this, some challenges still remain. Firstly, analog/RF circuits tend to have a complex state space with local minima and saddle points. State-of-the-art optimization algorithms [31] can get stuck in local minima, resulting in a non-optimal design. Secondly, quantitatively explaining the decisions made by the optimization algorithm is important for designer interpretability during design optimization. Current optimization algorithms provide no such feedback to the user.

We use Duplex for optimizing the analog circuits. We demonstrate that Duplex has 81% (up to $5\times$) more speedup as compared to state-of-the-art results and finds the global optimum for a design whose previously published result was a local optimum. We show our algorithm's scalability by optimizing a system-level post-layout charged-pump PLL circuit.

1.3.4 Supervised and unsupervised learning algorithm

Machine learning has become an integral part of modern data science. Engineers tend to increase the complexity of models to address challenging problems and larger data size. Training complex learners is very challenging because their loss function (which has to minimize) is nonconvex and has many local minima and saddle points. The Duplex algorithm can optimize the loss function and train the learner. As a proof-of-concept, we used Duplex to train two most common models in machine learning: classification using logistic regression, and clustering using k-mean clustering.

Our logistic regression model achieves an accuracy of 91% whereas the same model, trained with gradient descent algorithm, achieves 89% accuracy. Our duplex-based clustering algorithm can cluster our synthetic dataset similarly to k-mean clustering algorithm and achieve the same accuracy without getting stuck in local minima. In comparison to the gradient descent based optimizers, training with Duplex takes a longer time, but achieves a better accuracy.

We demonstrate that Duplex can optimize nonconvex loss/energy functions. We use Duplex for training supervised and unsupervised learners to solve classification and clustering problems. We show that learning with Duplex results in better accuracy than gradient descent.

1.4 Thesis contributions

In this thesis, our contributions are two-fold:

1. **Algorithmic contribution— Duplex optimization algorithm:** We introduce Duplex, a novel general optimization algorithm. The Duplex algorithm is a generalization of the Rapidly-exploring Random Tree (RRT) algorithm used in robotic motion planning. We introduce direction and space-separation to the Duplex algorithm. We utilize Duplex for solving directed search problems, non-convex optimization, and functional optimization. We present the detailed analysis of Duplex methodology in Section 1.2.

2. **Domain contributions— Problems solved in analog:** We apply Duplex to solve practical problems in analog validation and machine

learning. We formulate problems in analog validation as an optimization problem. Then we use Duplex to solve the problem and compute the optimum solution. Furthermore, we use Duplex in machine learning to solve problems in supervised and unsupervised learning. Duplex can learn from the data samples, train the models and infer relations between data by performing regression, classification, and clustering.

The Duplex algorithm is different from other walk-based optimization methods such as gradient descent or hill-climbing optimization. Duplex grows random trees and maintains the history. Duplex divides the state space of the problem into input, output and function spaces (*principle of separation*). The Duplex algorithm makes decisions in the function space depending on how close it is to the optimum solution. The Duplex algorithm avoids getting stuck in local minima and converges toward the optimum solution. Therefore, Duplex can be used to solve non-convex and functional optimization problems where the well-known optimization methods, such as gradient descent, are ineffective and will not produce optimum solutions.

Our second contribution is in the application domain. Traditionally, many analog validation problems are solved using Monte Carlo simulation. Monte Carlo simulation is a random walk and is not directed. Hence, the validation algorithms are very inefficient and take a long time. In this thesis, we formulate analog validation problems as optimization problems. We automated 6 keystone problems in analog validation and optimization (Table 1.1). The optimization objectives include finding the failure regions, maximizing distortions in the signal, minimizing functionals, and optimizing the design. We use the Duplex algorithm to optimize the objectives and solve the keystone problem.

To the best of our knowledge, we propose the first methodology for directed input stimuli generation with coverage and test compression algorithm. For optimizing analog circuits and eye diagram analysis, we improve the performance of the state of the art by factor of $5\times$ and $20\times$, respectively, and also provide global optimum solution and valuable feedbacks to the user.

1.5 Thesis organization

The remainder of this thesis is organized as follows. In Chapter 2, we cover the background materials and previous works that are closely related to the contributions of this thesis.

In Chapter 3, we describe the Duplex algorithm. We provide the background on Rapidly-exploring Random Trees (RRT). We describe our contributions over the RRT algorithm. We explain why Duplex algorithm is more efficient and applicable toward optimization problems. We define three types of optimization problems and describe how we solve them using the Duplex algorithm.

In Chapter 4, we apply Duplex for directed input stimuli generation for nonlinear analog circuits. Duplex utilizes random trees to explore the state space of analog circuits. We adapt duplex to include multiple objectives such as goal-oriented stimuli generation and increasing the test coverage. Duplex will automatically infer the goal regions and generate input stimuli directed toward the goal region while increasing the test coverage. We demonstrate that duplex is capable of generating significant, hard-to-find input stimuli and provides over two orders of magnitude speedup over Monte Carlo methods. We illustrate our technique by generating tests for an operational amplifier, voltage controlled oscillator (VCO), and ring modulator circuits.

In Chapter 5, we present a runtime monitoring algorithm for Duplex to verify design properties of nonlinear analog circuits. We use time-augmented random trees to simulate the analog circuits. The proposed runtime verification methodology consists of i) incremental construction of the time-augmented trees to explore the state-time space and ii) use of an incremental online monitoring algorithm to check whether or not the incremented random tree satisfies or violates specification properties at each iteration. In comparison to the Monte Carlo simulations, for providing the same state-space coverage, we utilize a logarithmic order of memory and time. We use a tunnel diode and a PLL circuit as case studies.

In Chapter 7, we present an efficient technique for analyzing eye diagrams of high-speed CMOS circuits in the presence of non-idealities like noise and jitter. Our method involves geometric manipulations of the eye diagram topology to find the area within the eye contours. We introduce random tree based simulations as an approach to computing the desired area. We use a high-speed CMOS inverter as a case study for generating worst-case eye diagram. We typically show 20× speedup in generating the eye diagram as compared to the state-of-the-art Monte Carlo simulation based eye diagram analysis. For the same number of samples, Monte Carlo produces an eye diagram that is 8.51% smaller than the ideal eye diagram. We generate an eye diagram that is 53.52% smaller than the ideal eye, showing a 47% improvement in quality.

In Chapter 8, we utilize Duplex for test compression. We introduce a methodology for automated test compression during electrical stress testing of analog and mixed signal circuits. This methodology optimally extracts only portions of a functional test that electrically stress the nets and devices of an analog circuit. We model test compression as a problem of optimizing functionals of the transient response. We present a random tree based approach to find optimal solutions for these computationally hard integrals. We demonstrate with an op-amp, VCO and CMOS inverter that the method consistently reduces the length of each test by 93%. We demonstrate our technique by compressing tests for VCO circuit, an opamp circuit and a CMOS inverter circuit in presence of process variations. We also provide a parallel version of the Duplex algorithm.

In Chapter 9, we use Duplex for optimizing the performance of analog circuits. Duplex determines the optimal design, the Pareto set and the sensitivity of circuit's performance metrics to its parameters. We optimize the performance of an opamp circuit and a charge-pump PLL circuit as case studies.

In Chapter 10 as a proof-of-concept, we demonstrate that the Duplex algorithm can be used for nonconvex optimization and training machine learning models in both supervised and unsupervised learning applications. We use Duplex to train a logistic regression model for solving binary classification problems. We achieve a very high-degree of accuracy for the given model. Secondly, we use Duplex for clustering unlabeled data. The Duplex algorithm can provide clustering without getting stuck in local minima.

In Chapter 6, we propose a methodology for reachability analysis of nonlinear analog circuits to verify safety properties. Our iterative reachable set reduction algorithm initially considers the entire state space as reachable. Our algorithm iteratively determines which regions in the state space are unreachable and removes those unreachable regions from the over-approximated reachable set. We use the State Partitioning Tree (SPT) algorithm to recursively partition the reachable set into convex polytopes. We determine the reachability of adjacent neighbor polytopes by analyzing the direction of state space trajectories at the common faces between two adjacent polytopes. We model the direction of the trajectories as a reachability decision function that we solve using a sound root counting method. We are faithful to the nonlinearities of the system. We demonstrate the memory efficiency of our algorithm through computation of the reachable set of Van der Pol oscillation circuit.

Finally Chapter 11 presents a summary of the work and concludes this thesis.

CHAPTER 2

PRELIMINARIES AND RELATIONSHIP TO EXISTING WORK

In this chapter, we present the definitions and background on the techniques used in this thesis. We first study the model for nonlinear systems that we use in this thesis in Section 2.1. Finally, we survey the established techniques for verification and validation of analog circuits.

2.1 Modeling and simulation of nonlinear and mixed-signal analog circuits

A nonlinear time-variant circuit is modeled as a differential algebraic equations (DAEs) through modified nodal analysis (MNA) [32] of the circuit's netlist. Let f and g denote the piecewise continuous time-variant nonlinear function governing the dynamics of the circuit, and $t \in [0, \infty)$. Let $\mathbb{S} \subseteq \mathbb{R}^n$ denote the continuous state space of the circuit. Let h be the piecewise continuous small perturbation function that results from modeling errors, aging, or uncertainties and disturbances. Let $\mathbb{U} \subseteq \mathbb{R}^m$ denote the input space of the circuit. $\mathbf{x}$ denotes the state variables, and $\mathbf{u}$ denotes the input variables of the circuit. $\mathbf{u}(t)$ is a piecewise continuous input signal. $\mathbf{x}(t)$ denotes the state of the circuit at time t. The initial state of the circuit is $\mathbf{x}(0)$. The initial state should be explicitly defined by the user; otherwise, it will be determined through DC operating point analysis [32]. A nonlinear analog circuit is described by an n-dimensional differential algebraic equation:[1]

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}(t), \mathbf{u}(t), t) + h(\mathbf{x}(t), t)\\ 0 &= g(\mathbf{x}, \mathbf{u}(t), t)\end{aligned}$$

[1]In this thesis, we use a bold character $\mathbf{v}$ for vectors and italic characters v_i for variables.

We consider that system as a perturbation of this nominal system:

$$F(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t) = 0 \tag{2.1}$$

A *solution* of the circuit in the time interval $[t_1; t_2]$ is the path taken by the circuit from state $\mathbf{x}(t_1)$ to state $\mathbf{x}(t_2)$. For a given state $\mathbf{x}(t_1)$ and input $\mathbf{u}(t_1)$, the differential constraints in Equation 2.1 determine the trajectory of the circuit in the interval $t \in [t_1\ t_2]$. The solution of the circuit derived from an action trajectory for the initial state $\mathbf{x}(t_i)$ at time $t = t_i$ is defined as an initial value problem (IVP) by:

$$\mathbf{x}(t) = \mathbf{x}(t_i) + \int_{t_i}^{t} f(\mathbf{x}(t'), \mathbf{u}(t'))dt' \tag{2.2}$$

For nonlinear analog circuits, since f is constructed from a netlist of analog circuits, f and its partial derivatives are continuously differentiable. In practice, g is usually unknown, but some information about it is known, such as its upper and lower bounds and piecewise continuity. Hence it is presented as an external term in the DAE model. For practical circuits, the solution of nonlinear analog circuits can be computed using a numerical ODE/DAE solver such as MATLAB or SPICE. Piecewise continuous input $\mathbf{u}(t)$ models a wide variety of inputs like continuous inputs (in analog circuits) and piecewise continuous inputs (like analog interfaces such as DAC circuits). Equation 2.1 models transient parameter variations that are due to changes in input $\mathbf{u}(t)$ and small perturbations in the circuit that result in changes in h in Equation 2.1.

2.2 Reachability analysis and safety definition

The *reachable set* is the set of all states that are reachable from the initial set of states for all possible solutions (Equation 2.2) (paths), for all admissible input signals U.

$$R_{x(0)}(U) = \bigcup_{x \in x(0)} \bigcup_{u \in U} \bigcup_{t \in [0, +\infty)} R(x, u, t) \tag{2.3}$$

where $R(x, u, t)$ is the state trajectory from state x. $R_{x(0)}$ denotes the reachable set from the initial set $x(0)$ for all $u \in \mathbb{U}$.

The over-approximated reachable set $\overline{R_{x(0)}}$ is defined as a set that satisfies $R_{x(0)} \subseteq \overline{R_{x(0)}} \subset \mathbb{S}$. Given the state space of an analog circuit $\mathbb{S}$, we want to verify safety properties. We define safety properties by specified sets of unsafe regions in the state space $\mathbb{R}_{\text{unsafe}}$. A safety property is satisfied if there is no possible trajectory from any of the initial states toward $\mathbb{R}_{\text{unsafe}}$. We conclude that the safety property has been satisfied when

$$R_{x(0)} \cap R_{\text{unsafe}} = \emptyset \tag{2.4}$$

On the other hand, $\overline{R_{x(0)}} \cap R_{\text{unsafe}} \neq \emptyset$ is not necessarily an indication of safety violation. This is an implication that we cannot yet determine the safety of the circuit.

2.3 Variational bayesian inference

In this section, we describe the *variational Bayesian inference* (VBI) algorithm [4]. We use the VBI algorithm to infer a mixture distribution of the given set of samples in Chapter 4. Let $\{\mathbf{x}_1, \ldots \mathbf{x}_n\}$ denote a set of samples from an N-dimensional sample space. We assume the samples are from an independent and identically distributed Gaussian mixture distribution with unknown mean and variance. A *mixture distribution* is a distribution whose density is the sum of a set of components. We want to compute the mean, variance and the weight of the components in the mixture distribution. The VBI algorithm infers the distribution of samples $\mathbf{x}_i$ as a **mixture Gaussian distribution** of the form

$$\sum_{i=1}^{K} \pi_i \, \mathcal{N}(\mu_i, \Lambda_i^{-1}) \tag{2.5}$$

where K represents the number of Gaussian components $\mathcal{N}$ in the mixture distribution with mean $\mu_{\mathbf{i}}$, and variance Λ_i^{-1} (Λ_i is the precision), and $\pi_{\mathbf{i}}$ denotes the weight of each component in the mixture.

An overview of the VBI algorithm is as follows. Variational Bayes fits the samples to a mixture Gaussian distribution (Equation 2.5) by iteratively computing and updating the parameters $\mu_{\mathbf{i}}$, $\mathbf{\Lambda_i^{-1}}$, and $\pi_{\mathbf{i}}$ for each of the K components in the mixture. Let (latent variable) z_{ij} indicate whether a corresponding sample $\mathbf{x}_i$ belongs to component j in the mixture. Let $\mathbf{z}_i$ denote a vector of z_{nk} for $k = 1 \dots K$. Each row j in $\mathbf{z}_i$ corresponds to the probability that this sample belongs to component j. So the $\mathbf{z}_i$ is a one-of-K vector where one of the elements, say j, is 1 (i.e. the sample z_{ij} probably belongs to component j) and all other $K-1$ elements are 0. Finally, let $\mathbf{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_n\}$. The variational Bayes models the variable $\mathbf{Z}$ and the parameters mean μ, precision $\mathbf{\Lambda}$, and mixture weight π as random variables (where the mean follows a Gaussian distribution, the precision follows a Wishart distribution, and the mixture weight follows a Dirichlet distribution). We refer our readers to [4] to drive the equations necessary to compute the mean μ, precision $\mathbf{\Lambda}$, and mixture weight π.

The conditional distribution of $\mathbf{Z}$, given the mixture weights π, is

$$p(\mathbf{Z}|\pi) = \prod_{i=1}^{n}\prod_{j=1}^{K} \pi_j^{\mathbf{z}_{ij}} \tag{2.6}$$

Additionally, the conditional distribution of the sampled state given the latent variables is

$$p(\mathbf{X}|\mathbf{Z}, \mu, \mathbf{\Lambda}) = \prod_{i=1}^{n}\prod_{j=1}^{K} \mathcal{N}(\mathbf{x}_i|\mu_j, \mathbf{\Lambda}_j^{-1})^{\mathbf{z}_{ij}} \tag{2.7}$$

where $\mu = \{\mu_k\}$ are the means of the components of the distribution and $\mathbf{\Lambda} = \{\mathbf{\Lambda}_k\}$ are the precisions. The covariance matrix will be computed by inverting the precision matrix $\mathbf{\Lambda}$.

We assume a Dirichlet[4] distribution for mixture weights π:

$$p(\pi) = Dir(\pi|\alpha_0) = C(\alpha_0)\prod_{i=1}^{K} \pi_i^{\alpha_0 - 1} \tag{2.8}$$

where $C(\alpha_0)$ is the normalization constant for Dirichlet distribution [4]. For mean and precision, we assume a Gaussian-Wishart [4] prior distribution, as follows:

$$\begin{aligned} p(\mu, \mathbf{\Lambda}) &= p(\mu|\mathbf{\Lambda})p(\mathbf{\Lambda}) && (2.9) \\ &= \prod_{i=1}^{K} \mathcal{N}(\mu_i|\mathbf{m}_0, (\beta_0\mathbf{\Lambda}_i)^{-1})\mathcal{W}(\mathbf{\Lambda}_i|\mathbf{W}_0, v_0) && (2.10) \end{aligned}$$

In [4], the responsibilities r_{nk} are modeled and computed as the expectations of the random variable z_{nk}. Therefore, computing the exact solution is difficult. The algorithm assumes that the variational distribution can be factorized between the variable $\mathbf{Z}$ and the parameters mean μ, precision Λ, and mixture weight π and approximates the mixture distribution. The joint distribution of all random variables is

$$p(\mathbf{X}, \mathbf{Z}, \pi, \mu, \mathbf{\Lambda}) = p(\mathbf{X}|\mathbf{Z}, \mu, \mathbf{\Lambda})p(\mathbf{Z}|\pi)p(\pi)p(\mu|\mathbf{\Lambda})p(\mathbf{\Lambda}) \tag{2.11}$$

where $\mathbf{X}$ is the set of samples and $\mathbf{Z}$ is the latent variable. We assume that variational distribution factorizes between the latent variables and parameters such that

$$q(\mathbf{Z}, \pi, \mu, \mathbf{\Lambda}) = q(\mathbf{Z})q(\pi, \mu, \mathbf{\Lambda}) \tag{2.12}$$

In order to compute the mean vector and precision vector of the mixture distribution, the algorithm iteratively alternates between two steps: i) computing the responsibility (expectations) of each cluster in explaining the samples, and ii) using the responsibilities to update the distribution parameters in order to maximize expectations. The algorithm iterates between the two steps until the distribution converges. The output of the algorithm is the mean μ, the precision Λ, and the weight mixture π of the mixture distribution (Equation 2.5). Further details of this technique can be found in [4]. Variational Bayes computes the mixture weights as $\pi_i = \frac{1}{n}\sum_{j=1}^{n} r_{ji}$ where r_{ij} are responsibilities of each sample with respect to each component in the distribution [4]. The responsibilities and weight coefficients of components that provide inadequate explanation of the samples will converge to zero. Therefore, after convergence, components with negligible mixture weights are discarded. As a result, the technique does not require prior information that specifies the exact number of components in the mixture distribution. In [4], this feature is referred to as *automatic relevance determination.*

An advantage of VBI over other clustering or inference algorithms is that the number of components K does not need to be known a priori. The VBI algorithm computes the number of components K automatically. Furthermore, the algorithm does not require prior information and uses conjugate priors to approximate the prior distribution using its parameters. The VBI approximates the computationally expensive integral that arises in the Bayesian inference by factorizing the prior distribution; therefore, the VBI algorithm is very fast. Although the VBI is very fast, the inference results are as accurate as those of other Monte Carlo Markov chain methods, such as Gibbs sampling for Bayesian networks [4]. Finally, the VBI algorithm can be implemented online. As a result, the algorithm can compute and update the sample distribution incrementally [33].

2.4 Established techniques for validation of analog circuits

We briefly describe our contributions in this thesis in the context of related work.

2.4.1 Analog test

We refer our readers to [11] for an introductory tutorial and to [34] for a review of the classic works. Researchers have focused on generating post-silicon tests for nonparametric testing [35, 36, 17], and parametric fault models [37]. Some techniques use learning algorithms to identify bad regions [38, 37].

Recently, researchers investigated generation of pre-silicon tests for analog circuits. That problem is closely related to that of runtime monitoring and falsification of analog and hybrid systems [20, 39, 40, 41, 42].

The RRT algorithm was originally developed in robotic motion planning [43]. In the classic RRT, the growth of RRTs is locally, but not globally, optimal. Several techniques have tried to address that issue [40, 42, 44]. In [39], the authors propose to introduce LTL properties into RRT to verify safety properties of hybrid systems for falsification. Dang and Nahhal [42] use RRT to generate counter-examples in analog and hybrid systems.

2.4.2 Runtime monitoring

Zaki et al. [20] survey recent literature on runtime monitoring and verification of analog and mixed-signal (AMS) designs. Researchers have employed a variety of techniques to analyze the transient behaviors of circuits in either an on-line or off-line fashion. Examples of such approaches include using interval arithmetic to validate the behavior of the circuit [45], using linear hybrid automation as a template monitor for online monitoring [46], and generating observers from PSL properties to monitor the simulation [47, 48].

The specification language we used in our work was first developed in [49, 50]. The tool described in those papers, *AMT*, synthesizes a timed automaton that monitors simulation traces for property violations. *AMT* has been used to verify some properties of DDR2 memory specification [51]. In other work, [52] propose use repeated SPICE simulation to explore the state-space of analog circuits for all possible discrete values.

In [39], the authors propose to introduce LTL properties into RRT to verify safety properties of hybrid systems for falsification. In a similar approach, [42] and [53] use RRT to generate counter-examples in analog and hybrid systems. Recently [54], used μ-calculus to reason about RRT in discrete-time control systems.

2.4.3 Reachability Analysis

Asarin et al.[55] provide an introduction and formal definition for reachability analysis. Most of reachability analysis techniques construct the reachable set from the initial set using forward reachability analysis [24]. Several techniques have investigated the usage of polytopes [24, 56, 57], zonotopes [58], or support functions [23]. Some reachability techniques are based on state space discretization methods [59, 60, 61, 62]. Another technique in control theory for verifying safety without actually computing the reachable set is using barrier certificates [63, 64].

Another related technique to ours is the backward reasoning technique [22]. Alur et al. [25] propose a technique for reachability analysis of linear hybrid systems using predicate abstraction. They propose to improve reachability analysis through vector field analysis and binary space partitioning to optimize predicate abstraction. Ratschan and She [65] propose recursive backward reasoning for hyper-boxes. In comparison to their work, we provide a more efficient partitioning algorithm using polytopes and a sound method for computing reachability decisions between adjacent polytopes for nonlinear analog systems.

2.4.4 Eye diagram analysis

There are three techniques to analyze the eye diagram: i) Monte Carlo simulations, ii) convolution-based analytical methods [9, 10], and iii) statistical methods [7, 8]. The *de facto* method for computing the eye diagrams is the Monte Carlo transient simulations [6, 32, 66, 67]. However Monte Carlo is too time-consuming and does not properly cover the simulation corners with high deviations. Researchers have worked on replacing transient simulations with convolution-based analytical methods [10, 9]. Analytical methods provide deterministic eye diagram, but are only applicable to the linear time-invariant systems. In [68], the authors construct the output waveforms using multiple edge responses. Statistical eye diagram analysis tools use statistical techniques to determine the eye diagram [7, 8, 69, 70, 71].

Recently, some efforts have been put into developing better models for worst-case eye diagrams. In [72], the authors use the step responses of pull-up and pull-downs to predict the worst-case analysis of eye-diagram of the high-speed channels. The authors construct the output waveforms using rising and falling transition responses [72]. In [68], the authors construct the output waveforms using multiple edge responses. Most of these techniques require applying a very long, often pseudo-random, bit sequence to the circuit. In practice, the length of the input bit sequence is limited, resulting in a larger eye diagram than the worst-case.

2.4.5 Test compression

Optimal ordering of analog tests is important to identify redundant tests and reduce total test time [73] [12]. Most failed tests can be strategically placed at the beginning of the test sequence in order to reduce the total test time [74]. A technique for optimal test ordering based on data from a small set of functional circuits is proposed in [13, 75]. A method for designing tests for parametric tests is proposed in [76]. Furthermore, efficient analog fault modeling can be used to lower production test time [11].

The total test time can be lowered by minimizing the total number of the tests in the batch [77]. Many researchers focused on selecting the subset of the tests to achieve the same coverage [14, 15]. The redundant tests can be identified by studying performance data from the simulation [77]. Recently, researchers used learning methods such as binary decision trees

[78] and test signatures [79] to identify redundant tests. Test signatures are used to predict performance metrics of the circuit and to identify redundant tests. Test signatures were initially proposed as a low-cost method by [79] to evaluate tests for RF circuits. Test signatures are used by [80] to identify redundant tests using regression methods. For SoCs, Golomb codes are used for on-chip compression of the test sequence [18]. Recently, a compressive-sensing testing method for 3D TSV was proposed in [19] where they use the sparsity of the testing data to design an on-chip test compressor and off-chip data recovery.

Automatic test generation tools [16, 81, 82, 83, 17, 84] are used to generate efficient tests, improve coverage, avoid redundant tests and lower the total test costs. The early literature on analog test generation is reviewed in [34]. Techniques for selecting the best test points in the analog circuit are studied in [85] and [86]. A technique for generating tests with minimum test time using a divide and conquer algorithm is proposed in [87]. Random trees areused to automatically generate tests for analog circuits with goal-oriented testing [88], with multiple objectives such as goal and coverage [89], and analyzing worst-case eye-diagrams [90].

2.4.6 Circuit optimization

Analog circuit optimization has been extensively studied in the past [3, 26, 30, 91]. Classic techniques relied on generic optimization techniques (such as simulated annealing) to optimize the circuit's performance [3, 26, 28, 91].

Recently, researchers used computational intelligence techniques to speed up circuit optimization [92, 93, 94]. [31] and [30] optimize the circuit by discretizing and exploring the state space. In [31], the authors optimize an operational amplifier, which we used in Section 9.4. Other specialized optimization techniques have been employed to address circuit optimization [95] with added objectives, such as yield [28, 31] or technology migration [94].

2.4.7 Optimization methods in machine learning

Optimization algorithms have a long research history in computer science, control, finance and many other disciplines. Classical optimization techniques are surveyed in [96]. Optimization problems in the continuous domain include convex [97], non-convex, functional (infinite dimension) optimization [98] and heuristic techniques [99] among others. Convex optimization techniques are reviewed in [97]. There are two approaches to address non-convex optimization problems: first order methods and second order techniques. First order methods, including stochastic gradient descent [100], Nesterov Momentum [101], AdaGrad [102], RMSProp [103, 104, 105], and Adam [106] rely on the gradient information to navigate toward the optimal solution. Second order techniques, based on Newton's methods, such as the SFO algorithm [107], use the Hessian information to compute the optimal solution. Many techniques combine first and second order methods, such as [108] from Google's brain team that uses L-BFGS algorithm. Heuristic optimization techniques are very effective at quickly finding approximate solutions when classical techniques fail to converge. Heuristic methods are based on simulated annealing [109, 110, 99] and genetic algorithms [111, 112, 113, 114].

2.4.8 Optimal control and motion planning

Optimization objectives often need to be optimized in dynamic contexts, *i.e.*, over time. Techniques from optimal control theory are used to optimize an integral of the output of the dynamic system. This is also known as functional optimization or infinite dimension optimization [98]. The most popular approaches toward solving the optimal control problems are using Euler-Lagrange [98, 115, 116] and Hamilton-Jacobi-Bellman equations [98, 117, 118]. Optimal motion planning techniques have been researched for autonomous robots [119] and agricultural robotics [120, 121, 122].

Classical robot motion planning algorithms have been surveyed in [43] and [123]. Popular sampling-based motion planning techniques are Rapidly-exploring Random Trees (RRT) [124, 43] and Probabilistic RoadMaps (PRM) [125, 126]. Optimal variations of the motion planning algorithms [127], such as RRT* [128], have been proposed to find the shortest-paths in the state space. Energy-optimal motion planning techniques based on optimal control are proposed in [129, 130, 131].

CHAPTER 3

THE DUPLEX RANDOM TREE OPTIMIZATION

3.1 Introduction

We propose Duplex, a random-tree based optimization algorithm for optimizing nonconvex functions and functionals. The Duplex algorithm is derived from the Rapidly-exploring Random Tree (RRT) algorithm in robotic motion planning. We describe the RRT algorithm in Section 3.2. To adapt the RRT algorithm for optimization application, we add direction to the random tree algorithm. We describe our contribution over the RRT algorithm in Section 3.3.

The Duplex algorithm has many advantages over traditional optimization algorithms based on gradient descent. The Duplex algorithm, similar to RRT, is probabilistically complete; hence it does not get stuck in local minima. Furthermore, the Duplex algorithm provides valuable feedback to the user, provides multiple candidate solutions by determining the Pareto frontier, and is also highly performance efficient, versatile and scalable. We describe and prove Duplex's properties in Section 3.7.

Duplex is very versatile and can address different optimization problems, including directed search in dynamic systems, nonconvex optimization, and functional optimization. Duplex differentiates and formulates different problem types by dividing the problem space into multiple smaller spaces (such as input, output, function). Then Duplex simultaneously grows multiple random trees in such spaces. We explain the Duplex principle of separation of spaces in Section 3.5. We cover the different problem types and their formulation in Duplex in Section 3.8. We formulate many problems in analog validation and machine learning into Duplex formulation and use Duplex to solve those problems.

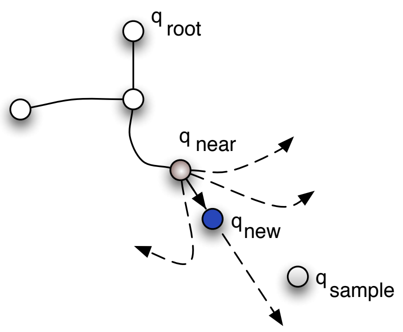


Figure 3.1: Growth of RRT through addition of a new node sampled from the state space.

3.2 Background on Rapidly-exploring Random Trees (RRT)

The Duplex algorithm is derived from the RRT algorithm. We briefly describe the RRT algorithm presented in [43]. The RRT algorithm was developed as a motion planning algorithm for robots. The objective of the RRT algorithm is to find a viable motion (path) from the robot's initial configuration of the robot to its destination. So the robot can move and get to its destination by executing the motion.

The RRT is a tree data structure. The tree is initialized through fixing of its root at a specified state[1] in the state space $\mathbb{S}$. The tree is then *grown* incrementally through the addition of edges between existing nodes and the new state selected from the state space. The selection of the new states determines the manner in which the tree grows in the state space. Typically, the new states are selected at random through **uniform sampling** of the state space.

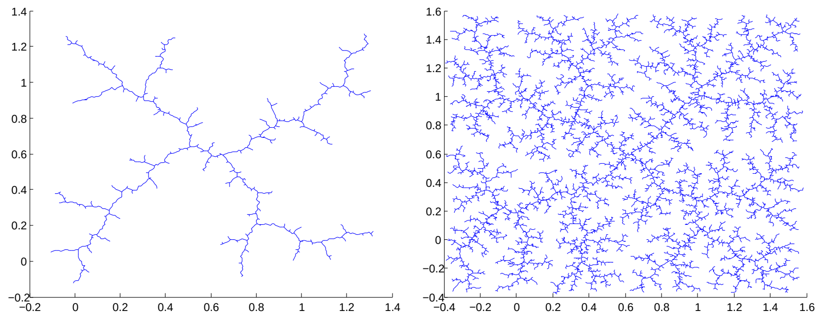
Let $\mathbb{G}$ be the RRT data structure. Each node of G corresponds to a state in $\mathbb{S}$, i.e., a unique set of values assigned to the state variables $\mathbf{x}$. Each edge represents a solution of the system from initial condition $\mathbf{x}$ for a given assignment of values to the input variables $\mathbf{u}$.

Algorithm 1 describes the growth of the tree $\mathbb{G}$ in the classic RRT algorithm [43]. At every iteration, the RRT algorithm generates a random state

[1]Throughout this thesis, *point* denotes a vector in $\mathbb{R}^n$. The *state* is a physical manifestation of the point in the state space $\mathbb{S} \subset \mathbb{R}^n$ (with corresponding scales and units). The *region* is a connected subset of the state space $\mathbb{S}$. Finally, a *node* is the state in the tree data structure (augmented with input $\mathbf{u}(t)$, time annotation t, and possible pointers to other nodes).

Algorithm 1 RRT algorithm using uniform sampling

```
1: $\mathbb{G}$.init $(\mathbf{x}(0))$
2: for $i = 1 \rightarrow MAX - ITER$ do
3:     $q_{sample} \leftarrow$ UniformSampling($\mathbb{S}$)
4:     $q_{near} \leftarrow$ FindNearestNodeInTree($\mathbb{S}, q_{sample}$)
5:     $q_{new} \leftarrow$ FindOptimumTrajectory($q_{near}, q_{sample}$)
6:     $\mathbb{G}$.expand($q_{new}$)
7: end for
```



(a) The RRT after 1000 iterations. (b) The RRT after 10000 iterations.

Figure 3.2: The classic RRT algorithm does not have any direction or bias and rapidly explores the entire reachable state space.

q_{sample} **uniformly distributed** in the state space. For every new generated state q_{sample}, the RRT algorithm will find the nearest state, q_{near}, and will determine which solution for any $u \in U$ will bring node q_{near} closer to the sampled state. The RRT determines the closest state by simulating different circuit trajectories and selecting the optimum one [42, 40] based on Euclidean distance. That process is called *shooting*. From the initial state q_{near}, the algorithm will randomly sample the input space $\mathbb{U}$ and generate the corresponding trajectory by *shooting* for a short time (Δt). The algorithm will then select the optimal trajectory as the trajectory that would result in the final state closest (based on Euclidean distance) to the q_{sample}. When the path is determined, the tree will expand from q_{near} toward q_{new} through the addition of the edge e_{new} to the tree. The algorithm stores the state q, time t, and trajectory u for each node in the tree, so later an input stimulus can be reproduced using that information. Figure 3.1 shows the growth of the RRT tree toward a given sample node. The RRT algorithm will terminate after a fixed number of iterations $MAX - ITER$.

3.2.1 Properties of the RRT algorithm

The RRT grows rapidly and quickly visits unexplored regions of the state space [43]. The RRT algorithm is *probabilistically complete*; *i.e.*, as the number of samples approaches infinity, the RRT covers the entire state space [43]. The RRT algorithm has an implicit *Voronoi bias* and therefore rapidly explores the total reachable state space, as shown in Figure 3.2. The RRT algorithm does not exhibit any bias toward the destination. In optimization problems, the goal is to reach the minimum of the function, not to explore the state space. As a result, the RRT algorithm is very inefficient for solving directed search or optimization problems that arise in analog circuit validation.

3.2.2 Comparison of the random tree algorithm and linear search

Traditional optimization methods are based on non-branching linear search algorithms such as gradient descent, simulated annealing, random walk, and hill climbing algorithms. The linear search starts from a randomly selected state and *walks* in the state space without branching to reach the optimum state.

The linear search algorithms do not branch the simulation and do not maintain the history of the previously visited states. If they get stuck in local minima, they have to backtrack which results in poor performance. Many algorithms avoid backtracking; these algorithms can get stuck in local minima and do not converge to the optimum solution in non-convex problems. Furthermore, they only consider one path per run, and cannot produce the Pareto frontier without multiple reruns. Random tree algorithm can address both of these issues.

3.3 Adding direction to the random tree algorithm

In search and optimization, the objective is not to explore the state space, but to reach the goal region and find a path to the optimum solution. The RRT algorithm, due to its uniform sampling of the state space, will generate many samples in the unreachable region and grows the random tree toward the boundaries of the reachable state space. In systems, a goal region can denote the set of states where the system would fail. Similarly, for a function, a goal region would be where the function would take the minimum value. In these cases, if we know the destination we do not need to explore the entire state space.

We improve the efficiency of the random tree algorithm by adding bias to the algorithm's growth. We *direct* the growth of the random tree toward the goal region by sampling from a biased distribution, centered in the goal region, instead of uniform sampling of the entire state space. As a result, the algorithm is more goal-driven and spends fewer iterations on exploring the unreachable regions in the space.

To the best of our knowledge, this is the first work that uses random trees for optimization by adding direction bias to the random tree algorithms. The random trees inherit the positive properties of the RRT algorithm such as probabilistic completeness and generating the Pareto frontier. Furthermore, adding direction will improve the efficiency of the algorithm and make it applicable to optimization problems. Finally, we introduce the Duplex principle which allows for solving different types of optimization problems such as nonconvex and functional optimization.

3.4 Duplex algorithm

The Duplex algorithm is an optimization algorithm that uses random trees to find the optimum solution. Duplex uses random tree search, a tree based simulation algorithm that also maintains the tree data structure as a record of the state space traversed. It maintains and simultaneously grows multiple homomorphic (mirrored) random trees: one in the state space and the other in the objective space. In the objective space, it uses the basic random tree search to find the globally optimal design or the goal region. In the state space, it decides which parameter needs to change to get closer to the goal region. This decision is made using a noisy gradient descent algorithm in combination with reinforcement learning [132] that evaluates the history of previous changes to the parameters in the parameter tree based on a reward function. There is no open ended search in the parameter modification phase (local step) of the algorithm.

Due to the probabilistic completeness property of random trees [43], Duplex does not get stuck in local minima. This is in contrast to random walk based methods like simulated annealing or gradient descent. The guidance in every step from the global search towards the local step decision helps in converging quickly to the optimal goal region.

3.4.1 Equivalence of search and optimization algorithms

The traditional random tree is a search algorithm. As a result, the objective of the algorithm is to find a path from the initial state to the goal state. Every search problem can be mapped to an optimization problem and vice versa (by defining the cost function as a distance to the optimum state). Therefore, a random tree search algorithm can also be used for optimization. The random tree search-turned-optimization algorithm would enjoy the benefits of random tree search, such as probabilistic completeness and generating Pareto frontier, although the algorithm would have a very poor performance due to lack of direction toward minimum.

Random trees are shown to consistently outperform random walk based search methods such as Monte Carlo simulations for search applications [89, 133, 134]. Efficiency improvement can be credited to the data structure maintained by the random tree algorithm during the simulation. While growing, it samples a new state in the goal region (desired solution set), and then determines which state is closest (in L_2-norm sense) to that sampled goal state among all of the previously visited states in the tree. It simulates a path between the closest state and the newly sampled state and adds the new state to the tree. This is in contrast to the memory-less sampling of points in the Monte Carlo based methods.

3.5 Duplex principle: separation of spaces

The Duplex algorithm optimizes different optimization problems such as search, nonconvex optimization and functional optimization. The key in solving totally different problems with the same algorithm is in how we formulate these problems using Duplex's principle of separation of spaces.

Two important aspects of the Duplex algorithm are as follows. Firstly, Du-

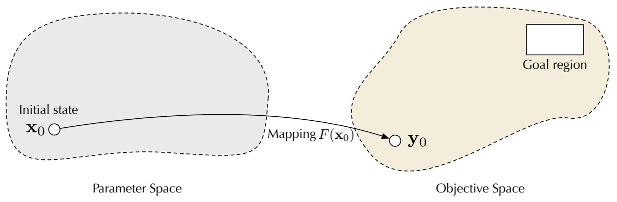


Figure 3.3: The input and objective spaces in multi-objective optimization.

plex partitions the objective functions into multiple spaces. Figure 3.3 shows the principle of space separation in Duplex. Secondly, it performs an efficient search in these multiple spaces using random trees. Duplex simultaneously constructs and maintains multiple different, but mirrored (homomorphic), random trees in the state, function and functional spaces. Intuitively, the output tree is the *mirror* of the input tree in the function space.

These trees represent different relationships. An edge in the input tree indicates that the two input states connected to that edge differ in *exactly one* variable. An edge in the output tree between indicates that the corresponding nodes in the input tree are connected. For each node $\mathbf{p}$ in the input tree, there exists a corresponding node in the output tree, and vice versa. The corresponding node in the output tree is computed by objective function with the given input.

3.6 Problems solved using the Duplex algorithm

We applied Duplex optimization to different problems in analog validation and machine learning. In each case, we formulate the problem as an optimization problem with specific auxiliary objectives. Then we use Duplex to optimize the objective and determine the optimum solution. In all of these cases, the objective function is non-traditional, non-convex or functional and there is no known algorithm to optimize this objective function efficiently.

We use Duplex for directed search in the space of nonlinear dynamic systems. The aim is to determine a path from the initial configuration to the final configuration. We use directed search for automatically generating input stimuli for checking circuits failure and improving test coverage. We develop a language for specifying and validating analog properties over the Duplex's tree data structure. We provide a technique to monitor logic properties during the execution of the Duplex algorithm.

We use Duplex for optimizing the transient response of the circuits. Specifically, Duplex can generate worst-case eye diagrams by maximizing signal distortion and can compress tests to execute them faster by generating time-optimal tests. The transient response of the circuit is an integral of the circuit's dynamic. Thus, optimizing the transient response is an instance of functional optimization for nonlinear systems.

Many different types of machine learning problems are optimizing an error/loss/energy of the current inferred hypothesis from the ideal answer. Therefore, optimization is the core of machine learning. We consider supervised learning using logistic regression with adaptive regularization and unsupervised learning based on the K-mean clustering algorithm. In supervised learning, we have the labeled data. We use the Duplex algorithm to minimize the squared error of the logistic regression hypothesis function. In the case of unsupervised learning, where we do not have labeled data, we model the distortion of each cluster as an energy function, and we use Duplex to minimize the distortion function.

3.7 Properties of the Duplex algorithm

3.7.1 Probabilistic completeness

The Duplex algorithm in finite dimension spaces is probabilistically complete. As the number of iterations goes toward infinity, the probability that the algorithm will find the optimum solution converges toward one.

We prove the probabilistic completeness by using a similar proof for the RRT algorithm [43, 135]. We show that the distribution of the samples in the random tree converges to the sampling distribution.

Let C denote the state space of the problem. If we have multiple spaces $C_1, \ldots, C_k$, we can combine them to form C. Assume C is nonconvex, bounded, open, and reachable. Let n denote the number of nodes in the random tree. Let $d(x)$ denote the random variable whose value is the distance of x to the nearest node in the random tree. For any $x \in C$ and positive real number $\epsilon > 0$, we have

$$\lim_{n \to \infty} P[d_n(x) < \epsilon] = 1. \tag{3.1}$$

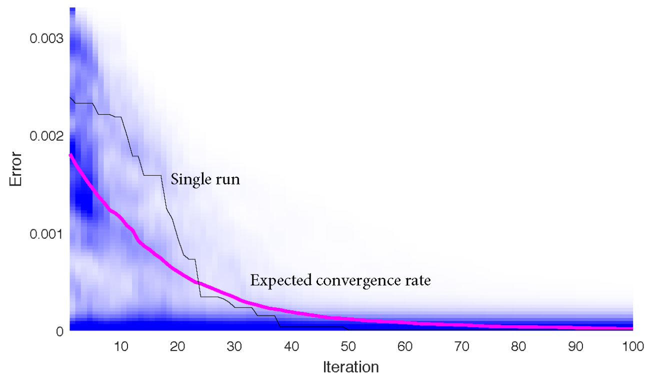


Figure 3.4: The convergence rate w.r.t. number of iterations for the Duplex algorithm for nonconvex optimization. Our algorithm converges very fast toward the optimum solution from any initial state. Duplex is not sensitive to the choice of initial state.

As the number of iterations n goes toward infinity, the probability that a node in the random tree is sampled arbitrarily close to the state x converges to one.

Proof: Let x denote any arbitrary state in state space C. Let x_0 denote the root of the random tree. Let $B(x)$ denote a ball of radius ϵ centered on x. The ball $B(x)$ has a strictly positive volume. Initially, we have $d_1(x) = |x, x_0|$. At each iteration, the probability that the new sampled node will be inside $B(x)$ is strictly positive. If all random tree nodes lie outside of $B(x)$, then $E[d_k] - E[d_{k+1}| > b$ for some positive real number $b > 0$. This implies that $\lim_{n\to\infty} P[d_n(x) < \epsilon] = 1$.

3.7.2 Rate of convergence

The Duplex algorithm quickly converges to the optimal solution. We empirically observed the Duplex algorithm has a sub-linear convergence rate for non-convex optimization. However, we could not mathematically prove this property of the Duplex algorithm for non-convex nonlinear systems.

Figure 3.4 shows the *visually weighted regression* plot for convergence rate for the Duplex algorithm for optimizing an inverter circuit using nonconvex optimization (Chapter 9). We measure error as the minimum distance from every node in the random tree toward the optimum solution. We execute Duplex for 100 independent runs with a random initial state and draw the overlapping convergence plots in the visually weighted regression plot. We

Algorithm 2 Abstract duplex algorithm

```
1: initialize the algorithm
2: while not converged() do
3:     $q_{\text{from}}$ = pick a node from database        ▷ Global step
4:     $q_{\text{new}}$ = update $q_{\text{from}}$              ▷ Local step
5:     Evaluate $q_{\text{new}}$
6:     Add $q_{\text{new}}$ to the database.
7: end while
```

also draw the average of all the convergence plot as the expected convergence rate. As shown in the convergence figure, Duplex quickly converges toward the goal region in the performance space. Figure 3.4 highlights two facts about the Duplex algorithm: 1) Duplex converges sub-linearly fast toward the goal region and 2) Duplex is very stable with respect to the choice of the initial state

In our experiment, we uniformly sampled the initial (root) state, so the variance at the beginning is very high. On the other hand, toward the end of the algorithm the variance in error is low because Duplex converges to the optimum results regardless of the choice of the initial state.

3.8 The Duplex optimization algorithm

3.8.1 Abstract Duplex Algorithm

The Duplex algorithm grows random trees in the state space in a directed manner. Every node in the random tree is a vector in the state space of the problem. The algorithm grows the random tree iteratively by adding nodes to the tree. At every iteration, the algorithm repeatedly makes two decisions: i) which node to branch the random tree from, and ii) where to go from the branched state. Each of these decisions impacts how Duplex is directed toward the goal region and avoids local minima.

The overview of Duplex is shown in Algorithm 2. In the beginning, the algorithm will initialize the random tree by constructing the root node. Then duplex iteratively adds nodes to the tree until convergence criteria are satisfied. At every iteration, the algorithm picks a node from the tree. How the algorithm selects q_{from} depends on the problem. Next, the algorithm will generate a new input q_{next} by modifying the node q_{from}. How the algorithm will update q_{next} also depends on the problem type. Finally, the algorithm evaluates the q_{next} according to the update rule and adds the new node q_{new} to the tree.

If we avoid branching the tree, the random tree becomes a random walk in the space. The duplex algorithm is a generalization of the gradient descent algorithm. The global step is set to pick the most recent node as the next node. Furthermore, the local step is to take the step in the direction of the gradient. The local update rule for gradient descent is shown in Equation 3.2 where f is the function that the algorithm is optimizing, γ is the learning rate and β is the white noise.

$$q_{new} = q_{from} - \gamma \bigtriangledown f(q_{from}) + \beta \tag{3.2}$$

We use Duplex algorithm to solve three class of optimization problems. We classify these problems according to the dimension of the problem space and type of objective functions into the following categories:

1. Type-I: Search in nonlinear systems
2. Type-II: Non-convex optimization in finite dimensions
3. Type-III: Functional optimization

In type-I problems we use Duplex to find a path from the initial state to the goal state in the state space of a nonlinear dynamic system. In type-II problems, we extend the duplex framework to optimize non-convex objective functions and find the minimum in finite dimensions. Finally, in type-III problems, we extend the algorithm to optimize functionals in infinite dimensional space. Sections 3.8.3-3.8.4 will describe these problems in detail. In each case, we formulate the problem in Duplex's formulation and use the Duplex algorithm to optimize the objective function.

3.8.2 Type-I: Search problems in finite dimensions

We define type-I problems as search problems in the state space of dynamic systems. The goal is to find a path from the initial state to the goal region if such path exists.

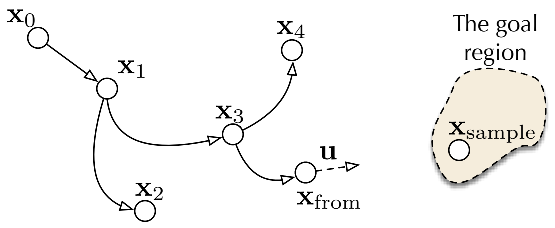


Figure 3.5: Growing random tree toward the goal region.

Algorithm 3 Duplex optimization for Type-I search

```
1: Set root of the tree at $\mathbf{x}_0$.
2: while $x^*$ is not reached do
3:     $q_{\text{sample}}$ = Generate a sample in the goal region
4:     $q_{\text{from}}$ = find the nearest node in the tree to $q_{\text{sample}}$      ▷ Global step
5:     $u$ = choose a trajectory from $q_{\text{from}}$ toward $q_{\text{sample}}$.      ▷ Local step
6:     $q_{\text{new}} = q_{\text{from}} + \int_t^{t+\Delta t} f(x,u,t)dt$.      ▷ Evaluate
7:     Add $q_{\text{new}}$ to the database.
8: end while
```

Problem Definition Given a nonlinear system f, a continuous space $\mathbf{R}^n$, an initial state $\mathbf{x}_0 \in \mathbf{R}^n$ and the boundary state $\mathbf{x}^* \in R^n$, find an input sequence $\mathbf{u}(t)$ such that

$$\mathbf{x}^* = \mathbf{x}_0 + \int_{t=0}^{T} f(\mathbf{x}, \mathbf{u}, t)dt \tag{3.3}$$

The Duplex algorithm solves type-I problems by growing a random tree in the space $\mathbf{R}^n$ from the initial state $\mathbf{x}_0$ toward the goal region $\mathbf{x}^*$. The tree is directed to grow toward the goal region. Figure 3.5 shows how the Duplex algorithm grows the random tree toward the goal region.

Algorithm 3 demonstrates how Duplex works. The algorithm will iteratively pick where to branch the random tree from (q_{from}) according to the global step. Then it will choose the optimum trajectory from the q_{from} using local steps. The algorithm iteratively repeats global and local steps until we find a sufficient number of samples in the goal region.

Global steps in the Duplex algorithm

At every iteration, we have multiple candidates to branch the simulation from because the state space has n dimensions, and we have multiple nodes

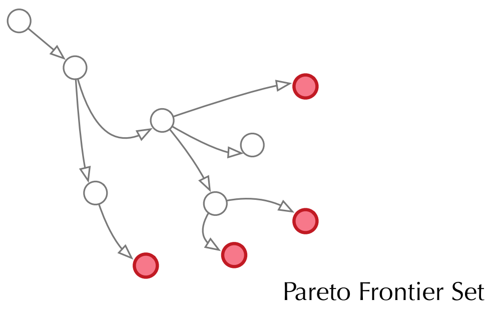


Figure 3.6: The Pareto frontier of the random tree.

in the random tree. Ideally, we wish to pick a node in the *Pareto front* of all nodes in the random tree (Figure 3.6). Pareto front is the set of all potential candidates in the random tree. Formally, Pareto frontier is defined as a relation between nodes. A node $\mathbf{x}$ is dominated by node $\mathbf{y}$ if

1. $e(\mathbf{x}_i) \leq e(\mathbf{y}_i)$ for all $1 \leq i \leq n$.
2. $e(\mathbf{x}_j) < e(\mathbf{y}_j)$ for at least one $j \in \{1, \ldots, n\}$.

The e_i function is the projection of the distance to the optimal solution in the i^{th} dimension. Pareto frontier is the set of all nodes that is not dominated by any other node.

To compute the exact Pareto set, we need to calculate the convex hull of all the nodes in the random tree, which is computationally very expensive. Instead, we choose an alternative method to sample the Pareto frontier which is computationally cheaper. At every iteration, we generate a sample q_{sample} inside the goal region. Our goal is to direct the random tree toward q_{sample}. We pick the nearest node q_{from} from the q_{sample} in the random tree. q_{sample} is in the Pareto frontier. We branch the simulation from the node q_{from}. The computational cost of each nearest neighbor query is $O(\log n)$, and the nearest neighbor search implementations are scalable regarding both the dimensions and number of samples. We use KD-tree data structure as a database for storing the nodes in the random tree and performing nearest neighbor queries.

Local steps in the Duplex algorithm

For the local steps, we have to determine what is the best input trajectory that can take us from q_{from} to the new sample q_{sample} in the goal region.

Duplex uses two strategies for determining the optimum trajectory depending on the availability of the gradient information.

Taking the local steps along the gradient If we have the gradient information available, the optimum trajectory is given by the Jacobian of the function f. We also add white noise η to the gradient to improve the performance of the algorithm around the saddle points. Finally, we add momentum to the trajectory to improve performance.

Learning the optimum local steps If the gradient information is not available, we use reinforcement learning to determine the optimum trajectory. We train a Q-function Q(x, u) where x is the states in the state space and u is the trajectories (actions). Next time for the given state x and trajectory u, we evaluate u using the Q function.

The Duplex algorithm for type-I problems is very similar to the RRT algorithm for motion planning. However, there are important differences between the two algorithms: i) Duplex is directed to grow toward the goal region. Hence, the algorithm is more efficient that the classic RRT. ii) The Duplex algorithm can optimize other objectives while growing the random tree such as improving coverage. An example of a type-I problem in analog validation is automated stimulus generation for nonlinear analog circuits. The test generation problem involves finding an input sequence (test stimuli) from the reset state of the circuit to its failure region. We formulate the stimuli generation problem as a type-I problem and use the Duplex algorithm to solve it in Chapter 4.

3.8.3 Type-II: Optimization problems in finite dimensions

Most practical problems have several (possibly conflicting) objectives that need to be satisfied. Multi-objective optimization is the problem of finding a vector of parameters which satisfies constraints and optimizes a vector function. We use duplex for multi-objective optimization of nonconvex nonlinear functions in finite dimensional spaces.

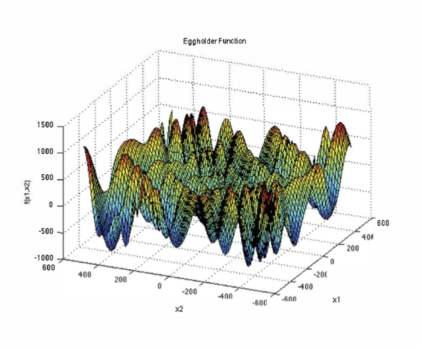
The multi-objective optimization problem is defined as follows:

$$\begin{aligned} min\ \mathbf{y} = F(\mathbf{x}) &= [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})] && (3.4) \\ subject\ to\ G(\mathbf{x}) &= [g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_k(\mathbf{x})] \geq 0 && (3.5) \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, 1 \leq i \leq n && (3.6) \end{aligned}$$

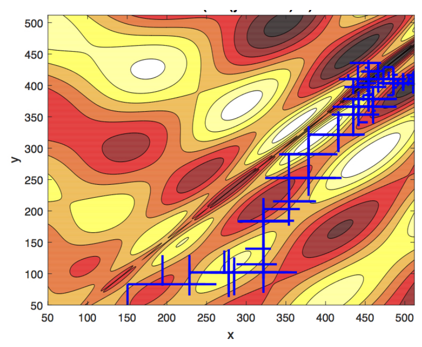
Vector $\mathbf{x} = (x_1, \ldots, x_n)$ is the parameter state. Let $\mathbf{y} = F(\mathbf{x})$ denote the objective vector. We want to optimize objective function F with respect to m non-convex nonlinear objective functions $f_1, \ldots, f_m$. The goal region is defined according to k inequalities $g_1, \ldots, g_k$. Furthermore, we have n lower and upper $[x_i^{(L)}, x_i^{(U)}]$ bounds for each parameter x_i.

Duplex maintains two disjoint spaces, namely the parameter (input) space $\mathbf{X}$ and the objective (function) space $\mathbf{Y}$. We generate an initial state $\mathbf{x}_0$ by sampling the parameter space. Then we simultaneously grow two random trees, namely the parameter tree and the objective tree in the parameter and objective space, respectively. The Duplex algorithm takes the global and local step similar to standard Duplex algorithm presented in Section 3.8.3. At every iteration, the algorithm generates a sample, say $\mathbf{y}_{\text{sample}}$, toward the goal region in the objective space. Then it searches the objective tree in the objective space for the nearest node toward $\mathbf{y}_{\text{sample}}$, say $\mathbf{y}_{\text{near}}$. Next, we find the corresponding parameter state $\mathbf{x}_{\text{near}}$ to $\mathbf{y}_{\text{near}}$ in the parameter space. Duplex branches from the nearest node $\mathbf{x}_{\text{near}}$ in the parameter tree. The algorithm selects the optimum input choice according to the noisy gradient descent algorithm combined with reinforcement learning to get closer to $\mathbf{y}_{\text{sample}}$. Eventually, the algorithm converges toward the goal region in the objective space.

Figure 3.7 shows how Duplex can optimize the egg-holder function. The egg-holder function is a very nonconvex function and has multiple local minima in its domain. The surface plot of the egg-holder function is shown in Figure 3.7. The Duplex algorithm creates two disjoint spaces: the parameter space (x_1, x_2), and the objective space $f(x_1, x_2)$. The algorithm then searches for the minimum of f in the objective space. As a result, the Duplex grows multiple branches toward the minimum. Although a few branches get stuck in local minima, finally the algorithm converges toward the global minimum, as shown in Figure 3.7. Figure 3.7.b shows the parameter tree rendered from the state space.



(a) The landscape of the egg-holder function with multiple local minima.



(b) Duplex converges to the global minimum without getting stuck in local minima.

Figure 3.7: Using Duplex for optimizing a non-convex function.

Type-II problems are generalizations of type-I problem. A type-I problem can also be formulated as a type-II problem by defining an error function $e(x) = |\mathbf{x}_0 - \mathbf{x}^*|$ and minimizing the error function e.

3.8.4 Type-III: Functional optimization in infinite dimensional space

Consider the nonlinear system that is expressed in the usual state space form as

$$\dot{x} = f(x, u, t) \quad (3.7)$$
$$y = h(x, u) \quad (3.8)$$

in which x, u and y denote system state, input and output, respectively. To characterize the performance of such systems, we consider optimality criteria that can be expressed as a performance functional of the form

$$J(x, u, y) = \int_{t_0}^{t_1} l(x, u, y)dt + M(x(t_1), y(t_1)) \quad (3.9)$$

in which l denotes the running cost at time t and M denotes a terminal cost. Note that we have expressed performance as a function of both state and output, for the sake of generality (even though, strictly speaking, this generality is redundant).

In many optimization problems, the goal is not just to reach the optimum

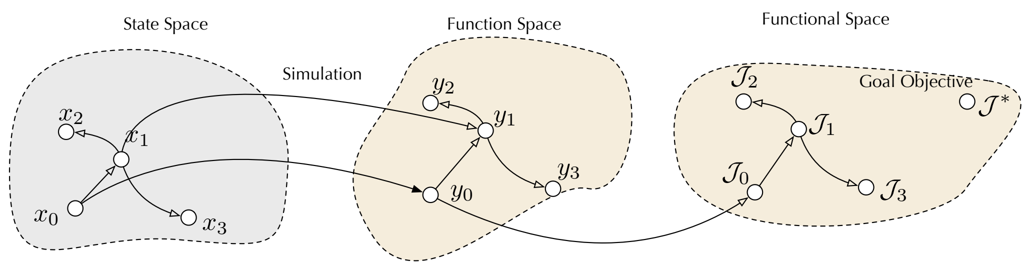


Figure 3.8: Partitioning the spaces into state space and function (objective) spaces in Duplex.

state, but to minimize a functional over the path to get there. For example, optimizing energy is a functional optimization, whereas minimizing power is a case of non-convex optimization. Functional optimization problems appear across a variety of disciplines from automated motion planning for driverless cars with objectives such as fuel efficiency and arriving time to optimizing for worst-case eye diagrams in analog circuits.

Previously, the functional optimization problems have been studied using optimal control technique. The most popular approach toward solving the optimal control problems was using Euler-Lagrange (E-L) [98] and Hamilton-Jacobi-Bellman (HJB) equations [98]. The E-L and HJB equations find the optimal solution, but they are very limited in scope and only apply to very simple systems.

The Duplex algorithm solves the functional optimization problem by augmenting its space model with functional spaces as shown in Figure 3.8. Then, the search is performed in three spaces simultaneously: i) the state space, ii) the function space, and iii) the functional space. Duplex uses the function space to enforce the boundary conditions. For every node in the function space, the Duplex algorithm evaluates the value of the functional from that node to the root of the tree and stores the final result as a node in the functional tree.

At every iteration, the algorithm generates two sample nodes q^{sample} in the function and functional space. The algorithm grows the tree toward these nodes to simultaneously enforce boundary conditions and minimize the value of the objective functional. It picks the nearest node q^{near} in the functional space to the q^{sample} and branches the simulation from the nearest node. The algorithm samples a new input in the state space, then computes the objective function and updates the random trees in the function and functional space accordingly.

In comparison to the previous work, Duplex relies on numerical computation and has virtually no limitation on what types of dynamics can be modeled. More complex objective functions can be easily formulated in their separate space, which allows for efficient modeling and prototyping real-world problems. The initial and boundary conditions are modeled separately in the function space and can be efficiently enforced. Empirically we observed that the Duplex algorithm efficiently converges toward the optimum value and is two orders of magnitude faster than random-walk based methods.

Consider the Dido isoperimetric [98] functional optimization problem as shown in Figure 3.9. The Dido problem is a classic problem in optimal control. The algorithm asks for what is the largest enclosed area given fixed initial, boundary and perimeter conditions. Specifically, we are interested in minimizing

$$J(y) = \int_a^b y(x)dx \text{ s.t. } y(a) = y(b) = 0 \tag{3.10}$$

$$C_0 = \int_a^b \sqrt{1 + y'(x)^2}dx \tag{3.11}$$

$$\tag{3.12}$$

In this problem, the state space $x \in R^2$, there are no dynamics and the function space y, has infinite dimensions. Figure 3.9 shows the functional space of the random tree and the result of the algorithm, an optimized path capturing the maximum area. The optimum solution is an arc of a circle and the algorithm will eventually converge toward the optimal solution.

3.9 Online resources

We developed a toolset for evaluating and demonstrating the Duplex algorithm. This toolset, along the benchmarks, is released under an open source license and is available online.

The Duplex algorithm, as presented in this chapter, is developed in C++ and released in the Duplex optimization toolbox [136]. We later ported the scripts to C++ for supporting multi-objective stimuli generation, runtime monitoring, eye diagram analysis and test compression [137]. RRT reposi-

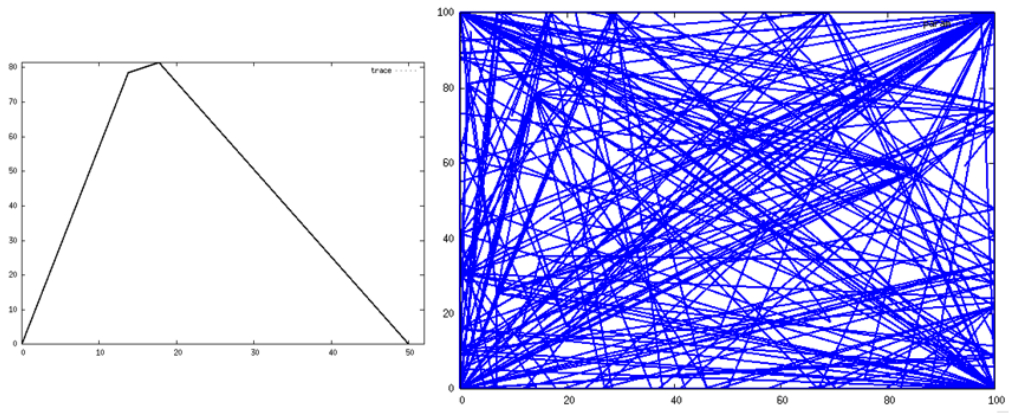


Figure 3.9: The result of the Dido optimization problem using Duplex algorithm.

tory [137] also includes the initial prototype of the directed test generation algorithm that was implemented in MATLAB. The results from Chapter 4, 5, 7, and 8 are partially generated using [137]. We finally created the Duplex optimization toolbox that follows the algorithm as described in this chapter in [136]. The reachability analysis tool was implemented in [138].

The case studies that we used in this thesis are stored in [139]. We implemented the Urbana SAT solver [140] that uses Duplex to solve the boolean satisfiability problem. We implemented the Rapidly-exploring Random Forests algorithm for test generation and reachability analysis [141]. Finally we implemented a game, where we use Duplex to find the Nash equilibrium of capacitated selfish replicated games [142].

3.10 Chapter summary

In this chapter we introduced the Duplex optimization algorithm. We described our contributions over the classic RRT algorithms: i) direction in search, and ii) separation of spaces. Finally, we showed which optimization problems can be solved with the Duplex algorithm and how we can solve them using the Duplex.

CHAPTER 4

DIRECTED INPUT STIMULI GENERATION

4.1 Introduction

4.1.1 Validating analog circuits by simulation

We motivate the reasons to generate input stimuli automatically, as well as the reasons to prefer directed stimuli over random stimuli in analog validation. Our intended use case for this technology is in pre-silicon and post-silicon validation, as a replacement to random Monte Carlo simulations. In current practice, three types of input stimuli are applied to the netlist. The first is generic stimuli from the design house's repository of standard stimuli for the circuit, like applying a sine input to an opamp circuit. The second input stimuli are random Monte Carlo simulations to check for behaviors under different operating conditions. The designer then manually writes test benches designed specifically to check the circuit's corner case behavior and functionality.

In this process, neither the first nor second set of stimuli is capable of exciting corner cases and critical functionality. The most complex part of the verification is done manually. While this was acceptable in the era where analog was relegated to a few standard, non-integrated circuits such as small amplifiers and regulators, etc., such custom crafting of verification artifacts cannot scale to today's systems. Today, analog and mixed signal chips form a majority of modern systems-on-a-chip (SoCs). Automated stimulus generation is therefore a critical need for current and future analog and mixed signal designs.

In current practice, the automated part of the verification is in the second type of stimuli, *i.e.* Monte Carlo simulations [143, 144]. Monte Carlo based methods simulate the circuit using randomly generated inputs. They do not take into account the circuit structure, topology, or state space to target their simulations. On the other hand, *directed simulation* can focus the simulations to expected or known objectives that capture the desired functionality. Objectives can be simple, such as reaching a specific state (say equilibrium), output saturation or safety. Objectives can also be complex behavior-based, like locking, operating regions of active elements, stressing interconnects, etc. Objectives are especially useful during IP integration, where generating stimuli for integrating a netlist into an SoC is a complex problem. For instance, if unsafe regions (e.g. overshooting voltages or excessive current through the IO) in the interface between the SoC and analog netlist are set as goal objectives, an input stimulus can be generated to check for erroneous behavior. This is an increasingly common practical scenario. In the absence of objective based directed inputs, random stimuli may not even reach the desired objectives within acceptable time and resource limits. We believe that this significant chasm in the analog and mixed signal verification process can be bridged by introducing directed stimulus generation. In order for analog verification to scale to the designs of the future with numerous and complex analog components, directed simulations are critically important. To calibrate, in digital circuits, directed input stimuli form the majority of the pre-silicon verification process. Random stimuli are introduced much later for simulating unexpected or corner case scenarios, and aborted after a pre-decided number of cycles.

4.1.2 Generating directed input stimuli using Duplex

We use the Duplex algorithm for *automatic directed input stimulus generation* for validating nonlinear analog circuits. Ours is a simulation based approach that can be used to exercise interesting or relevant behaviors of the circuit in a targeted manner. *Goals* such as operating modes of active components, stable operating states, failure regions, stressing interconnects, equilibrium states, and other relevant behavior can be triggered using our approach. These goals can be user specified or automatically inferred by our algorithm. Input stimuli that can reach these goal regions are then generated. Coverage goals can also be specified in our algorithm, such that the generated input stimuli can meet them. We define *coverage* as uniformity of the visited states in the reachable state space.

We formulate the directed input stimuli generation as a type-I duplex optimization problem. In type-I problems, Duplex grows a single random tree in the state space of the circuit. The Duplex can be manipulated to provide *local direction* concerning which state the simulation should branch from next, as well as *global direction* concerning which region in the state space the simulation should be directed. The Duplex tracks the states it has visited so far by maintaining a tree data structure that it updates every iteration. The classic RRT grows the tree structure by sampling states from a uniform distribution over the state space. In [145], we augmented the RRT with a time dimension, to be able to generate time-variant transient input stimuli. For this work, we will use these time-augmented RRTs.

In [88], we introduced goal-orientedness in analog input stimulus generation. In this work, we have developed that idea further into a directed input stimulus generation methodology that can simultaneously optimize for goals as well as *coverage*. We also propose in this work, *Multi-Objective RRTs (MORRTs)*, which introduce a biasing and feedback loop into the regular RRTs, to bias the growth of the RRTs towards a goal and/or a coverage objective. Traditional RRTs simulate the next state by sampling from a default uniform distribution of states. With MORRTs, we provide alternate distributions for the RRT to sample from. These alternate distributions are biased in favor of goal regions and/or increased coverage. While in [88] we used a clustering algorithm, in this work, we infer these goal and coverage distributions automatically using variational Bayesian inference (VBI)[4], a statistical inferencing algorithm. The VBI algorithm infers a goal distribution from an initial learning phase where it samples states from the user defined or frequently occurring states. It infers a coverage distribution by analyzing the state space distribution of the previously visited states of the MORRT. This provides an integrated methodology to generate high coverage input stimulus directed towards goal regions.

4.1.3 Benefits of using Duplex methodology

We demonstrate that the MORRT algorithm is able to generate tests in goal regions significantly better than Monte Carlo. It takes the random Monte Carlo simulations 199× more iterations and 188× more time to reach the goal regions, as compared to our directed approach. The time overhead incurred in every iteration is higher for the MORRT than Monte Carlo, but we show that it is no greater than 22% in our experimental results. The computational overhead in the MORRT is because of i) inferring the goal distribution, and ii) searching for the closest node to the desired goals. The MORRT stores context through a data structure that represents the visited state space. The memory overhead as a result of maintaining this data structure is not significant.

We present extensive and detailed experimental results on several circuits. We used a Josephson junction circuit, an op-amp and a high-speed VCO circuit. The op-amp is an 8-dimensional CMOS circuit. The VCO netlist is extracted from the post-layout circuit. We demonstrate that our learning strategy with VBI is able to identify the goal regions effectively. We also show that the input stimuli generated by the MORRT algorithm are more efficient than the traditional RRT in achieving objectives. For the Josephson junction circuit, we obtained several stimuli cases that drove the circuit into undesirable states. Such undesirable behavior is known to be hard to detect using conventional test-generation methods [146]. We also demonstrate that our tests can validate correct behavior as well as reveal anomalous behavior. Finally, we quantify the coverage and goal-orientedness of MORRT in terms of the *star discrepancy metric* used by [147].

In [88] we used the traditional RRT algorithm for generating goal-oriented input stimuli for nonlinear analog circuits. We used a grid-based clustering algorithm to identify the goal regions and biased the growth of the RRT toward those regions. Our contributions over [88] are as follows. In this work, we introduce coverage as an objective for input stimulus generation along with goals. We introduce the MORRT algorithm that can generate tests with respect to both high coverage and goal-orientedness. We introduce a biasing technique based on a feedback loop in the MORRT algorithm to favor its growth towards a desirable part of the state space. We use the variational Bayesian inference algorithm to infer the goal distribution and coverage distributions. We introduce a parameter that provides a knob between the goal-orientedness and high coverage simultaneously. We provide extensive experimental results including a CMOS circuit (over [88]) that show the efficacy, efficiency and scale of the MORRT.

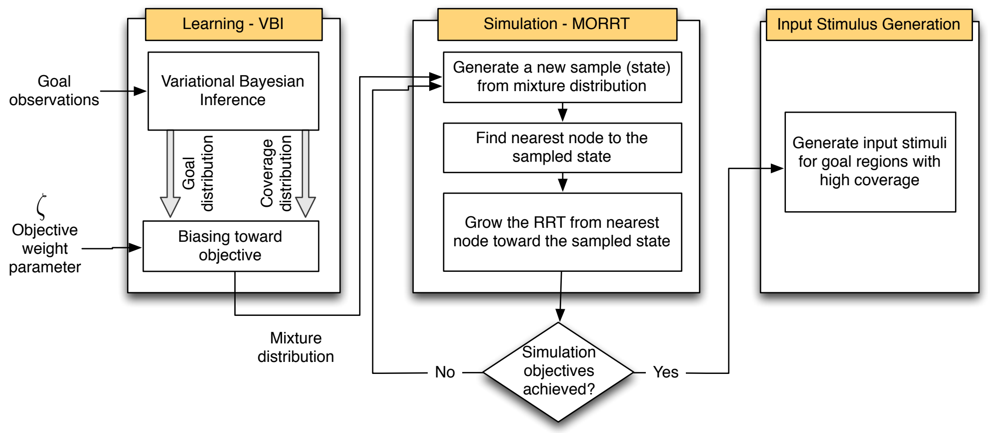

Learning - VBI
Simulation - MORRT
Input Stimulus Generation
Goal observations
Variational Bayesian Inference
Goal distribution
Coverage distribution
ζ
Objective weight parameter
Biasing toward objective
Mixture distribution
Generate a new sample (state) from mixture distribution
Find nearest node to the sampled state
Grow the RRT from nearest node toward the sampled state
Simulation objectives achieved?
No
Yes
Generate input stimuli for goal regions with high coverage

Figure 4.1: Framework of our directed input stimulus generation technique (Section 4.2)

4.1.4 Chapter organization

The rest of this chapter is organized as follows. In Section 4.2 we overview the Duplex methodology for directed input stimuli generation. We describe the Duplex algorithm in Section 4.3. We show the experimental results in Section 4.4.

4.2 Framework of our automated directed input stimulus generation algorithm

The input to our algorithm is an analog circuit netlist (MATLAB or HSPICE netlist). We determine the state space of the circuit by converting it to an ODE [32]. The output of our algorithm is a set of input stimuli. Each input stimulus is a piecewise linear input waveform signal that is assigned to each transient input source in the netlist such as current, voltage and other transient variable sources that are inputs to the circuit (Section 4.3.4, Figure 4.3).

Figure 4.1 shows an overview of our automated directed input stimulus generation framework. There are three key components: i) learning, ii) simulation, and iii) input stimulus generation. The purpose of the learning component is to infer goal distributions and coverage distributions that the MORRT simulation phase can sample from. This phase provides the bias for the subsequent MORRT growth. We use the variational Bayesian inference algorithm (Section 2.3) to infer the goal distribution and coverage distributions. Goals can either provided by the user or automatically generated by our learning algorithm. VBI infers a goal distribution from set of training states. The training states are generated from frequently occurring regions in the state space. To determine coverage distribution, we employ the VBI algorithm in a non-standard way (Section 4.4.3. We exploit the fact that the MORRT maintains a data structure to keep track of visited states, and *feedback* the visited states to the VBI algorithm. The algorithm then infers the distribution that will bias the sampling in favor of higher coverage of the state space.

The output of the learning phase is a *mixture distribution* that combines both the goal and coverage distributions according to ζ, a weight factor. ζ can be dialed up or down by the user to reflect the extent to which he wants goal orientation and/or high coverage in the generated tests. The purpose of the simulation component is to simulate the MORRT. The MORRT is a random tree grown in the state space of the circuit. In each iteration, the next state to be simulated (node of the tree) is generated from the mixture distribution. The MORRT then finds the node of the tree nearest to the newly sampled state and simulates an optimum trajectory path from that node to the new state (Section 4.3.3). If the goal regions are reached, or the coverage goal is reached during simulation, we invoke the final component.

The purpose of the input stimulus generation phase is to extract a test from the MORRT simulations at a given state. We extract the path from the initial state (the root of the MORRT) towards the goal region (the leaf of the MORRT) by traversing the tree (Section 4.3.4).

4.3 Proposed directed input stimulus generation algorithm: Multi-Objective RRT

The details of our Multi-Objective RRT algorithm are explained in Algorithm 4. The MORRT algorithm generates biased states from a distribution $\mathbb{M}$ which is a mixture of two distributions: the goal distribution $\mathbb{G}$ and the coverage distribution $\mathbb{H}$. The mixture distribution $\mathbb{M}$ is defined as:

$$\mathbb{M} = (1 - \zeta) \times \mathbb{H} + \zeta \times \mathbb{G} \tag{4.1}$$

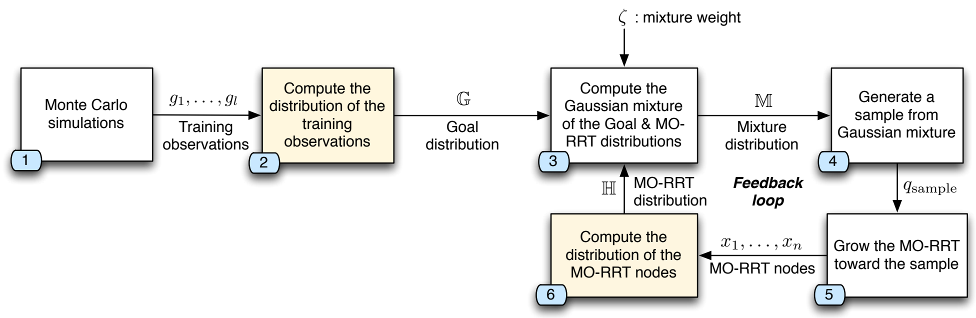


Figure 4.2: Detailed block diagram of the learning phase of the Multi-Objective RRT algorithm. First, we identify the goal distribution (block 1 and 2). We grow the MORRT by sampling states from the mixture distribution. We feed the MORRT states back to the learning algorithm to update the mixture distribution (blocks 6 and 3). Shaded regions corresponds to the VBI algorithm.

where $\mathbb{G}$, $\mathbb{H}$ and $\mathbb{M}$ are the CDF of the goal, coverage and MORRT sampling distributions. The primary input to the algorithm is a mixture weight parameter ζ such that $0 \leq \zeta \leq 1$, which tunes the algorithm between the two objectives. A higher ζ causes more states to be generated from the goal distribution $\mathbb{G}$. Higher ζ biases our algorithm to be more directed toward the goal-oriented traces. On the other hand, a lower ζ generates more states from coverage distribution $\mathbb{H}$ and increases the coverage of our algorithm in the reachable state space. The other inputs to the algorithm are the state space $\mathbb{S}$, the input space $\mathbb{U}$, and the initial condition $\mathbf{x}(0)$ of the circuit. The outputs of the algorithm are the MORRT data structure and a set of input stimuli that drive the circuit from the given initial conditions ($\mathbf{x}(0)$) to the goal region.

For simplicity, first we explain the case where $\zeta = 1$ and the algorithm is purely goal-oriented, as in [88]. In this case, the goal states $\{g_1, \ldots, g_l\}$ are provided by the users. If the user does not know the goal region, we generate a few training states using a uniform distribution over the entire state space. We simulate the training states and record the terminating states as goal states $\{g_1, \ldots, g_l\}$ (Algorithm 4, line 6). We determine the Gaussian mixture distribution of the goal states $\mathbb{G}$ using the VBI algorithm (Algorithm 4, line 7).

In the simulation phase (Algorithm 4, lines $10 - 17$), we grow the MORRT in the state space. We draw states from the Gaussian mixture distribution $\mathbb{M}$. Since $\zeta = 1$, the algorithm is purely goal-oriented, and $\mathbb{M} = \mathbb{G}$. Much as in

Algorithm 4 Multi-Objective RRT algorithm

```
1: $\zeta$: Mixture weight parameter
2: Goal distribution $\mathbb{G}$: Mixture Gaussian distribution of the goal states
3: Coverage distribution $\mathbb{H}$: Mixture Gaussian distribution of the visited
   states in MORRT
4: Sampling distribution $\mathbb{M}$: Mixture Gaussian distribution of the goal ($\mathbb{G}$)
   and coverage ($\mathbb{H}$) distributions.
5: $MAX - ITER$: Maximum iteration of the algorithm
6: $\{g_1 \dots g_l\}$ = training observations
7: $\mathbb{G} \leftarrow$ Variational Bayesian inference $(g_i)$
8: $\mathbb{M} \leftarrow \mathbb{G}$
9: $\mathbb{RRT}$.init $(\mathbf{x}(0))$
10: for $i = 1 \rightarrow MAX - ITER$ do
11:     $q_{goal} \leftarrow$ Generate a random state from $\mathbb{M}$
12:     $q_{near} \leftarrow$ Find nearest node in the MORRT $(\mathbb{S}, q_{goal})$
13:     $q_{new} \leftarrow$ Find optimum trajectory $(q_{near}, q_{goal})$
14:     $\mathbb{RRT}$.expand $(q_{new})$
15:     $\mathbb{H} \leftarrow$ Variational Bayesian inference (MORRT)
16:     $\mathbb{M} \leftarrow$ Gaussian mixture distribution $(\mathbb{G}, \mathbb{H}, \zeta)$
17: end for
18: return input stimuli from MORRT
```

the classic RRT algorithm (Algorithm 1), we grow the MORRT by finding the node nearest to the sampled state and then finding the optimum trajectory from that node toward the sampled state. After the fixed number of iterations $MAX - ITER$, we exit the exploration phase, and then we generate input stimuli from the MORRT (Algorithm 4, line 18). We generate stimuli for the circuit by analyzing the MORRT data structure. Each stimulus can be used to drive the circuit from a given initial state to the goal region that we identified in the state space (Section 4.3.4).

Now, if $\zeta < 1$, we have to balance the goal objective with coverage. The sampling distribution $\mathbb{M}$ is a mixture Gaussian distribution of the distribution of the goal states and the MORRT states. The mixture weight in distribution $\mathbb{M}$ is proportional to $\zeta\mathbb{G}$ and $(1-\zeta)\mathbb{H}$. To compute the sampling distribution $\mathbb{M}$, we perform the learning phase to identify goal distribution $\mathbb{G}$, and then we let $\mathbb{M} = \mathbb{G}$. Initially, we let $\mathbb{M} = \mathbb{G}$ since the MORRT does not yet exist (Algorithm 4, line 8). As the algorithm iterates, the MORRT grows to explore the reachable state space. We update $\mathbb{M}$ at every iteration using the feedback from the states visited thus far by the MORRT (Algorithm 4, line 16).

Each state $x_1, \ldots, x_n$ in the MORRT is a visited state that is reachable from the initial state. At each iteration, we update the distribution of the MORRT states $\mathbb{H}$ using the VBI algorithm (Algorithm 4, line 15). After updating the distribution $\mathbb{H}$, we update the sampling distribution $\mathbb{M}$ based on the mixture weight ζ. If ζ is closer to 1, it means that $\mathbb{M}$ will be closer to $\mathbb{G}$, making the algorithm more goal-oriented. A low ζ means that $\mathbb{M}$ will be closer to the distribution $\mathbb{H}$, making the algorithm generate more states in the already-visited regions of the state space to increase the coverage.

4.3.1 Inferring the goal and coverage distributions

We utilize the *variational Bayesian inference (VBI)* (Section 2.3) [4] algorithm to determine the distribution of the goal states and visited states (MORRT states) in the learning phase of the MORRT algorithm (Figure 4.2). In the learning phase, we compute the distribution of the goal states from the training data $\{g_1, \ldots, g_l\}$ (Figure 4.2, block 2). As the MORRT algorithm explores the state space, we compute the distribution of the reached state space from the MORRT nodes $\{x_1, \ldots, x_n\}$ (Figure 4.2, block 4).

Identifying the distribution of goal states $\mathbb{G}$

We infer the goal distribution from the goal states. The goal states are states from the goal region. The goal observations can be directly provided by the user. In case the user already knows the goal region in the state space, he can manually generate states around that goal region and use them as goal observations. The goal state does not have to be a solution of the circuit or even reachable. MORRT will find an input stimulus that drives the circuit from the initial state toward those goal states. In practice, occasionally the user is unable to provide the goal states or generating goal states might be labor-intensive. In case that the user does not provide the goal observations, our algorithm samples the state space of the circuit by performing a limited number of training simulations. In each simulation, we generate a random initial state as well as the duration of the simulation from a uniform distribution. At the termination of each simulation, we record the final state of the simulation. We refer to these terminating simulation states as *goal states*. We use the concentration of the goal states as a measure of the importance of a region. We assume that the distribution of the goal observations is Gaussian around the goal region.

The VBI algorithm will determine the optimum number of goal regions that provides the best explanation of the goal states by optimizing the mixing coefficient (Equation 2.5). We have no prior information about the distribution of the goal states. We set the mean to the mean of goal states and set the variance to 1, as the initial values to the algorithm. Distribution of the goal states is computed only once, and after the learning phase it remains constant. The VBI algorithm computes the mean and variance of the distribution. On convergence, the output of the algorithm consists of four parameters: $\boldsymbol{\mu}_{\mathbb{G}}, \boldsymbol{\Lambda}_{\mathbb{G}}, \boldsymbol{\pi}_{\mathbb{G}}$, and number of components, $K_{\mathbb{G}}$. We compute the mixture Gaussian distribution as a goal distribution using Equation 2.5:

$$\mathbb{G}(\boldsymbol{\mu}_{\mathbb{G}}, \boldsymbol{\Lambda}_{\mathbb{G}}) = \sum_{i=1}^{K_{\mathbb{G}}} \boldsymbol{\pi}_{\mathbb{G}_i} \, \mathcal{N}(\boldsymbol{\mu}_{\mathbb{G}_i}, \boldsymbol{\Lambda}_{\mathbb{G}_i}^{-1}) \tag{4.2}$$

Infer the coverage distribution $\mathbb{H}$

As the MORRT algorithm explores the reachable state space, it provides a very accurate snapshot of the reachable state space. Each MORRT node is a visited state in the state space that is reachable from the initial state. After adding each new state to the MORRT, we compute the mean and variance of the distribution of the MORRT states. After the VBI algorithm converges, the output of the algorithm consists of four parameters, $\boldsymbol{\mu}_{\mathbb{H}}, \boldsymbol{\Lambda}_{\mathbb{H}}$, and $\boldsymbol{\pi}_{\mathbb{H}}$, and the number of components, $K_{\mathbb{H}}$. We compute the mixture Gaussian distribution as a coverage distribution using (Equation 2.5):

$$\mathbb{H}(\boldsymbol{\mu}_{\mathbb{H}}, \boldsymbol{\Lambda}_{\mathbb{H}}, \boldsymbol{\pi}_{\mathbb{H}}) = \sum_{i=1}^{K_{\mathbb{H}}} \boldsymbol{\pi}_{\mathbb{H}_i} \, \mathcal{N}(\boldsymbol{\mu}_{\mathbb{H}_i}, \boldsymbol{\Lambda}_{\mathbb{H}_i}^{-1}) \tag{4.3}$$

Unlike the goal distribution, $\mathbb{G}$, that is fixed throughout the growth of the MORRT, at each iteration the coverage distribution, $\mathbb{H}$, evolves as the MORRT explores the state space. Therefore the number of components and the mixture distribution itself are dynamic, whereas the goal distribution is static and does not change throughout the algorithm. Mixture distribution $\mathbb{M}$ gets updated because of $\mathbb{H}$ at every iteration.

4.3.2 Biasing towards objectives

The most important part of the MORRT algorithm is generation of the biased states (Figure 4.2 - block 3). In our algorithm, we bias the states toward the distribution of the goal region or the reachable space according to the parameter ζ. The goal distribution and the coverage distribution are the mixture Gaussian distribution $\mathbb{G}$ and the $\mathbb{H}$ determined by the VBI algorithm (Equations 4.2 and 4.3). The biased mixture distribution is a mixture of the goal and coverage distributions proportional to the weight parameter ζ:

$$\mathbb{M}(\mathbf{x}) = (1-\zeta) \times \mathbb{H}(\boldsymbol{\mu}_{\mathbb{H}}, \boldsymbol{\Lambda}_{\mathbb{H}}, \boldsymbol{\pi}_{\mathbb{H}}) + \zeta \times \mathbb{G}(\boldsymbol{\mu}_{\mathbb{G}}, \boldsymbol{\Lambda}_{\mathbb{G}}, \boldsymbol{\pi}_{\mathbb{G}}) \tag{4.4}$$

As a result of our weight mixture, if the user specifies $\zeta = 0$, then the biased distribution $\mathbb{M}$ is the same as the coverage distribution, and our algorithm is completely coverage-driven. Similarly, if the user specifies $\zeta = 1$, then the biased distribution $\mathbb{M}$ is equal to the goal distribution $\mathbb{G}$, and our algorithm is completely goal-oriented. Our algorithm will mix the goal distribution and coverage distribution according to the weight mixture ζ. We study different choices of ζ in the experimental results section. We consider only finite mixture models. Note that although G is independent of ζ, H is dependent on ζ. However, since ζ is constant throughout the algorithm, we ignore it in the inference of the coverage distribution.

4.3.3 MO-RRT based circuit simulation

We grow the MORRT in the simulation phase of the algorithm to explore the state space. At every iteration, we generate a state from the mixture distribution of the goal and coverage distributions. We find the nearest node in the MORRT from the sampled state. The nearest node is computed according to the Euclidean distance between nodes. Next, we determine the optimum trajectory from the nearest node towards the sampled state. Each trajectory is an assignment to the circuit's inputs from the nearest node. We randomly sample different inputs and pick the one that takes us closer to the sampled state. Finally we simulate the circuit from the nearest node using the optimum trajectory for the simulation time Δt. We add the result of the simulation as a new node to the MORRT and continue.

Generating a state from the mixture distribution and picking a nearest node provides a global direction in the MORRT. Therefore when more states are generated from the goal distribution, the growth of the MORRT is biased toward the goal regions. Similarly, picking an optimum trajectory gives a local direction to the MORRT.

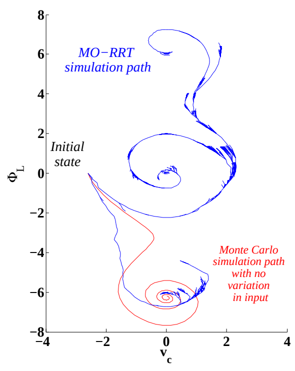
4.3.4 Extracting input stimuli from MORRT

Each leaf in the MORRT corresponds to an input stimulus. The algorithm will record each input $\mathbf{u}(t)$ used in each edge of the MORRT on the edge. For each leaf we can extract the unique input sequence that drives the circuit from the initial state (root of the MORRT) to that leaf. To generate a stimulus, we select the desired circuit states as our targets and choose the appropriate target node (q_{target}) in the tree that is inside our target regions. We extract a (unique) path between the target node and a root of the tree (the initial state). By traversing that path in reverse, we can generate the input sequence $\mathbf{u}(t)$ that would take us from the initial state (q_{root}) to our desired state (q_{target}). An example input stimulus and corresponding trace are shown in Figure 4.5c.

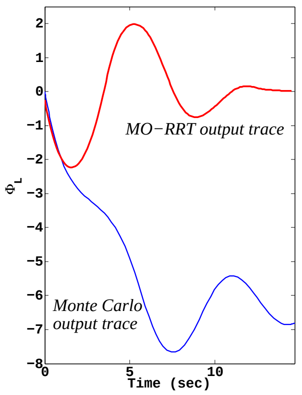
Figure 4.3a shows the state space of the Josephson circuit (explained in Section 4.4.1) where the initial state is selected at state $(-2.6, 0)$. The ideal input drives the circuit to an equilibrium state at $(0, 6)$, which results in the output trace shown in Figure 4.3b. However, if we use Multi-Objective RRT, we can explore alternative goal regions and obtain different results (Figure 4.3a). MORRT can explore other goal regions around equilibrium states $(0, 0)$ and $(0, 6)$. After the MORRT reaches a state in the goal region, we can extract an input stimulus by traversing the path from that state toward the root of the RRT. Finally, we can obtain the input sequences (Figures 4.3c and 4.3d) and report them to the user as input stimuli.

4.4 Experimental results

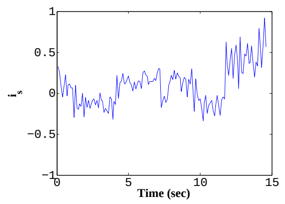
We developed a prototype tool to evaluate the accuracy and efficiency of our algorithm. The tool's input is the circuit netlist in SPICE format. We perform the modified nodal analysis to obtain the DAE model (Equation 2.1) for the netlist. Next, we construct the MORRT data structure and grow the



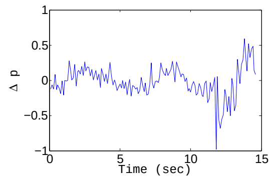
(a) The state space of the Josephson circuit. The MORRT algorithm explores very different paths from the nominal simulation. Our algorithm is able to excite behaviors of the circuit that are normally very hard to test.



(b) An output trace w.r.t. time computed using RRT is very different from the nominal simulation. Our algorithm is able to find an input stimulus that shows the circuit not functioning.



(c) Input sequence $i_s(t)$ extracted from the output trace.



(d) Input sequence $\Delta p(t)$ extracted from the output trace.

Figure 4.3: An example of the input stimuli generated for the Josephson circuit (Section 4.4.1) from the MORRT. Generating input stimulus for Josephson circuit is difficult using conventional Monte Carlo methods.

RRT in the state space of the circuit according to Algorithm 4. We utilize MATLAB's *ode45* and the Synopsys HSPICE numerical ODE/DAE solver to simulate the circuit and obtain the optimum trajectories.

We applied our algorithm to two nonlinear systems: a Josephson junction circuit with complex and nonlinear dynamics (Section 4.4.1), and a CMOS opamp circuit 8-dimensional state space (Section 4.4.4). In the Josephson circuit, we show how we tuned the bias of our algorithm from goal-orientedness to high coverage. We compared our algorithm against Monte Carlo simulation for generating directed tests and coverage criteria. For the op-amp circuit, we showed how our technique can be used in practical situations for generating stress and/or functional tests. Furthermore, we showed that the auto-generated directed tests are shorter and more efficient than manually generated tests.

4.4.1 Effects of ζ on growth of the MORRT

The Josephson junction circuit is shown in Figure 4.4. The Josephson junction is a time-invariant nonlinear inductor governed by Equation 4.5. As

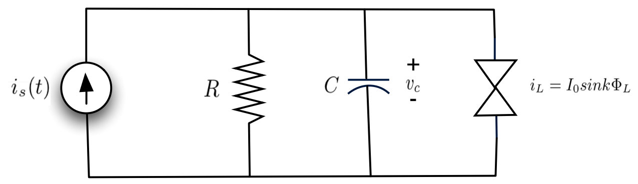


Figure 4.4: Josephson junction circuit.

before, we set $\Delta t = 0.1$. We executed the classic RRT algorithm for 3,000 iterations. We executed the Multi-Objective RRT algorithm for total of 3,000 iterations (including 2700 iterations for growing the RRT and 300 training iterations) for various choices of ζ.

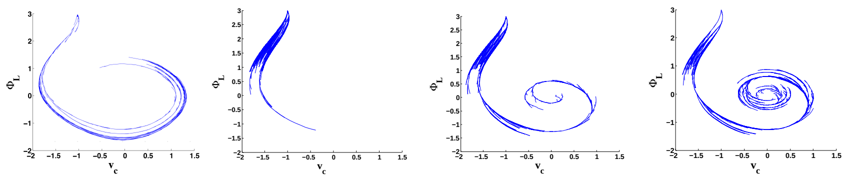
$$i_L = I_0 \times sin(k\Phi_L) \tag{4.5}$$

Therefore, the differential system for the circuit in Figure 4.4 is

$$\begin{aligned} \dot{v}_c &= \frac{1}{C}(\frac{1}{R}v_c - I_0 sin(k\Phi_L) + i_s(t)) && (4.6) \\ \dot{\Phi}_L &= v_c && (4.7) \end{aligned}$$

where $I_0 = 1, R = 4$, and $C = 1$. The inputs of the circuit are the current source ($i_s(t)$) and the variation in $\Phi_L(\Delta_\Phi)$, which are both within the range of $[-0.1, 0.1]$.

Figure 4.5 shows the results of the MORRT algorithm versus the classic RRT algorithm. The initial state of the circuit was chosen at state (-1,3). We identified regions around the state (0,0) as our goal regions; the algorithm thus guided the tree toward the state (0,0) and generated many traces toward that state. As shown in Figure 4.5, in MORRT, the algorithm did not waste any states in the irrelevant regions and quickly converged directly towards the goal state (0,0). On the other hand, in classic RRT, the algorithm spent a lot of its states in uninteresting regions and eventually did not converge toward the goal state (0,0). Figure 4.5 also shows the correlation between the mixture weight parameter ζ and the growth of the MORRT. When ζ was relatively low, the algorithm spent a lot of states inside the reachable state space to increase the coverage. However, as ζ increased, the MORRT grew toward the goal regions. As we show later in Section 4.4.3, we observed that setting $\zeta = 0.5$ yielded the best trade-off between coverage and concentration


(a) Classic RRT. (b) The MORRT algorithm (ζ = 0). (c) The MORRT algorithm (ζ = 0.1). (d) The MORRT algorithm (ζ = 0.3).

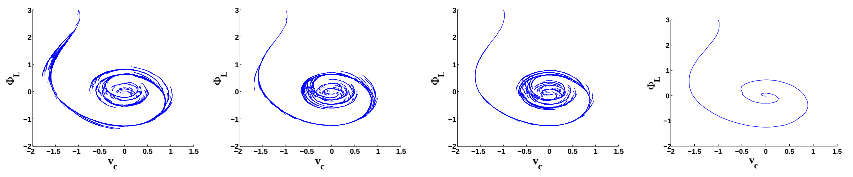

(e) The MORRT algorithm (ζ = 0.5). (f) The MORRT algorithm (ζ = 0.9). (g) The MORRT algorithm (ζ = 1.0). (h) Trace extracted from our goal-oriented algorithm.

Figure 4.5: Exploring the state space of a Josephson junction circuit using the classic RRT and MORRT. Figure 4.5a shows the classic RRT algorithm; for the given number of iterations (3,000), the algorithm did not converge. Figures 4.5b to 4.5g show the various MORRT results for different increasing values of ζ. Finally, Figure 4.5h shows a trace extracted from our algorithm. For the same number of iterations, the MORRT algorithm will converge faster and provide more coverage of the region around the equilibrium state (0,0).

of tests around the goal regions.

4.4.2 Performance comparison of MORRT vs. Monte Carlo

To evaluate the performance of our algorithm against Monte Carlo, we performed two experiments. We used the Josephson case study used in Section 4.4.1. The initial state was selected at state $(-2.6, 0)$ (similar to Section 4.3.4 Figure 4.3a). The goal region was selected within $0.5-$ball around the stable equilibrium state $(0, 0)$. Each Monte Carlo simulation simulates the circuit for $t = 2.5$s and takes 250 iterations. Similar to Monte Carlo, we set $\Delta t = 0.1$ in MORRT algorithm. We performed two experiments:

1. **Experiment I:** We ran both MORRT and M.C. for 3,000 iterations. We reported total execution time and the number of relevant tests in each case.

Table 4.1: Performance comparison of MORRT vs M.C.

Experiment I: Running both algorithm for 3,000 iterations		
	MORRT	Monte Carlo
Number of goal input stimuli	487	0
Total execution time	9.66 sec	7.47 sec

Experiment II: Running both algorithm until finding 30 goal input stimuli		
	MORRT	Monte Carlo
Total iterations	1248	249250
Total execution time	3.57 sec	672.13 sec

2. **Experiment II:** We ran both MORRT and M.C. algorithms until finding 30 goal-oriented test stimuli ending within the $0.5-$ball around the equilibrium state $(0, 0)$. We reported total execution time and total iterations in each case.

Table 4.1 shows the result of the performance comparison. Given the same number of iterations, Monte Carlo was 22.6% faster than our algorithm. However, MC did not find any goal input stimuli, whereas our algorithm was able to generate 487 input stimuli directed toward the goal region. This experiment demonstrates that performance overhead of our algorithm w.r.t. the Monte Carlo simulation is as low as 22.6%.

On the other hand, for generating 30 goal input stimuli, we performed significantly better than the Monte Carlo. It takes the random Monte Carlo simulations 199× more iterations and 188× more time to generate 30 goal input stimuli, as compared to our directed approach.

4.4.3 Measuring coverage and goal-orientedness

We used a quantitative approach to measure coverage and goal-orientedness to evaluate our algorithm. For goal-orientedness, we measured the number of traces generated in the vicinity of the goal region. For coverage, we used a discrepancy metric to measure how much the visited states were equidistributed in the reachable state space.

To evaluate the coverage of the MORRT algorithm, we measured the discrepancy of the nodes in the MORRT. We used the star discrepancy [147, 42] to compute the discrepancy. The star discrepancy has previously been used by [42] to guide and bias the RRT algorithm. In analog circuits, classic measures of computing coverage, such as branch coverage or path coverage, are not applicable, because of the continuous dynamics of the analog circuits. On the other hand, geometric discrepancy measures can express how well the states are equi-distributed in the reachable state space or around the goal regions. The star discrepancy measures the uniformity of the distribution of a state within a region. We relate a high coverage with uniformity of a state's distribution inside the reachable state space.

Let P denote the state set $\{x_1, \ldots, x_n\}$ inside the k-dimensional unit cube $B = [0, 1)^k$. Let $\mathbb{I}^*$ be the class of all k-dimensional sub-intervals I of B of the form $I = \prod_{i=1}^{k}[0, \beta_i]$ such that $0 \leq \beta_i \leq 1$. The local discrepancy is defined as

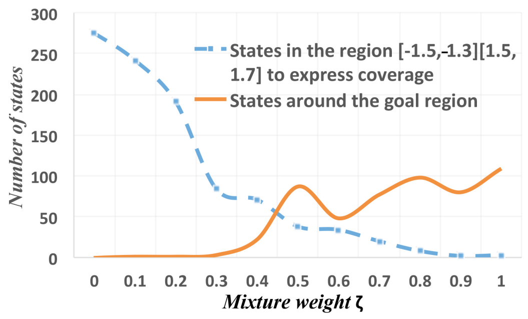
$$D(P, I) \equiv \left|\frac{A(P, I)}{k} - Vol(I)\right| \tag{4.8}$$

where $A(P, I)$ is the number of states of P that are inside I and $Vol(I)$ is the volume of the sub-interval I. The star discrepancy is the supremum of all local discrepancies. The star discrepancy [147, 42] of the state set P in the box B is defined as

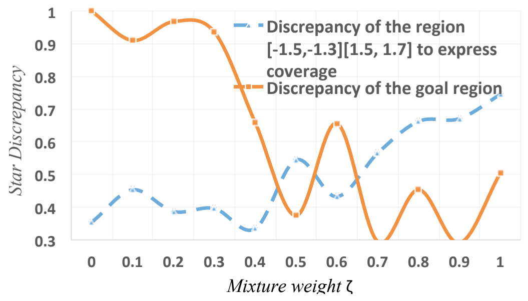
$$D^*(P, B) \equiv \sup_{i \in I} D(P, i) \tag{4.9}$$

The term *star* reflects the fact that every sub-interval in $\mathbb{I}^*$ has a vertex at the origin.

The range of the star discrepancy is the set $(0, 1]$, where low discrepancy means a more uniform set and a higher discrepancy indicates greater nonuniformity. In general, generating a low-discrepancy random sequence is very difficult. We estimated the coverage of the MORRT algorithm with respect to the mixture weight ζ. We used the results from the Josephson junction circuit. To estimate the coverage, we set the box B equal to the interval $[-1.5, -1.3] \times [1.5, 1.7]$ and we computed the star discrepancy of RRT states inside B. To measure goal-orientedness, we set the box G at $[-0.1, 0.1] \times [-0.1, 0.1]$ (the goal region), and we counted the number of MORRT nodes inside B. Figure 4.6 shows the discrepancy and goal-orientedness results of the MORRT algorithm. Figure 4.6a gives the goal-orientedness of our algorithm w.r.t. parameter ζ. In the MORRT, we can extract one trace for each state in the goal region. As shown in Figure 4.6a, as



(a) Number of goal traces with respect to ζ.



(b) Star discrepancy with respect to ζ.

Figure 4.6: Effects of ζ on discrepancy and number of states in MORRT.

we increase the ζ, we bias the growth of the MORRT toward the goal region (in this case, the origin state). The results in Figure 4.6a are confirmed by the results in Figure 4.5. The increase in the number of states around the origin indicates that our algorithm is more goal-oriented as ζ increases. Figure 4.6a clearly shows the correlation between the mixture weight ζ and the number of goal traces. Figure 4.6b illustrates the coverage of our algorithm w.r.t. parameter ζ. As the figure clearly shows, increasing ζ will cause the discrepancy in the box $[-1.5, -1.3] \times [1.5, 1.7]$ to increase. Increased discrepancy indicates less uniformity and less coverage. Therefore, when we use a lower ζ, we achieve a lower discrepancy (in the range of $[0.3, 0.5]$). Moreover, Figure 4.6b shows that if ζ is increased (i.e., our algorithm is more goal-oriented), the MORRT will be grown toward the goal region, and we will have more states inside the goal region. Therefore, our generated input stimuli will be more goal-oriented. Based on Figure 4.6, we recommend an optimum value for ζ that yields an acceptable degree of coverage and goal-orientedness. Our general recommendation is that ζ be set to 0.5. However, depending on the application, the user can choose a higher or lower value for ζ to customize the algorithm.

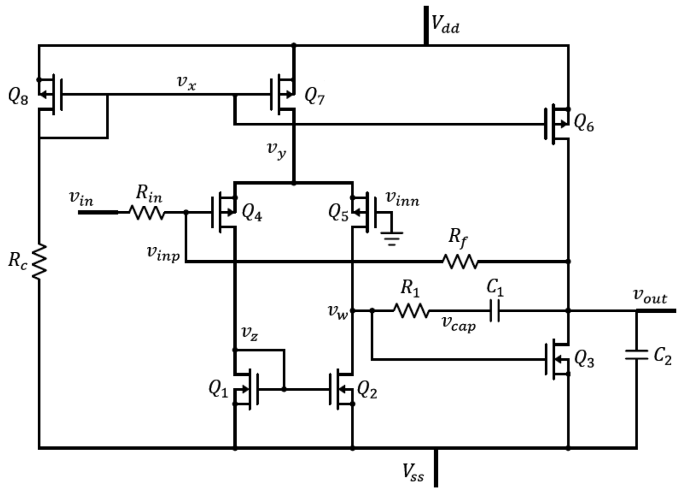


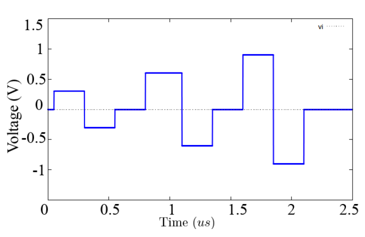
Figure 4.7: Schematic of the opamp circuit.

4.4.4 Generating input stimuli for CMOS operational amplifier circuit

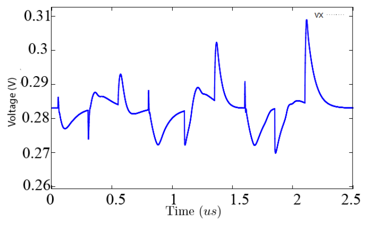
In this section, we demonstrate how MO-RRTs is used in practical situations. We used a 2-stage CMOS operational amplifier integrator circuit, as shown in Figure 4.7, to show practicality and scalability of our algorithm. We used this opamp in a voltage divider configuration with unity gain. The opamp was designed in 0.18 μm library. We sat $VDD = -VSS = 0.9$V. Each test was applied to the V_{in} signal. The output of the opamp was saturating at 0.2V and -0.8V, respectively.

We used Synopsys HSPICE to simulate the circuit. The input to our tool was the op-amp netlist in HSPICE format. The output of the tool was a PWL signal that could be used as an input stimuli to the V_{in} voltage source. The state space consists of the following variables: $\{v_{cap}, v_{dd}, v_i, v_{inn}, v_{inp}, v_o, v_{ss}, v_w, v_x, v_y, v_z, v_w, t\} \in R^{12,1}$ (Figure 4.7). We ran each experiment for 10,000 iterations. Each iteration consisted of a small HSPICE simulation for duration of $dt = 10\mu$s. Each experiment took approximately 15 minutes to complete on a Core-i5 machine equipped with 16GB of memory. In order to analyze the time-variant op-amp circuit, we augmented each state in the MORRT with time notation [145]. The root of was time-annotated with zero. For each other state, the time annotation was the time of the parent node plus the duration of the transient simulation from the parent to the state.

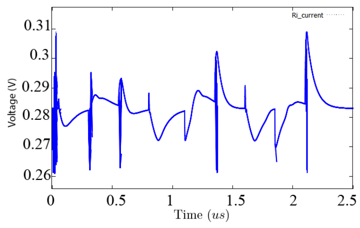
We used our tool to generate three types of input stimuli: i) functional tests, where the objective was to reach a specific region (namely, the saturated



(a) The input signal to generate signal profile



(b) The current profile of the resistor R_1



(c) The MORRT algorithm generates a stress tests for resistor R_1.

Figure 4.8: Generating tests for stressing the resister R_1

outputs) in the state space, ii) stress tests, where the objective was to put as much current or voltage through circuit components or nodes as possible, and iii) combining different input stimuli together.

Generating functional tests

We generated a test that would saturate the output of the opamp circuit. We manually generated a random vector where $v_o = 0.2$V and $v_o = -0.8$V to learn the goal region. It is not required for the training observations to be simulated or even reachable. We used those states as a training observations, effectively bypassing the Monte Carlo simulation step (Figure 4.2 block 1) in our algorithm. Next, we used our learning algorithm to identify the goal region from these states and to compute the goal distribution. Finally, we used the MORRT to generate input stimuli that could saturate the outputs of the circuit at 0.2V and -0.8V. We ran the algorithm twice to reach the output voltage 0.2V and -0.8V. Figure 4.9a shows the output of the circuit when the MORRT was directed toward saturating the output at 0.2V. We extracted multiple input stimuli from the MO-RRT that saturates the output voltage. As long as the goal region is reachable, the MO-RRT algorithm can effectively find an input stimuli that directs the circuit toward the goal region.

We observed that the lengths of the input stimuli generated by MO-RRT are very compact and efficient.

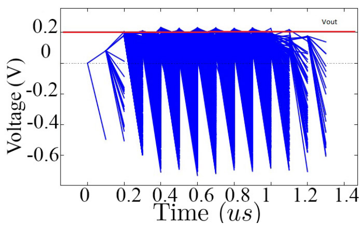
Generating stress tests

We used our algorithm to generate stress tests for different components on the circuit. For stress testing the components, initially we computed a profile for each node in the circuit. We applied the input signal v_{square} as shown in the Figure 4.8a to compute the minimum and maximum value of the current through the resistor. Figure 4.8b shows the voltage v_x and the current through resistor $R_1 = 2.1\text{K}\Omega$. The maximum current through R_1 is determined to be 0.57mA ($= \frac{v_x - v_{ss}}{R_1}$). Next, we applied the MO-RRT algorithm and biased the growth of the tree towards the region with maximum current where $i_{R_1} = 0.6$mA. For the given resistor R_1 in the circuit, the objective of the algorithm was to put as much as 0.6mA current through that resistor. Our algorithm selected the plane $i_{R_1} = 0.6$mA as the goal region and generated tests that would reach that region. Figure 4.8c shows the result. Our algorithm was able to compute several input signals that would stress the resistor R_1 to its maximum allowed current. The automated generated tests requires $4us$ to finish, whereas the manually generated test by the designer requires 200μs to finish. Furthermore, the input stimuli determined by our technique was shorter and more efficient that the profile signal. We repeated the same experiments for all other nodes in the circuit and successfully reached the goal objectives using MO-RRT.

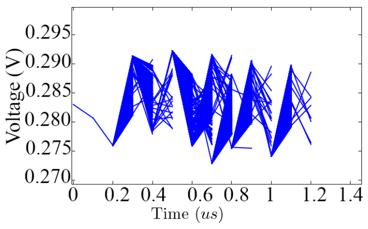
Combining multiple input stimuli

Finally, we used the MO-RRT algorithm to combine different test stimuli. We wanted to find a test that saturates the output voltage v_o at 0.2V and stresses the resistor R_c by maximizing the voltage v_x. Figure 4.9a shows the MORRT G_1 that is used to generate tests for saturating the output voltage. Similarly, we computed another MORRT G_2 that maximizes the voltage v_x and stresses the resistor R_c. The MO-RRT G_1 can be used to generate tests that stresses R_c (and vice versa) but it is not efficient because most of the states do not reach maximum current through R_c.

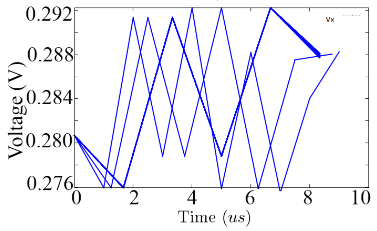
The objective of the experiment was to learn the goal regions from G_1 and



(a) The MO-RRT generates a test that saturates the output at $0.2v$.



(b) The voltage v_x in the MO-RRT of the combined tests for saturating output v_o and stressing resistor R_c.



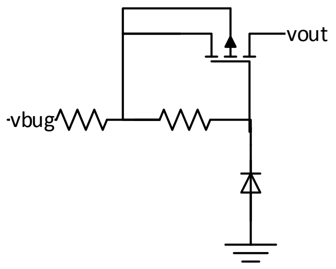
(c) Extracting tests from the MO-RRT that saturates the output and maximizes the current through R_c.

Figure 4.9: Combining different tests. The MO-RRT can learn the goal regions from two given test sets and generate a combined tests that simultaneously reaches both goal regions.

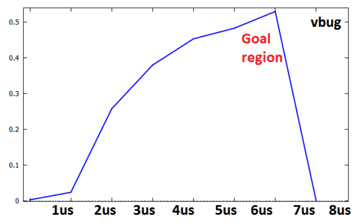
G_2 and generate a new MO-RRT that can simultaneously reach both goal regions. We are only interested in the dimensions of the v_o and v_x. First we collected the terminating states in MO-RRT G_1 and G_2. We generated a new set of learning states from those terminating states where the output voltage was obtained from G_1 and the v_x was obtained from G_2. We computed a new goal distribution from the learning states. The goal distribution is a normal distribution with the mean $(v_o, v_x) = (0.2V, 0.3V)$. We grow the MO-RRT toward the goal region $(0.2V, 0.3V)$. Figure 4.9b shows the MO-RRT toward the combined goal regions. In comparison to the previous result, the MO-RRT for the combined goal region explored both goal regions simultaneously and combined the two test sets. Figure 4.9c shows the extracted tests from the combined MO-RRT that saturates the output and stresses resistor R_c.

Finding design bugs in the opamp

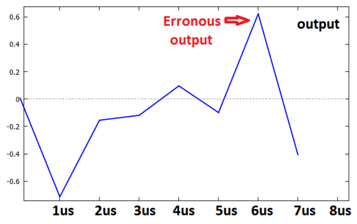
In order to show how MORRT can be used to find bugs in the circuit design, we intentionally introduced a bug into the opamp design. We emulated a



(a) Schematic of the introduced bug in the opamp circuit.



(b) The input stimulus generated by the MORRT that excites the bug in the opamp design.



(c) The erroneous output of the opamp.

Figure 4.10: Combining different tests. The MO-RRT can learn the goal regions from two given test sets and generate a combined test that simultaneously reaches both goal regions.

bug by adding a small voltage limiter subcircuit to the opamp circuit as shown in Figure 4.10a. We connected the output of the voltage limiter to the second differential input of the opamp at v_{inn} node (this node was initially grounded). Using this circuit, when the v_{bug} increases more than 0.5V, the transistor turns on and increases the voltage of the node v_{inn}. There were two inputs to the circuit: the input signal v_{in} and the second faulty input v_{bug}.

To excite the bug, we searched for combination of input signal that resulted a in positive v_{inn} (normally this signal was grounded). We set the $v_{inn} = 0.5$ as the goal objective and executed the MORRT for 1,000 iterations (1 minute). The MORRT algorithm successfully found multiple stimuli ending in the region where $v_{inn} = 0.5$. Each stimulus resulted in the erroneous output in the opamp. Figure 4.10b and 4.10c shows the input stimulus v_{bug} and corresponding output of the opamp, respectively. The spike in the opamp's output was caused by reaching the goal region in the input stimuli.

4.4.5 Generating input stimuli for voltage controlled oscillator circuit

Voltage controlled oscillators (VCO) are widely used in RF circuits, frequency synthesizers and phased-locked loops. We generated input stimuli for a post-layout 1 GHz VCO circuit in TSMC-0.18μm process, shown in Figure 4.11, to validate its functionality and interface. The netlist was extracted from the VCO layout with all the parasitic capacitance and resistors. Figure 4.12a shows the oscillation of the VCO circuit where $V_{\text{bias}} = 750$mV, $V_{control} = 630$mV and $V_{dd} = 1.8$V. It takes 10.8ns for the output to reach its peak-to-peak maximum. The output oscillates between 0.7V and 1.34V.

We defined three transient inputs to the circuit ($0 \leq V_{bias} \leq 1$, $0 \leq V_{control} \leq 1$ and $1.8 \leq V_{dd} \leq 2$) to model transient input and power noise in the circuit. We executed the MORRT for 20,000 iterations (60 minutes). We used the MORRT to generate input stimuli for the following tests:

- Reaching the maximum output voltage (1.34V) as soon as possible without oscillating. This stimulus is useful to check the peak-to-peak swing of the VCO circuit. Figure 4.12b shows the result of the experiment. The MORRT successfully generated stimuli that drive the output voltage to 1.3V.

- Stimuli for cutting off the output voltage. In order to cut off the output, we selected the output at 0V as our goal region and ran the MORRT. Note the $v_{out} = 0$ was not included in the initial output swing of the VCO circuit. The MORRT was able to find many stimuli that minimized the output as low as 0.05V. There are always small leakage current and capacitive charge that prevent the output from becoming 0. The length of the test was 0.65ns.

4.4.6 Intermediate results for our algorithm

We used a tunnel diode circuit [148] to show how our technique works under parameter and input variation. The circuit's behavior was specified by the following differential equations:

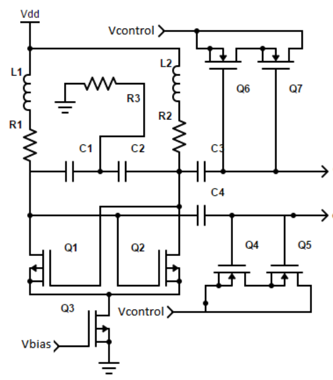


Figure 4.11: Schematic of the VCO circuit.

$$\dot{i}_L = \frac{1}{C}(i_L - h(v_r)) \tag{4.10}$$

$$\dot{v}_r = \frac{1}{L}(-i_d - R \times i_L + E) \tag{4.11}$$

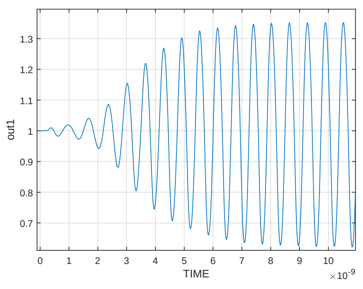
where i_L is the current through the inductor L, and v_r is the voltage across the tunnel diode. In our configuration, the circuit parameters were set to $E = 1.2\text{V}, R = 1.5\text{K}\Omega, C = 2\text{pF}$, and $L = 5\mu\text{H}$ (Figure 4.13). The tunnel diode's behavior was governed by a nonlinear current-voltage relation

$$h(x) = 17.76x - 103.79x^2 + 229.62x^3 - 226.31x^4 + 83.72x^5 \tag{4.12}$$

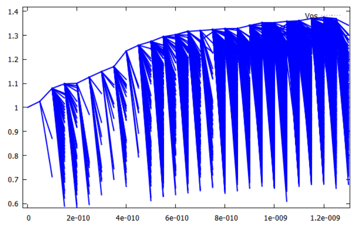
The circuit had three equilibrium points at approximately $(0.063, 0.758)$, $(0.285, 0.61)$, and $(0.884, 0.21)$. Two equilibrium points can easily be identified using the learning step of the algorithm. We selected an unstable equilibrium point $(0.285, 0.61)$ as the initial state and the root of the MORRT.

We set $\Delta t = 0.1$. First, we executed the classic RRT algorithm for 3,000 samples. Then we executed the MORRT algorithm and a total of 3,000 samples (including 2,700 samples for growing the MORRT and 300 training samples) for various choices of ζ. We analyzed the circuit under disturbance input variables for the current throughout the diode (modeled as $I_d = h(V_d) + \Delta p_1$) and voltage variation from the DC source (modeled as $E = U_0 + \Delta p_2$). The range of the disturbances was $(p_1, p_2) \in [-0.1, 0.1] \times [-0.1, 0.1]$.

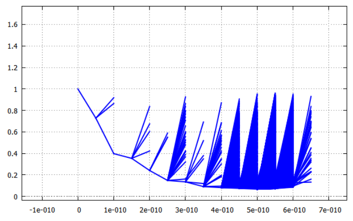
Figures 4.15a and 4.15b show the results of the analysis using both classic RRT and our Multi-Objective RRT algorithm. As shown in the figure, in comparison to the classic RRT algorithm, ours generates more traces ending



(a) The default output of the VCO circuit. The output oscillates between (0.7V,1.3V).



(b) Testing the peak swing of the VCO output using random tree.



(c) Random tree generates a tests to cut-off output.

Figure 4.12: Generating input stimuli for VCO circuit.

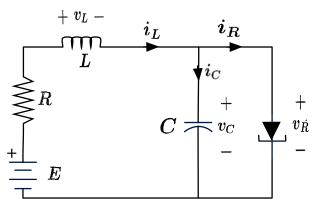
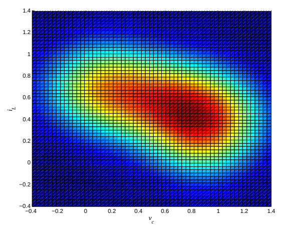


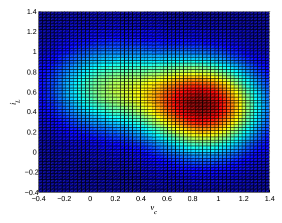
Figure 4.13: Tunnel-diode circuit.

in the target regions (in this case, around two equilibrium points of the tunnel diode circuit); the classic algorithm uses the uniform sampling and would waste a lot of samples to find its path toward equilibrium points. We biased the growth of the MORRT by sampling from the mixture of goal and coverage distributions. Figure 4.14 shows the probability distribution function of the Gaussian mixture of the goal and coverage distributions determined using the VBI algorithm (Section 2.3).

4.4.7 Scaling the MORRT input stimulus generation

To demonstrate the scalability of the MORRT input stimulus generation algorithm, we applied it to the *ring modulator* circuit with 15 dimensions. The ring modulator circuit is used for amplitude modulation or frequency


(a) Mixture distribution with $\zeta = 0.2$.


(b) Mixture distribution with $\zeta = 0.5$.

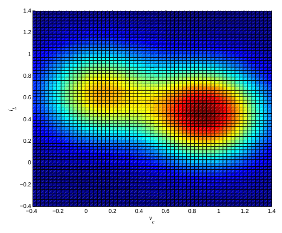
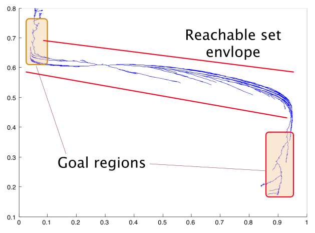
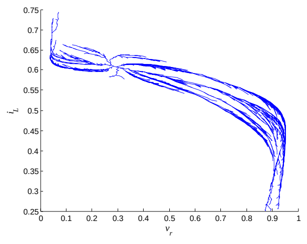

(c) Mixture distribution with $\zeta = 0.9$.

Figure 4.14: Effect of mixture weight ζ on the mixture Gaussian distribution $\mathbb{M}$. The mixture distribution converges toward the distribution of the goal region $\mathbb{G}$ for goal-oriented MORRT with higher ζ. On the other hand, a lower ζ with coverage-driven objective ensures that $\mathbb{M}$ is closer the MORRT distribution $\mathbb{H}$.



(a) Classic RRT.


(b) MORRT ($\zeta = 0.5$).

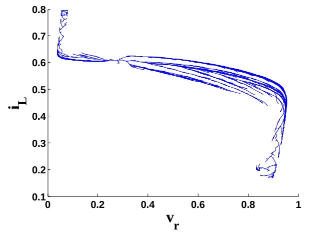

(c) MORRT ($\zeta = 1.0$).

Figure 4.15: Tunnel diode results for classic and MORRT algorithm. While the classic RRT algorithm will generate a lot of samples to find its path towards two stable equilibrium points (the boxed regions), the MORRT algorithm will rapidly converge and will generate more traces in relevant regions and explores more regions of the state space. Moreover, the MORRT will provide a better coverage in the reachable state space (the enveloped region).

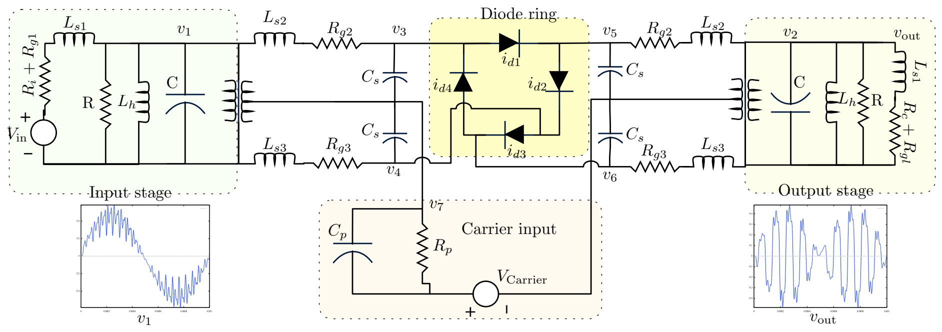


Figure 4.16: Schematic of the ring modulator circuit. The ring modulator consists of four parts: the input stage, the carrier stage, the output stage, and the diode ring. The circuit modulates the input signal V_{in} with carrier signal V_{Carrier}.

mixing. It produces the output signal v_{out} from the low-frequency signal V_{in} multiplied by the signal V_{Carrier} in the time domain. A schematic of the classic ring modulator is shown in Figure 4.16. The main part of the circuit consists of four nonlinear diodes arranged in a clockwise ring configuration. The dynamics of the circuit are governed by a 15-dimensional time-variant ODE function f. The first seven equations describe the voltage relations, whereas the rest of the equations describe the current throughout the nodes. The ODE function f and its parameters are described in [149]. In our implementation, we set $C_s = 1$ nF. The initial state was the point zero. We used the Modified Extended Backward Differentiation ODE/DAE solver MEBDFDAE to simulate the system [150]. MEBDFDAE is a very efficient solver for simulating stiff and nonstiff ODE and DAE systems. Figure 4.16 shows a transient output signal v_{out} and the output of the input stage v_1 of the circuit for nominal inputs without any variations.

The state space of the circuit is $y \in \mathbb{R}^{15}$. The simulation time was $t \in [0, 10^{-3}]$. The inputs to the circuits were the following sinusoidal voltage sources. The variation parameters $\Delta_0 \times \Delta_1$ were uniformly sampled from interval $[-0.1, 0.1] \times [-0.05, 0.05]$.

$$\begin{aligned} V_{in}(t) &= 0.5\sin(2000\pi t) + \Delta_0 \\ V_{\text{Carrier}}(t) &= 2\sin(20000\pi t) + \Delta_1 \end{aligned}$$

The current through each diode in the ring was modeled as

$$i_d = I_S(e^{\frac{V_D}{nV_T}} - 1) = \gamma \times (e^{\delta V_D} - 1) \tag{4.13}$$

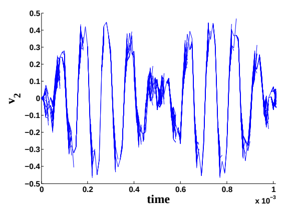
where $\gamma = 40.67286402 \times 10^{-9}$ and $\delta = 17.7493332$ (see [149] for details).

In order to analyze the time-variant inputs V_{in} and V_{Carrier}, we augmented each state y in the MORRT with time notation [145]. The root of the tree had the time annotation zero. For each other state, the time annotation was the time of the parent node plus the duration of the transient simulation from the parent to the state. In our case study, each edge was a transient simulation with duration 10μs. Augmenting MORRT with time dimensions allowed us to model time-variant behaviors, such as oscillation in the state-time space [145].

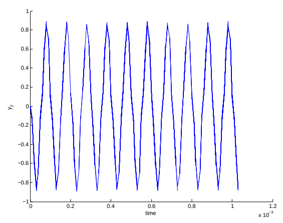
We manually provided the algorithm with 300 training observations that were relevant to the functionality of the ring modulator. Our algorithm inferred the goal distribution (with two components) from the training observation using VBI. We performed an initial exploratory simulation of the state space using the classic RRT for 100 samples. We biased the time dimension in the initial exploration to ensure the transient progress of the RRT [145]. After the initial exploration, we used our Multi-Objective RRT algorithm (with ζ set to 0.5) to explore the state space of the circuit. In this case study, we were particularly interested in the input-output relation of the circuit. So we selected 5 voltages (including v_1 and v_{out}) as the variables for the VBI algorithm and biasing. By choosing only a subset of variables instead of the entire variable set, we reduced the order of the learning algorithm and gained efficiency. Moreover, by removing the uncorrelated signals from the learning phase, we were able to generate more accurate results. For the rest of the variables, we used uniform sampling similar to that of the classic RRT. We emphasize that the MORRT algorithm does not necessarily need to be applied to whole systems, but only to the dimensions of interest from the design perspective.

Figure 4.17 shows the results of our algorithm for the ring modulator circuit. As we mentioned earlier, the ring modulator is a time-variant circuit. Therefore, we augmented each state with a time annotation to preserve timing information. The expected output of the circuit and corresponding input (after the input stage) are shown in the circuit schematic (Figure 4.16). Figure 4.18a shows the expected signal v_1 with respect to signal v_{out}. Figure 4.18b illustrates the state space of the circuit and the state space that we used in our learning algorithm.

After the MORRT algorithm was executed, the signal v_{out} was extracted



(a) The output of the circuit $v_{\text{out}}(t)$ with no disturbance; this signal is the multiplication of the signal $V_{\text{in}}(t)$ by $V_{\text{carrier}}(t)$.

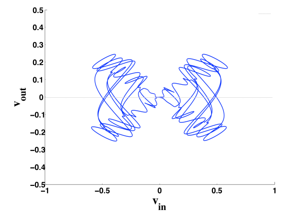


(b) Output of the carrier stage $v_7(t)$ with added perturbation in the MORRT algorithm.

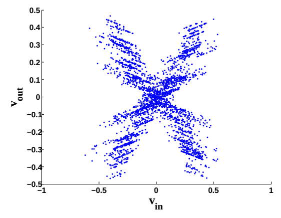
Figure 4.17: The output of the carrier stage is relatively clean according to the specification, because of the RC filter in the design. Therefore, the output perturbation is propagated from the input stage through the diode ring. The cause of the bug in the circuit is the poor input stage filter design.

from the MORRT. There were many divergences from the expected output because of the disturbances on the input signals. Between the two input signals (V_{in} and V_{Carrier}), the perturbation on V_{in} has a greater effect on the circuit outputs because the RC filter (formed by resistor R_p and capacitor C_p) at the output of the carrier stage absorbs most of the high-frequency disturbances that we put on the carrier signal. That observation is supported in Figure 4.17b, which shows the output of the carrier stage; the output is very clean.

Finally, Figure 4.18b shows the scatter plot of the MORRT projected into the v_1 and v_{out} dimensions. During the initial learning phase, the VBI algorithm identified the origin as the goal region. As a result, many of the samples were generated around the origin point. In Figure 4.18b, we removed the edges of the MORRT to demonstrate the concentrations of the samples around the origin. Many of the MORRT nodes were generated around the center region in the state space that was identified as the goal region. The rest of the samples were generated almost uniformly in the reachable state space to improve the coverage of the MORRT. As shown in Figure 4.18a, the rest of the samples were uniformly generated in the reachable state space to improve the coverage of the MORRT.



(a) Projection of signal v_1 with v_{out} with no disturbance to delineate the reachable state space. The circuit is simulated once for 1 ms.



(b) Scatter plot of v_1-v_{out} projection of the circuit with MORRT.

Figure 4.18: Exploring the reachable state space using MORRT. For each leaf node in the MORRT, we can extract the input sequence that will generate that trace. The VBI algorithm inferred the distribution at the origin as the goal region. As a result, most of the traces are focused around the center of the region.

4.5 Chapter summary

In this chapter, we used Duplex for pre-silicon directed input stimulus generation for nonlinear analog circuits. We modeled the input stimuli generation problem as a type-I duplex search problem. We added objectives such as goal-oriented and coverage to the Multi-Objective RRT algorithm. We demonstrated that Duplex is useful for generating goal-oriented and high coverage input stimulus for analog circuits in order to check their functionality during the design process.

CHAPTER 5

RUNTIME MONITORING OF RANDOM TREES

5.1 Introduction

Verifying nonlinear analog circuits is a major challenge and an ongoing topic of intensive research. Formal verification methods exhaustively analyze all possible behaviors of the circuit statically. They present a very daunting computational challenge in the domain of analog circuits. A less rigorous, but more practically viable, alternative is runtime verification and monitoring of properties. In runtime verification, a property monitor checks whether a finite set of simulation traces would satisfy or violate a given property specification [49].

Current approaches for runtime verification are inefficient. Runtime verification has two components: the generation of traces and behavior (simulation) and checking and monitoring of properties against the simulated traces. Existing approaches (See [20]) for runtime verification generate transient traces of the system using Monte Carlo simulation and monitor properties against these traces.

In this chapter, we introduce a technique for runtime verification that is based on the Rapidly Exploring Random Tree (RRT) algorithm [43]. Our technique simulates the state-space of a nonlinear analog circuit in a manner different from Monte Carlo simulations. The RRT is a tree data structure that grows rapidly by performing exploratory simulations of system behavior. We use the incremental nature of RRT growth patterns to monitor properties of interest incrementally. Hence, our RRT-based runtime verification framework provides a novel simulation as well as property-checking and monitoring methodology. Previously, RRTs have been extensively used in robotic motion planning [43] and reasoning [54], safety falsification [39, 151] and test generation [42].

In order to adapt traditional RRTs to runtime monitoring of analog circuits, we introduce the *Time-augmented RRT (TRRT)* algorithm. RRT, being a *tree* data structure, spans the state-space and does not contain loops and cycles. This makes the RRT unsuitable for checking properties like oscillation that need to traverse a cyclic path in the state-space. The RRT explores the entire state-space uniformly without any bias towards a particular dimension. Thus the growth along the time dimension might be very small for a large state-space with many dimensions. We address these issues in TRRT data structure. We augment the state-space of an analog circuit with the time dimension, providing a state-time space for the time-augmented RRT to grow. The state-time space adds the time dimension to the $n-$dimensional state-space vector. In order to ensure forward progress in time, we introduce a sampling bias in the time dimension, without violating the probabilistic completeness property of the time-augmented RRT.

Our runtime verification technique based on TRRTs works as follows. Design specification properties are provided to describe the temporal and logical behavioral model of the analog signals. Given initial state(s) and input parameters of a system (including uncertainty and variation parameters), our algorithm randomly samples the state-time space of the circuit and expands the TRRT toward the new samples through simulation. The TRRT is constructed incrementally starting from the specified initial state. At every incremental iteration, an edge corresponding to a single simulation trace from the previous (initial) state to the next (final) state is added to the tree. As a result, the constructed TRRT consists of many randomized simulation traces as edges. At every iteration, our monitoring algorithm checks the newly added edge against given properties for violation. The properties we check include both analog properties, meaning input/output properties that do not involve previous state information, and temporal properties, i.e., properties that are stateful. We use the STL/PSL properties [49], with a few modifications for TRRT-based checking. We define the operator *Norm* for computing distance the between vectors as well as the jitter property in a manner that is not amendable to Monte Carlo simulation, but is verifiable with TRRTs.

Our technique has many benefits over state-of-the-art approaches for runtime verification. First, ours is more efficient. A major source of inefficiency in Monte Carlo simulations is the overlapping of simulation traces. For a given circuit, a majority of the simulation traces browse the same path in the state-space during runtime. Therefore, they share the same path and overlap with each other. Repeated simulation of the same path in the circuit does not provide any new information, and results in poor performance. However, the TRRT-based method incrementally grows the TRRT in the state-time space. TRRT is persistently aware of the state-space all the time. TRRT, being a tree data structure, always grows toward unique samples in the state-space such that two traces never coincide. In TRRT the direction of the growth is always toward a state as yet unexplored and every simulation trace is unique. At every iteration, the tree traverses and covers more of the state-space. Consequently, TRRT does not allow repeated sampling of the same sequence of nodes and prohibits overlapping of traces in the state-space. Since we conceptually arrange Monte Carlo's linear simulation traces in a tree data structure, we have an average logarithmic efficiency in simulation performance and memory to achieve the same state-space coverage as Monte Carlo.

Second, our monitoring algorithm proceeds incrementally. For most cases, we only have to check the incremented edge to the TRRT to decide whether a property has been violated. Thus, our incremental monitoring algorithm using TRRT is more efficient than monitoring the entire trace [54].

Finally, by using TRRTs, we efficiently verify properties that require a comparison over the entire trace. Such properties include jitter and deviation. Monte Carlo simulations, as they simulate only one trace at a time and possess no knowledge of the state-space, are not well suited to checking those properties. On the other hand, the TRRT can maintain information and verify multiple traces simultaneously because of its step-by-step growing data structure.

We model circuit states and inputs as continuous finite variables without discretizing them. We use SPICE to simulate circuit behavior. Our methodology accurately models the continuous-time behavior of analog circuits.

Our main contributions are as follows.

- We propose an incremental property checking and runtime monitoring algorithm for nonlinear analog circuits that utilizes TRRT to verify design specification properties. Our algorithm is incremental in nature and more efficient than previous strategies for runtime verification.

- We introduce a Time-augmented Rapidly-exploring Random Tree (TRRT) algorithm. TRRT has an augmented time dimension and a biased sampling algorithm in that dimension. TRRT provides the same coverage as Monte Carlo, while utilizing the logarithmic order memory and time. TRRT prohibits simulation trace overlap.

- We define the semantics for our property specification language.

To demonstrate the effectiveness of the proposed approach, we applied our technique in several case studies. We first used our methodology on a tunnel diode circuit as a proof of concept. Then we showed the scalability and practicality of our technique by using it to verify a PLL circuit.

5.2 Analog property specification

We use a property specification language based on STL/PSL to define our properties [49, 152, 50]. We use a subset of the operators defined in STL/PSL, and define a few of our own to suit the RRT verification framework. As in PSL, the analog layer is used to describe the properties of continuous variables and vectors, and the temporal layer is used to reason about the temporal behavior of the circuit.

We define the **Norm** operator for both the analog and temporal layers. We also express the **jitter** property in a manner that is conducive to RRT-based verification. We describe the syntax and semantics of all the operators we use in both layers. The rest of this section describes the syntax and semantics of the analog and temporal layers of properties.

5.2.1 Syntax

Syntax of the analog layer

The grammar for the analog syntax is

$$\phi ::= \texttt{var} | \texttt{const} | f(\phi, \ldots, \phi) \tag{5.1}$$

`var` is a continuous finite variable, and can be a single-dimensional vector, much as x_i denotes to a single waveform in simulation, or can be any subset of the circuit's state vector, like $< x_{i_1}, x_{i_2}, \ldots, x_{i_{n'}} >$, where the index set

$\{i_1, \ldots, i_{n'}\}$ is specified by the user. `const` is a finite constant. Finally, f can be any of the following functions:

- Shift
- Binary operators, including $\{+, -, \times, /\}$[1]
- Norm

The semantics of the above functions are described in Section 5.2.2.

Syntax of the temporal layer

We define the temporal layer to reason about time. Let ϕ be an atomic proposition. For every state (sub)vector $\mathbf{x}$, we associate a time instance of the form $\mathbf{x}(t)$ where $t \in \mathbb{T}$. Set $\mathbb{T}$ is the set of all possible times, and it is defined as $T := \{x | x = k \times \Delta t, k \in \mathbb{Z}^+\}$ where Δt is the minimum discrete time resolution. The time interval is an array of $\mathbf{x}(t)$ with a variable t. Time intervals can be fixed, like $t \in [30, 40]$, or relative, like $t \in [t, t+300]$. In both cases, we write them as $t \in [t_i, t_j]$. Moreover since we monitor the behavior of the system for a finite time interval, temporal modalities are bounded to intervals of the form $[i, j]$, where $0 < i < j \leq T_{max}$, and $i, j \in \mathbb{T}$ where $T_{max} = \sup(\mathbb{T}) = k_{max} \times \Delta t$.

Similar to [49], we define the temporal layer as follows:

$$\begin{aligned} \varphi \;\; = & \;\; p \mid \phi[a:b] \; \star \; \phi[a:b] \mid \texttt{not}\; \varphi \mid \varphi \bullet \varphi \\ \mid & \;\; \texttt{eventually}[a:b]\; \varphi \mid \varphi \; \texttt{until}[a:b] \; \varphi \mid \mathcal{J}[a:b](\varphi) \end{aligned}$$

v is a propositional variable; the comparison operator $\star$ includes $\{\leq, <, \geq, >, \doteq, \not\doteq\}$. The logical operator $\bullet$ includes logical and, or, xor, xnor, nand and nor. The semantics of the above functions and operators are defined in Section 5.2.2.

We use the notation ϕ for analog and φ for temporal formulas.

[1]With the exception that $\frac{\phi_1}{\phi_2}$ is well-defined if and only if $0 \notin \phi_2$.

5.2.2 Semantics

Semantics of analog layer

The semantics of the analog layer are defined as the function of f over the state (sub)vector ϕ. f can be any of the following functions:

- Shift: Shift is defined as changing the index of time dimension of a variable along the same trace on the simulation, i.e., $\texttt{shift}(\phi, \texttt{const})[t] = \phi(t + \texttt{const})$.
- Binary function[2] f: $f(\phi_1, \phi_2)[t] = f(\phi_1[t], \phi_2[t])$.
- **Norm**(ϕ, p), Norm(ϕ_1, ϕ_2): returns the p-norm of ϕ, or the L^2-norm for computing the distance between ϕ_1 and ϕ_2, assuming both propositions have the same dimension. Norm is used to measure the distance of the state-vector against another vector or a constant. L^2-norm is defined as $\texttt{Norm}(x, y) := ||x - y|| = \sqrt{\sum_{j=1}^{n'} (x_{i_j} - y_{i_j})^2}$, and the p-norm, assuming $p \geq 1$, is defined as $\texttt{Norm}(x, p) := ||x||_p = (\sum_{j=1}^{n'} |x_{i_j}|^p)^{\frac{1}{p}}$.

Semantics of the temporal layer

For the temporal layer, we mostly use the same semantics proposed in [49], with some modifications as follows. Comparison operator $\star$ includes $\{\leq, <, \geq, >, \doteq, \not\doteq\}$. To reason about equivalence in the analog domain, we use the notation $x(t) \doteq y(t')$ to indicate that $||x(t) - y(t')|| < \epsilon$. Equivalence operator is satisfied if and only if for any given $t \in [t_i, t_j]$, state $x(t)$ remains within ϵ-envelope around $y(t')$ for any given $t' \in [t'_i, t'_j]$. The following definition easily expands to $\leq, \geq$, and $\not\doteq$.

The logical operator $\bullet$ includes logical and, or, xor, xnor, nand & nor. The semantics of `not` are defined as $\mathbb{M} \models (\texttt{not}\varphi)$ `iff` $\mathbb{M} \not\models \varphi$. Similarly, $\mathbb{M} \models (\varphi_1$ `and` $\varphi_2)$ `iff` $\mathbb{M} \models \varphi_1$ `and` $\mathbb{M} \models \varphi_2$ and so on. $\mathbb{M}$ is the circuit's simulation abstraction provided by the RRT $\mathbb{G}$.

The semantics of *Until* are defined in [49]. The *Eventually* operator can be defined using *Until* as follows. The temporal modalities $\Diamond$ (*eventually*, sometimes in the future) and $\Box$ (*always*, from now on forever) are derived as

[2]For simplicity, we use the notation $\diamond$ for the binary function f. Therefore, $(\phi_1 \diamond \phi_2)[t] = \phi_1[t] \diamond \phi_2[t]$.

follows: $\Diamond\varphi \equiv$ *true* `until` φ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$. By combining the above, we obtain the *infinitely often* $\varphi \equiv \Box\Diamond\varphi$ and the *eventually forever* $\varphi \equiv \Diamond\Box\varphi$.

Jitter is an undesired deviation from true periodicity of an assumed periodic original signal. The basic jitter operator ($\mathcal{J}(x,t)$) will compute the deviation in time using
$\mathcal{J}(f,x,t) = \max\limits_{0 \le t \le T_{max}} (x(t) - x(t-f)) - \min\limits_{0 \le t \le T_{max}} (x(t) - x(t-f))$

For periodic signals, the above definition is equivalent to a maximum deviation of state vector $x(t)$ from other state vectors at the same period (including other states at time $t, t-f, t-2f, \dots$). We use that idea to create a recursive $\mathcal{J}$ operator model for computing jitter in RRT later in Section 5.3.2. We also extend the jitter operator to compute the deviation of non-periodic signals.

5.3 TRRT-based runtime verification algorithm

Our goal is to verify that an analog circuit $\mathbb{M}$ satisfies a property Φ. A summary of the steps (Figure 5.1) in our TRRT-based runtime verification algorithm is as follows:

1. We construct a TRRT $\mathbb{G}$ (Section II.B) to represent a set of feasible traces of the analog circuit $\mathbb{M}$. We construct $\mathbb{G}$ by initializing it to a known operating point of the circuit and then growing it step by step.

2. In each step, we employ a *monitor* to check whether the property Φ is violated by any of the traces represented by the TRRT $\mathbb{G}$.

3. If the monitor does not detect a violation, we grow the TRRT by one more step. We repeat this process iteratively until a violation is detected or a user-specified limit on the number of steps is reached.

In each step of our algorithm, we grow the TRRT $\mathbb{G}$ by adding a single edge to the tree (Section II.B). Each edge in $\mathbb{G}$ corresponds to a single transient circuit simulation of length Δt. For several types of properties, our monitor can detect a violation by checking only the newly added edge of the TRRT. Therefore, for those types of properties, our algorithm is highly efficient since it can perform verification in an incremental manner.

With each step, the TRRT grows. TRRT will quickly explore and provide a high coverage of the reachable state-time space of the circuit $\mathbb{M}$ [42].

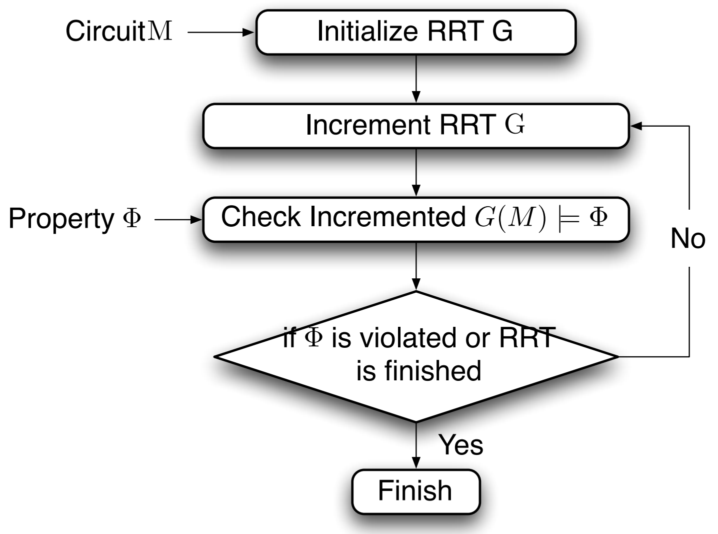

Circuit $\mathbb{M}$
Initialize RRT G
Increment RRT $\mathbb{G}$
Property Φ
Check Incremented $G(M) \models \Phi$
No
if Φ is violated or RRT is finished
Yes
Finish

Figure 5.1: Flowchart of TRRT-based runtime monitoring algorithm.

Algorithm 5 TRRT-based runtime monitoring algorithm

```
1: InitializeTRRT($\mathbf{x}(0)$)
2: InitializeMonitor($\varphi$)
3: for $i = 1 \to K$ do
4:     $q_{\texttt{sample}}$ = UniformSampling($\mathbb{S}$)
5:     $q_{\texttt{sample}}[n] = \texttt{rand}([\frac{i}{k} \times \texttt{rand}(0, T_{\texttt{max}})], T_{\texttt{max}})$
6:     $q_{\texttt{near}}$ = FindNearestNodeInTree($\mathbb{S}, q_{\texttt{sample}}$)
7:     $u(t)$ = GenerateNewInput($\mathbb{S}$)
8:     $q_{\texttt{new}}$ = Simulate($\mathbb{M}$, $q_{\texttt{near}}, u(t), \Delta t$)
9:     $\mathbb{G}$.expand($q_{\texttt{new}}$)
10:    Monitor($\mathbb{G}$, $q_{\texttt{new}}$, $\varphi$)
11:    if ($\mathbb{G} \not\models \varphi$) return False
12: end for
```

Therefore, we can have more confidence in the verification results that we obtain using our algorithm than the verification results obtained from random Monte Carlo simulations. State-time space is the circuit's state-space augmented with a dimension corresponding to time.

Algorithm 5 shows our TRRT-based runtime monitoring algorithm. The inputs to our algorithm are i) the circuit $\mathbb{M}$, ii) the initial state of $\mathbb{M}$, and iii) the properties φ to be verified on $\mathbb{M}$.

We now describe the steps of our algorithm in detail.

5.3.1 Constructing the TRRT for the circuit

This section describes how we extended the TRRT algorithm for runtime verification of analog circuits.

In real-world circuits, a state might be revisited during circuit operation. Oscillation circuits are very good examples of that scenario. Traditional TRRTs are tree data structures that span the state-space, and therefore do not contain any cycles. Such TRRTs cannot be used for verifying oscillation properties. In order to address that issue, we modify TRRTs to include a notion of time as well.

For a circuit with n state variables, we construct a TRRT $\mathbb{G}$ of $n+1$ dimensions corresponding to the state-time space of the circuit. State-time space is the n-dimensional state-space augmented with a dimension corresponding to time. Each node in the tree $\mathbb{G}$ is an $n+1$-dimensional vector denoted by $< x_1, x_2, \ldots, x_n, t >$ that corresponds to an n-dimensional state vector $< x_1, x_2, \ldots, x_n >$ and a time variable t. Let q be a point in the state-time space of circuit $\mathbb{M}$. We use the notation $q[x]$ to indicate the state vector and $q[t]$ to indicate the time variable of q.

Each edge in the TRRT $\mathbb{G}$ denotes a transient simulation of the circuit. We annotate each edge of $\mathbb{G}$ with the simulation time stamp t and the inputs to the circuit at that time $u(t)$. We use a discrete time step Δt for simulating each edge of the TRRT. Therefore, in the process of constructing the TRRT, we discretize the behavior of the circuit. We choose Δt to be small enough such that we do not discard the relevant behavior of the circuit. The minimum time step Δt should satisfy the *Nyquist* criterion $\frac{1}{\Delta t} \geq 2f_{\mathtt{max}}$, where $f_{\mathtt{max}}$ is the maximum operating frequency of the circuit [52].

To build the TRRT $\mathbb{G}$, we first select an initial state as the root of the tree $\mathbb{G}$. Our algorithm performs standard DC operating point analysis over M and sets the computed operating point as the initial state of the circuit. Alternatively, we allow the user to specify the initial state of the circuit.

In our algorithm, we grow the TRRT in a step-by-step manner. In each step, we obtain a point $q_{\mathtt{sample}}$ by randomly sampling the state-time space of $\mathbb{M}$. We then grow the TRRT towards the point $q_{\mathtt{sample}}$ as follows. We first find the closest node (in terms of Euclidean distance) to $q_{\mathtt{sample}}$ in the TRRT $\mathbb{G}$, namely $q_{\mathtt{near}}$. As in the classical TRRT algorithm, we determine the best possible trace from $q_{\mathtt{near}}$ toward $q_{\mathtt{sample}}$ and generate an input $u(t)$ to the circuit to follow that trace. We simulate the circuit $\mathbb{M}$ from state $q_{\mathtt{near}}$ for simulation time Δt using input $u(t)$, and determine the resulting state $q_{\texttt{new}}$. We then add $q_{\texttt{new}}$ as the new reached state to the TRRT $\mathbb{G}$.

The transient circuit simulation always progresses in time. We wish to model that in the TRRT growth as well. In order to achieve that while growing the TRRT, we filter out the candidates for the closest node that have a time annotation higher than that of $q_{\texttt{sample}}$. In other words, $q_{\texttt{sample}}[t] \geq q_{\texttt{near}}[t]$. Therefore, we ensure that the time notation of the parent node in $\mathbb{G}$ is always smaller than that of its children. As a result, our TRRT correctly models the progression of time in simulation traces of the circuit.

The classical TRRT algorithm tries to explore the entire space uniformly with no bias towards any particular dimension. Therefore, if there are many circuit state variables, the TRRT's growth in the time dimension may be very small. Consequently, it may take a large number of growth steps before the TRRT contains long simulation traces. Therefore, we modify the classical TRRT algorithm to improve the efficiency of our methodology. We modify the classical TRRT algorithm by introducing a small bias towards the time dimension. We ensure that the bias does not alter the probabilistic completeness property of the TRRT.

Let $T_{\texttt{max}}$ be the maximum simulation time specified by the user. We bias our random number generator by adding a probabilistic offset to the time variable. The default random generator for $q_{\texttt{sample}}$ is $q_{\texttt{sample}}[t] = \texttt{rand}(0, T_{\texttt{max}})$. Function $\texttt{rand}(a, b)$ uniformly samples a number in the interval (a, b). We wish to shift the time bias as i, the number of iterations, increases. Therefore, we use the following probabilistic offset to bias the time: $\frac{i}{k} \times \texttt{rand}(0, T_{\texttt{max}})$. That bias does not violate the probabilistic completeness property of TRRTs, since $\lim_{K\to\infty} \frac{i}{k} \times \texttt{rand}(0, T_{\texttt{max}}) = 0$.

With our bias, we calculate the time index of each new sample as

$$q_{\texttt{sample}}[t] = \texttt{rand}([\frac{i}{k} \times \texttt{rand}(0, T_{\texttt{max}})], T_{\texttt{max}}) \tag{5.2}$$

where k is the maximum number of iterations and i is the current iteration of the algorithm. The t index in each node corresponds to the time variable.

We use the notation $\mathbb{G}_i$ to indicate the TRRT $\mathbb{G}$ at the i^{th} iteration of the algorithm. After adding a new edge to the $\mathbb{G}_{i-1}$, the monitoring algorithm *Monitor* checks the incremented tree $\mathbb{G}_i$ against the property Φ.

5.3.2 TRRT-based incremental monitoring algorithm

The monitoring algorithm *Monitor* first parses the analog property Φ (Section 5.2) into a parser tree $\mathbb{P}$. The parser breaks down the formula into smaller sub-formulas. Each sub-formula can be an analog or temporal formula as described in Section 5.2. The parser performs that procedure recursively until all the sub-formulas are *atomic propositions*. An atomic proposition is a formula that is either *true* or *false* and cannot be broken down into simpler sub-formulas. Every leaf in $\mathbb{P}$ corresponds to an atomic proposition.

Monitor starts by checking the atomic propositions in the leaves of the parser tree $\mathbb{P}$. For every atomic proposition φ, the monitoring algorithm marks every node q in TRRT $\mathbb{G}$ such that $q \models \varphi$. Algorithms 6 and 7 describe how *Monitor* checks analog and temporal properties, respectively. The algorithm then moves from the leaves of $\mathbb{P}$ upwards to the top formula (i.e., the root of $\mathbb{P}$), checking every sub-formula stored in $\mathbb{P}$. *Monitor* terminates when the root of $\mathbb{P}$ is reached.

If the atomic proposition is an analog formula, *Monitor* employs Algorithm 6 to evaluate it. The evaluation of an analog formula involves computations using scalar data. These computations do not involve sequences in time. The algorithm marks every node in $\mathbb{G}$ in which the proposition evaluates to *true*.

The shift function can be implemented by traversing the TRRT $\mathbb{G}$ backward in a trace from a leaf toward the root of the tree. The TRRT consists of a set of simulation traces that are continuous in time. Therefore, a path from any node to the root of the tree is a complete *reversed* simulation trace of the circuit. Hence, traversing the tree backward through each node's parents is the same as moving backward in simulation. Similarly, the *maxSibling* and *minSibling* functions, which we use later in jitter computation, traverse the TRRT among siblings of each node (instead of parents) and would return a sibling with the maximum or minimum value. The basic operators and norm functions are computed according to the semantics in Section 5.2.2. Finally, the algorithm moves on to the next leaf in the parser tree.

Since most analog and temporal properties are associated with a single node, by adding a new node, we do not have to check the entire TRRT $\mathbb{G}$ to verify those properties. In most cases, verification of satisfaction and violation can be deduced by only checking the last node q_{new} against the incremented tree. As a result, because of the iterative construction of TRRT, algorithm 6 can be performed at $O(1)$ for most operators. The shift operator

Algorithm 6 Analog checking algorithm *Monitor($\mathbb{G}, q_{\texttt{new}}, \phi$)*

```
 1: Analog Formula $\phi$, TRRT $\mathbb{G}$, new node $q_{\texttt{new}}$
 2: switch $\phi$ do
 3:     case const
 4:         return const
 5:     case $f(\phi_1, \ldots, \phi_n)$
 6:         for $i = 1$ to $n$ do
 7:             Check($\phi_i$)
 8:         end for
 9:         switch $f$ do
10:             case Shift
11:                 $q_{\texttt{parent}} = q_{\texttt{new}}$
12:                 while Parent($q_{\texttt{parent}}$)$[t] \neq \phi_2$ do
13:                     $q_{\texttt{parent}}$ = Parent($q_{\texttt{parent}}$)
14:                 end while
15:             case Binary Operator $\diamond$
16:                 if $f$ is division function then
17:                     Check $\phi_2 \notin (0 - \epsilon, 0 + \epsilon)$
18:                 end if
19:             case Norm
20:                 Compute $L_2$ or $L_p$ norm $||\phi_1||_{\phi_2}$ or $||\phi_1 - \phi_2||$
21:             case Max(Min)-Sibling
22:                 for Node $q_{\texttt{sibling}}$ in $Child(Parent(q_{\texttt{new}}))$ do
23:                     Max = Max($q_{\texttt{sibling}}$)
24:                     Min = Min($q_{\texttt{sibling}}$)
25:                 end for
26:         return $f(\phi_1, \ldots, \phi_n)$
27: if $\mathbb{G} \models \phi$ then
28:     mark $q_{\texttt{new}}$
29: end if
```

is an exception with the worst-case complexity of traversing the tree from a leaf to the node, which is $O(\log n)$.

In order to verify a formula with temporal operators, *Monitor* employs Algorithm 7. This algorithm analyzes sequences of nodes in $\mathbb{G}$, each of which corresponds to a transient simulation in the circuit. An interval in which the proposition is defined is specified in the proposition. At every iteration, the algorithm checks whether $\mathbb{G}$ satisfies or violates the temporal formula for traces that lie within that interval. An example would be a decision on whether $\mathbb{G} \models (x[t, t + 100] < y[0, 100])$.

In order to incrementally decide whether $q_{\texttt{new}} \models$ `Eventually` φ, we add a Boolean variable `IsEventuallySatisfied` to each node in TRRT $\mathbb{G}$. `IsEventuallySatisfied` is true, if and only if at least one node along the path from $q_{\texttt{new}}$ along its parents to the root of $\mathbb{G}$ satisfies φ (honoring the time interval [a,b], on the path and filtering out other nodes). Algorithm 7 shows how we compute and update **Eventually** operator on TRRT.

To incrementally decide whether $q_{\texttt{new}} \models \varphi_1$ `until` φ_2, we add two additional variables to each node in TRRT. The first variable is $\texttt{always}\varphi_1\texttt{TillNow}$ which indicates that along the path from $q_{\texttt{new}}$ to its parents in the interval $[a, b]$, the proposition φ_1 is always satisfied. Similarly, $\texttt{NumberOf}\varphi_2\texttt{TillNow}$ $\in \{0, 1, \texttt{many}\}$, which counts the number of nodes that satisfy proposition φ_2. In **Until** proposition, violation occurs when φ_1 does not hold until φ_2. Our method for finding the violation is sketched in Algorithm 7.

The TRRT algorithm incrementally builds the tree by adding simulation traces edge by edge. As a result, for the majority of formulas in our semantics, checking only the new edge is enough to verify or find a violation of the formula over $\mathbb{G}$. However, for some temporal properties we may have to go back in time or more concisely traverse the TRRT $\mathbb{G}$ from the new leaf node upward, through its parents, towards the root of the tree until we can determine the status of the property. In the worst-case, that can take $O(\log n)$, where n is the size of TRRT $\mathbb{G}$. Since the size of the tree is finite, and by definition of the tree, there is no loop in the tree, our algorithm always terminates.

5.4 Experimental results and discussion

To evaluate our approach, we have implemented our algorithm in the C++ language. For simulating the circuit, we used Synopsys HSPICE and developed the interface between HSPICE and our tool.

We show the results for two nonlinear systems. The first case study involves a tunnel diode oscillator that we used as a proof of concept. The second case study is a phase-locked loop (PLL) circuit.

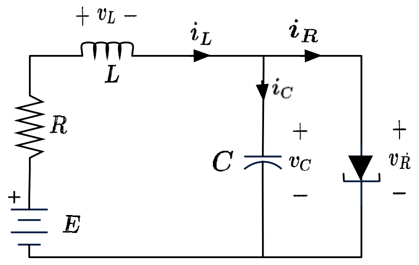


Figure 5.2: Tunnel-diode oscillator circuit.

5.4.1 Tunnel diode oscillator

To illustrate our methodology, we considered a tunnel diode oscillator that is a well-known nonlinear analog circuit. The resonant tunneling of the tunnel diode allows the current to decrease as voltage increases for some range of voltages. We used the circuit shown in Figure 5.2.

This circuit has a two-dimensional state-space. The state equations are modeled as

$$\dot{i_L} = \frac{1}{L}(v_{in} - R \times i_L - v_C) \tag{5.3}$$

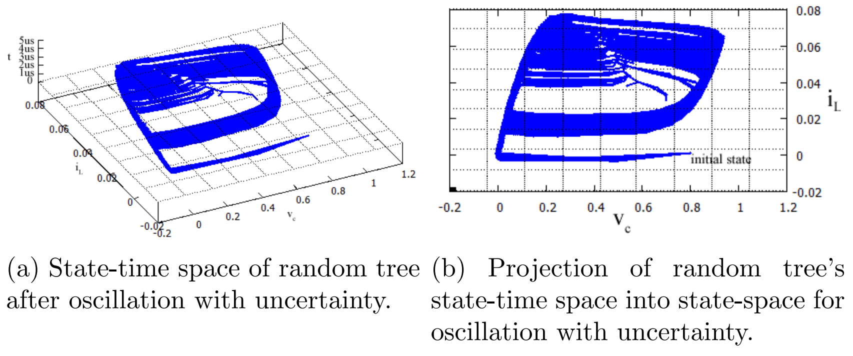
$$\dot{v_c} = \frac{1}{C}(-i_d(v_c) + i_L) \tag{5.4}$$

where $i_d(v_c)$ describes the nonlinear tunnel diode behavior.

We wished to verify whether or not for a given variation in the voltage source and uncertainty parameters in tunnel-diode models and for given initial conditions, the circuit satisfies the following oscillation property. We modeled the voltage source variation as $V_{in} = V_0 + p_1$, where $V_0 = 300\text{mV}$ and the tunnel diode uncertainty was modeled as $i_d(x) = x^3 - 1.5x^2 + 0.6x + p_2$. We assumed that the distribution of both variation parameters (p_1 and p_2) was uniform.

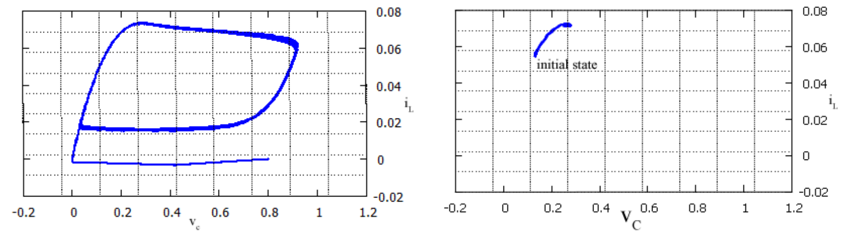
The oscillation property under consideration is as follows [153]. For oscillation, the current i_L should cycle between 0.02 and 0.06 indefinitely. Within the time interval $[0, 1\mu\text{s}]$, infinitely often whenever the $Norm(i_L)$ reaches 0.02, it will reach this value again within the time interval $[0, 6e-7]$. Also, the same property applies for i_L with amplitude 0.06. Formally $\forall\Box[0 : 1\mu\text{s}](\forall\Diamond[0 : 0.6\mu\text{s}](i_L \leq 0.02)) \wedge \forall\Box[0 : 1\mu\text{s}](\forall\Diamond[0 : 0.6\mu\text{s}](i_L \geq 0.06))$

Figure 5.3 shows the results of the tunnel-diode analysis using TRRT. Figure 5.3a shows the state-time space of the tunnel diode circuit with the voltage source variation modeled by $p_1 \in$[-0.05mv, 0.05mv] and $p_2 \in$



(a) State-time space of random tree after oscillation with uncertainty.

(b) Projection of random tree's state-time space into state-space for oscillation with uncertainty.



(c) Random tree's state-space projection for oscillation.

(d) Random tree's state-space projection of non-oscillating circuit.

Figure 5.3: Random tree outputs for tunnel diode oscillator.

$[-0.005, 0.005]$. The TRRT time resolution was set to $\Delta t = 10\mu$s, and we executed the algorithm for 20,000 iterations. Figure 5.3b shows the projection of the state-time space into the time dimension, which is the state-space of the circuit. As shown in Figure 5.3b, for many simulation traces, the circuit oscillates fully; however, for some branches of the TRRT, the oscillation was limited and did not meet the specification. Those branches failed to satisfy the design constraints for oscillation. Thus, this circuit is not verified. Figure 5.3c shows the same circuit for the same initial condition, but with reduced variation parameters $p_1 \in$[-5mV, 5mV] and $p_2 \in$[-0.5mV, 0.5mV]. We did not find any violation of the specification in this circuit. This example shows that even for a common initial state, the bound on variation parameters can lead to the violation of design specifications. The final experiment, shown in Figure 5.3d, was the tunnel diode example. The parameters in the tunnel diode model were set up by the same bounded variation as shown in Figure 5.3c, but with a different initial state. In this case, the circuit would not oscillate at all.

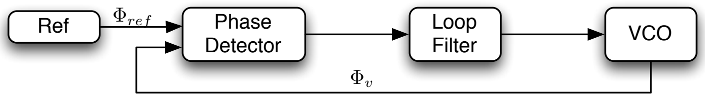


Figure 5.4: Phase-locked loop (PLL) circuit

5.4.2 Phased-locked loop circuit

PLL is a circuit that generates an output signal whose phase is related to the phase of an input reference signal. PLL circuit typically consists of a reference signal generator, a voltage-controlled oscillator (VCO), a phase-frequency detector (PFD), a loop filter, and a feedback loop.

Figure 5.4 shows the basic architecture of a PLL. In our simulation, we set the initial condition in the unlocked state of the PLL. When the PLL is out of lock, the frequencies of the input and output signals are different. The filter suppresses the higher harmonics. Consequently, there will be a DC component that will pull the average output frequency of the VCO up or down until the PLL locks. When the output of the filter is stable, PLL has locked to the input signal. We used a PLL circuit with $\Phi_{ref} = 1$ MHz and $f_0 = 1.01$ MHz. The input signal was generated through a fixed sinusoidal voltage source. HSPICE simulation was performed at the highest run level for maximum accuracy. To verify the uncertainty parameters in PLL, we added variation parameters to sources at the phase-detection block at each iteration. The variations were uniformly generated from interval $[-0.001, 0.001]$. We executed the TRRT for 30,000 iterations. Our PLL circuit had 17 states.

We used the output signal of the loop filter to verify the locking of the PLL. When PLL locks, the output signal eventually becomes stable, and there is no more deviation in the signal, except for some small deviations due to the phase-detector operations. We define the `PLL-locking-property` as eventually forever the jitter on the analog signal $Norm(v_1 - v_2)$ becomes less than 50mV in $2us$ interval where v_1 and v_2 are outputs of the loop filter block. Therefore eventually forever jitter on $v_1 - v_2$ is smaller than 50mV. That property means that the deviation of the given signal has to become less than 0.05 in a 2μs interval, so that it can be considered stable. Thus, we can assume that PLL has locked.

Figure 5.5 shows the deviation trace of the norm distance between the loop filter's output generated by TRRT under uncertainty parameters in the phase-detector block. Our algorithm finds no violation in the pll-locking-

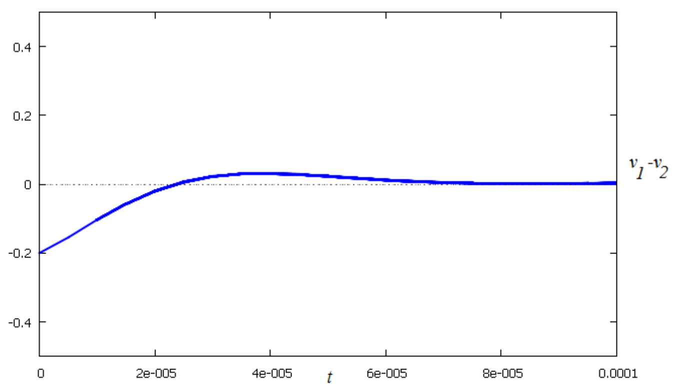


Figure 5.5: The TRRT trace of signal deviation for a loop filter.

property, so we assume the output of the loop filter eventually becomes stable forever.

5.5 Chapter summary

In this chapter, we used Duplex for runtime monitoring of analog circuits. We proposed a runtime verification algorithm to check the random trees against given specification. Our runtime verification methodology consists of i) incremental construction of the random trees to explore the state-time space and ii) use of an incremental online monitoring algorithm to check whether or not the incremented random tree satisfies or violates specification properties at each iteration.

Algorithm 7 Temporal checking algorithm *Monitor(*$\mathbb{G}, q_{\texttt{new}}, \varphi$ *)*

```
1: Temporal Formula $\varphi$, TRRT $\mathbb{G}$, new node $q_{\texttt{new}}$
2: switch $\varphi$ do
3:     case $v$
4:         return $v$
5:     case $\phi_1 \star \phi_2$
6:         check if $q_{new} \models \phi_1 \star \phi_2$ // Analog properties
7:     case $\varphi_1 \bullet \varphi_2$
8:         check if $q_{new} \models \varphi_1 \bullet \varphi_2$ // Temporal properties
9:     case Eventually $\varphi$[a,b]
10:        if $q_{\texttt{new}} \models \varphi$ or Parent($q_{\texttt{new}}$).Is$\varphi$Satisfied then
11:            $q_{\texttt{new}}$.Is$\varphi$Satisfied = true
12:        end if
13:    case $\varphi_1$ until $\varphi_2$
14:        if $q_{\texttt{new}} \models \varphi_2$ then
15:            $q_{\texttt{new}}$.NumberOf$\varphi_2$TillNow =Parent($q_{\texttt{new}}$).NumberOf$\varphi_2$TillNow
   + 1
16:        end if
17:        if $q_{\texttt{new}} \models \varphi_1$ and Parent($q_{\texttt{new}}$).Always$\varphi_1$TillNow then
18:            $q_{\texttt{new}}$.Always$\varphi_1$TillNow = true
19:        end if
20:        if $q_{\texttt{new}}$.NumberOf$\varphi_2$TillNow = 1 and
   Parent($q_{\texttt{new}}$).Always$\varphi_1$TillNow=false then
21:            return violation
22:        end if
23:    case $\mathcal{J}$
24:        for Every child node $v_i$ in Shift($q_{\texttt{new}}$, t) do
25:            $\mathcal{J}(q_{\texttt{new}})$ = max($\mathcal{J}(v_i)$)-min($\mathcal{J}(v_i)$)
26:        end for
27: if $\mathbb{G} \models \varphi$ then
28:     mark $q_{\texttt{new}}$
29: end if
```

CHAPTER 6

REACHABILITY ANALYSIS

6.1 Introduction

We propose a methodology for reachability analysis of nonlinear analog circuits. Our method reduces the approximation error and is computationally efficient. Our technique can be applied to general nonlinear systems while providing a precise analysis for handling polynomial nonlinear systems. Our algorithm is faithful to the nonlinear nature of the system and does not linearize the system at any point. Consequently, it provides a tightly over-approximated reachable set that is close to the exact reachable set. Most of the previous techniques compute reachability starting with the initial state and iteratively growing the reachable set [24, 58]. That approach is called forward reachability analysis [24]. In contrast, we start with an over-approximation that constitutes the entire reachable space of the system. We compute the boundaries of the reachable set by iteratively determining which regions in the over-approximated reachable set are unreachable. Next, we remove those regions from the reachable set to reduce its size. We call our method the *iterative reachable set reduction*.

Our algorithm works as follows. Initially, the entire state space is marked as the reachable set. Then we compute and refine the boundaries of the reachable set from the outside. At every iteration, our algorithm recursively partitions the reachable space into convex polytopes. For a given polytope, an adjacent polytope is one that shares a face with it. We determine if every polytope is reachable from its adjacent reachable polytopes. If we determine that a polytope is not reachable from any of its adjacent neighbor polytopes, then that polytope is marked as unreachable. We remove those unreachable polytopes from the reachable set to refine the over-approximated reachable set.

A polytope is reachable if there is a feasible trajectory toward it from any of its adjacent reachable polytopes. Therefore, we examine the direction of state space trajectories over the common face of every adjacent neighbor polytope. The direction of a state space trajectory is modeled as a multi-variable *reachability decision function* whose domain is the common face between the two adjacent polytopes. We call a function *existentially positive* if there exists a point in its domain where the sign of the function is positive.[1] If the reachability decision function is existentially positive on the common face between the target and its adjacent polytope, we determine that the target polytope is reachable. If none of the corresponding reachability decision functions are existentially positive, we declare the target polytope unreachable.

To determine whether a function is existentially positive on its domain, we check whether that function has any roots in that domain. Current root finding algorithms are known to be numerically unstable [32]. However, in our context, we would like to determine the existence of a root in a domain rather than finding the location of the root. Hence it is sufficient to count the roots without actually finding them. We employ a root counting method based on *Sturms theorem* [154] for nonlinear polynomial systems. Although the root counting method provides precise analysis for polynomial nonlinear circuits, in the case of general nonlinear circuits, it is not applicable. In the general case, we use root finding methods (like the *Newton-Raphson* method or the *Quasi-Newton* method [32]) to determine whether the function is existentially positive. By using root counting for polynomial nonlinear systems, we provide an accurate solution for proving reachability without linearizing the system at any point. The polytopes that represent the partitions of the state space get progressively smaller with every iteration. The over-approximation error of the reachable set is non-increasing and becomes smaller.

Typical implementations of polytope partitioning suffer from memory-related efficiency issues. We circumvent these problems by using the *Space Partitioning Tree (SPT)* data structures to model the state space. SPT is the generalized Binary Space Partitioning Tree (BSPT) in higher dimensions[155]. Previously, BSPT has been used in computer graphics [155], CAD, and verification [25]. We use SPT for recursive partitioning of the state space and as a data structure for storing and accessing geometric objects. Partitioning of the state space into polytopes results in generation of many polytopes for

[1]Existential positivity is defined over a ball in $\mathbb{R}^n$ space. A function can be existentially positive and negative over a ball at the same time.

modeling the state space. SPT models polytopes in the state space using hyperplane division instead of modeling each polytope individually. Consequently, the complexity order of the number of generated polytopes becomes polynomial. Also, SPT is very efficient at enumerating adjacent polytopes [155] and extracting the boundaries of the reachable set [156]. Those properties make SPT a suitable underlying data structure for our reachability algorithm.

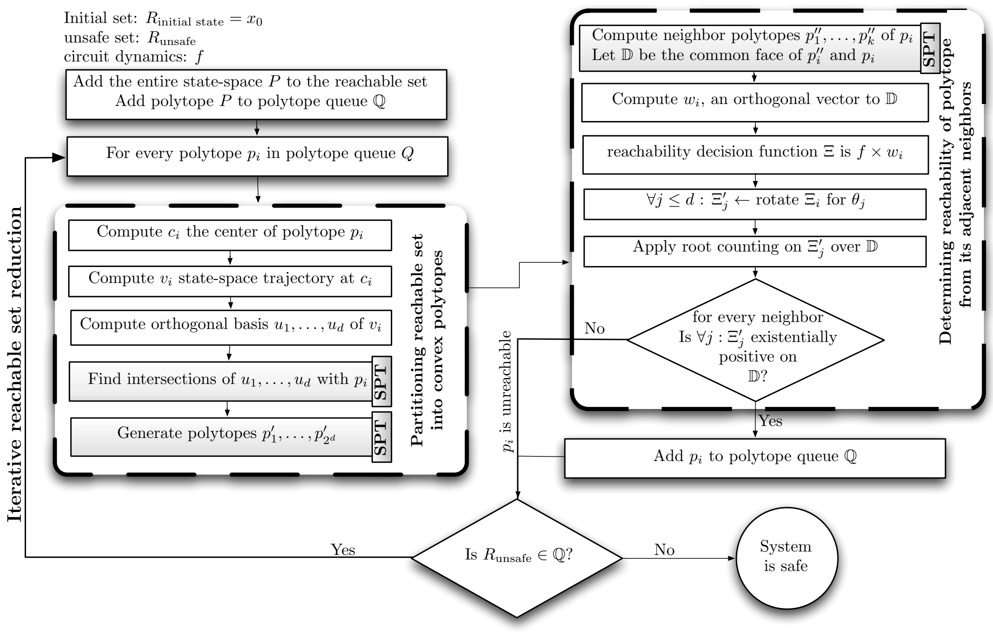
Our major contributions are as follows.

- We propose the iterative reachable set reduction algorithm, for reachability analysis of analog circuit systems. Our algorithm iteratively reduces the volume of the reachable set in an outside-in manner and converges quickly on a result.
- Our algorithm can be used to verify nonlinear analog circuits. We are faithful to the nonlinearities of the system. Our algorithm is accurate and does not introduce any linearization error into the reachable set.
- Our algorithm utilizes Space Partitioning Trees (SPT) to efficiently model the state space partitioning. Due to usage of SPT, our algorithm is more memory efficient than the state-of-the-art.

Since our over-approximations are conservative abstractions of the reachable set, our algorithm will never declare an unsafe state as safe, but it might declare a safe state as unsafe. We prove this soundness of our algorithm. Our algorithm will always converge to the exact reachable set, or an over-approximation of it. We demonstrate empirically that the algorithm converges in a few iterations to a tight approximation. We compute the reachable set of a nonlinear *Van der Pol* oscillation circuit, a standard circuit used in this analysis. To increase the confidence in our results, we run several transient simulations using a numerical simulator to approximately delineate and illustrate the reachable set. The transient simulations closely follow the output of our algorithm.

6.2 Iterative reachable set reduction algorithm

The objective of reachability analysis is to determine if there exists a trajectory from the set of initial states that eventually reaches the set of unsafe



Initial set: $R_{\text{initial state}} = x_0$
unsafe set: R_{unsafe}
circuit dynamics: f
Add the entire state-space P to the reachable set
Add polytope P to polytope queue $\mathbb{Q}$
For every polytope p_i in polytope queue Q
Compute c_i the center of polytope p_i
Compute v_i state-space trajectory at c_i
Compute orthogonal basis $u_1, \dots, u_d$ of v_i
Find intersections of $u_1, \dots, u_d$ with p_i
SPT
Generate polytopes $p'_1, \dots, p'_{2^d}$
SPT
Partitioning reachable set into convex polytopes
Iterative reachable set reduction
Compute neighbor polytopes $p''_1, \dots, p''_k$ of p_i
Let $\mathbb{D}$ be the common face of p''_i and p_i
SPT
Compute w_i, an orthogonal vector to $\mathbb{D}$
reachability decision function Ξ is $f \times w_i$
$\forall j \leq d : \Xi'_j \leftarrow$ rotate Ξ_i for θ_j
Apply root counting on Ξ'_j over $\mathbb{D}$
for every neighbor Is $\forall j : \Xi'_j$ existentially positive on $\mathbb{D}$?
Determining reachability of polytope from its adjacent neighbors
No
p_i is unreachable
Yes
Add p_i to polytope queue $\mathbb{Q}$
Yes
Is $R_{\text{unsafe}} \in \mathbb{Q}$?
No
System is safe

Figure 6.1: Overview of the iterative reachable set reduction algorithm. The exterior loop is the iterative reachable set reduction algorithm. For each polytope, our algorithm partitions it. Then, for each new partition, our algorithm decides on the reachability of those partitions from the reachable set. The parts of our algorithm that use SPT for computation are marked with SPT labels.

states under the circuit's differential regime. Our algorithm achieves this objective by iteratively identifying unreachable states. To identify the unreachable regions, we will recursively partition the over-approximated reachable set $\overline{\mathbb{R}}_x$ into convex polytopes.

Our iterative reachable set reduction algorithm consists of four major components: i) the main iterative reachable set reduction loop, ii) the state space partitioning algorithm, iii) a process for determining the reachability of adjacent neighbor regions, and iv) an SPT data structure for state space modeling. Figure 6.1 illustrates the important phases of our algorithm that are described in the following subsections.

The inputs to our algorithm are i) the state space of a nonlinear analog circuit, along with ii) the governing differential equations, iii) the set of initial states in the state space, and iv) the set of unsafe states.

Let $\mathbb{S} \subseteq \mathbb{R}^d$ be the continuous state space of an analog circuit where d is the number of the state variables. Let $R_{x(0)} \subseteq \mathbb{S}$ be the reachable set of the circuit from the initial set $x(0)$, and $\overline{R} \subseteq \mathbb{S}$ be an over-approximation of R_x, so $R_x \subseteq \overline{R}_x$. We assume the state space is bounded. That assumption is not limiting, because the target of our algorithm is an analog circuit. For example, a user can define the voltage variable to be bounded by $[-V_{cc}, +V_{cc}]$, where V_{cc} is the value of the voltage source, and so on. We assume that any region outside those bounds is unreachable, and we consider the region inside the bounds to be the state space of the circuit.

6.2.1 Iterative reachable set reduction

In this section, we describe the primary loop of our algorithm, the iterative reachable set reduction. Algorithm 8 shows that this loop will recursively remove the unreachable regions from the reachable state space.

We model the partitioned regions as convex polytopes which are identified through intersections of hyperplanes. At every iteration, we analyze the polytopes that are at the boundaries of the reachable set. This implies that those polytopes share boundaries with some unreachable set. For every generated polytope P_i, if P_i is adjacent to some unreachable set, then we analyze the reachability of P_i from its neighbors. We analyze the reachability of the polytope by checking the direction of state space trajectories of adjacent partitions in the reachable set. If we prove there is no feasible trajectory from any of the adjacent reachable regions toward the polytope P_i, we determine that P_i is unreachable and we remove it from the reachable set $\overline{\mathbb{R}}_x$. Otherwise, we recursively partition the P_i to further refine the reachable set.

In Algorithm 8, we create a queue of reachable polytopes. For each of those polytopes P_i, if P_i is at the boundary of reachable set, we further divide P_i to get a finer partition. Next, for each of the sub-partitions of P_i (called P_i's children) we determine if they are reachable from the reachable set. We enqueue any of those sub-partitions that are reachable and discard other sub-partitions to get a more accurate over-approximation. This process continues until a predefined number of polytope divisions n is reached; at that point, the algorithm terminates. In our algorithm, the volume of polytopes rapidly gets smaller and our algorithm converges to an approximated reachable set very fast.

To determine the reachability of each sub-partitions, we analyze the direction of the state space trajectories toward those sub-partitions. The polytope

Algorithm 8 Iterative reachable set reduction algorithm

```
1: Circuit $\mathbb{S}$, Initial States $\mathbb{I}$, Iteration bound $n$
2: Queue $\mathbb{Q}$
3: ReachableSet $\overline{R_x}$
4: $\mathbb{Q}$.push($\overline{R_x}$)
5: while $\neg$ $\mathbb{Q}$.empty() do
6:     Polytope $\mathbb{P} \leftarrow \mathbb{Q}$.pop()
7:     if $\mathbb{P}$ is adjacent to an unreachable region or $iteration < n$ then
8:         iteration $\leftarrow$ iteration+1
9:         partition($\mathbb{P}$)
10:        for $P_j$ in $\mathbb{P}$.getChildren() do
11:            Determine the reachability of $P_j$ from its adjacent neighbors
12:            if $P_j$ is reachable then
13:                $\mathbb{Q}$.push( $P_j$ )
14:            end if
15:        end for
16:    end if
17: end while
```

Algorithm 9 Partitioning the polytope

```
1: Circuit $\mathbb{S}$, Convex Polytope $P$
2: $c$ = center point of $P$
3: $v$ = $\mathbb{S}$.state space trajectory at ($c$)
4: $u_1, \ldots, u_d$ = GramSchmidt($\mathbb{S}$, $v$)
5: $i_1, \ldots, i_d$ = Compute intersections of $u_1, \ldots, u_d$ hyperplanes with $P$
6: $P_1, \ldots, P_{2^d}$ = Polytopes defined by $i_1, \ldots, i_d$ points and $P$
7: return $P_1, \ldots, P_{2^d}$
```

is reachable under either of two conditions: i) the polytope is part of the initial state, or ii) there exists a trajectory from at least one of the polytope's reachable adjacent neighbors toward it. In those cases, we conclude that the polytope is reachable.

6.2.2 Partitioning the reachable set into convex polytopes

We partition the continuous space of analog circuits to obtain a discrete model for the state space. Our algorithm partitions the reachable state space into convex polytopes. The partitioning is based on the direction of state trajectories at the center of each polytope. Let d denote the dimension of the system.

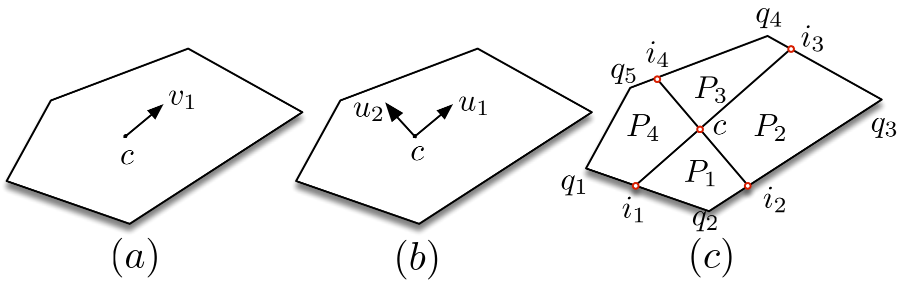


Figure 6.2: Partitioning a polytope based on state space trajectories.

Algorithm 9 shows an overview of our partitioning algorithm for a given polytope P. Initially, the entire state space constitutes the first polytope. At every subsequent iteration, we recursively divide the polytope by partitioning it using hyperplanes. We compute those hyperplanes using a vector basis that is orthogonal to the state space trajectory at the center of the polytope. Let c be the center of convex polytope P. Let v be the trajectory vector of the system at c (Figure 6.2.a). Using Gram-Schmidt orthonormalization process [157, 60], we construct an orthogonal basis vector set u such that $\{u_1, \ldots, u_d : u_i.u_j = 0, \forall 1 \leq i, j \leq d, i \neq j, v \in u\}$ (Figure 6.2.b). Vectors $u_1, \ldots, u_d$ form a set of d hyperplanes that divides the polytope P into 2^d convex polytopes $P_1, \ldots, P_{2^d}$ (Figure 6.2.c). Accordingly, our algorithm computes the intersection of each hyperplane with the faces of the polytope P. Then we compute the polytopes generated by the intersection of those hyperplanes and the polytope P. For example, in Figure 6.2.c, the new polytope P_1 is defined by the sequence of points $< c, i_4, q_5, q_1, i_1 >$ and so on. For recursively partitioning the state space, we utilize space partitioning tree (SPT) algorithm. SPT divides the state space into convex polytopes defined by intersection of hyperplanes. SPT algorithm allows an efficient storing and accessing of the polytopes in polynomial time [155].

6.2.3 Determining the reachability of adjacent polytopes

After generating a new polytope, we determine whether that polytope is reachable from the adjacent reachable polytopes. To determine the reachability between adjacent polytopes, we evaluate the direction of trajectories at the common face of the polytope and adjacent polytopes from the reachable set. If we can prove that for all common faces with the reachable set, there is no trajectory from the reachable set toward that polytope, we can deduce that polytope is unreachable and remove it from the reachable set. As shown in Figure 6.2, in 2-dimensions, the shared face between two adjacent polytopes R_1 and R_2 is the line from p_1 to p_2. Therefore for checking reachability of R_1 from its adjacent reachable neighbor R_2, we should check if there is any trajectory from R_2 to R_1 at the common face between two polytopes. We need to find at least a single trajectory from R_2 that goes toward R_1. We reformulate this as an analytical *reachability decision function*.

The direction of the trajectories over the bounded face of the polytope is presented as a multi-variable reachability decision function. Let w be an orthogonal vector from the center of the p_1, p_2 face toward R_1. We use a cross product of the RHS of the circuit's ODE f (Section 8.2) with the w vector to determine the direction of trajectories. For example, for a 2-dimensional system, the direction of vector trajectories is defined by the following *reachability decision function* Ξ:

$$\Xi(x, y) = f(x, y).w = det \begin{pmatrix} p_2.x - p_1.x & f_1 - p_1.x \\ p_2.y - p_1.y & f_2 - p_1.y \end{pmatrix} \tag{6.1}$$

where $p_1 =< x_1, y_1 >$ and $p_2 =< x_2, y_2 >$.

We define *existential positivity* of a function to determine if there is any interval in which the function ξ is positive on its domain. The existential positivity property for function ξ is defined as if there exists any *ball* $B_\epsilon(t) \in D_\xi$ such that for some $\mathbf{x} \in B_\epsilon(t)$ we have $\xi(\mathbf{x}) > 0$, where D_ξ denotes to domain of ξ.

The existence of a trajectory from R_2 toward R_1 is equivalent to the existential positivity of the reachability decision function Ξ on the function's domain (line p_1 to p_2, i.e., $\mathbb{D} = \{< x, y >: \lambda p_1 + (1-\lambda)p_2 =< x, y >, \lambda \in [0, 1]\}$). Therefore, when $\Xi(x, y) > 0$, the direction of the trajectories is toward R_1 and $\Xi(x, y) < 0$ is an indication of the direction of the trajectories toward R_2.

To determine existential positivity of higher dimensional functions, we transform them to lower dimensions. Therefore, we reduce that function to a weaker form by transforming it from a single d-dimensional function into d single-dimensional functions using rotation transformation. At lower dimensions (like $d = 1$), we use a root-counting method using the *Sturm's theorem* [154] and root-finding method using the *Newton-Raphson* algorithm

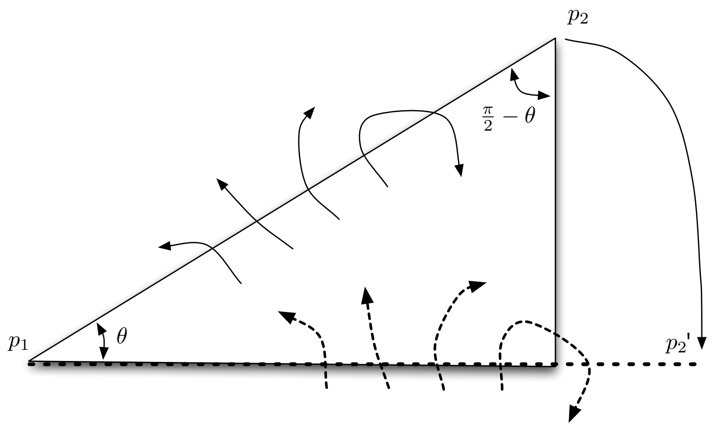


Figure 6.3: Determining existential positivity of the reachability decision function. Our algorithm rotates the function θ degrees to align it to the axis. Therefore, other variables become constant, and the reachability decision function becomes a single-dimensional basis function.

[32] to determine the existential positivity of the function. We call these single-dimensional functions the *basis functions* of the reachability decision function.

The domain of the basis function becomes paraxial by applying rotation transformation to the common face between two adjacent polytopes (Figure 6.3). Therefore except for only one axis, all other variables are constant. By applying d rotation transformations to the decision function for each axis, the reachability decision function is reduced to d single-dimension basis functions. If all of those d single-dimension functions are existentially positive on their basis domains (which are the intervals obtained by rotating the domains of the decision functions), we conclude that the reachability decision function is existentially positive on its domain. For example, in two dimensions the rotation transformation is as follows:

$$\begin{bmatrix} \Phi(x) \\ \Phi(y) \end{bmatrix} = \begin{bmatrix} \Xi_1(x,y) \\ \Xi_2(x,y) \end{bmatrix} \times \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

where Ξ_1 and Ξ_2 are the RHS of the system's ODE, and θ is the angle between the face and the axis. $\Phi(x)$ and $\Phi(y)$ are the basis functions.

For a single-dimensional basis function, our algorithm uses two different methods for determining its existential positivity. Existential positivity for a function depends on whether the function has any roots in its domain. If the root does not exist, it means that the function is not changing signs on its interval. Therefore we evaluate the basis function at the midpoint of the function's domain. If the function evaluates to positive, then we can conclude existential positivity of the basis function. Otherwise, the basis function is not existentially positive. We use two methods for determining the existence of a root in the basis function.

- Root counting method. For polynomial systems, we count the number of roots in any given domain using Sturm's theorem [154], which defines the number of real roots of a polynomial system in any interval using the changes in the signs of the values of the Sturm's sequence.

  Therefore if the total number of roots of the basis function in its domain is more than one, we deduce that the basis function is existentially positive on its domain. The benefit of the root counting method is that it always returns an exact result. However, Sturm's theorem can only be applied to nonlinear polynomial systems.

- Root finding method. Instead of counting the number of roots, our algorithm computes the roots of the function. If our algorithm is able to find at least one root in the domain of the basis function, we conclude that the basis function is existentially positive on its domain. Our algorithm uses the Newton-Raphson [32] algorithm to find roots of nonlinear functions.

6.2.4 Modeling the state space using a space partitioning tree (SPT)

We use a space partitioning tree (SPT) algorithm to model and store the polytopes generated in the state space. The space partitioning tree algorithm is the general n-dimensional case of the binary space partitioning algorithm used in [25, 155].

The polytopes are modeled in the SPT tree as shown in Figure 6.4. First, the entire state space is modeled as the root of the tree. Then we partition the root into 2^2 polytopes using two hyperplanes. Those hyperplanes are added to the SPT to model the generated polytopes. The tree is built and maintained on-the-fly during execution of the reachability algorithm.

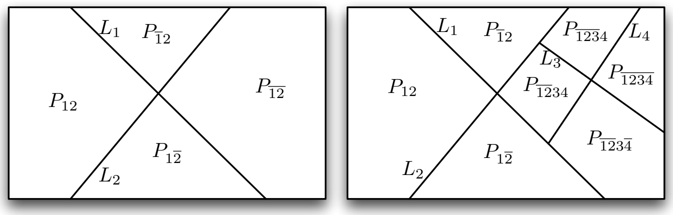


Figure 6.4: State space partitioning using hyperplanes. The polytopes are defined by the intersections of the hyperplanes in the state-space.

Let d denote the number of dimensions. We are constructing the SPT in R^d space. At each iteration, we add d hyperplanes to the tree to model the 2^d convex polytopes. Each hyperplane can be defined in $\mathcal{O}(d)$ memory. Therefore the memory complexity of our algorithm is $\mathcal{O}(nd^2)$ where n is the number of divisions that our algorithm performs. SPT allows access of logarithmic complexity to each polytope inside the state space while it utilize a moderately small memory foot-print. SPT also facilitates an efficient enumeration of adjacent polytopes to a region. Finally, SPT allows for fast computation of boundary of reachable set without computing the union of all reachable sets [156].

6.2.5 Sketch of soundness proof

To prove soundness of the iterative reachable set reduction algorithm, we show that if the region is unsafe, our algorithm will never declare that the region is safe. On the other hand, if the region is safe, our algorithm does not make any guarantees on it. We do not generate any false positives. Let $\mathbb{S} \subseteq \mathbb{R}^n$ denote the continuous state space of the circuit. Let $R_{x(0)}$ and R_{unsafe} denote the initial set and unsafe set of states respectively. A region is a set of states in $\mathbb{S}$. An unsafe region $\mathbb{X}$ is a region such that $\mathbb{X} \in R_{\text{unsafe}}$. The safe regions are in $\mathbb{S} - \mathbb{X}$. Let the sequence $< p_i, p_{i+1}, \ldots, p_j >$ denote the set of convex polytopes in the state space such that p_t and p_{t+1} are adjacent.

We use proof by contradiction. Given an unsafe region $\mathbb{U} \in R_{\text{unsafe}}$, let us assume that our algorithm declares this unsafe region as safe. Initially, the algorithm considers the entire set of states as reachable. This includes the safe as well as unsafe states in the system. The algorithm does not declare any unsafe region as safe. Hence, it is initially sound. Let us say that in iteration i, the algorithm declares (erroneously) that a region $\mathbf{U}$ is safe. In every subsequent iteration, it will declare $\mathbf{U}$ as safe. Initially we assumed that the system was unsafe. This implies that there is at least one trajectory from initial state $x_0 \in R_{x(0)}$ to state $x_f \in \mathbf{U}$. Let us call this trajectory T. By construction, T will cross some subsequence of adjacent polytopes $< p_0, p_1, \ldots, p_k >$ such that $x_0 \in p_0$ and $x_f \in p_k$. The polytope p_0 is reachable because $p_0 \cap R_{x(0)} \neq \emptyset$. Since the algorithm declared the system safe, the polytope p_k should be unreachable. Therefore, for some $1 < j \leq k$ our algorithm has determined that p_j is reachable but p_{j+1} is unreachable. This would mean that there is no trajectory from p_j toward $p_j + 1$. But T is a trajectory from p_0 to p_1 to $\ldots p_j$ to p_{j+1} to $\ldots$ to p_k. Therefore there is a trajectory from p_j to p_{j+1}. This is a contradiction. Hence the soundness of the algorithm is proved.

6.3 Experimental results

We implemented the iterative reachable set reduction algorithm in a prototype tool in the C++ language to evaluate its accuracy and efficiency. We chose an Apple Macbook Pro laptop equipped with a Core i7 processor and 8 GB memory as our computing platform. We ran our algorithm over a Van der Pol oscillation circuit to compute its reachable set. A Van der Pol oscillator is a nonconservative oscillator with nonlinear damping. A Van der Pol oscillator is a fundamental example in nonlinear oscillation theory [148]. It has a periodic solution that attracts every solution in the state space (except the zero solution). It is governed by two dimensional equations.

$$\dot{x} = y \tag{6.2}$$

$$\dot{y} = \epsilon(1 - x^2) \times y - x \tag{6.3}$$

In our experiment, we let $\epsilon = 1$, which was a medium value for ϵ and resulted in a medium distortion in the oscillation. The state space was defined as a bounded box $\mathbb{S} = [-10, 10] \times [-10, 10] \subset \mathbb{R}^2$. The initial set was a box $[-3.0, -2.8] \times [3, 3.2] \subset \mathbb{S}$. Figure 6.5 shows the reachable set of the Van der Pol oscillator circuit. The reachable set is the grey region. The unreachable states are in white. Figure 6.5 also shows the hyperplanes and the polytopes that our algorithm generated to compute the reachable set. As shown in

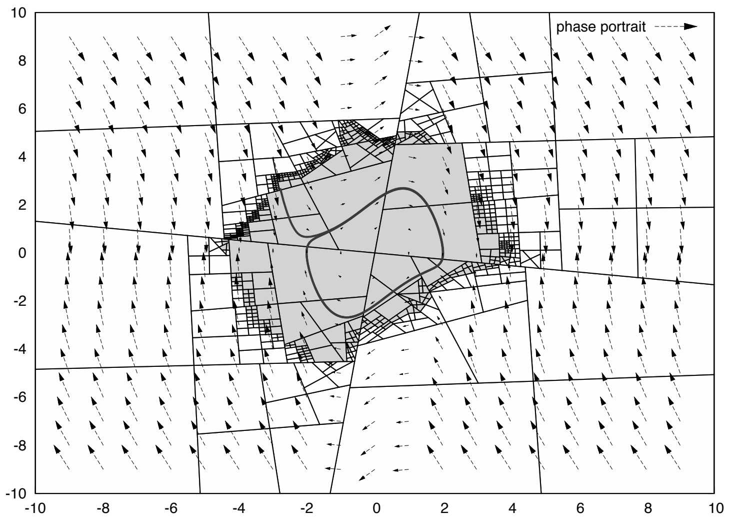


Figure 6.5: Reachable set for the Van der Pol oscillator using our iterative reachable set reduction algorithm. The reachable set is in grey and the unreachable states are in white. The polytopes at the boundaries of the reachable set shrink rapidly in volume.

the figure, our algorithm rapidly removed huge portions of the state space that were unreachable from the reachable set during the first few iterations. Then our algorithm eventually converged on the more refined and accurate reachable set. Our algorithm took **0.05 seconds** to compute the reachable set on our computing platform.

The iterative reachable set reduction algorithm can be effectively used for safety verification through computation of the reachable set. In many real test cases, only a few iterations are required to generate a coarse approximation of the reachable set and hence prove safety. That makes our algorithm a suitable candidate for safety verification. At any point during the execution, if we can prove safety, our algorithm terminates. If after reaching a predefined number of polytope partitioning, we have been unable to prove safety, our algorithm terminates without deciding on the safety of the system.

To make Figure 6.5 more informative, we added a quiver plot of the vector field (marked with arrows). We have also shown a transient simulation from a sampled point in the initial set. The transient trace was simulated using a numerical ODE solver (explicit embedded Runge-Kutta Prince-Dormand (8,9) method available in GNU-gsl-odeiv2 package) to delineate the reachable

Table 6.1: Space partitioning tree statistics. During the execution of the iterative reachable set reduction algorithms, most of the generated polytopes are at the boundaries of the reachable set and they rapidly get smaller in volume. † indicates that the number is a two-dimensional volume.

Statistical information obtained from the SPT	Value
Number of generated polytopes	1000
Number of leaves in the tree	751
Number of reachable leaves in the tree	491
Number of generated hyperplanes	500
Maximum depth of the SPT	9
Average depth of the SPT	7.34
Volume of smallest polytope in the leaf	4.63e-4 †
Volume of biggest polytope in the leaf	26.40 †
Volume of biggest reachable polytope in the leaf	6.25 †
Average volume of a polytope in the leaf	0.53 †
Average volume of a reachable polytope in the leaf	0.14 †

set. The circuit was simulated for $t = 20$ with $\delta t = 0.02$ time steps.

Table 6.1 shows some statistics of the space partitioning tree. We terminated the execution after 250 iterations. The SPT was built in two dimensions on-the-fly during the execution of our algorithm. In the end, the algorithm explored and divided 751 out of the possible 4^9 polytopes. The maximum depth of the tree was a logarithmic order of the generated polytope. In the end, the boundary of the reachable set consisted of 214 polytopes.

Among the different components of our iterative reachable set reduction algorithm, the component that allows for optimizations is the partitioning technique. Hence, a change in the partitioning algorithm significantly impacts the quality of results. Initially, we used hyper-rectangle partitioning, where each polytope's face was aligned to the axes. Similar to [60], we observed that the hyperbox data structure was causing a massive over-approximation of the reachable state space. Also, we tried other methods for polytope partitioning, such as partitioning of each polytope into two polytopes using the binary space partitioning tree (instead of d polytopes). However, the results were unsatisfactory. With those observations, we decided that polytope partitioning with d hyperplanes, resulting in 2^d polytopes at each iteration, is the optimum implementation of our algorithm.

To evaluate the reachability determination between adjacent polytope algorithm, we used a sampling-based method. To determine the reachability of a polytope from its adjacent polytopes, our algorithm incorporated a sampling scheme as follows. Our algorithm created many sample points at the borders of the polytope and simulated them for a δt time. Then our algorithm determined if the final state of the simulation had ended up inside the polytope or in the adjacent polytope. As a result, for 100 samples for each common face of each polytope, there was no significant difference between the sampling based reachability decision and our algorithm. However the sampling-based method took significantly more time (**4.48 seconds**) to compute the reachable set.

6.4 Chapter summary

In this chapter, we proposed a technique for reachability analysis of nonlinear analog circuits to verify safety properties. Our algorithm iteratively determines which regions in the state space are unreachable and removes those unreachable regions from the over-approximated reachable set. We use the State Partitioning Tree (SPT) algorithm to recursively partition the reachable set into convex polytopes. The algorithm can verify safety properties in near real-time and is very memory efficient. We computed the reachable set of the Van der Pol oscillation circuit.

CHAPTER 7

WORST-CASE EYE DIAGRAM ANALYSIS

7.1 Introduction

7.1.1 Monitoring signal integrity using the eye diagrams

Transient circuit simulation using Monte Carlo simulations is the most commonly used eye diagram analysis technique for nonlinear time-variant circuits [6]. However, Monte Carlo simulations can take very long (between days to weeks) [6] to fully analyze variations in the channel and circuits. Their coverage of simulation corners is also not as high as desired. Statistical [7, 8] and convolution-based analytical methods [9, 10] are fast and high coverage, but their scope is limited to linear time-invariant circuits. Also, they produce the final eye diagram contour, but not the corresponding input simulation trace. We present a simulation based eye diagram analysis technique as an alternative to Monte Carlo based methods. We analyze nonlinear time-variant circuits such as CMOS circuits. We argue for the higher coverage of simulation corners using our method as compared to Monte Carlo in the same time. Put differently, we produce the same quality eye as Monte Carlo as much as 20× faster. We also provide the input traces for an eye.

7.1.2 Using Duplex algorithm for generating worst-case eye diagram

We model the eye diagram analysis as a type-III duplex optimization problem. We use geometry to model the eye diagram as a type-III optimization problem. The worst-case eye diagram of the circuit corresponds to the minimum of the objective function in our optimization problem. Secondly, we use Duplex to minimize the objective function. The Duplex algorithm determines the input to the circuit and accordingly simulates the circuit to compute the eye diagram.

In current practice, the eye diagram is used as an output. We use the eye diagram itself to compute the worst case behavior of the design. If the eye diagram was representing non-ideal signal behavior, its contours would be distorted, depicting a noisy signal. Our method exploits this relationship by reversing the order and *distorting the eye diagram itself.* We model the distortion of the eye diagram for parameters such as noise margin and jitter as a *distortion functional* of that parameter. For example, the noise margin distortion functional models the area inside the contours of the eye diagram. We define jitter and overshoot/undershoot distortion functionals as well. We model these functionals such that an objective function comprising their weighted sum can optimize for the worst case eye diagram.

The sampling based optimization approach we use is based on random trees. We use the random tree algorithm to optimize the objective function and determine the contours of the eye diagram for the given input sequence. Using random tree simulation, we avoid repetitive exploration of the same regions, which is a known problem in Monte Carlo simulations [6]. Furthermore, we provide a better coverage of simulation corners than Monte Carlo transient simulations. These reasons make the random tree more attractive as an optimizing option than other standard optimization algorithms. Much of our efficiency and scalability results from the choice of the random tree as an optimization tool

There are two circuit inputs to our method. The first input is a deterministic logical input bit pattern to the circuit. The second input is a set of *nondeterministic perturbation parameters* that model variations, uncertainty in modeling and noise such as voltage fluctuation, input noise and signal timing variations. We model the perturbation parameters as truncated Gaussian random processes. We generate the eye diagram for the pre-determined input bit-sequence. We assume truncated Gaussian distribution for all the perturbation parameters. We automatically determine and cover the corner cases for each perturbation parameter. Finally, we generate the eye diagram corresponding to the selected bit pattern for the worst-case corner of all the perturbation parameters. Using our method, we can, with high accuracy, generate the absolute worst-case eye diagram of the circuit.

7.1.3 Benefits and contributions of Duplex algorithm

Our geometric approach of manipulating the eye diagram using integrals and optimization has many benefits. Our approach is quantifiable and precise with an optima. Our formulation is very efficient and does not impose any significant computational overhead because it can be computed and updated incrementally at every iteration of the algorithm. Our algorithm is adaptable to different scenarios by adjusting the perturbation parameters for computing area in the eye, as well as optimization objectives. The random tree algorithm is simulation based and we can compute the eye diagram of nonlinear time-variant analog circuits including high-speed CMOS circuits.

We use a post-layout CMOS inverter circuit as a proof of concept. To produce the same eye diagram, our random tree algorithm utilizes samples more efficiently and requires 20.66× fewer samples and 20.14× less absolute time. Alternatively, if we execute both Monte Carlo and random tree for the same amount of samples, our algorithm provides better coverage of the simulation corners while only imposing 1% absolute runtime overhead in comparison to the Monte Carlo. Finally, we demonstrate the scalability of our algorithm by computing the worst-case eye diagram of a post-layout 7-stage CMOS ring oscillator circuit. We added 35 variation parameters to this circuit, making the input space have 35 dimensions, while the state space has 210 dimensions.

Our contributions in this work are as follows. We present an efficient method for eye diagram analysis of nonlinear analog circuits. We geometrically model the worst case eye diagram as an area under the eye contours. We introduce a random tree algorithm to optimize the distortion functionals for parameters like noise margin, jitter, etc. We demonstrate how a random tree approach is best suited for this optimization problem. We show that our methodology provides a much higher quality eye than Monte Carlo, while being time efficient and scalable.

7.2 The eye diagram

An eye diagram [5] is a two-dimensional plot generated by repeatedly sampling and superimposing a signal (Figure 7.1). Let V_{OL} denote the logic level 0 and V_{OH} denote the logic level 1. The eye diagram consists of samples

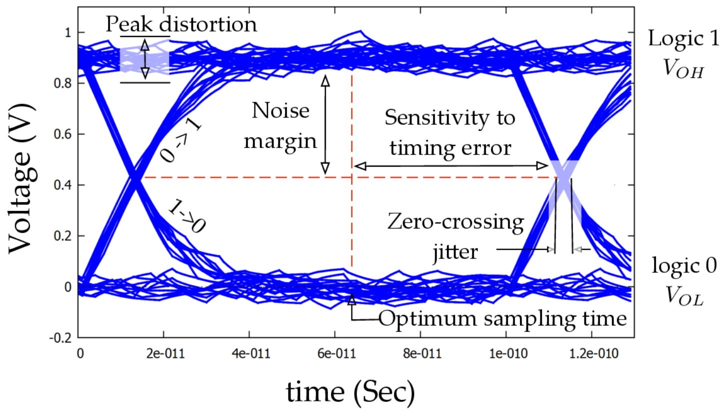


Figure 7.1: Eye diagram.

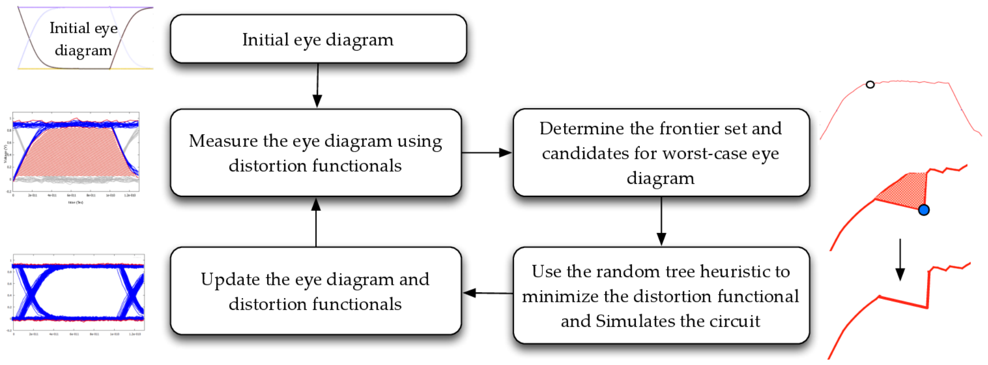

Initial eye diagram
Initial eye diagram
Measure the eye diagram using distortion functionals
Determine the frontier set and candidates for worst-case eye diagram
Use the random tree heuristic to minimize the distortion functional and Simulates the circuit
Update the eye diagram and distortion functionals

Figure 7.2: The high-level description of our approach. We use the eye diagram as a feedback in our approach and minimize the distortion functionals using the random tree algorithm.

corresponding to the signal value 0, signal value 1, transition from $0 \rightarrow 1$ and transition from $1 \rightarrow 0$. Important signal features such as *noise margin*, *peak distortion* such as voltage *overshoot/undershoot* and *jitter* can be measured from the eye diagram. The noise margin denotes the height of the eye, V_{OH} to V_{OL} at peak to peak, which determines the amount of the additive noise at the output. Peak distortion is the amount of the noise as overshoot and undershoot on V_{OH} and V_{OL} voltages. The jitter is determined by the width of the eye.

7.3 Our approach for eye diagram analysis

We generate the initial eye diagram for any pre-determined input bit sequence. The input bit sequence is defined by the user and can be arbitrarily long. We make a simplifying assumption that the elements of the bit sequence are independent and there is no interdependence between symbols. On the other hand, the focus of this chapter is on the worst-case corners in perturbation random processes that affect the input signal such as voltage fluctuations, noise, timing variations, etc. We focus on transient variations in power, input and timing variations such as jitter and rise time and fall time.

We analyze the eye diagram. For each simulation corner, we determine the worst case input that generates the eye diagram with minimum noise margin, maximum jitter, etc. Our methodology (Figure 7.2) consists of two important phases: i) Measuring the eye diagram using distortion functionals such as noise margin and jitter functional (Section 7.4), and ii) Using random tree optimization to minimize the distortion functional (Section 7.5). For the given perturbation input corner, the eye diagram with minimum distortion functional corresponds to the eye diagram of the circuit. For the given corner, worst case input determined by our algorithm corresponds to the eye diagram of the circuit.

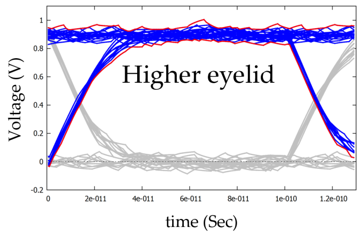
7.4 Geometric measurement of the eye diagram

In this section we model the eye diagram analysis as a multi-objective optimization problem.

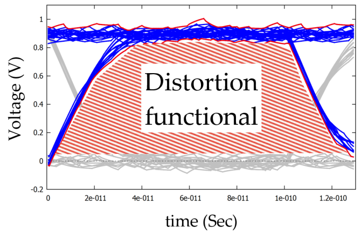
7.4.1 Geometric measurement of the eye output

We propose a formulation for the eye diagram analysis problem. We maximize the signal envelope of the eye diagram. Equivalently, we can minimize the *eye closure*, *i.e.* area inside the eye diagram contour. The eye closure determines various signal integrity parameters such as noise margin, jitter and voltage overshoot/undershoot.

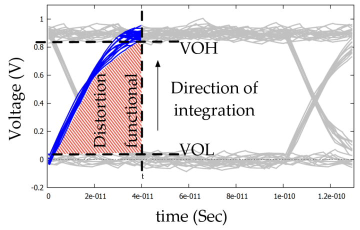
Let $\{b_0, \ldots, b_n\}$ denote the n-bit input bit sequence for the circuit. Let $\{\mathcal{Y}_0, \ldots, \mathcal{Y}_p\}$ denote the p perturbation random processes. Each random process $\mathcal{Y}_i$ follows a truncated Gaussian distribution $\mathcal{N}(\mu_i, \sigma_i)$. Let $\{d_1, \ldots, d_p\}$ denote the maximum distance of the perturbation samples from the mean of the distribution. For example, to determine an eye diagram of an inverter



(a) The higher eyelids in the eye diagram.



(b) The g_1 functionals measures the area inside the higher eyelid that we wish to minimize.



(c) The g_3 functional is a Lebesgue integral [158] that measures the jitter distortion.

Figure 7.3: The distortion functionals.

circuit with the input bit sequence is 00110, we add voltage fluctuation on input signal Y_1, where Y_1 follows a Normal distribution $\mathcal{N}(0, 0.05^2)$ and the input voltage can deviate up to $d_1 = 6\sigma = 0.3$V in that input corner.

Let v denote the output signal of the circuit. Let w denote the window size of the eye diagram analysis where $w = 2 \times T$, where T is the period of the signal v. Let s denote the set of signal samples in the eye diagram. Each sample is a pair of voltage and time, denoted by $s(v)$ and $s(t)$, respectively. In order to analyze the eye diagram, we decompose the eye diagram into *higher and lower eyelids*.

Definition 1 (Higher and Lower eyelid)**.** The *higher eyelid* is the set of signal samples corresponding to $1 \to 1$ and $0 \to 1 \to 0$ transitions in the eye diagram (Figure 7.3a). Similarly, the lower eyelid corresponds to $0 \to 0$ and $1 \to 0 \to 1$ transitions.

We define important eye diagram specifications such as noise margin, jitter and voltage overshoot/undershoot w.r.t. the signal envelope of each eyelid.

Definition 2 (Frontier set)**.** The *frontier set* is the signal envelope of the lower and higher eyelid.

Noise margin functionals: We measure the **noise margin** using the integral of the eye diagram contour of the minimum of the higher eyelid

(As shown in Figure 7.3b) and maximum of lower eyelid w.r.t. time. The minimum of higher eyelid corresponds to a weak logical 1 and maximum of lower eyelid corresponds to a weak logical 0. These integrals denote the area within the intersection of higher and lower eyelids. Minimizing this area as a result of minimizing these integrals results in lower noise margin.

The noise margin functionals measure the area inside the higher and lower eyelids. *We define these functionals in such a way that minimizing them results in lower noise margin in the eye diagram.* Let $s_1(t)$ and $s_2(t)$ denote the minimum higher eyelid and maximum lower eyelid at the time t, respectively. We define *noise margin distortion functional* as

$$\begin{aligned} g_1 &= \int_0^w s_1(t) - s_1^{Utopian}(t)dt \\ g_2 &= \int_0^w s_2^{Utopian}(t) - s_2(t)dt \end{aligned} \tag{7.1}$$

where the Utopian functions $s_1^{Utopian}(t)$ and $s_2^{Utopian}(t)$ denote the minimum and maximum for output voltages and w is the time window of the eye diagram. Without loss of generality assume $s_1^{Utopian}(t) = V_{OL}$ and $s_2^{Utopian}(t) = V_{OH}$.

Jitter, unlike noise margin or overshoot/undershoot, is a mapping from voltage to time. Although minimizing noise margin integrals also minimizes jitter as well, we emphasize jitter performance in high-speed IO circuits separately. We measure jitter using the *Lebesgue* integral [158] of the frontier set from V_{OL} to V_{OH} w.r.t. voltage as shown in Figure 7.3c. *We define these functionals in such a way that minimizing them results in higher jitter in the eye diagram.* Let $s_3(v)$ and $s_4(v)$ denote the states with the maximum and minimum time annotation (right-most and left-most samples) in the higher eyelid in the first and second period respectively. The *jitter distortion functionals* g_3 and g_4 are defined as

$$\begin{aligned} g_3 &= \int_{v_{OL}}^{v_{OH}} s_3(v) - s_3^{Utopian}(v)dv \\ g_4 &= \int_{v_{OL}}^{v_{OH}} s_4(v) - (W - s_4^{Utopian}(v))dv \end{aligned} \tag{7.2}$$

where $s_3^{Utopian}(v)$ and $s_4^{Utopian}(v)$ denote the rise time and fall time of the signal, respectively. Figure 7.3c shows the result of the jitter distortion functional g_3 where $s_3^{Utopian} = 40\text{ps}$. Similarly, g_5 and g_6 are defined for the lower eye lid as well.

Overshoot and undershoot are computed using the integral of maximum of higher eyelid and minimum of lower eyelid using the area outside the higher and lower eyelid curves. Minimizing these integrals increases the maximum of higher eyelid and minimum of lower eyelid and results in maximum overshoot and undershoot, respectively. Let s_7 and s_8 denote the set of maximum higher eyelid and minimum lower eyelid samples of the signal. The *overshoot and undershoot distortion functionals* are defined as

$$\begin{aligned} g_7 &= \int_0^w s_7^{Utopian}(t) - s_7(t)dt \\ g_8 &= \int_0^w s_8(t) - s_8^{Utopian}(t)dt \end{aligned} \tag{7.3}$$

where $s_7^{Utopian}(t)$ and $s_8^{Utopian}(t)$ are some values that the signal will surely never reach. For CMOS digital circuits, we use $s_7^{Utopian}(t) = 1.2\text{V}$ and $s_8^{Utopian}(t) = -0.2\text{V}$.

7.4.2 Computing the worst-case corner for the eye diagram

We define the *eye closure* functional as

$$g(\{\mathcal{Y}_0, \ldots, \mathcal{Y}_p\}) = \sum_{i=1}^{8} \omega_i g_i(x(\{b_0, \ldots, b_n\}, \{\mathcal{Y}_0, \ldots, \mathcal{Y}_p\})) \tag{7.4}$$

where $x(\{b_0, \ldots, b_n\}, \{\mathcal{Y}_0, \ldots, \mathcal{Y}_p\})$ is a sample in the eye diagram. $\{b_0, \ldots, b_n\}$ is the input bit sequence specified by the user. $\{\mathcal{Y}_0, \ldots, \mathcal{Y}_p\}$ is the perturbation random processes. The weights $\omega_1, \ldots, \omega_8$ are defined by the user s.t. $\sum \omega_i = 1$ and specify the importance of each distortion functional in the shape of the eye diagram. We want to minimize the eye closure. To minimize g, we have to minimize each distortion functional g_i. The eye diagram with minimum g corresponds to the eye diagram of the circuit. Since bit sequence b_i is selected by the user, our objective is to find the random processes $\mathcal{Y}_i$ that result in the minimum g and the eye diagram of the circuit.

To the best of our knowledge, there is no direct analytical method to optimize or even solve the objective function g [98]. We thereby use a simulation

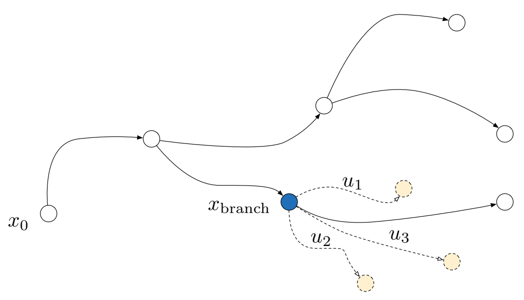


Figure 7.4: The growth of the random tree algorithm.

based optimization approach that provides a close approximation to the eye diagram of the circuit.

7.5 Minimizing distortion functionals using random trees

7.5.1 The random tree algorithm

We use a random tree (Figure 7.4) to simulate the circuit. The tree is incrementally *grown* by adding an edge between an existing node and a new state. Each node is a point from the state space of the analog circuit. Each edge is a short SPICE simulation of the circuit with a specific input trajectory. At each iteration, we select a node q_{from} where we wish to branch. To determine which input trajectory to take, we randomly *shoot* multiple trajectories from q_{from} in order to determine an *optimum trajectory* of the circuit at q_{from}. We provide details of how we compute the optimum trajectory in the next section. Next, we select the optimum trajectory and simulate the circuit from q_{from} to get the new node q_{new}. Finally the tree is expanded from q_{from} to q_{new}.

7.5.2 Our algorithm to minimize distortion functionals

We use random tree algorithm to minimize the distortion functionals and obtain the eye diagram with minimum eye closure (Algorithm 10). Initially, we apply an initial bit pattern[1] to the inputs of the circuit to exercise the

[1] In this work, we used the bit sequence 00110 to expose the $0 \rightarrow 0$, $0 \rightarrow 1$, $1 \rightarrow 0$ and $1 \rightarrow 1$ transitions and clock signaling in the eye diagram. Any other pre-determined or

initial eye diagram. At every iteration, we choose which distortion functional we wish to minimize from $g_1, \ldots, g_8$ with probability ω_i (Equation 7.4). The random tree algorithm simulates the circuit and samples perturbation inputs that reduce the distortion functionals. This process continues until we converge to the eye diagram of the circuit.

At every iteration, let g_i denote the distortion functional that we wish to minimize. The random tree algorithm randomly picks the node q_{from} from the *frontier set* s_i of the distortion functional g_i.

After selecting $q_{\text{from}} \in s_i$ we compute the perturbation input that decreases the distortion functional g_i. The distribution and amplitude of the perturbation are defined by the user. We sample a finite number of trajectories from q_{from} by linearizing the circuit at q_{from} and computing the optimum trajectory from the *Jacobian* matrix [32]. We pick a trajectory $y_0, \ldots, y_p$ that minimizes the distortion functional g_i. We simulate the circuit from the node q_{from} for time Δt using the input trajectory $y_0, \ldots, y_p$ to get the new node q_{new}. The user defines the simulation step Δt. Finally we add the new node q_{new} to the random tree and update the eye diagram.

We terminate the algorithm after reaching the maximum number of iterations. We also terminate if we converge to the final eye diagram where the consecutive change in the value of distortion functionals is below the threshold.

The output of the random tree algorithm is the eye diagram of the circuit, parameters for the worst case IO excitation, and the input sequence (bit pattern and variation in parameters such as noise and voltage fluctuations) corresponding to the output eye diagram.

7.6 Experimental results and discussions

In order to evaluate our algorithm, we implemented a tool in C++ and developed the interface with Synopsys HSPICE for simulating analog circuits. Our experiments were performed on a Core-$i52500K$ processor equipped with 16GB memory.

random input bit sequence can be used.

Algorithm 10 Our algorithm for minimizing the distortion functionals

```
1: Input: bit sequence $b_0 \dots b_n$
2: Input: perturbation random processes $\{\mathcal{Y}_0, \dots, \mathcal{Y}_p\}$
3: $\mathbb{G}$.init ()
4: Initial eye diagram $\mathcal{I}$ = Simulate the circuit with input bit sequence $b_0 \dots b_n$
5: while terminating condition not met do
6:     $g_i$ = Select objective with probability $\omega_i$
7:     $s_i$ = Select frontier set of $g_i$ from $\mathcal{I}$
8:     $q_{\text{from}}$ = Select a node randomly from the set $s_i$
9:     $\{y_0, \dots, y_p\}$ = Find an optimum trajectory $y_0, \dots, y_p$ from random processes $\{\mathcal{Y}_0, \dots, \mathcal{Y}_p\}$ from $q_{\text{from}}$ that reduces $g_i$
10:    $q_{\text{new}}$ = simulate the circuit from $q_{\text{from}}$ using input trajectory $\{y_0, \dots, y_p\}$
11:    $\mathbb{G}$.exapnd($q_{new}$)
12:    Update the eye diagram $\mathcal{I}$ and its frontier sets
13: end while
```

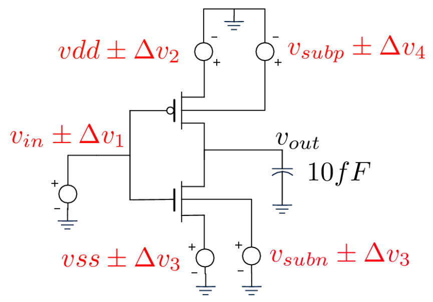
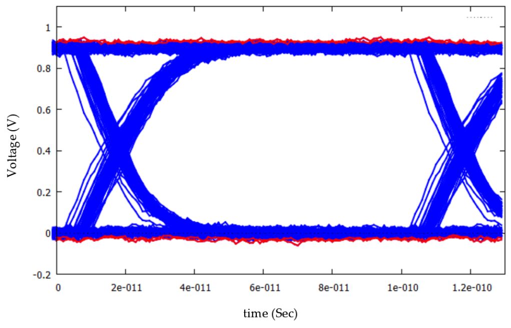


Figure 7.5: Schematic of CMOS inverter circuit.

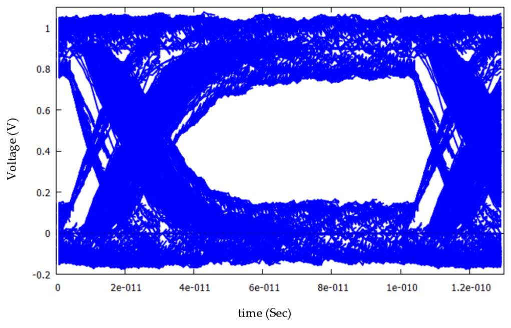
7.6.1 Efficiency of random tree algorithm

We use a post-layout CMOS inverter circuit (Figure 7.5) to evaluate the efficiency of the Duplex algorithm vs. Monte Carlo transient simulations for computing the eye diagram of the circuits. The inverter has 31 dimensions. The input to the circuit is a binary signal with non-ideal rise and fall time and input jitter. There are 5 variation sources in this circuit on inputs, power networks and substrate network (Figure 7.5). Our algorithm automatically adds amplitude and timing noise to the input signal and DC noise to other variations sources.

Efficiency: Figure 7.6a shows the eye diagram of the inverter circuit obtained using Monte Carlo transient simulation for 50,000 iterations. Each iteration is a small simulation step for Δt = 1ps. The frequency of the



(a) Eye diagram of the CMOS circuit using Monte Carlo simulation of the circuit.



(b) Eye diagram computed using random tree algorithm.

Figure 7.6: The worst-case analysis of the eye diagram in Monte Carlo vs. our algorithm. Given the same number of iterations, our algorithm generates an eye diagram that is 47% smaller than the eye diagram generated using Monte Carlo simulation.

input signal is 10GHz. We simulated the circuit for $\frac{50{,}000}{100\text{ps}/1\text{ps}} = 500$ random bits. The DC noise follows normal distribution with standard deviation 0.05. The jitter and rise/fall time noise follows a Normal distribution $\mathcal{N}(5ps, 1ps)$ and $\mathcal{N}$(10ps, 2ps), respectively. As shown in Figure 7.6a, the Monte Carlo algorithm does not converge to the eye diagram of the circuit within the limited number of iterations.

Figure 7.6b shown the eye diagram obtained using our algorithm. We run the random tree algorithm for the same number of iterations as Monte Carlo (50,000). Perturbation parameters were sampled from Gaussian distributions. In our algorithm we set the simulation corner to $6-\sigma$ deviation from the mean of the distribution. For example, input timing variation (jitter) follows a $\mathcal{N}$(5ps,1ps) Gaussian distribution, but can take a value from the range of $[0, 5 + 6 \times 1\text{ps}]$. The probability of a sample with 6σ deviation in Monte Carlo is 0.00034%, but our algorithm was able to quickly find such

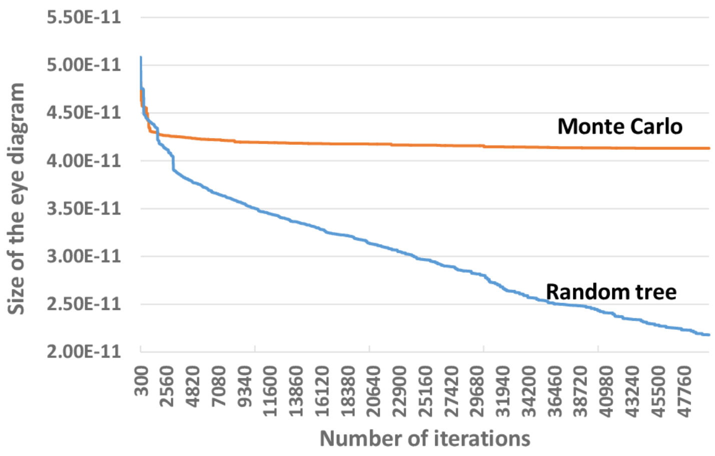


Figure 7.7: The convergence rate of random tree algorithm vs. Monte Carlo for the eye diagram analysis. The random tree algorithm converges much faster that Monte Carlo.

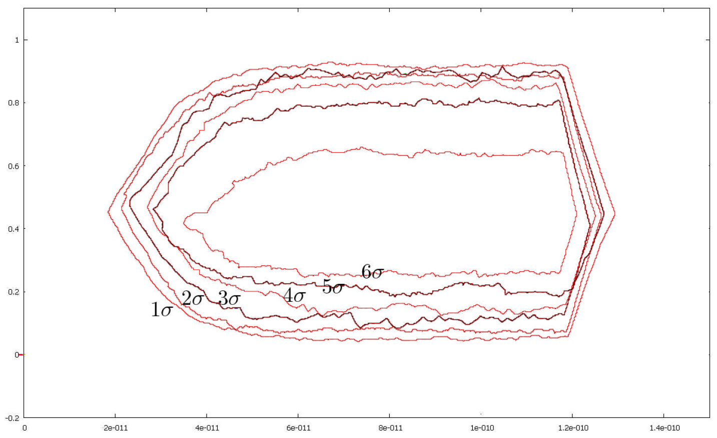


Figure 7.8: The size of the eye diagrams for different maximum deviations for simulation parameters.

corners.

As a result our algorithm was much faster and more efficient than the Monte Carlo. Given the same number of iterations, our algorithm produces a more accurate eye diagram and converges faster than Monte Carlo. In terms of absolute runtime, our algorithm does not impose any significant computational overhead. In our tool, the runtime of the Monte Carlo for 50,000 iterations was **141 minutes** whereas random tree took **143 minutes**,[2] which shows only 1% runtime overhead.

[2]In our implementation, at every iteration in both Monte Carlo and random tree, we had to execute the HSPICE software as an external tool which takes an substantially long time for license checkout.

Figure 7.7 shows the progress of our algorithm vs. Monte Carlo in each iteration. At every iteration, we reported the size of the eye diagram using Equation 7.4. Using Monte Carlo, the objective size decreased quickly at the beginning of the simulation, but the rate of convergence slowed down very quickly after a few bits. The random tree algorithm, on the other hand, rapidly converged to a smaller eye closure.

Figure 7.8 shows the eye diagram contour for different maximum deviations from the means of random processes. We executed out tool for different maximum distance from the mean of the distribution for every perturbation parameter. We plotted the contour of the generated eye diagram for 1σ distance to 6σ distance. As shown in the figure, as we increase the distance from 1σ to 6σ, the enemy closure becomes smaller and the signal integrity declines.

Finally, we extract input stimuli for generating the eye diagram. Statistical methods and analytical convolution-based methods are unable to do this. Figure 7.9 shows the scatter plot of the input stimuli for power voltage VDD that generates the logical 1 in our eye diagram. Each input stimulus was a path from the root of the tree to a node in the frontier set s_1 corresponding to the minimum of higher eyelid. The output sequence of the eye diagram was 11001. Most tests in Figure 7.9 initially follow the ideal path (the initial eye diagram in Figure 7.2) which is highlighted in the Figure 7.9 at voltage 0.9V. Figure 7.9.a shows the histogram of the VDD inputs in the input sequences. We removed the ideal paths from the histogram (where VDD=0.9V) for clarity. Most of the samples for generating a weak logic 1 came from the tail of the voltage distribution at 0.7V (In Monte Carlo, this distribution was $\mathcal{N}(0.9\text{V}, 0.05\text{V})$. Figure 7.9.b shows the scatter plot of the input stimuli extracted from the random tree. There were total of 130 input sequence corresponding to the minimum of higher eyelid ($\frac{\text{window-size}}{dt} = \frac{130\text{ps}}{1\text{ps}} = 130$). Figure 7.9.b shows that the worst-case higher-eyelid consists of three separate part. In each part, the signal followed the ideal path for some time and then diverged into that part. The worst case higher eyelid occurred during the $1 \rightarrow 1$, $1 \rightarrow 0$ and $0 \rightarrow 1$ transition. However, most of the samples (including the samples determining the noise margin at $t = 60\text{ps}$) were from the $0 \rightarrow 1$ transitions. This information can be used for debugging and validating the circuit.

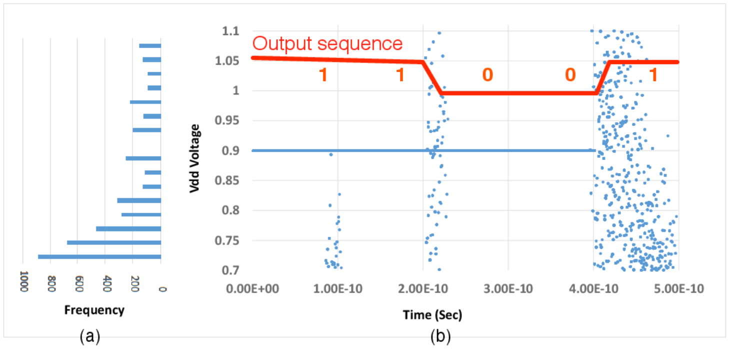


Figure 7.9: The scatter plot of the VDD inputs for generating the frontier set s_1. The left side figure shows the histogram of the VDD inputs samples (we excluded the samples from the ideal path). The right side is the scatter plot of input stimuli drawn over time, which identifies three separate component in the worst-case eye diagram.

7.6.2 Scalability of our algorithm

The random tree simulation, similar to Monte Carlo, is highly scalable and can be used on industrial circuits. We analyzed the 7-stage post-layout CMOS ring oscillator circuit in 45nm process to demonstrate scalability. The ring oscillator consists of an odd-number of CMOS inverters (Figure 7.5) arranged in a ring architecture. As a result, the circuit was unstable and oscillated as expected. We added 35 variation parameters to the circuit and analyzed the eye diagram for worst-case at the output. The state space of the circuit was 210 dimensions and the input space was 35 dimensions. Unlike the inverter circuit, the ring oscillator did not have any digital input signal, so we excluded the input jitter and rise/fall time model in the input space. Furthermore, the output oscillates so there is no $1 \to 1$ and $0 \to 0$ transitions in the eye diagram. As a result, we did not model the overshoot and undershoot functionals (g_7 and g_8) in the objective function by setting $\omega_7 = \omega_8 = 0$.

Figure 7.10 shows the eye diagram of the ring oscillator circuit obtained using our algorithm after 20,000 iterations. Our algorithm took **62 minutes** to compute the eye diagram.

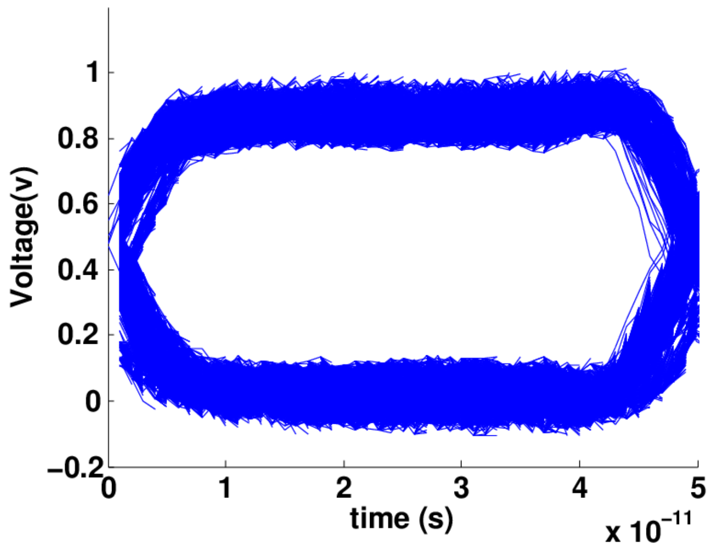


Figure 7.10: The eye diagram of ring oscillator circuit computed using our technique.

7.7 Chapter summary

In this chapter, we used Duplex for worst-case analysis of the eye diagram of CMOS circuits in presence of non-idealities in the circuit such as noise, voltage fluctuations and jitter. We argued that the classic definitions are inadequate in measuring important factors in the eye diagram and then proposed new objectives for analyzing the worst-case eye diagram. We modeled the eye diagram analysis as a type-III Duplex functional optimization problem. We modeled the objective function for noise margin and jitter as an area of the eye diagram curve. We used the Duplex algorithm to minimize the area of the eye diagram and determine the worst case eye diagram. Our experimental results demonstrated that the Duplex algorithm can be effectively and efficiently utilized to find worst-case signal integrity failures in the high speed CMOS circuits. Importantly, these worst-cases are not easily obtainable using classic simulation techniques such as Monte Carlo.

CHAPTER 8

TEST COMPRESSION

8.1 Introduction

8.1.1 Problem and Motivation

Test compression for analog and mixed signal circuits is a challenging problem. Analog circuits are nonlinear systems and work with continuous signals. Compressing continuous inputs is very complex and an instance of functional optimization. Compressing tests while maintaining the same precision and recall specification is challenging. Compressing analog tests may increase the rate of false positives and results in unnecessary losses. Therefore, test compression algorithm should guarantee functional equivalency and ensure the circuit will behave identically under the original and compressed tests. Test compression methods often require extra circuitry, both on-chip and off-chip, to minimize communication time and compressing the tests. The additional circuitry increases the cost of the testing. Finally, optimizing RF tests are still an open challenge because compressing RF tests in time inherently destroys frequency properties of the initial test.

To reduce test time, we introduce an automated test compression methodology for analog and mixed signal circuits. We apply our test compression algorithm to stress testing in this work. The intent of the stress test is to expeditely push vulnerable sections of certain corner-case die to manifest as hard defects. During the stress test, a functional test sequence is executed on the IC under high electrical activity. For analog circuits, this implies close to functionally achievable high currents and voltages on internal nets and node pairs respectively. The maximum and minimum values of voltage and current are obtained from the functional testing profile of a circuit. The input test sequences in stress testing are obtained from functional tests. However, the length of each functional test can be reduced, as long as it exercises the maximal and minimal electrical characteristics of the circuit.

Stress testing typically has three phases as shown in Figure 8.1: i) setup phase, ii) execution phase, and iii) stress phase. During setup, the circuit is reset, and DC inputs are set. For SoCs, the test is uploaded to the chip during the setup phase. Transient inputs are applied during execution, driving the circuit from an initial state to the final stressed circuit state. The circuit remains in the stressed condition for the remaining test duration. Some techniques compress the tests by reducing the communication time during the setup phase [18, 19]. However, these methods often require an extra on-chip circuitry and can be expensive. On the other hand, the duration of the stress phase is fixed and cannot be reduced. The focus of this chapter is the execution phase where the objective is only to drive the circuit to a stressed state. Since functional tests are not created with the purpose of achieving high electrical activity at different internal regions in optimal time, a lot of time is spent in the execution phase on finding the inputs that would stress the circuit. After our shortened execution phase, the stress phase can continue as usual, by stressing critical nodes repeatedly in the final state. Decreasing the duration of execution phase will not have any impact on the functionality of the stress test and does not require any additional cost or hardware on the circuit or the ATE machine.

8.1.2 Our methodology for test compression

In [159], we proposed a technique for automated test compression for electrical stress testing of analog circuits. We modeled the test compression as an optimization problem. We used our technique to compress tests for practical and scalable analog circuits. We analyze the output response of the analog circuit, instead of the input stimuli itself. We define two tests as functionally equivalent if they have the same initial value and boundary value in the state space with respect to a target electrical quantity. The test is finished when it reaches the boundary value within the test duration. The *length* of the test is the time duration for the test to reach its boundary value from the initial value. For instance, in an inverter, all input signals that can drive the output to V_{OH} from reset are functionally equivalent. For example, for testing V_{OH} in an inverter circuit, we are only interested in checking if the

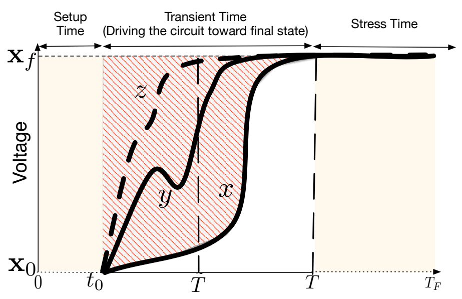


Figure 8.1: Compressed test z with the optimal time among all functionally equivalent tests $\{x, y, z\}$.

output voltage can reach a particular value. Hence, all input signals that can drive the output voltage from the reset state to the V_{OH} are functionally equivalent, regardless of their length. So any input signals that can drive the output voltage from the reset state to the V_{OH} are functionally equivalent regardless of how long each input signal takes.

Given an output transient response of the system, our objective is to find a functionally equivalent, but shorter test. In Figure 8.1, given test x, tests y and z are functionally equivalent to x because their initial value x_0 and boundary value x_f are the same. z is the shortest test. Note that shorter test length corresponds to smaller area on the left side of the transient response curve. If we now pull the curve while keeping the initial and boundary values the same, we get shorter tests.

We formalize this intuition as follows. We formulate test time minimization as a functional [98] optimization problem.[1] We express the output transient response of the system as an *electrical flux* functional [98]. Electrical flux[2] is a quantity that captures voltage as well as time. *Electrical flux functional* measures the area to the left of the output transient. This is the area that we get by integrating vertically, i.e. along the value axis as shown in Figure 8.1. If we now optimize the electrical flux functional as our objective function, we can achieve the goal of test time compression. Since the initial and boundary values of voltages are fixed, this is tantamount to minimizing the time taken for the voltage to reach boundary value from its initial value. We prove for smooth and differentiable nonlinear analog circuits that the test with

[1]Functional integration refers to the field of calculus where the domain of an integral is a space of functions. This is distinct from the use of functional tests in the manufacturing process of chip design. An example of a functional is a function applied to the state space or the output response of the circuit, such as integrating the output curve.

[2]Should not be confused with magnetic flux.

minimal test time corresponds to the test with minimum flux functional (Section 8.2.1).

For minimizing this electrical flux functional, there are no closed form analytical solutions to the best of our knowledge. We therefore provide a *simulation based minimization approach.* This simulation based approach is based on *random trees.* Random trees use numerical methods (such as SPICE) to simulate the circuit. While random trees have some benefits we describe in the chapter, simulated annealing and other numerical optimization algorithms could well be applied to solve this problem. Since we use a simulation based optimization method, we do not guarantee the minimal solution, but a near-minimal one. Empirically, we observe that our tests provide significant compression over the original tests.

We demonstrate the effectiveness of our test compression methodology on three CMOS circuits. We use a post-layout CMOS inverter circuit (45nm, 31 dimensions) with all the extracted parasitics as an illustrative example. We show that we can achieve up to 94% reduction in the inverter test length. Our main case study is a CMOS opamp circuit (0.18μm library, 12 dimensions). We show that we can consistently achieve on average 93% reduction in test length for multiple functional and burn-in stress tests for the op-amp. We analyze tests for a voltage controlled oscillator designed in 0.18μm circuit to show scalability and achieve compression up to 88%.

Our technique compresses the tests in the time dimension, destroying frequency properties of the test. Hence, it cannot be used to compress frequency tests such as bandwidth. On the other hand, stress tests, functional tests and defect screening are independent of the frequency properties of the circuit and adapt very well to our algorithm.

Variations in manufacturing process, especially at deep sub-micron levels, introduce uncertainties in device geometries and mismatch in sizing. Variation causes variances in the timing and output performance of the circuit which decreases yield. As a result, testing analog circuits becomes very challenging due to unpredictability of the circuit's behaviors. If the performance of the output metric is outside the test specification, the circuit fails the test, which results in decreased yield for that set and device. It is essential to design (and compress) tests to cover the worst-case execution and achieve minimum false positives in order to improve yield. Our previous technique [159] did not handle test compression in the presence of process variation.

Test generation and compression is a very time-consuming task. We have to spend hours in the lab to cut picoseconds from the test time. Compressing each test for a practical-sized circuit can take hours [159] on a typical workstation. The downside of test compression is increase in the engineering R&D time and increased time-to-market.

8.2 Test compression as a flux functional optimization problem

Let T in Eq 2.2 denote the *length* of the test sequence $u(t)$. $\mathbf{x}$(T) is the *boundary* value, indicating the final state of the test. In practical applications, the solution of nonlinear analog circuits can be computed using a numerical ODE solver like SPICE.

There may be multiple boundary values $\mathbf{x}_1, \ldots, \mathbf{x}_f$ on the solution of the test sequence. A test has to visit states $\mathbf{x}_1, \ldots, \mathbf{x}_f$ in sequence to be passed. We divide this problem into $n-1$ single boundary test compression. Each boundary value $\mathbf{x}_i$ is computed using

$$\mathbf{x}_{i+1}(t_{i+1}) = \mathbf{x}_i(t_i) + \int_{t_i}^{t_{i+1}} f(\mathbf{x_i}(t), \mathbf{u_i}(t))dt \tag{8.1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ are the initial and the boundary values for the test compression problem. For sake of simplicity, in this chapter we are only concerned with an initial value $\mathbf{x}_0(0)$ and a single boundary value $\mathbf{x}_f(T)$. Multiple boundary values can be divided into multiple single boundary value problems. Dynamics of analog circuits are smooth and continuous and satisfy the local Lipschitz property [98]. This ensures the existence and uniqueness of the solutions. We can concatenate these divided tests back together to form the original test.

Definition: *Functionally equivalent tests*: We define two tests (input signals) as functionally equivalent if and only if for the same nonlinear system, their initial and boundary values are the same. In other words, two input signal u and $\tilde{\mathbf{u}}$ are functionally equivalent when

$$\int_0^T f(\mathbf{x}, t, \mathbf{u})dt = \int_0^{\tilde{T}} f(\mathbf{x}, t, \tilde{\mathbf{u}})dt \tag{8.2}$$

where T and $\tilde{T}$ denote the length of the input signal u and $\tilde{u}$, respectively. This implies that for both input test u and $\tilde{u}$ if initial values $\mathbf{x}_0$ are equal then their boundary values $\mathbf{x}_f$ are equal. For test compression, the length of the compressed, functionally equivalent tests should be less than the original. So $\tilde{T} \leq T$.

The objective of the test compression problem is to find, among all possible input stimuli $\tilde{u}$ which are functionality equivalent to the original input stimuli u, the input stimuli that requires minimum amount of time to reach the boundary value $\mathbf{x}_f$ from the initial value $\mathbf{x}_0$ as shown in Figure 8.2. We propose the following objective function for optimization:

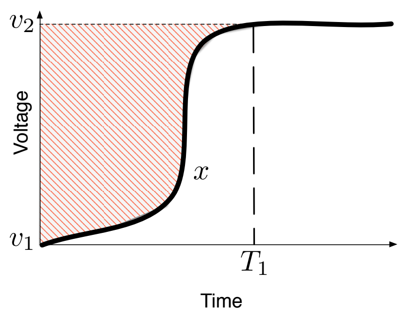
$$\min \int_{\mathbf{x}_0}^{\mathbf{x}_f} t(\mathbf{x}) d\mathbf{x} \tag{8.3}$$

where t($\mathbf{x}$) is the time dimension of each state in the solution. $\mathbf{x}$ denotes the state and is a vector in $\mathbf{R}^n$.

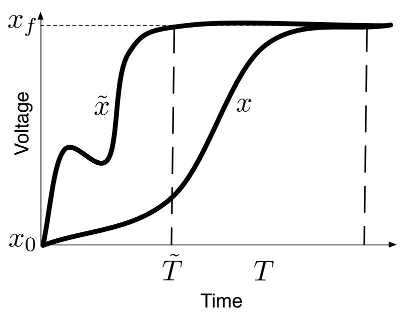
Equation 8.3 is an integral along the solution path $x(t)$ describing the electrical *flux* of the circuit. Flux is a physical entity that captures voltage and time simultaneously. The flux is quantified by the *Weber* metric and expressed in terms of voltage times seconds ($Wb = V \times s$). We can measure the flux by either integrating voltage dv or time dt. Since we want to minimize time while keeping the boundary voltages fixed, we define the **test compression objective function** using a *Lebesgue* integral [98] in equation 8.3. Figure 8.1 shows the area that the Equation 8.3 is computing. The Lebesgue integral measures the area by integrating the value (y axis) over time, in contrast to the standard horizontal time axis integration.

Since $\mathbf{x}_0$ and $\mathbf{x}_f$ are constant, minimizing this integral results directly in minimizing T. Although decreasing the flux functional does not necessarily minimize time, the test with minimum flux functional coincides with the test with minimum time. This implies that minimizing the flux functional is necessary, but not sufficient, for minimizing the test time. When we converge to the minimum of the flux functional, we also provably converge to the test with optimum time. Since the time required to reach the boundary value in output is the same as input test time, minimizing Equation 8.3 results in compressing the input test signal too.

Equation 8.3 is directly computing a function of the state of the circuit. An input signal is just another dimension in the state space. Hence, minimizing



(a) Objective functional of the test compression.



(b) Compressed test will reach the boundary value sooner.

Figure 8.2: The modeling of the test compression problem.

this integral will minimize the time required for a test to reach the boundary value.

To the best of our knowledge, there is no closed form analytical solution for Equation 8.3 when the system f is a nonlinear function. We therefore optimize Equation 8.3 using simulation based algorithm based on random trees.

8.2.1 Sketch of proof for minimality of flux functionals

Reducing the flux functional is the necessary, but not sufficient, condition for optimizing test time. The optimal test (with minimum length) also has the minimum flux functional. We prove this by contradiction. First, we prove the statement for low-dimension cases and then generalize it to higher dimensions.

Theorem: The optimal test (with minimum length) has the minimum flux functional.

Proof: Assume that the solution y of the nonlinear system f with input u is smooth, 2^{nd}-order differentiable and locally Lipschitz. Assume there exists an optimal test u_1 with the transient solution y_1 such that the flux functional of y_1 is not minimal. Let $T_{\min}$ denote the length of u_1. Assume there exists a test u_2 with transient solution y_2 such that y_2 has the minimum flux functional among all admittable transient solutions in the system. The length of u_2 is bigger than $T_{\min}$.

Because of the *first mean value theorem for integrals*, $\mathbf{x}_2$ and $\mathbf{x}_1$ are going to intersect at some point x^* at time t^* (Figure 8.3). We construct an alternative

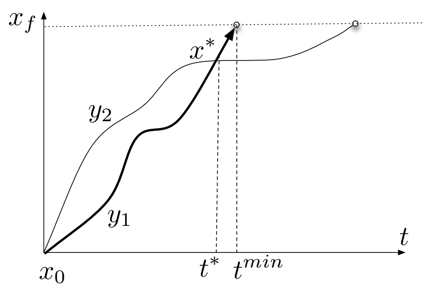


Figure 8.3: Contradiction test with minimum time but not minimum flux functional.

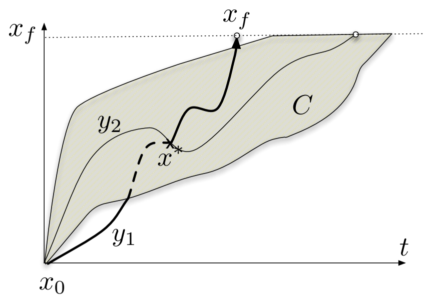


Figure 8.4: Generalization of the proof to higher dimensions.

solution x_3 by applying the input u_2 till time t^* and then the input u_1 to the circuit till time T_{min}. The combined test has a corner point and is not smooth anymore, so we use Wierstrass-Erdmann theorem [98] to construct the test. The length of the test u_3 is T_{min} which indicates that this is a minimal test. On the other hand, the flux functional of $\mathbf{x}_3$ is less than x_2 which is a contradiction. Therefore a test with minimum time is also a test with smallest flux functional.

The same idea can be applied in higher dimensions as well. Assume the smooth manifold C is the set of all admittable transient solutions of the system with minimum flux functional (Figure 8.4). Any trace below and above manifold C has higher and lower flux functional, respectively. Similarly, assume test u_1 with solution y_1 is time-optimal. Trace y_1 intersects with manifold C at x^*. Similarly, according to existence and uniqueness theorem, we can construct an alternative tests y_3 that follows y_2 till reaching x^* and then follows y_1. y_3 has less flux functional and is time-optimal, which is a contradiction.

8.3 Optimizing Functionals Using Random Trees

In this section, we present the Duplex optimization algorithm to find a optimal solution to Equation 8.3. The solution is the compressed test sequence $\tilde{\mathbf{u}}$. The input to our algorithm is the circuit netlist and an input test signals in piecewise linear (PWL) format.

8.3.1 Random Trees

A random tree is a tree-based simulation algorithm that generates and maintains a tree data structure during simulation. It is able to backtrack to different previously visited states, which makes it more versatile than a random walk based simulation algorithm like Monte Carlo. It is incrementally *grown* by adding an edge between an existing state and a new state. Each node of the tree is a point in the state space of the analog circuit. Each edge is a short SPICE simulation of the circuit with a specific input trajectory. At each iteration, it selects a state $x_{\mathrm{b}ranch}$ to branch from. The random tree randomly *shoots* multiple trajectories $u_1, \ldots, u_m$ from $x_{\mathrm{b}ranch}$. It then selects the *optimum trajectory* u_{opt} and simulates the circuit from $x_{\mathrm{b}ranch}$ to get the new node $x_{\mathrm{n}ew}$. Finally the tree is expanded from $x_{\mathrm{b}ranch}$ to $x_{\mathrm{n}ew}$. The selection of the locally optimum trajectory at every iteration of the tree's growth can be biased according to heuristics pertaining to the objective function.

Time annotation in random trees The root of the tree is the initial value state with time 0. Let $\mathbb{G}$ be the random tree data structure. Each state in $\mathbb{G}$ is augmented with time annotation. Therefore a state v in $\mathbb{G}$ is a pair of $(x_1, x_2, \ldots, x_n, t)$ where x_is are the set of values assigned to the state variables of the circuit and t is the sample time.

8.3.2 Random tree algorithm to minimize flux functional

Figure 8.5 shows steps of the random tree algorithm, as applied to minimize our test compression functional formulation. Initially we grow the random tree according to the original test input u. We determine the initial value $\mathbf{x}_0$ and the boundary value $\mathbf{x}_f$ from the original test (Figure 8.5-block 0).

A state $\mathbf{x}$ is *time-optimal* if and only if $\mathbf{x}$ belongs to the convex hull of

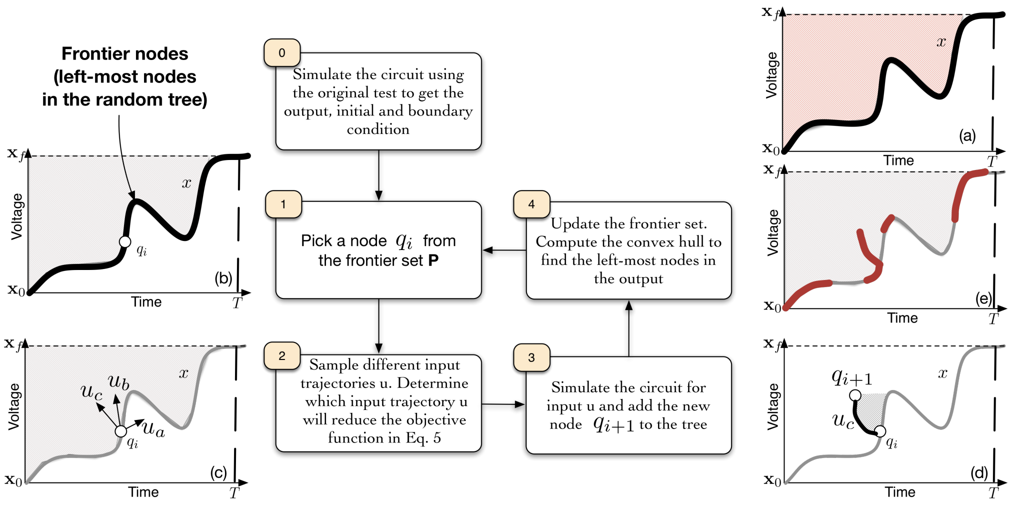

Frontier nodes (left-most nodes in the random tree)
0
Simulate the circuit using the original test to get the output, initial and boundary condition
1
Pick a node q_i from the frontier set **P**
2
Sample different input trajectories u. Determine which input trajectory u will reduce the objective function in Eq. 5
3
Simulate the circuit for input u and add the new node q_{i+1} to the tree
4
Update the frontier set. Compute the convex hull to find the left-most nodes in the output
Voltage
Time
x_f
x_0
T
x
q_i
q_{i+1}
u_a
u_b
u_c
(a)
(b)
(c)
(d)
(e)

Figure 8.5: Random tree algorithm to minimize flux functional.

states and has a minimum time among other states for the same voltage. The *frontier set* is the set of time-optimal states in the random. Intuitively, the frontier set is the set of left-most states on the signal envelope of the random tree. The frontier set contains the candidate states that can minimize the objective function. We update the frontier set by computing the convex hull of the states in the random tree algorithm and sorting the nodes according to their times.

Each iteration in our algorithm consists of the steps (0-4) shown in Figure 8.5. First, the state x_i is selected to branch from. The algorithm picks a state from the frontier set (Fig 8.5-block 1). Secondly, the algorithm determines the input test u_i. It randomly samples multiple different input trajectories and picks a trajectory that reduces the objective function using the Jacobian[98] of the circuit at the state x_i. The Jacobian linearizes the circuit in vicinity of x_i to choose the optimum trajectory. Thirdly, the algorithm then simulates the nonlinear circuit from x_i for time dt by applying the input trajectory u_i to get the next state x_{i+1}. The simulation time for each edge is small (Fig 8.5-block 2). Finally the algorithm adds the state x_{i+1} to the random tree and updates the frontier set accordingly. If the state x_{i+1} reduces the objective function (Equation 8.3), it changes the convex hull of the nodes and updates the frontier set. The frontier set is highlighted in Figure 8.5-e with the bold lines (Fig 8.5-block 3). The algorithm terminates whenever i) the objective function converges to a minimum, or ii) maximum number of iteration is reached. We compute the compressed input sequence

$\tilde{\mathbf{u}}$ by traversing the tree from the root to a node in the frontier set that satisfies the boundary value $\mathbf{x}_f$ and concatenating the input u_i on each edge into a sequence $\tilde{\mathbf{u}}$. The result is the compressed input sequence $\tilde{\mathbf{u}}$.

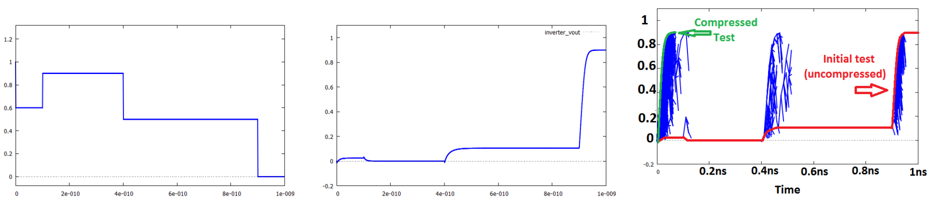
Choosing the optimum trajectory at each iteration: In our case, the random tree algorithm operates in two modes: 1) Test compression mode, where the objective is to compress the given test in time, and 2) Directed test compression mode, with the additional constraint of the test's boundary conditions. We use a technique presented in [88] to bias the growth of the random tree toward the final boundary state.

In our case, we select the optimum trajectory (input) at every iteration according to i) how much the given trajectory decreases the objective function, and ii) how close it drives the test to the boundary condition. We select the optimum trajectory (input) at every iteration according to (i) how much the given trajectory decreases the objective function. and ii) how close it drives the test to the boundary condition. We generate a few sample inputs v_i from a uniform distribution. For each sample, we compute the approximate circuit trajectory u_i from v_i using the Jacobian of the circuit. We rank the sampled inputs according to the Equation 8.4.

$$r_{u_i} = \alpha \Delta f + (1-\alpha) d(x_f, x_i + \Delta t u_i) \tag{8.4}$$

where α is the weight determining mode of the operation, x_f is the boundary condition, $d(x_f, x_i)$ is the Euclidean distance between x_f and current state x_i, and Δf is the approximate negative difference in flux functional. The formula $x_i + \Delta t u_i$ gives us the approximate result of the simulation for a small simulation time Δt.

When our algorithm operates in test compression mode we choose a trajectory that minimizes the flux functional in Equation 8.3. When we want to enforce the boundary conditions as well, the algorithm chooses the trajectory u_i that minimizes the flux functional and takes us closer to the boundary condition x_f. In Figure 8.5-c, u_c is selected since it takes us closer to x_f.



(a) Initial test input applied to the inverter.

(b) Output of the inverter circuit for the given test.

(c) Compressed output of the inverter circuit, showing 94% compression in the stimuli length.

Figure 8.6: We use the inverter circuit as an illustrative example for test compression.

8.3.3 Termination

The random tree algorithm will terminate after running for a fixed number of iterations. After termination, the algorithm lists every state that meets the boundary conditions of the test. We sort these states according to their finishing time. Then we pick the state with the minimum finishing time. We extract the compressed test sequence by traversing the random tree from the candidate state toward the root of the tree. The resulting test is guaranteed to be functionally equivalent to the original test since the candidate state meets the boundary condition.

8.3.4 Compressing tests for the CMOS inverter circuit: an Illustrative example

We compressed functional tests for a post-layout CMOS inverter circuit (Figure 8.6). Figures 8.6a-8.6b show an example input stimulus and the corresponding output for the inverter circuit. We extracted the netlist from the layout of the 65nm inverter circuit with all the parasitics. We wish to test whether the V_{OH} could reach 0.9V. Initially the designer provided an input sequence $\mathbf{u}(t)$ shown in Figure 8.6a in the PWL (piecewise linear) format. The corresponding output of the inverter $\mathbf{x}(t)$ is shown in Figure 8.6b. It takes 950ps for the circuit to reach V_{OH} for the given input sequence. In this case, we have initial value $\mathbf{x}_0 = 0$V and boundary value $\mathbf{x}_f$ =0.9V. The initial test took 950ps to reach the objective so the length of the initial test is 950ps.

Figure 8.6c shows a test for an inverter where $V_{OH} = 0.9v$ was the stress test objective. After $300,000$ iterations, the random tree reached V_{OH} in 52ps with a compression rate of 94%. The random tree tried to switch the input to 0 at 100ps and 400ps before converging to optimum switching time as 0ps. Figure 8.6c shows multiple output traces overlapped in one figure. We pick the path that reaches the final state $v_{out} = 0.9$ first.

8.4 Test compression in the presence of process variation

Process variation, especially at sub-65nm technology process, introduces variability in analog designs. The variation in the chip manufacturing may introduce randomness in physical and electrical characteristic of the analog circuits. The process variation is due to variation in process parameters, variation in lithography process, *etc.* These variabilities include uncertainties in device geometries and mismatch in sizing.

From the test compression perspective, process variation can increase the rate of false positives and decrease yield. For example, a compressed test for the ideal circuit might not pass on the worst-case circuit with variation, even if the design is still within the specification limit. It is important to design and compress tests for the worst-case behavior of the circuit. However, the worst-case corners of the circuits are test-dependent and are not known a priori.

In this work we consider static process variation. Static variation affects physical model of the circuit such as transistor widths, lengths, etc. The static variations are randomly sampled at the beginning of the simulation and will not change during the simulation. We assume process variation follows a truncated Gaussian distribution. The analog circuit can be viewed as a statistical entity with n random variables that model the process variation. Let $V = \{v_1, \ldots, v_n\}$ be a set of n random variables. Each variable $v_i \in V$ is independent and has a real value and belongs to the interval $[v_i^{min}, v_i^{max}]$, corresponding to a different process corner. We assume for each input test u_i, there are n instances $u_{i1}, \ldots, u_{in}$ corresponding to different process corners.

We propose two approaches to test circuits in the presence of process variation. These approaches are independent and can be used at the same time. Firstly, we can compress every test for different corners where the electrical

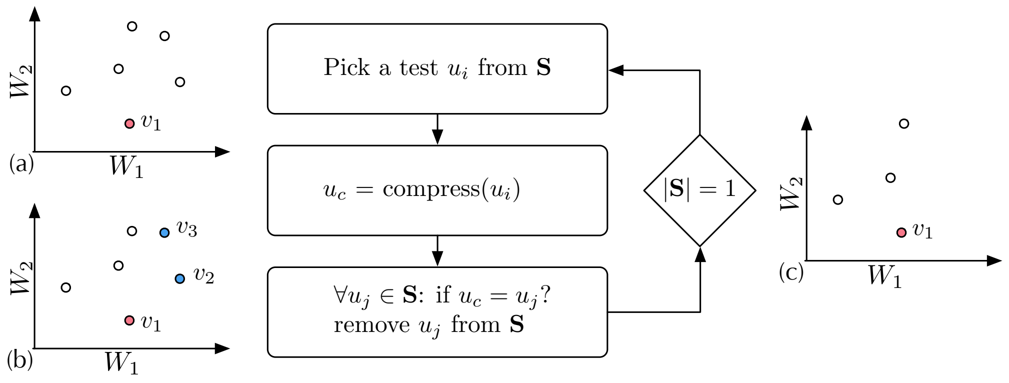


Figure 8.7: The flowchart of the greedy algorithm for compress test in the presence of process variation.

activity excitation profile might vary. A circuit can have a slightly different electrical activity excitation profile in different process corners for the same functional test. During testing, the user can apply the appropriate test for each specific die by identifying where it lies in the process spectrum using on-die process monitors such as ring oscillators. This approach requires compressing all test patterns. Secondly, we observed many of the test patterns for the slower corners are functionally equivalent and can be used for the faster corners as well. In this approach we use a greedy algorithm to reduce the total number of tests n in the presence of process variation.

We use a greedy algorithm to find the worst-case corner for the given compressed test. This algorithm will identify the most applicable test for different corners and use those tests to replace as many tests as possible in the faster corners. The algorithm will eventually converge to a compressed test that is slow enough to cover every the worst-case corners.

The test compression algorithm in the presence of process variation is shown in Algorithm 11. Let u denote the analog stress test for the circuit M. Let $u_1, \ldots, u_n$ denote the instances of the test u for different process corners. The timing specifications are varied among these instances, but they are functionally equivalent. These instances are automatically generated using our goal-oriented input stimuli generation algorithm [88] to be functionally equivalent to the test u for different process corners v_i. Let set S denote the set of all tests $u_1, \ldots, u_n$ for different process corners.

At every iteration, the algorithm randomly picks a process corner v_i and the corresponding test u_i. We use our test compression algorithm (Figure 8.5)

Algorithm 11 Test compression in the presence of process variation

```
1: Inputs: circuit netlist, process corners $V = v_1, \dots, v_n$, input tests $u_1, \dots, u_n$
2: Initialize the test
3: Let Queue $S = \{u_1, \dots, u_n\}$
4: while S.size()==1 or converged() do
5:     $u_i$ = pick an input test from $S$
6:     $u_c$ = compress input $u_i$ for the given process corner $v_i$
7:     for all $u_j \in S$, simulate the circuit with test $u_j$ and process corner $v_j$
8:     Remove all tests $u_j \in S$ from $S$ that are functionally equivalent to $u_c$
9:     Push $u_c$ to the end of the queue $S$
10: end while
```

to compress the test u_i and obtain the compressed version u_c. For each remaining process corners v_j, we simulate the circuit with both input u_j and u_c to see if u_c is functionally equivalent to the tests u_j or not. If u_c can replace the uncompressed test u_j, it means that the process corner v_i is slower than v_j. Therefore a test for the process corner v_i can also be used to test the circuit at the process corner v_j. We will remove the test u_i from the set S and use u_c instead. Finally, we push back the compressed test u_c to the end of the queue. The algorithm will terminate when we only have 1 test left in the queue. The last test is guaranteed to be functionally equivalent to all other tests in the original test batch. The algorithm also terminates when we cannot replace any more tests. We use the last remaining test a compressed test for all process corners.

Figure 8.7 shows the overview of our greedy algorithm for test compression in the presence of process variation. The figure also shows an example of different process corners $v_1, \dots, v_n$ for the circuit. At the beginning, our algorithm randomly picks process corner v_1 and corresponding test u_1 from the set **S**. The algorithm compresses test u_1 to obtain u_c. Say u_c is functionally equivalent to the tests u_2 and u_3 in the set **S** from corners v_2 and v_3 in our example in Figure 8.7.b. As a result, we can use corner v_1 instead to replace corners v_2 and v_3. Next, the greedy algorithm removes corner v_2 and v_3 from the set **S**. The algorithm continues until there is only one process corner is left in **S**, which corresponds to the worst-case corner for that test.

8.5 Parallel test compression

Nowadays, powerful multi-core workstations are inexpensive and ubiquitous. The engineers, on the other hand, are very expensive. For a large-scale circuit, compressing every test can take tens of engineering man-hours. It is essential to utilize parallelization to minimize the time required by the test compression algorithm in order to increase efficiency, minimize research and development costs and reduce time-to-market.

We introduce algorithmic parallelism to our test compression algorithm. During our experiments, we observed that a majority of the test compression runtime (about 98%) was spent on simulating the circuit using HSPICE (block 3 in Figure 8.5). Furthermore, the SPICE simulations are executed independently and tend to be more parallelizable. Executing multiple simulations concurrently greatly reduces the compression runtime. On the other hand, the other steps of the random tree algorithm, including picking nodes from the frontier set, generating inputs and updating the frontier sets, take up less than 2% of the runtime of the algorithm. Parallelizing them does not improve performance significantly. We execute those blocks sequentially in order to avoid memory corruptions and deadlocks.

Figure 8.8 shows the parallel version of the random tree algorithm. Assume we have n processors core $p_1, \ldots, p_n$. We use thread p_1 as the master and $p_1, p_2, \ldots, p_n$ as worker threads. Thread p_1 runs the compression algorithm and maintains the random tree data structure. All accesses to the random tree, including insertions and lookups in the frontier set, are synchronized in a single-thread. We use superscript notation $q_i^{(1)}$ to indicate that processor p_1 is working on node q_i. At the beginning of the i^{th} iteration, thread p_1 selects n nodes from the frontier set $\mathbf{P}$, namely $q_i^{(1)}, \ldots, q_i^{(n)}$. The algorithm generates n sample trajectories $u_i^{(n)}, \ldots, u_i^{(n)}$ for each node q_i. Next, the algorithm spawns n threads, each simulating the netlist from initial state $q_i^{(j)}, 1 \leq j \leq n$ with the input trajectory $u_i^{(j)}$. Each thread is assigned to a worker thread p_j for execution and *concurrently* simulates the circuit. After the simulation is finished, the algorithm updates the frontier set with the new nodes $q_{i+1}^{(1)}, \ldots, q_{i+1}^{(n)}$.

To evaluate the efficiency of parallelism, we compressed the input to an opamp circuit (in Section 9.4) using both single-threaded and multi-threaded version of our algorithm. We executed both versions on a machine equipped

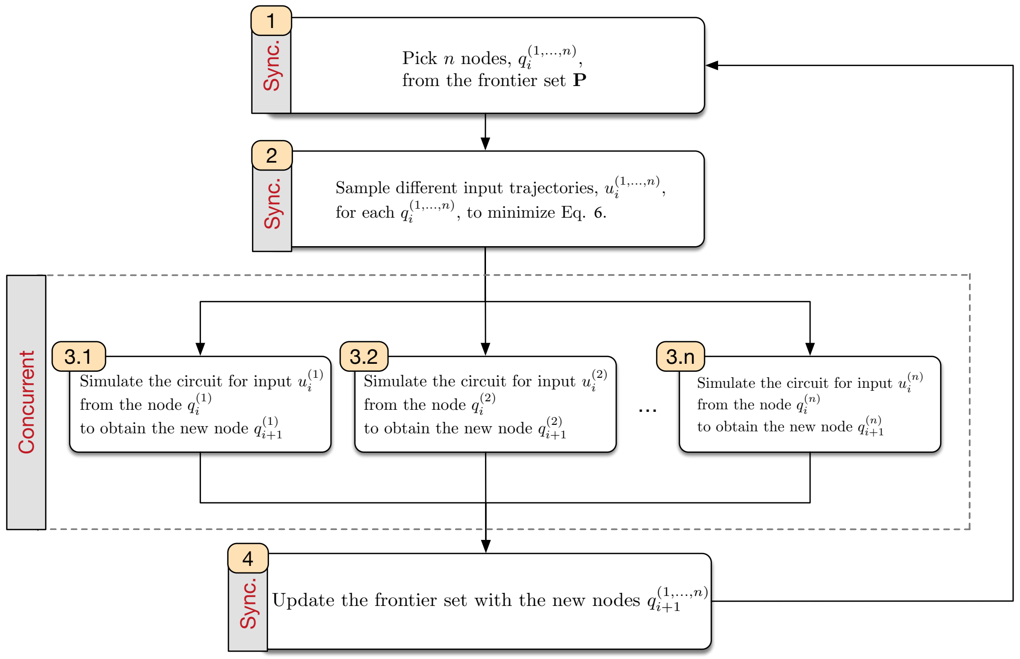

1
Sync.
Pick n nodes, $q_i^{(1,\dots,n)}$, from the frontier set $\mathbf{P}$
2
Sync.
Sample different input trajectories, $u_i^{(1,\dots,n)}$, for each $q_i^{(1,\dots,n)}$, to minimize Eq. 6.
Concurrent
3.1
Simulate the circuit for input $u_i^{(1)}$ from the node $q_i^{(1)}$ to obtain the new node $q_{i+1}^{(1)}$
3.2
Simulate the circuit for input $u_i^{(2)}$ from the node $q_i^{(2)}$ to obtain the new node $q_{i+1}^{(2)}$
...
3.n
Simulate the circuit for input $u_i^{(n)}$ from the node $q_i^{(n)}$ to obtain the new node $q_{i+1}^{(n)}$
4
Sync.
Update the frontier set with the new nodes $q_{i+1}^{(1,\dots,n)}$

Figure 8.8: Parallel version of the random tree algorithm to minimize flux functional. The SPICE simulations are executed concurrently.

with quad-core single CPU with 6MB $L3$ cache and 16GB memory with no hyper-threading. Utilizing parallelization, reduced the runtime of our algorithm from 3 hours to less than 45 minutes, yielding 70% overall speedup versus single-threaded execution.

8.6 Experimental results

We developed a prototype tool in $C{+}{+}$ to evaluate the accuracy and efficiency of our algorithm. The input to our tool was the circuit netlist in HSPICE format. The user defines the test input signals as a PWL (Piecewise linear) sources. Our tool used Synopsys HSPICE to simulate the circuit. The output of our tool was another PWL signal that can be used to test the circuit. We ran each experiment for 100,000 iterations. Each iteration consisted of a small HSPICE simulation for duration of $dt = 1\mu$s. Each experiment took approximately 3 hours to be completed on a quad-core Core-i5 machine equipped with 16GB of memory. When we enabled parallel version of our algorithm, the same experiment took less than 45 minutes. Table 8.1 lists the parameters that we used in our experiments.

Table 8.1: Parameters of the random tree

parameter	value	description
max-iter	10000	The maximum number of iterations
tree size	10000	The size of tree, also the number of HSPICE simulation
dt	$1\mu s$	The length of each edge in the random tree
total time	3 hrs	Total runtime of the random tree algorithm per test
α	0.5	The weight of objectives for ranking states in Equation 6

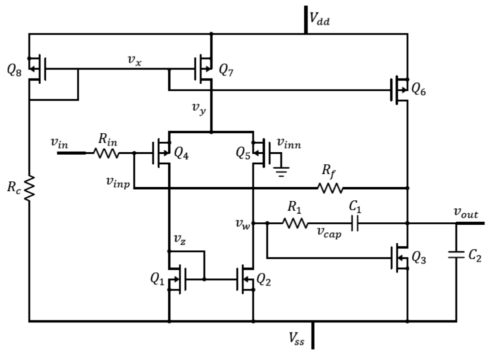
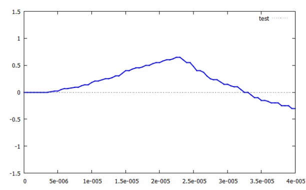


Figure 8.9: Schematic of the operational amplifier circuit.

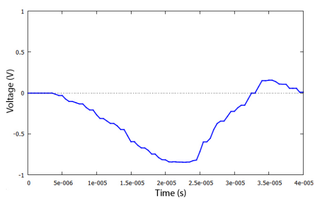
We used an operational amplifier circuit, shown in Figure 8.9, to show practicality, scalability and effectiveness of our algorithm. We used this opamp in a voltage divider configuration with unity gain. The opamp was designed in 0.18μm library. We sat $V_{DD} = -V_{SS}$ =0.9V. Each test was applied to the V_{in} signal. The output of the opamp was saturating at 0.2V and -0.8V respectively. The state space consists of the following variables (Figure 8.9):

$$\{v_c, v_+, v_i, v_{in}, v_{ip}, v_o, v_-, v_w, v_x, v_y, v_z, v_w, t\} \in R^{12,1} \tag{8.5}$$

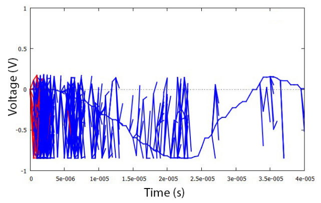
Compressing functional tests: Our first set of results consists of compressing the input stimuli given by the user. The given stimuli stress tests or checks the functionality of the circuit. The output of the opamp circuit saturates at 0.2V and -0.8V. For the saturation test, we applied the input signal shown in Figure 8.10a to the V_{in}. The input signal is a standard test with a period of a sine wave at 10KHz combined with a ramp voltage and a white noise. Figure 8.10b shows the output of the opamp. The circuit was saturated at -0.8V at time 23μs. The circuit was saturated again at time 36μs when the output reached 0.2V. The test takes 40μs to finish. We used our test compression algorithm to compress the saturation test input for the



(a) Uncompressed test input to the opamp.

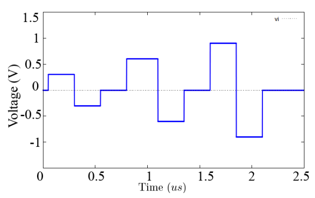


(b) Original test output of the opamp.

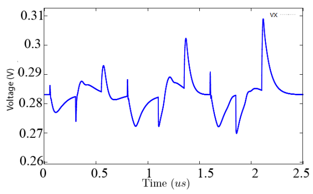


(c) Random tree after the test compression algorithm.

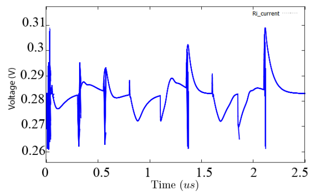
Figure 8.10: Saturation test for the opamp circuit.



(a) Stress testing resistor R_1.



(b) The current profile of the resistor R_1.



(c) Random tree stressing resistor R_1.

Figure 8.11: Using random trees to compress stress tests for circuit's components.

opamp circuit. Figure 8.10c shows the random tree as a result of our algorithm. The random tree algorithm found an alternative test that reached both saturation voltages -0.8V and 0.2V in 2μs and 1μs, respectively. We extracted a compressed test of the length 3μs, achieving a compression ratio of 92.5%.

Compressing stress tests: We used the input shown in Figure 8.11a to stress opamp's current source and determine maximum drivable current through resistor $R_c = 2.1\text{K}\Omega$. Current profile simulation showed the maximum current was 576mA, when switching from V_{DD} to V_{SS} in the input at 2.12μs. Our algorithm found an alternative test that could drive the same current from resistor R_c in 80ns, whereas the length of the original test was 2.12μs. The compression ratio for the test was 96%. In our test, the switch from V_{SS} to V_{DD} occurred at the beginning of the test. The output of the random tree is shown in Figure 8.11c.

Compression ratio consistency: We created a profile for each node in the opamp by applying the input signal in Figure 8.11a to the V_{in} and simulating the circuit. We computed the maximum and the minimum value for

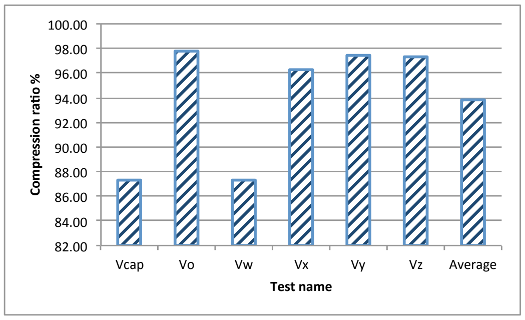
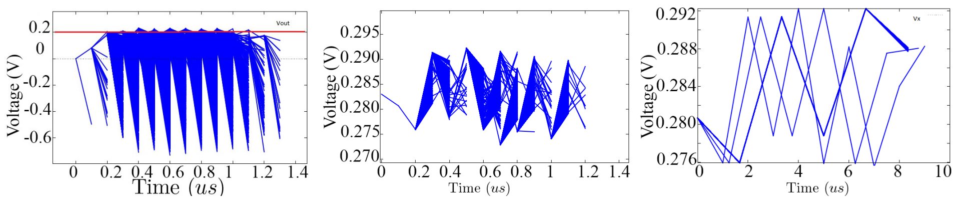


Figure 8.12: Compression ratio for different tests.

the signal. We extracted a test for reaching the extrema of the signal from the profile. Finally, we used our technique to compress each test. Figure 8.12 shows the compression ratio for each tests. On average, we can achieve 93% compression ratio for these tests. **All of our compressed tests are functionally equivalent to the original tests.**

Directed test compression mode We demonstrate an example of the directed test compression mode of our algorithm. In this mode, our algorithm simultaneously generates and compresses tests that are time-optimal. We use the technique presented in [88] to guide the growth of the random tree toward the boundary condition. At each iteration, we choose a trajectory toward the boundary condition with the minimum flux functional as described in Section 8.3. We set α in Equation 8.4 to 0.5. Sometimes (specially for stress testing) the user knows the objective of the test (such as reaching a specific voltage or current through a node), but the original input test is hard to find. We designed our algorithm to handle such cases where only the final boundary value of the test is known, but the path from the initial to final state is unknown.

Generating and compressing single tests: We set up our algorithm to generate a test that would saturates the output of the op-amp. We set the hyper planes $v_o = 0.2\text{V}$ and $v_o = -0.8\text{V}$ as the goal boundary conditions. We used the random tree to generate two input stimuli that saturates the outputs of the circuit at 0.2V and -0.8V. We ran the algorithm twice to reach the output voltage 0.2V and -0.8V. Figure 8.13a shows the output of the circuit when the random tree was directed toward saturating the output at 0.2V. We extracted multiple input stimuli from the random tree that saturates the



(a) Auto-generated and compressed test to saturate the output.

(b) The voltage v_x in the random tree of the combined tests for saturating output v_o and stressing resistor R_c.

(c) Extracting combined tests from random tree.

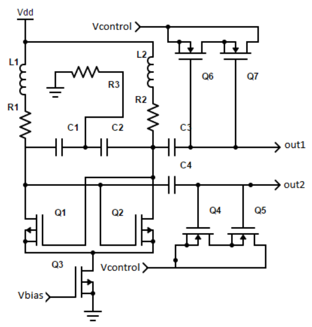
Figure 8.13: Combining different tests.

output voltage. The length of the generated test is 30μs.

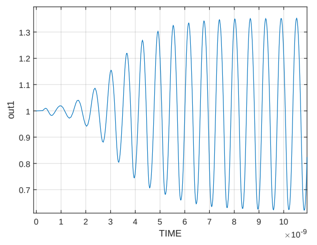
Combining multiple tests We used the random tree with directed test generation to combine multiple tests into a single test. The combined test should reach the boundary conditions of all of those tests. We generated two sets of tests g_1 and g_2. Tests g_1 saturate the output voltage of the opamp and g_2 tests stress the resistor R_c by maximizing the current through it. The objective was to generate a new random tree that can *simultaneously reach both boundary planes* g_1 and g_2. First we collected the terminating states in random tree g_1 and g_2 as in [88]. The goal distribution is a mixture Gaussian distribution with the mean $(v_o, v_x) = (0.2, 0.3)$. We grow the random tree toward the boundary state (0.2V, 0.3V). Figure 8.13b shows that the random tree explored both the boundary planes simultaneously and combined tests. Figure 8.13c shows the extracted tests from the combined random tree that saturates the output and stresses resistor R_c.

Compressing tests for voltage controlled oscillator: Voltage controlled oscillator (VCO) circuits are widely used in RF circuits, frequency synthesizers and phased-locked loops. We used a VCO circuit, as shown in Figure 8.14a, to demonstrate the practicality and scalability of our algorithm. Figure 8.14b shows the standard output of the VCO circuit where V_{bias} =750mV, $V_{control}$ =630mV and V_{DD} = 1.8V. It takes 10.8ns for the output to reach its peak-to-peak maximum. The maximum output is 1.34V.

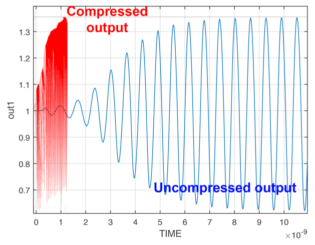
We used our tool to compress tests for the VCO circuit. We defined three transient inputs to the circuit: $0.75 \leq V_{bias} \leq 1$, $0.65 \leq V_{control} \leq 1$ and $1.8 \leq V_{DD} \leq 2$. We executed the random tree for 10,000 iterations. The algorithm



(a) Schematic of the VCO circuit.



(b) The uncompressed output of the VCO circuit.



(c) The compressed output of the VCO circuit.

Figure 8.14: Compressing tests for VCO circuit. Our technique compressed VCO swing tests by 88%.

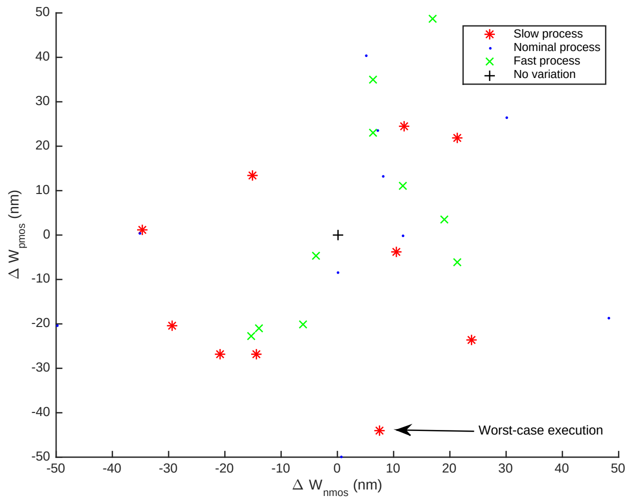


Figure 8.15: Process variation in the width of NMOS and PMOS transistors in the inverter circuit and worst-case corner.

took 30 minutes and produced an output that reached the maximum output voltage within 1.2ns. In comparison to the original tests, the compression ratio was 88%. Our generated input stimuli cannot be used to test the oscillation of the VCO, but it can be used to validate the output swing, defect screening and stress testing the circuit.

Compressing tests in the presence of process variation We used the inverter circuit, as shown in Figure 9.1, designed at 45 nm process with process variation as a case study. We modeled process variation in transistor width as a truncated Gaussian distribution with σ =25nm. The nominal width of NMOS and PMOS transistors were 415 nm and 630 nm, respectively. We randomly generated a batch of 10 different variation instance for the three process corner, including fast, nominal and slow. The total number of tests was 30. We set the length of the test for the slowest corner to be 100 ps, i.e. at 100 ps, the output must be equal to V_{OH}. Figure 8.15 shows the variation in the width of the NMOS and PMOS transistor in the inverter circuit.

We compressed the batch using our greedy test compression algorithm (Algorithm 11). The random tree in the inner loop of the greedy algorithm was executed four times to compress four different corners. The algorithm finally picked the process corner $(\Delta W_{\text{nmos}}, \Delta W_{\text{pmos}}) =$ (8 nm, -45 nm) as the worst-case corner and compressed test for that corner. The compressed test is applicable to all other corners in the circuit.

In this particular experiment, we were checking the output-high of the inverter circuit, which would stress the PMOS transistor. As a result, the greedy algorithm found the narrowest PMOS transistor as the worst-case corner for that particular test. The worst-case corners are *test dependent* and for different tests, the worst-case corner will be different. Worst-case corners cannot be identified during design without first understanding and analyzing test stimuli.

8.7 Chapter summary

In this chapter, we used Duplex for automated test compression for electrical stress testing of analog and mixed signal circuits. Duplex optimally extracts only portions of a functional test that electrically stress the nets and devices of an analog circuit. We modeled the test compression as a type-III Duplex functional optimization problem. We demonstrated with an op-amp, VCO, and CMOS inverter that the method consistently reduces the length of each test by an average of 93%. Duplex can compress tests in the presence of process variation and utilize parallel processing to speed up the compression algorithm.

CHAPTER 9

CIRCUIT OPTIMIZATION

9.1 Introduction

9.1.1 Optimizing performance of analog circuits by searching the parameter space

The goal of performance optimization is to find an optimum assignment to the circuit's parameters, such as transistors' widths and lengths, that optimizes the circuit's performance metrics such as gain, bandwidth and power.

We model the performance optimization problem as a type-II duplex optimization problem. Duplex determines the optimal design, the Pareto set and the sensitivity of circuit's performance metrics to its parameters. Duplex does not get stuck in local minima and provides valuable feedback to the user in the form of performance to parameter sensitivity graphs and Pareto set. Duplex is also highly performance efficient and scalable, as demonstrated in our results.

9.1.2 Using Duplex algorithm for optimizing analog circuits

Duplex uses random tree search, a tree based simulation algorithm that also maintains the tree data structure as a record of the state space traversed. It maintains and simultaneously grows two homomorphic (mirrored) random trees; one in the parameter space and the other in the performance space. In the performance space, it uses the basic random tree search to find the globally optimal design by expanding the tree toward the *goal region.* In the parameter space, it decides which parameter needs to change to get closer to the goal region. This decision is made using a reinforcement learning algorithm [132] that evaluates the history of previous changes in the parameter tree based on a reward function. Duplex does not get stuck in local minima because of the probabilistic completeness property of random trees[43]. This is in contrast to random walk based methods like simulated annealing. The guidance in every step from the global search towards the local step decision helps in converging quickly to the optimal goal region.

The choice of random trees contributes to most of the advantages of Duplex. Random trees generally search the space more efficiently than Monte Carlo based simulation methods [43, 89, 133, 134], contributing to Duplex's efficiency. Additionally, during the course of the simulation, a unified tree structure connecting performance and parameter space of the circuit is maintained. This helps generate by-products like Pareto surfaces and sensitivity analysis that provide design insights. Computing the exact Pareto surface is typically computationally very expensive [30, 28] since Pareto sets are high dimensional surfaces in the parameter space. Duplex uses a statistical inference algorithm to infer the distribution of optimal states in the tree in the parameter space. It uses the inferred distribution of optimal parameter states as a Pareto surface. In addition, Duplex keeps track of how variations in a given parameter cause performance metrics to change and retrospectively generates a performance to parameter sensitivity graph. This is in contrast to typical algorithms that do not record circuit information during the optimization process.

9.1.3 Benefits and contributions of using the Duplex algorithm

Duplex is a scalable algorithm and can optimize system-level post-layout circuits. We demonstrate duplex's scalability by optimizing a system-level post-layout 1.6 GHz charged-pump PLL circuit [160] (with 131 CMOS transistors) as shown in Figure 9.10. Duplex's scalability is due to its open-ended search being restricted to the performance space, that tends to be much smaller than the parameter space, greatly reducing the size of the search space. The size of the parameter space depends on the size of the circuit. The PLL circuit has over 100 CMOS transistors, but the performance space has only 5 dimensions (Table 9.2). The complexity of Duplex is not dependent on circuit size, allowing it to scale easily.

Duplex is computationally highly efficient. We demonstrate that Duplex has an 81% (up to 5×) more speedup as compared to state-of-the-art results [31] on the same design (two stage operational amplifier [31]). Notably, this design has local minima. Although a few branches in the parameter tree grow toward the local maxima, Duplex used the performance tree to grow toward the global optimum and successfully converged toward the global maximum (resulting in an opamp with 5 GHz bandwidth), unlike [31] which got stuck in a local minima (reporting 2 GHz as a maximum bandwidth for the circuit) (Figure 9.6). We also demonstrate Pareto surface computation and sensitivity analysis (Section 9.4.1). Duplex is a stable algorithm with little variance in its execution with different initial states. We demonstrate Duplex's stability by running it multiple times on a CMOS inverter circuit and optimizing the inverter for power and delay (Figure 9.7).

Our contributions are as follows. We present Duplex random tree search for performance space optimization of analog circuits that is more efficient than state-of-the-art. Duplex is inherently scalable and does not get stuck in local minima. We provide a simple and efficient technique for Pareto surface generation. We also present a technique to analyze the relative sensitivity of a parameter with respect to performance. With Duplex, we present the idea of optimization by simultaneously traversing dual spaces.

9.1.4 Chapter organization

In this chapter, initially we provide a model for the performance optimization problem that is similar to type-II duplex problems in Section 3.8.3. Then we propose using Duplex to solve the optimization problem in Section 8.3. Finally, we demonstrate the Duplex algorithm by optimizing an opamp circuit and a charge-pump PLL circuit. We show that Duplex is 5× faster than the state-of-the-art and finds the global optimum for a design whose previously published result was a local optimum. We show our algorithm's scalability by optimizing a system-level post-layout charged-pump PLL circuit in Section 9.3.

9.2 Optimization model

For a given circuit topology and process technology, performance of the circuits is measured with respect to metrics such as gain, slew rate and bandwidth. The circuit's performance depends on parameters such as transistor width, length, resistor and capacitor values. Let $P \in \mathbb{R}^n$ and $Q \in \mathbb{R}^m$ denote the parameter and performance space, respectively. Let n and m denote the number of parameters and performance metrics, respectively. We refer the points in the performance and parameter space as a *performance and parameter states*, respectively.

We model physical constraints of the circuit and manufacturing process as constraints in the parameter space. For example, for the inverter circuit in Figure 9.1, transistor M_1 could have a width constraint $1\mu\text{m} < M_1 \leq 10\mu\text{m}$. The parameter space may also have equality constraints enforced by layout design rules. For example, the width of transistor M_1 should be twice the width of transistor M_2.

The parameter and performance variables can each have different scales (say Nano to Giga) and measuring units. We normalize across them by mapping every variable to the interval $[0, 1]$. In general, the size of a parameter space n is related to the number of components (size) of the circuit.

The constrained parameter space is a subset of the parameter space, bounded by the physical constraints of the circuit. A constrained parameter space is modeled as an intersection of k inequalities:

$$P = \{\mathbf{p} \mid \mathbf{C}\mathbf{p} \leq \mathbf{b}\} \tag{9.1}$$

where $\mathbf{C}$ and $\mathbf{b}$ are $k \times n$ and $n \times 1$ matrices. For each $\mathbf{p}$ in the set P, all sizing requirements of the circuit are met.

A *performance (parameter) variable* is a variable in the performance (parameter) space. A *performance (parameter) state* is a vector value assignment to all the performance (parameter) variables in the circuit. The relationship between parameter and performance spaces is shown in Figure 9.2. An instance of the circuit with a given parameter state corresponds to a specific performance state. This can be viewed as an *onto*, or many-to-one mapping f from many parameter states to one performance state. We can only evaluate mapping f point wise using numerical simulation (such as HSPICE).

The reachable performance space is the image Q of the constrained

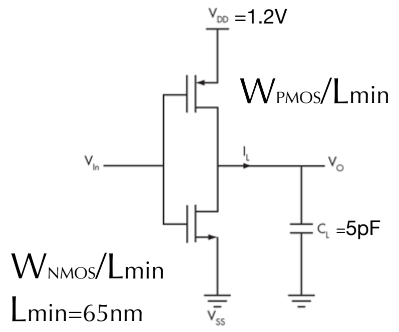


Figure 9.1: Schematic of an inverter circuit that we use as an illustrative example. We want to optimize the width of NMOS and PMOS transistors to minimize dynamic power and delay.

parameter space P in the performance space.

$$Q = \{\mathbf{q} \mid \mathbf{q} = f(\mathbf{p}) \text{ where } \mathbf{p} \in P\} \tag{9.2}$$

The goal region is a subset of the reachable performance space where the performance of the circuit is within the acceptable range, set by the designer.

$$Q_{\text{goal}} = \{\mathbf{q}_{\text{goal}} \mid \mathbf{G}\mathbf{q}_{\text{goal}} \leq \mathbf{d}\} \tag{9.3}$$

where $l \times m$ matrix $\mathbf{G}$ defines the constraints on the goal region. **The optimal state** is any state in the *goal region* of the performance space.

For the inverter in Figure 9.1, parameter space is R^2 and consists of the widths of NMOS and PMOS transistors. The optimization goal is to minimize power, rise and fall time delay of the circuit. Specifically, we want *power* $\leq$ 100μW and *delay* $\leq$10 ps. An example of the parameter state is $(w_{pmos}, w_{nmos}) = (4\mu\text{m}, 2\mu\text{m})$. Similarly, a performance state is a vector $(p, d_{rise}, d_{fall}) = (50\mu\text{W}, 5\text{ps}, 4\text{ps})$.

9.3 The Duplex random tree search algorithm

9.3.1 Random tree search

The random tree is a tree structure [89, 43, 134] that is constructed in the continuous space $\mathbb{R}^n$. Each node in the tree is a vector in $\mathbb{R}^n$. Each node can

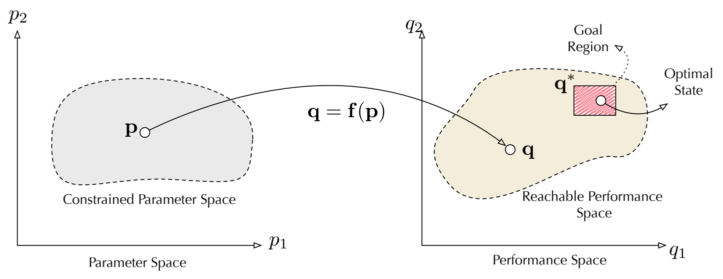


Figure 9.2: The relation between *constrained parameter space* (left) and the *reachable performance space* and the *goal region* (right).

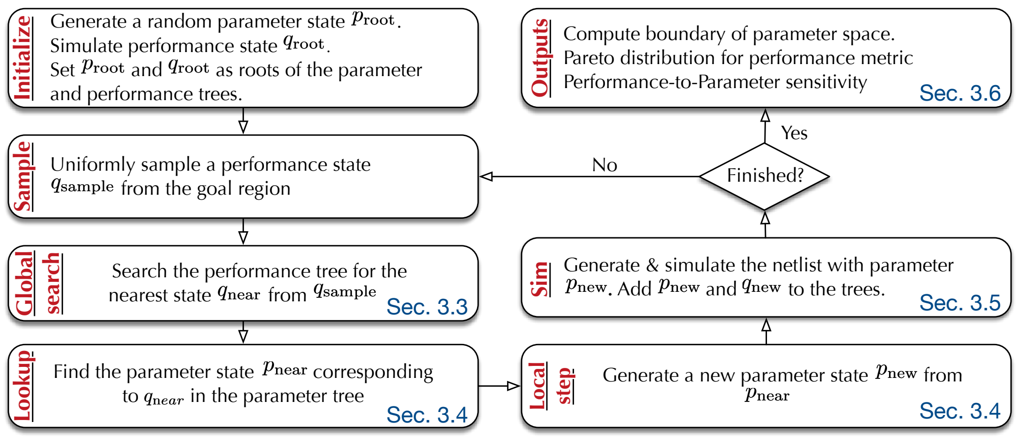

Initialize
Generate a random parameter state p_{root}.
Simulate performance state q_{root}.
Set p_{root} and q_{root} as roots of the parameter and performance trees.
Sample
Uniformly sample a performance state q_{sample} from the goal region
Global search
Search the performance tree for the nearest state q_{near} from q_{sample}
Sec. 3.3
Lookup
Find the parameter state p_{near} corresponding to q_{near} in the parameter tree
Sec. 3.4
Local step
Generate a new parameter state p_{new} from p_{near}
Sec. 3.4
Sim
Generate & simulate the netlist with parameter p_{new}. Add p_{new} and q_{new} to the trees.
Sec. 3.5
Finished?
No
Yes
Outputs
Compute boundary of parameter space.
Pareto distribution for performance metric
Performance-to-Parameter sensitivity
Sec. 3.6

Figure 9.3: Flowchart of the Duplex random tree search algorithm for performance optimization.

have multiple children. The tree is initialized by fixing its root to a specific state in the space. The random tree is constructed incrementally.

Random trees are shown to consistently outperform random walk based search methods such as Monte Carlo simulations for search applications [89, 133, 134]. Efficiency improvement can be credited to the data structure maintained by the random tree algorithm during the simulation. While growing, it samples a new state in the goal region (desired solution set), and then determines which state is closest (in $\mathcal{L}_2$-norm sense) to that sampled goal state among all of the previously visited states in the tree. It simulates a path between the closest state and the newly sampled state and adds the new state to the tree. This is in contrast to the memory-less sampling of points in the Monte Carlo based methods.

Duplex simultaneously constructs and maintains two different, but mirrored (homomorphic), random trees in the performance and parameter space.

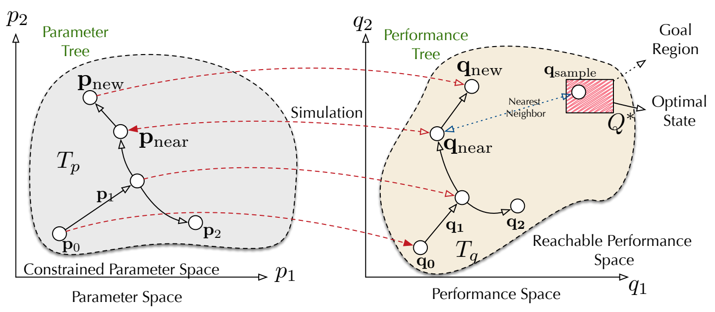


Figure 9.4: Growing parameter and performance tree in the parameter and performance space.

Figure 9.4 shows the parameter and performance random tree growing in the parameter and performance space. Intuitively, the performance tree is the *mirror* of the parameter tree in the performance space. Let T_q denote the performance tree and T_p denote the parameter tree. Let $\mathbf{q}^{(i)}$ and $\mathbf{p}^{(i)}$ denote the i^{th} nodes in the performance and parameter tree, respectively. Let $\mathbf{Q}^*$ denotes the goal region in the performance space.

These trees represent different relationships. An edge in the parameter tree indicates that the two parameter states connected to that edge differ in *exactly one* variable. An edge in the performance tree between indicates that the corresponding states in the parameter tree are connected. For each node $\mathbf{p}$ in the parameter tree, there exists a corresponding node in the performance tree, and vice versa. The corresponding node in the performance tree is computed by simulating the circuit with the given parameters.

For the inverter circuit, each node in the parameter tree is a two-dimensional vector $\mathbf{p} = (w_{\text{nmos}}, w_{\text{pmos}})$, corresponding to the width of NMOS and PMOS transistors. Each node in the performance tree is an assignment of vector of performance metrics $\mathbf{q} = (power, d_{\text{rise}}, d_{\text{fall}})$. Performance node $\mathbf{q}$ is computed by simulating the inverter circuit with the parameter vector $\mathbf{p}$ using HSPICE.

9.3.2 The Duplex algorithm

Figure 9.3 shows the flow of the Duplex algorithm. Duplex advances toward the goal region $\mathbf{Q}^*$ in the performance space. At the i^{th} iteration, it navigates the performance space to get closer to the goal region ($\mathbf{q}^{(i)}_{sample}$). This is the *global search step*. When it finds a close enough state ($\mathbf{q}^{(i)}_{near}$) to the goal region, it looks up the corresponding mirror image of that state in the parameter space ($\mathbf{p}_{near}^{(i)}$). For the mirror state, it finds a neighbor state by perturbing a single parameter in the mirror state ($\mathbf{p}_{new}^{(i+1)}$). This *local step* in the parameter space is the action the algorithm takes based on the guidance from the performance space. A performance state corresponding to the neighbor state is added in the performance space ($\mathbf{q}_{new}^{(i+1)}$). The algorithm continues until it reaches an optimal state in the performance space.

9.3.3 Global search steps in performance space

Duplex biases the search by growing the performance tree toward the goal region. In every iteration, it uniformly samples the goal region to find a candidate optimal state $\mathbf{q}_{\text{sample}}^{(i)}$. Duplex's objective in this iteration is to get closer to $\mathbf{q}_{\text{sample}}^{(i)}$. It finds the closest state $\mathbf{q}_{\text{near}}^{(i)}$ as per Euclidean distance in the performance tree from $\mathbf{q}_{\text{sample}}^{(i)}$. It uses the KD-tree algorithm[43] for efficient search of the tree in the performance space. For this $\mathbf{q}_{\text{near}}^{(i)}$, it then looks up the corresponding state in the parameter space and finds $\mathbf{p}_{\text{near}}^{(i)}$.

9.3.4 Local coordinated steps in the parameter space

In this phase, Duplex's objective is to find a state $\mathbf{p}_{\text{new}}^{(\mathbf{i+1})}$ in the parameter space that is a neighbor of $\mathbf{p}_{\text{near}}^{(\mathbf{i})}$ such that its image $\mathbf{q}_{\text{new}}^{(\mathbf{i+1})}$ will be closer to the goal region. It perturbs exactly one parameter in the parameter state $\mathbf{p}_{\text{near}}^{(\mathbf{i})}$ to obtain a new neighbor state $\mathbf{p}_{\text{new}}^{(\mathbf{i+1})}$. There are three reasons why duplex only perturbs a single parameter (coordination) at each iteration: Firstly, optimizing a circuit with multiple parameters can be done by iteratively optimizing single parameters in rotation, as in the coordinated descent algorithm [4]. Secondly, for circuit design, explaining the results of the learning algorithm is very valuable. By using coordinated steps instead of the gradient, Duplex is able to use reinforcement learning and compute the sensitivity of performance metrics to parameters. Finally, some parameters might not have significant impact on the performance metrics. By coordinating one parameter at a time, we can find these less significant parameters and i) avoid perturbing them in the future and ii) report them to the designer.

Duplex uses a reinforcement learning algorithm [132] to determine which parameter variable ($p_j \in \mathbf{p}_{\text{new}}^{(i+1)}$) to perturb. It uses an annealing learning rate [4] to determine how much to perturb the j^{th} parameter in $\mathbf{p}_{\text{new}}^{(i+1)}$.

Reinforcement learning

Since we treat the circuit as a blackbox, the gradient information is not available. Without the gradient, analytically computing an optimal local step is not possible. Instead, Duplex relies on the history of the previously taken steps to learn what is the best step in the future. Duplex keeps a history of how influential each parameter is in getting closer to the goal region. Every parameter state $\mathbf{p}_{near}^{(i)}$ has a reward vector Q associated with it, that is initialized to all ones at the root of the parameter tree. After each iteration, Q is updated, depending on whether changing the j^{th} parameter resulted in the corresponding performance state getting closer to the goal region or not.

The new neighbor state $\mathbf{p}_{\mathbf{new}}^{\mathbf{(i+1)}}$ differs by the parent parameter state only in parameter j, so we compute the reward vector according to Equation 9.4.

$$Q(\mathbf{p}_{near}^{(i)}, j) \leftarrow Q(\mathbf{p}_{near}^{(i)}, j) + \gamma(\|\mathbf{q}_{new}^{(i+1)}, Q^*\| - \|\mathbf{q}_{near}^{(i)}, Q^*\|) \tag{9.4}$$

where $\|.\|$ is the distance from the performance state $\mathbf{q}$ and the goal region and γ is the discount rate. Next time the parameter state $\mathbf{p}$ is chosen, Duplex uses weighted uniform sampling on $Q(\mathbf{p})$ to select the parameter j.

The reward vector is inherited only by children states of a parent state. Reward vectors on two different paths do not influence each other. This is necessary to avoid making global mistakes in the random tree. Therefore, even if one branch of the tree is stuck in a local minima, the other branches are not affected.

Annealing learning rate

The learning rate α in Duplex is set such that initially we search the **space**, then we **converge** toward the optimum state. Duplex determines the length of each step, the extent to which the new parameter state should differ from the parent state, according to a *learning rate* α. Initially the length of the steps are very high (the search phase), but as we get closer to the optimum state, we anneal (gradually lower) the length of each step in order to converge toward the optimum.

The learning rate depends on the step length of the parent state, and a parameter K, and the initial step length α_0 specified by the user.

$$\alpha_{p_{\text{new}}} = \frac{\alpha_0}{1 + K \times \alpha_{p_{\text{near}}}} \tag{9.5}$$

The sign of the step length is chosen randomly. Duplex adds or subtracts the value of $\alpha_{p_{\text{new}}}$ to the j^{th} parameter in the parameter state vector. In Duplex, unlike other learning algorithms, the learning rate is dependent on the depth of the tree and not the number of iterations. After determining which parameter to change and how much to change that parameter, Duplex generates the new parameter state $\mathbf{p}_{\text{new}}^{(i+1)}$.

9.3.5 Generating the new performance state

From the neighbor state $\mathbf{p}_{\text{new}}^{(i+1)}$, Duplex generates the new performance state by using a numerical simulator like SPICE to evaluate the sampled parameter. The values of the performance metrics (gain, bandwidth etc.) form the state vector of $\mathbf{q}_{\text{new}}^{(i+1)}$. The pair ($\mathbf{p}_{\text{new}}^{(i+1)}$, $\mathbf{q}_{\text{new}}^{(i+1)}$) is added to the parameter and performance trees respectively and the reward vector for p_{new} is updated.

9.3.6 Other outputs: Pareto distribution and sensitivity analysis

After reaching the goal region, the algorithm generates the Pareto surface of the design space. Let S denote the set of performance states in the goal region. Duplex computes the Pareto set by gathering all the corresponding parameter states to the set S. It infers the mixture distribution of the Pareto set using variational Bayesian inference [4].

Duplex also analyzes the *sensitivity* of each performance metric j to each parameter i. It records the result in the sensitivity variable $\mathbf{s}s_{ij}$. Duplex traverses the random tree to determine the number of times a parameter has changed and the extent to which it has changed. In a manner similar to covariance computation, the relative change in the performance due to a parameter is of interest.

Let $f_{j,\mathbf{q}}$ denote the value of performance metric j at state $\mathbf{q}$. At each iteration, if changing parameter i results in change in $f_{j,q}$, we record the difference in variable $\delta_{\mathbf{q}_{new},i,j}$:

$$\delta_{\mathbf{q}_{new},i,j} = |f_{\mathbf{q}_{new},j} - f\mathbf{q}_{near}, j| \tag{9.6}$$

where $\mathbf{q}_{near}$ is the parent state of $\mathbf{q}_{new}$. So $\delta_{\mathbf{q},i,j}$ is the difference of performance j between the new state $\mathbf{q}$ and its parent when we change parameter i. Duplex updates the sensitivity matrix according to $\Delta s_{ij} = |\frac{\delta_{\mathbf{q},i,j}}{f_{j,\mathbf{q}}}|$.

$$\mathbf{s}_{ij} = \sum_{\mathbf{q}} |\frac{\delta_{\mathbf{q},i,j}}{f_{j,\mathbf{q}}}| \tag{9.7}$$

After termination, Duplex normalizes each row (j) in the sensitivity matrix $\mathbf{s}$.

9.3.7 Termination and complexity analysis

The objective of the Duplex algorithm is to reach a goal region. The Duplex algorithm will terminate when it finds sufficient optimal states in the performance tree within the goal region, or if it has reached the maximum number of iterations.[1]

Duplex's approach toward search is a twofold: 1) Global search in the performance space, where it searches for the nearest visited state and biases the search toward the goal region, and 2) local search in the parameter space, where it takes the best action from the given parameter state according to the past simulation history. In comparison to the local search, global search typically provides significant efficiency improvement; However it is very expensive and does not scale beyond 100 dimensions. In duplex, we only perform global search in the performance space, which typically is very small (to the order of tens of dimensions). Since the dimension of performance space is usually very small (in comparison to the parameter space) search in the performance space is very efficient. In our implementation, we used KD-tree data structure [43] as our database for closest state search queries.

[1]Duplex is, in a certain sense, a search algorithm for high-dimensional continuous spaces. Thus, it is technically different from other optimization techniques such as simulated annealing or gradient descent. Unlike optimization methods, Duplex does not try to optimize an objective function after reaching the goal region and meeting the performance requirement of the circuit. Although search algorithms can be used as an optimization algorithms and vice-versa.

Therefore, the complexity of search for Duplex is $O(n \times m \times \log(n))$ where n is the number of iterations and m is the number of performance metrics.

The Duplex algorithm, unlike conventional search methods such as simulated annealing, does not get stuck in local minima of the performance space. Even if some branches of the random tree do get stuck in local minima, the algorithm simultaneously grows other branches outside the minima and converges toward the global optimum. Therefore, the probability of finding the optimum state goes to 1 as times goes toward infinity. This is based on the probabilistic completeness property of the random tree search algorithm [43].

9.4 Experimental results

In order to show the effectiveness, efficiency and scalability of Duplex algorithm we used three case-studies: i) a CMOS inverter, which we used as a proof-of-concept and to analyze the performance and stability of the Duplex algorithm, ii) an amplifier circuit (from [31]), which we used to demonstrate the efficiency of the algorithm and to show that our algorithm does not get stuck in local minima, and iii) a system-level post-layout charge-pump PLL circuit, which we used to demonstrate Duplex's scalability and practicality for high-dimensional system-level circuit.

We use the Duplex algorithm to explore the performance space of a two-stage operational amplifier with frequency compensation [31] as shown in Figure 9.5.[2] The opamp circuit has many parameter and performance variables, demonstrating the scalability and efficiency of the Duplex algorithm. The circuit was designed in 65 nm library and the supply voltage was 1.2V. The main objective is to meet the bandwidth requirement of the circuit. There are 7 design variables in the circuit: (the capacitor C_c, the bias current I_{bias}, and the widths of transistors W_1, W_3, W_5, W_6 and W_8. The other parameters can be calculated from these parameters. Let λ =30 nm. The lengths of all transistors are set to $L_{\text{m}in} = 10\lambda$ to meet an acceptable output resistance and intrinsic gain.

We executed Duplex to optimize the parameters in order to meet the specifications shown in Table 9.1. We designed the circuit in the same process, used the same performance specification and applied the same inputs as [31].

[2]We selected the same opamp case-study as [31] in order to compare Duplex with the state of the art.

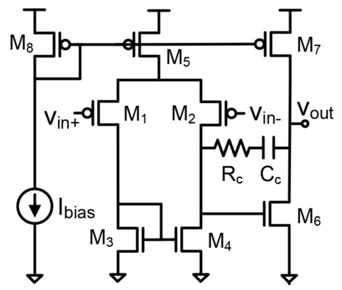


Figure 9.5: Schematic of a two-stage operational amplifier.

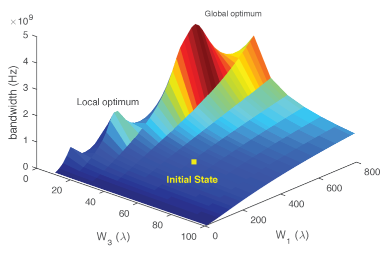
Table 9.1: Performance specification for the opamp circuit and the result of circuit optimization. Duplex determines the optimum value for the parameters and performance metrics of the circuit.

Performance metric	Performance Spec. set by. designer	Optimum Value computed by Duplex
Power	< 0.5mW	0.4763mW
Phase margin	$> 45°$	109°
Gain margin	> 5dB	11.22dB
DC gain	¿30dB	76.59dB
Slew rate	$> 10\frac{\text{V}}{\mu\text{sec}}$	$55.65\frac{\text{V}}{\mu\text{s}}$
Bandwidth	¿2GHz	5.766GHz

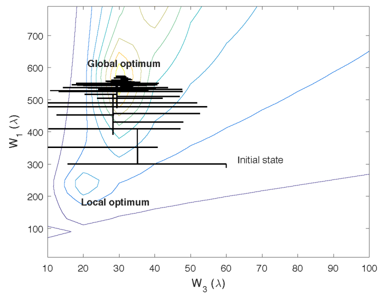
It took approximately 20 minutes and 857 HSPICE simulations to perform the optimization and generate 100 optimal states within the goal region on a Windows machine equipped with a Core-i5 processor and 16GB memory to optimize the opamp circuit. The majority of the time was spent on HSPICE simulation and the Duplex's performance overhead was negligible.

Jung et al. [31] reported 4625 SPICE simulations to compute the optimal design. **In comparison, Duplex finished in 857 HSPICE simulations, demonstrating a** 81% **more performance efficiency than [31].** Furthermore, Duplex improved the quality of the optimization results by increasing the circuit's bandwidth to 5.7GHz, up to 250%, in comparison to [31] where they reported the optimized bandwidth of 2.2GHz. Notably, the opamp design demonstrates how Duplex escapes getting stuck in local minima.

Figure 9.6 shows how Duplex simultaneously explores the parameter and performance space and avoids the local optima. On the left, Figure 9.6a shows the circuit's bandwidth w.r.t. size of transistors M_1 and M_3, assuming



(a) The bandwidth w.r.t. the width of transistor M_1 and M_3. The bandwidth objective has one global maximum and multiple local maxima in the state space.



(b) The parameter tree explores the space and converges to the global maxima, while not getting trapped in the local maxima.

Figure 9.6: Using Duplex for optimizing the bandwidth of the op-amp.

other parameters are set to optimal value. Let w_1 and w_3 denote the width of transistor M_1 and M_3, respectively. Due to symmetry, size of transistors M_2 and M_4 are equal to M_1 and M_3, respectively. There is one global maximum, located at $(w_1, w_3) = (590\lambda, 30\lambda)$; however, there are multiple local maxima throughout the space. The objective of Duplex was to maximize bandwidth without getting stuck in local maxima. We set the initial state at $w_1, w_3 = (300\lambda, 60\lambda)$ and executed Duplex. Figure 9.6b shows the contour plot of the bandwidth. We rendered the parameter tree over the contour plot to show how Duplex explores the parameter space. Even though a few branches in the parameter tree grow toward the local maximum at $(250\lambda, 20\lambda)$, Duplex used the performance tree to grow toward the global optimum and successfully converged toward the global maximum.

As the algorithm got closer to the goal region, the annealing step length caused Duplex to take smaller steps and remains within the goal region. As a result, many samples were generated within the goal region. At each iteration, Duplex only changed one parameter, making all edges in the parameter tree parallel to the $w_1 - w_3$ axis. Hence, many of the states where the w_1 or w_3 were unchanged are not shown in the projected figure. In order to increase the bandwidth, Duplex aggressively increased the size of transistor M_1. The opamp's unity-gain bandwidth can be approximated as [31] $w_c = \frac{g_{m1}}{C_c}$. This suggests that the bandwidth can be increased by transconductance of the first stage, which in turn can be achieved by sizing up the input transistors M_1 and M_2. Bandwidth can also be increased by reducing the compensa-

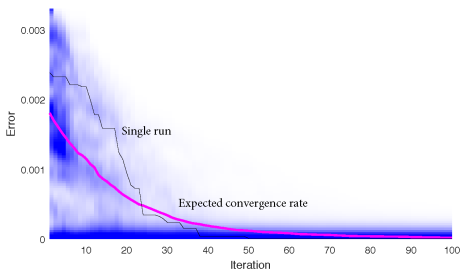


Figure 9.7: The convergence rate w.r.t. number of iterations for the Duplex algorithm for the inverter case study. Our algorithm converges very fast toward the optimum design from any initial state. Duplex is not sensitive to the choice of initial state.

tion capacitor C_c or increasing the bias current I_{bias}. Duplex automatically performed all of these optimizations in order to meet the specification.

We use the CMOS inverter from Figure 9.1 to demonstrate a few outputs of Duplex. The inverter is designed in 65 nm process.

Figure 9.7 shows the *visually weighted regression* plot for convergence rate for the Duplex algorithm for the inverter case study. We measure error as the minimum distance from every step in the performance tree toward the center of the goal region. We executed Duplex for 100 independent runs with a random initial state and draw the overlapping convergence plots in the visually weighted regression plot. We also draw the average of all the convergence plot as the expected convergence rate. As shown in the convergence figure, Duplex quickly converges toward the goal region in the performance space. Figure 9.7 highlights two facts about the Duplex algorithm: 1) Duplex converges exponentially fast toward the goal region and 2) Duplex is very stable with respect to the choice of the initial state

In our experiment, we uniformly sampled the parameters for the initial (root) state in the parameter space, which is the reason for high error variance in the beginning. On the other hand, toward the end of the algorithm the variance in error is low because Duplex converges to the optimum results regardless of the choice of the initial state.

9.4.1 Performance to parameter sensitivity

We visualize the performance to parameter sensitivity matrix s_{ij} using a bipartite sensitivity graph as shown in Figure 9.8. The left side of the sensitivity

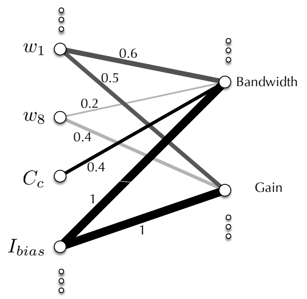


Figure 9.8: The sensitivity graph visualizing performance to parameter sensitivity for opamp case-study.The Edges are annotated with the sensitivity of a performance metric (a node in the right side) to a particular parameter (a node on the left side).

graph denotes the parameters of the circuit (such as width of transistors or bias current) and the right side denote the performance metric measured by the Duplex algorithm (such as bandwidth, power and gain). The thickness of each edge between a parameter and performance node denote the sensitivity. Due to the lack of space, we only showed the partial graph of the most important nodes and leave out the rest.

Each row of the sensitivity matrix is normalized. Hence, for each given performance metric, one parameter has an edge of thickness 1.0, denoting the most influential parameter to that performance, and the other parameters have values between $[0, 1]$. For the opamp circuit, the sensitivity graph implies bandwidth depends on the bias current i_{bias}, compensation capacitor C_c and the width of transistors M_1 and M_8. This observation supports our bandwidth analysis earlier. Similarly, the gain is very sensitive to the width of transistor M_1 and sizing of the current mirror M_8, and the bias current. However, the gain is not sensitive to the compensation capacitor. We also observed that the biasing current was the most sensitive parameter in the design.

9.4.2 Pareto distribution inference

To compute the Pareto distribution, we collect the parameter samples that result in acceptable performance from the circuit. Figure 9.9 shows the Gaussian mixture distribution of those samples for the opamp circuit, projected to W_5, I_{bias} plane, where W_5 is a width of transistor M_5 and I_{bias} is the bias current. The mean of the Pareto distribution indicates the optimal value of the

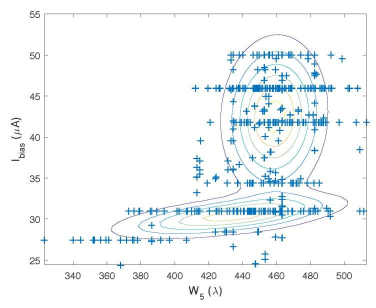


Figure 9.9: Distribution of the optimal parameters for the opamp circuit. Duplex computes the Pareto set as a mixture Gaussian distribution by inferring the distribution of the samples in the goal region. Pareto surface is computed from the CDF of the pareto distribution. We use the mean of the distribution as the optimum state.

parameter. Furthermore, we can generate more optimal design parameters from the Pareto distribution and predict the yield for the circuit.

9.4.3 Optimizing the PLL circuit

The charge-pump PLL (CP-PLL) [160, 161] is one of the key building blocks in many analog IPs and SoCs. The PLL can be used in various applications such as clock synchronization and jitter mitigation.

We used a low-noise 1.6 GHz CP-PLL circuit as a system-level example to demonstrate Duplex's scalability. The schematic of the CP-PLL circuit is shown in Figure 9.10 [161]. The CP-PLL circuit consists of five blocks: phase detector, charge-pump circuitry, loop filter, voltage controlled oscillator (VCO) and frequency divider, arranged in a feedback configuration [161]. The circuit has 131 CMOS transistors and 140 parameters (including the width of the transistors and a few resistors and capacitors). The circuit is designed using TSMC 0.18μm process, using supply voltage 1.8V. The reference clock was set at 200 MHz.

The first block in the CP-PLL circuit was the phase detector circuit. The phase detector compares the clock produced from the VCO with the reference clock and produces an error signal proportional to the phase difference between its inputs. The phase detector block was implemented using a NOR gate, hence it was balanced but not very power efficient. We minimized the total power dissipation of the phase detector while maximizing the gain K_d. We set the width of the transistors in the NOR gate as a parameter.

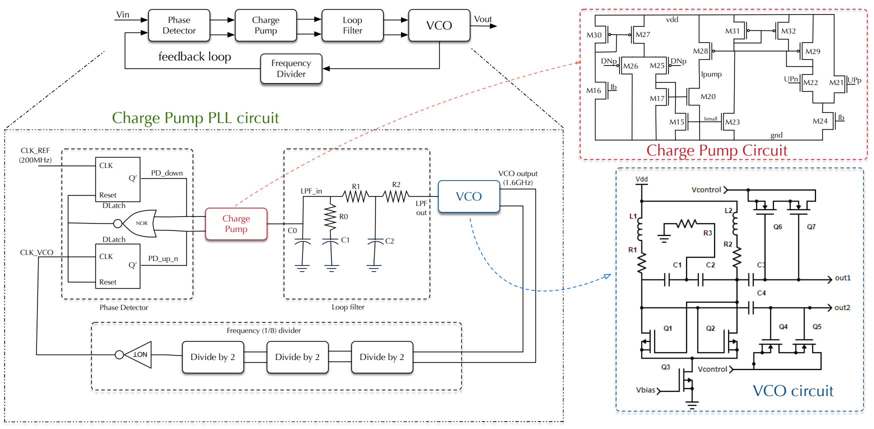


Figure 9.10: Schematic of a post-layout charge-pump PLL circuit.

The charge pump was a set of symmetrical current sources. Transistors $M_{21}, \ldots, M_{26}$ supply the pump-up current to the loop filter. It consists of an input differential pair $M_{21}M_{22}$, current mirror load M_{29}, output current source M_{28}, and pull-up transistors $M_{31}M_{32}$. A similar circuit is used to generate the pump-down current. We set the size of the input differential transistors pairs as a parameter for duplex in order to make the charge-pump circuit fully balanced. After the charge pump block, there was a filter to remove the high-frequency components of the signal introduced by the phase detector circuit. The loop filter was implemented as a simple RC network and consisted of three resistors and capacitors that created a three-pole one-zero network [160]. The CP-PLL's stability was largely dependent on the value of capacitor C_1, and the bandwidth was dependent on the value of resistor R_1. After duplex optimization, the algorithm determined the optimum value for capacitor C_1 was 48 pF and for resistor R_1 was 54 KΩ.

The voltage controlled oscillator (VCO) [160] was supposed to produce a clock at $1.6GHz$. The input stage consists of $M_4, \ldots, M_7$ transistors which are used as varactors for frequency tuning. $M_1, \ldots, M_3$ offers negative conductance to fulfill the oscillating pre-conditions. R_0 and R_1 are parasitic resistance of L_0 and L_1. The output frequency of the VCO depends on the value of m_{mult}. The algorithm was optimizing the bias voltage of the VCO circuit. For the VCO circuit, we ensured the gain of the circuit was more than $25\frac{\text{Meg}}{\text{V}}$. The output of the VCO passes a series of 2:1 dividers, reducing the frequency from expected 1.6 GHz to 200 MHz[160].

We optimized the PLL design such that it would meet the operating frequency of the PLL at 1.6 GHz while optimizing the performance of phase

Table 9.2: Result of Duplex optimization of the CP-PLL circuit. Duplex determines the optimum value for the parameters and performance metrics of the circuit.

Perf. metric	Perf. Spec. set by. designer	Opt. Val. computed by Duplex
PD Gain K_d	$> 1\mu$A/Deg	1.16μA/deg
Frequency	1.6GHz ±0.01	1.6025GHz
VCO PhaseNoise	¡-60dB@60K	-96.25dB@60K
VCO gain	¿25MEG/V	39.1 MEG/V
PD Power	¡20mW	12.85mW

detector gain and power and the VCO gain and phase noise. We executed duplex for 350 iteration which took approximately 13 hours. The algorithm found multiple configurations that satisfied the performance requirements of the CP-PLL circuit. Table 9.2 shows the result of the optimization.

9.5 Chapter summary

In this chapter, we used duplex algorithm to optimize performance metrics of analog and mixed signal circuits. We modeled the circuit performance optimization as a Duplex type-II nonconvex optimization problem. Duplex determines the optimal design, the Pareto set and the sensitivity of circuit's performance metrics to its parameters. We demonstrated that Duplex is 5× faster than the state-of-the-art and finds the global optimum for a design whose previously published result was a local optimum. We showed our algorithm's scalability by optimizing a system-level post-layout charged-pump PLL circuit.

CHAPTER 10

BEYOND ANALOG: APPLICATION OF DUPLEX IN MACHINE LEARNING

Machine learning is the science of building models from the data and making predictions. Machine learning combines different aspects of statistics, information theory, control and computer science. Optimization is the core of all machine learning algorithms. Vapnik first defined machine learning an optimization problem in [162]. A general approach to many machine learning algorithms is to model the prediction error as a cost function. The process of training the model is equivalent to minimizing the cost function. Similar cost functions for unsupervised learning algorithms such as clustering also exist based on energy functions. We can formulate many supervised and unsupervised machine learning problems as an optimization problem and train the model using the Duplex algorithm by minimizing the cost function.

The machine learning models should not be very sensitive to the input changes. This is achieved by implementing low variance complex models that avoids over fitting. A drawback of added complexity to the model is a nonconvex landscape of the model's cost function. Training such a model is very challenging. The optimization algorithms often get stuck in local minima without converging toward the optimal solution. Furthermore, the cost function has multiple saddle points. First order optimization methods suffer a performance penalty around the saddle points. The gradient is close to zero around the saddle points and slows down the descent algorithm. On the other hand, second order Newtonian methods do not scale with respect to the size of the data.

We use Duplex to optimize the cost function in supervised and unsupervised learning setup. Duplex is scalable, very efficient and does not get stuck in local minima, which results in more accurate models.

10.1 Supervised learning and classification

In supervised learning, we learn from the labeled data. Let X denote the feature (input) space, Y denote the label (output) space, and D denote the distribution of samples over $X \times Y$. The input to the algorithm is set S of N training data $S = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$. Each pair $(x^{(i)}, y^{(i)})$ consists of the input feature vector $x^{(i)}$ and the output label $y^{(i)}$. The algorithm is learning a function $f : X \to Y$ from the input space X to the output space Y. In order to measure how well the model fits the training data we define a loss function $L : Y \times Y \to R^{\geq 0}$. The loss function $L(z; y)$ measures the loss whenever we predict y as z. The expected loss of f is defined as *risk*:

$$R(f) = \mathbf{E}_{(x,y)\ D}[L(f(x); y)] \tag{10.1}$$

The purpose of the supervised learning algorithm is to search for a function f that minimizes the risk R. Finding the global minimizer of the risk function $\arg\min_{f \text{ is a function}} R(f)$ is fundamentally impossible. However, we can approximate the risk with the training error

$$\hat{R}(f) \equiv \mathbf{E}_{(x,y)\ S}[L(f(x); y)] \approx R(f) \tag{10.2}$$

by assuming that good performance on the training set translates to good performance on every sample we see in the future.

We use logistic regression algorithm for classification. Assume the output labels take binary values $y_i \in \{0, 1\}$.

$$\begin{aligned} P(y = 1|x) &= h_\theta(x) = \frac{1}{1 + exp(-\theta^T x)} = \sigma(\theta^T x), & (10.3) \\ P(y = 0|x) &= 1 - P(y = 1|x) = 1 - h_\theta(x). & (10.4) \end{aligned}$$

The function $\sigma(z) = \frac{1}{1+exp(-z)}$ is often called the sigmoid function and has a range $[0, 1]$ which allows us to interpret it as a probability. Intuitively, we are searching for a value of θ such that the probability $P(y = 1|x) = h_\theta(x)$ is large when x belongs to the class 1 and small otherwise. We define the following cost function to determining how well the model does:

$$J(\theta) = \frac{-1}{N} \sum_{i}^{N} (y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))). \tag{10.5}$$

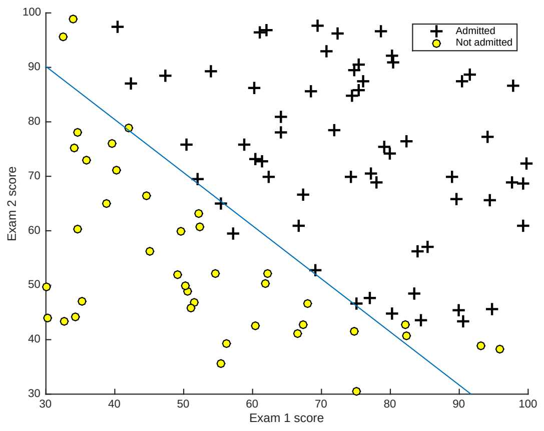


Figure 10.1: Duplex algorithm cluster samples together by minimizing the distortion function.

By minimizing the loss function J, we train the logistic regression model. We formulate the learning problem as a type-II duplex problem. The input space is the space of vector $\theta = [\theta_0, \theta_1, \theta_2]$. The training set consists of 100 samples (x_1, x_2), each labeled with y_1 output. The output dimension is the space of parameter J. At each iteration, we compute the global step by generating a new sample using a zero-mean normal distribution (the minimum of the loss function is at zero).

The gradient of function J with respect to input parameter θ is

$$\frac{dJ(\theta)}{d\theta_j} = \frac{1}{N} \sum_{i=1}^{N} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{10.6}$$

We take the local steps along the direction of the gradient with added white noise using

$$\theta = \theta - \alpha * \frac{dJ(\theta)}{d\theta} + \beta; \tag{10.7}$$

where X is the vector of input, Y is the vector of features and α is the learning rate and β is annealing white noise.

We trained the logistic regression model with the Duplex algorithm. Figure 10.1 shows the decision boundary computed using the Duplex algorithm. After 1000 iterations, the accuracy of the model on the validation data was 91%. In comparison, the gradient descent algorithm converged after 100 iterations and its accuracy was 88%.

10.2 Unsupervised learning and clustering

In many machine learning problems, the data is not labeled. The goal of the unsupervised learning algorithm is to infer the hidden structures from the unlabeled data and cluster them together. Since there is no label, there is no error function or risk associated with a solution.

An efficient method for clustering unlabeled data is K-means. The K-means algorithm splits the data into K clusters. Assume we have N samples $\{x^{(1)}, \ldots, x^{(N)}\}$ such that $x^{(i)} \in \mathbf{R}^n$. We wish to cluster these samples into K clusters, defined by their centroids $\{\mu_1, \ldots, \mu_K\}$ such that every sample belongs to the cluster with the closest centroid. The distortion of the samples in the given cluster configuration is defined as

$$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2 \tag{10.8}$$

where $c(i)$ denotes the cluster that the i^{th} sample belongs to. The objective of the algorithm is to find c and μ that minimize the function J. The distortion function J is a nonconvex function and has discontinuities; therefore standard optimization algorithm such as gradient descent are not guaranteed to converge toward the global minimum.

The classic K-means algorithm repeatedly assigns each sample $x^{(i)}$ to a cluster with closest centroids μ_j using

$$c^{(i)} = \arg\min_j ||x^{(i)} - \mu_j||^2 \tag{10.9}$$

Then updates the clusters by moving the centroids to the mean of the samples assigned to that cluster.

$$\mu_j = \frac{\sum_{i=1}^{N} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{N} 1\{c^{(i)} = j\}} \tag{10.10}$$

The K-means algorithm repeats the following two steps until the centroids do not move anymore. We used Duplex to solve the clustering algorithm. We model the input (centroid) space as an $N \times K$-vector $\{\mu_1, \ldots, \mu_k\}$. We break the distortion functional into the sum of its components for each cluster as

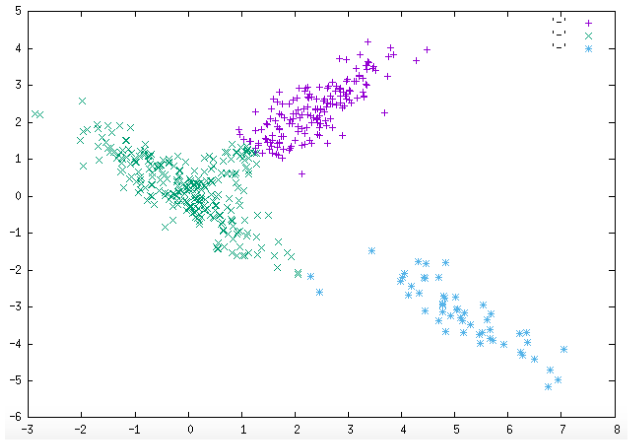


Figure 10.2: Duplex algorithm cluster samples together by minimizing the distortion function.

follows.

$$J_j(c,\mu) = \sum_{i=1}^{N} 1\{c^{(i)} = j\} ||x^{(i)} - \mu_{c^{(i)}}||^2 \tag{10.11}$$

We formulate the clustering problem as a type-II optimization problem. We use Duplex to minimize the distortion function. We simultaneously grow two random trees in the centroid space $\mathbf{R}^{N\times K}$ and the distortion space $\mathbf{R}^{K}$. At every iteration, we generate a goal sample in the distortion space where the distortion is close to zero. Then we pick the nearest node in the distortion random tree according to Euclidean distance. We find the corresponding node in the centroid tree. We update one of the centers in that node and randomly assign a new center. We evaluate the new node according to Equation 10.11 and add the new node to the distortion tree. We repeat this procedure until we converge to the optimal solution.

We clustered a synthetic 2D dataset of 500 samples as shown in Figure 10.2 to evaluate the Duplex algorithm. The final value of the distortion functional after 1000 iterations was 1.38444. In comparison, the K-means algorithm converged to the distortion value of 1.38999 after six iterations and ten restarts. In comparison to the classic Kmean algorithm, Duplex can take a longer time, but it will converge to a better solution.

10.3 Chapter summary

In this chapter, we used Duplex for optimizing the cost function of different machine learning algorithms. First, we use Duplex for logistic regression as an example of supervised learning problem. Then we used Duplex for clustering an unsupervised learning example. We showed that the Duplex algorithm is capable of optimizing nonconvex functions and has a high accuracy.

CHAPTER 11

CONCLUSION

Our mission was to provide automation to analog design flow to improve the quality and reliability of analog circuits. To achieve this mission, we proposed the Duplex methodology and optimization algorithm. We formulated challenging problems in analog validation and optimization as a Duplex optimization objectives (state search, nonconvex and functional optimization problems). Then we used the Duplex optimization algorithm to optimize the objective function and solve the analog validation and optimization problem.

We have presented Duplex, a methodology for nonconvex and functional optimization. We used Duplex to address the broad range of challenging problems in analog design automation. The Duplex algorithm brought global direction to the search and used it to find the optimum solutions quickly. Furthermore, Duplex attacked more complicated problems by using the principle of space separation, dividing the problem space into input, output and function spaces. Duplex utilized random tree data structure to simultaneously explore these spaces and used how close it was in the higher spaces to guide the random tree in lower spaces.

The Duplex algorithm provided a lot of advantages over the state-of-the-art techniques, both in the analog domain and beyond. We showed the Duplex algorithm enjoyed theoretical, as well as empirical, convergence guarantees toward the global optimum solution. We found optimal solutions for the problems that their previous state-of-the-art results were local optimums. Globally optimum solutions result in increased bandwidth and higher performance in analog, as well as higher classification accuracy in machine learning. The duplex algorithm is very performance efficient and very concurrent in nature. We consistently demonstrated that Duplex provides at least two orders of magnitude speedup over Monte Carlo simulations. Finally, the Duplex algorithm is very scalable. We demonstrated scalability in practice by using Duplex to optimize system-level and post-layout circuits. Finally, Duplex maintains the search history and uses the random tree to provide valuable feedback to the user. We computed the circuit's sensitivity, the distribution of worst-case inputs, and the Pareto set of optimal samples and reported them to the user.

We demonstrated the breadth of scope of Duplex methodology by applying it to solve the keystone problems in analog validation, optimization and beyond. We addressed challenging open problems in analog validation such as automatically generated directed input stimuli while simultaneously improving coverage, compressing analog tests in time for stress and functional testing, and worst-case eye diagram analysis. We provided new results, found design bugs, provided two orders of magnitude efficiency improvements, computed the circuit's response up to 6-σ deviation in inputs, compressed tests up to 96%, combined analog tests together, and computed distributions of worst-case input corners in the eye diagram. We formally verified the circuit by implementing reachability analysis and runtime monitoring algorithms. We designed our analog specification language and developed an incremental model checker to monitor the execution of the Duplex algorithm against the specification property. We optimized analog circuits for getting the best performance. We improved the state-of-the-art, both regarding efficiency (5$\times$ speedup) and accuracy (found globally optimum solution). We computed the circuit's performance to parameter sensitivity and the Pareto frontier by analyzing the samples in the random tree.

Finally, we generalized the Duplex algorithm as an optimization toolbox. We observed that problems in analog circuits share the same characteristics as problems in machine learning, motion planning and optimal control. We found that the knowledge and the technique that we obtained for optimizing problems in analog domain can also be used to solve problems in machine learning. We employed the Duplex algorithm to train supervised and unsupervised learning models for classification and clustering. We computed the clustering of unlabeled data and improved the accuracy of a binary classifier.

REFERENCES

[1] GBI Research. *Analog Integrated Circuits (IC) Market to 2016.* GBI research group market report No. GBISC029MR, Available online at `http://www.gbiresearch.com` (accessed Oct 2015), 2012.

[2] ITRS group. *International Technology Roadmap for Semiconductors.* Available online at http://www.itrs.net/Links/2011ITRS/2011Chapters/2011Design.pdf (accessed on Oct 2013), 2011.

[3] Georges G. E. Gielen, HCC Walscharts, and Willy M C Sansen. Analog circuit design optimization based on symbolic simulation and simulated annealing. *IEEE Journal of Solid-State Circuits*, 25(3):707–713, 1990.

[4] Christopher Bishop. *Pattern Recognition and Machine Learning.* Springer, New York, 2006.

[5] A Bruce Carlson, Janet C Rutledge, and Paul Crilly. *Communication Systems.* 5th edition. McGraw-Hill, January 2001.

[6] Christian P. Robert and G. Casella. *Monte Carlo Statistical Methods (Second ed.).* Springer, New York, February 2004.

[7] P Kumar Hanumolu, B Casper, R Mooney, Gu-Yeon Wei, and Un-Ku Moon. Analysis of PLL clock jitter in high-speed serial links. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 50(11):879–886, November 2003.

[8] Bryan K Casper, Matthew Haycock, and Randy Mooney. An accurate and efficient analysis method for Multi-Gb/s Chip-to-chip signaling schemes. *Symposium on VLSI Circuits*, February 2004.

[9] G Balamurugan, B Casper, J E Jaussi, M Mansuri, F O'Mahony, and J Kennedy. Modeling and analysis of high-speed I/O links. *IEEE Transactions on Advanced Packaging*, 32(2):237–247, 2009.

[10] Akira Tsuchiya, Masanori Hashimoto, and Hideotoshi Onedera. Optimal termination of on-chip transmission-lines for high-speed signaling. *IEICE Transactions on Electronics*, E90-C(6):1267–1273, June 2007.

[11] L S Milor. A tutorial introduction to research on analog and mixed-signal circuit testing. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 45(10):1389–1407, 1998.

[12] L Milor and A L Sangiovanni-Vincentelli. Minimizing production test time to detect faults in analog circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(6):796–813, 1994.

[13] Nourredine Akkouche, Salvador Mir, and Emmanuel Simeu. Ordering of analog specification tests based on parametric defect level estimation. *VLSI Test Symposium (VTS)*, pages 301–306, 2010.

[14] Xin Li, Rob R Rutenbar, and Ronald D Blanton. Virtual probe: a statistically optimal framework for minimum-cost silicon characterization of nanoscale integrated circuits. *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 433–440, 2009.

[15] Mohamed A El-Gamal, Abdei-Karim S O Hassan, and Hany L Abdel-Malek. A new approach for the selection of test points for fault diagnosis. *ISCAS'95 - International Symposium on Circuits and Systems*, 3:2019–2022, 1995.

[16] Asma Laraba, H G Stratigopoulos, Salvador Mir, Hervé Naudet, and Christophe Forel. Enhanced reduced code linearity test technique for multi-bit/stage pipeline ADCs. *IEEE*, pages 1723–6734, 2012.

[17] H G Stratigopoulos. Test metrics model for analog test development. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(7):1116–1128, July 2012.

[18] A Chandra and K Chakrabarty. Test data compression for system-on-a-chip using Golomb codes. *18th IEEE VLSI Test Symposium*, pages 113–120, 2000.

[19] Hantao Huang, Hao Yu, Cheng Zhuo, and Fengbo Ren. A compressive-sensing based testing vehicle for 3D TSV pre-bond and post-bond testing data. In *ISPD '16: Proceedings of the 2016 on International Symposium on Physical Design*, pages 19–25, New York, New York, USA, April 2016. Arizona State University, ACM.

[20] Mohamed H. Zaki, Sofiène Tahar, and Guy Bois. Formal verification of analog and mixed signal designs: A survey. *Microelectronics Journal*, 39(12):1395–1404, December 2008.

[21] Eugene Asarin, Venkatesh P. Mysore, Amir Pnueli, and Gerardo Schneider. Low dimensional hybrid systems – decidable, undecidable, don't know. *Information and Computation*, 211:138–159, February 2012.

[22] Jörg Preußig, Olaf Stursberg, and Stefan Kowalewski. Reachability analysis of a class of switched continuous systems by integrating rectangular approximation and rectangular analysis. In *HSCC '99: Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*. Springer-Verlag, March 1999.

[23] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, May 2010.

[24] G Frehse. PHAVer: algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3):263–279, 2008.

[25] Rajeev Alur, Thao Dang, and Franjo Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *Transactions on Embedded Computing Systems*, 5(1):152–199, February 2006.

[26] Georges G. E. Gielen and R Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. In *Proceedings of the IEEE*, pages 1823–1824. IEEE, 2000.

[27] C Toumazou and C A Makris. Analog IC design automation. I. Automated circuit generation: new concepts and methods. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(2):218–238, 1995.

[28] Saurabh K Tiwary, Pragati K Tiwary, and Rob A Rutenbar. Generation of yield-aware Pareto surfaces for hierarchical circuit design space exploration. *Design Automation Conference*, pages 31–36, 2006.

[29] G Yu and P Li. Hierarchical analog/mixed-signal circuit optimization under process variations and tuning. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 2011.

[30] G Stehr, H E Graeb, and K J Antreich. Analog performance space exploration by normal-boundary intersection and by Fourier-Motzkin elimination. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(10):1733–1748, 2007.

[31] S Jung, J Lee, and J Kim. Variability-aware, discrete optimization for analog circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 33(8):1117–1130, 2014.

[32] Lawrence T Pillage, Ronald A Rohrer, and Chandramouli Visweswariah. *Electronic Circuit and System Simulation Methods*. McGraw-Hill Professional Publishing, 1995.

[33] Vaclav Smidl and Anthony Quinn. *The Variational Bayes Method in Signal Processing*. Springer, 2006.

[34] P Duhamel and J Rault. Automatic test generation techniques for analog circuits and systems: A review. *Circuits and Systems, IEEE Transactions on*, 26(7):411–440, 1979.

[35] R Voorakaranam and A Chatterjee. Test generation for accurate prediction of analog specifications. In *VLSI Test Symposium, 2000. Proceedings. 18th IEEE*, pages 137–142. IEEE Comput. Soc, 2000.

[36] A Abderrahman, B Kaminska, and E Cerny. Optimization-based multifrequency test generation for analog circuits. *Journal of Electronic Testing*, 9(1-2):59–73, 1996.

[37] Chen-Yang Pan and Kwang-Ting Cheng. Test generation for linear time-invariant analog circuits. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(5):554–564, 1999.

[38] Parijat Mukherjee, G Peter Fang, Rod Burt, and Peng Li. Efficient Identification of Unstable Loops in Large Linear Analog Integrated Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(9):1332–1345, 2012.

[39] E Plaku, L E Kavraki, and M Y Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 34(2):157–182, 2009.

[40] Jongwoo Kim and J. M. Esposito. Adaptive sample bias for rapidly-exploring random trees with applications to test generation. In *American Control Conference, 2005 Proceedings of the 2005*, 2005.

[41] A Julius, G Fainekos, M Anand, and I Lee. Robust test generation and coverage for hybrid systems. *Hybrid Systems: Computation and Control*, 4416:329–342, January 2007.

[42] Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, February 2009.

[43] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[44] Sudhi Proch and P Mishra. Directed test generation for hybrid systems. *2014 15th International Symposium on Quality Electronic Design (ISQED)*, pages 156–162, 2014.

[45] MH Zaki, S Tahar, and G Bois. A practical approach for monitoring analog circuits. In *Proceedings of the 16th ACM Great Lakes*, January 2006.

[46] Goran Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, August 2005.

[47] G Al Sammane, M H Zaki, Z J Dong, and S Tahar. Towards assertion based verification of analog and mixed signal designs using PSL. *Forum on Specification and Design Languages*, pages 293–298, 2007.

[48] Ying-Chih Wang, A. Komuravelli, P. Zuliani, and E. M. Clarke. Analog circuit verification by statistical model checking. In *Design Automation Conference (ASP-DAC), 16th Asia and South Pacific*, 2011.

[49] D Nickovic and O Maler. AMT: A property-based monitoring tool for analog systems. *Formal Modeling and Analysis of Timed Systems*, January 2007.

[50] O Maler and D Nickovic. Monitoring temporal properties of continuous signals. *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 71–76, 2004.

[51] Kevin D. Jones, Victor Konrad, and Dejan Ničković. Analog property checkers: a DDR2 case study. *Formal Methods in System Design*, 36(2):114–130, June 2010.

[52] Tathagato Rai Dastidar and P. P. Chakrabarti. A verification system for transient response of analog circuits. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):31–es, August 2007.

[53] T Nahhal and T Dang. Test coverage for continuous and hybrid systems. *Computer Aided Verification*, pages 1–46, May 2007.

[54] S Karaman and E Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. *2012 Americal Control Conference*, pages 2222–2229, 2012.

[55] E Asarin, G Schneider, and S Yovine. Algorithmic analysis of polygonal hybrid systems, part I: Reachability. *Theoretical Computer Science*, September 2012.

[56] A Chutinan and B H Krogh. Computational techniques for hybrid system verification. *Automatic Control, IEEE Transactions on*, 48(1):64–75, January 2003.

[57] M Greenstreet and I Mitchell. Reachability analysis using polygonal projections. *Hybrid Systems: Computation and Control*, January 1999.

[58] A Girard. Reachability of uncertain linear systems using zonotopes. *Hybrid Systems: Computation and Control*, pages 1–15, January 2005.

[59] W Hartong, R Klausen, and L Hedrich. Formal verification for non-linear analog systems: Approaches to model and equivalence checking. *Advanced Formal Verification*, pages 205–245, 2004.

[60] Sebastian Steinhorst. *Formal Verification Methodologies for Nonlinear Analog Circuits*. PhD thesis, Universität Frankfurt, Frankfurt, August 2010.

[61] Sebastian Steinhorst and Lars Hedrich. Model checking of analog systems using an analog specification language. *Design, Automation and Test in Europe*, pages 324–329, March 2008.

[62] Hallstein Asheim Hansen, Gerardo Schneider, and Martin Steffen. Reachability analysis of non-linear planar autonomous systems. In *FSEN'11: Proceedings of the 4th IPM international conference on Fundamentals of Software Engineering*. Springer-Verlag, April 2011.

[63] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. *Hybrid Systems: Computation and Control, volume 2993 of Lecture Notes in Computer Science*, pages 271–274, February 2004.

[64] Christoffer Sloth, George J. Pappas, and Rafael Wisniewski. Compositional safety analysis using barrier certificates. *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control - HSCC '12*, pages 115–229, January 2012.

[65] S Ratschan and Z She. Recursive and backward reasoning in the verification on hybrid systems. In *Proceedings of the 5th Int Conf on Informatics in*, January 2008.

[66] Mathworks. *MATLAB eye diagram analysis*. Available online at http://www.mathworks.com/help/comm/ref/commscope.eyediagram.html Accessed April-2014, 2014.

[67] Zhaoqing Chen and G Katopis. Searching for the worst-case eye diagram of a signal channel in electronic packaging system including the effects of the nonlinear I/O devices and the crosstalk from adjacent channels. In *Electronic Components and Technology Conference, 2009. ECTC 2009. 59th*, pages 1106–1113. IEEE, 2009.

[68] JiHong Ren and Kyung Suk Oh. Multiple edge responses for fast and accurate system simulations. *IEEE Transactions on Advanced Packaging*, 31(4):741–748, November 2008.

[69] Mike Peng Li, Masashi Shimanouchi, and Hsinho Wu. Advanements in high-speed link modeling and simulation. *Custom Integrated Circuit Conference*, 2013.

[70] Vladimir Stojanovic and Mark Horowitz. Modeling and analysis of high-speed links. *Custom Integrated Circuits Conference*, pages 589–594, 2003.

[71] Dan Oh, Jihong Ren, and Sam Chang. Hybrid statistical link simulation technique. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 1(5):772–783, May 2011.

[72] Rui Shi, Wenjian Yu, Yi Zhu, Chung-Kuan Cheng, and Ernest S Kuh. Efficient and accurate eye diagram prediction for high speed signaling. In *ICCAD '08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design.* IEEE Press, November 2008.

[73] L Milor and A Sangiovanni-Vincentelli. Optimal test set design for analog circuits. In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pages 294–297. IEEE Comput. Soc. Press, 1990.

[74] S D Huss and R S Gyurcsik. Optimal ordering of analog integrated circuit tests to minimize test time. In *Design Automation Conference, 1991. 28th ACM/IEEE*, pages 494–499. IEEE, 1991.

[75] N Akkouche, S Mir, E Simeu, and M Slamani. Analog/RF test ordering in the early stages of production testing. *VLSI Test Symposium (VTS), 2012 IEEE 30th*, pages 25–30, 2012.

[76] Jin Chen and A Ramachandran. A novel test set design for parametric testing of analog and mixed-signal circuits. In *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pages 474–480. IEEE Comput. Soc, 1997.

[77] Chieh-Yuan Chao, Hung-Jen Lin, and L Miler. Optimal testing of VLSI analog circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):58–77, November 2006.

[78] Sounil Biswas and R D Blanton. Reducing test execution cost of integrated, heterogeneous systems using continuous test data. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(1):148–158, 2010.

[79] R Voorakaranam, S Cherubal, and A Chatterjee. A signature test framework for rapid production testing of RF circuits. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 186–191. IEEE Comput. Soc, 2002.

[80] L Balado, E Lupon, J Figueras, M Roca, E Isern, and R Picos. Verifying functional specifications by regression techniques on Lissajous test

signatures. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(4):754–762, 2009.

[81] P N Variyam, J Hou, and A Chatterjee. Efficient test generation for transient testing of analog circuits using partial numerical simulation. In *17th IEEE VLSI Test Symposium*, pages 214–219. IEEE Comput. Soc, 1999.

[82] Andrzej Kuczyński. Parametric faults detection in analog circuits using polynomial coefficients in NN learning. *International Conference on Signals and Electronic Systems ICSES*, pages 249–252, 2010.

[83] A Singhee and R A Rutenbar. Statistical blockade: very fast statistical simulation and modeling of rare circuit events and its application to memory design. *IEEE Trans Comput-Aided Des Integr Circuits Syst*, 28(8):1176–1189, 2009.

[84] S D Huynh, S Kim, M Soma, and Jinyan Zhang. Automatic analog test signal generation using multifrequency analysis. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 46(5):565–576, 1999.

[85] A Halder and A Chatterjee. Automated test generation and test point selection for specification test of analog circuits. In *Quality Electronic Design, 2004. Proceedings. 5th International Symposium on*, pages 401–406. IEEE Comput. Soc, 2004.

[86] T Golonek and J Rutkowski. Genetic-algorithm-based method for optimal analog test points selection. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 54(2):117–121, 2007.

[87] A V Gomes and A Chatterjee. Minimal length diagnostic tests for analog circuits using test history. In *Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings*, pages 189–194. IEEE Comput. Soc, 1999.

[88] Seyed Nematollah Ahmadyan, Jayanand Asuk Kumar, and Shobha Vasudevan. Goal-oriented stimulus generation for analog circuits. In *49th Design Automation Conference (DAC-2012)*, 2012.

[89] Seyed Ahmadyan and Shobha Vasudevan. Automated transient input stimuli generation for analog circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pages 1–1, October 2015.

[90] Seyed Nematollah Ahmadyan, Shobha Vasudevan, Eli Chiprout, Chenjie Gu, and Suriyaprakash Natarajan. Fast eye diagram analysis for high-speed CMOS circuits. In *Design, Automation Test conference in Europe*, 2015.

[91] Xin Li, Jian Wang, L T Pileggi, Tun-Shih Chen, and Wanju Chiang. Performance-centering optimization for system-level analog design exploration. In *Proceedings of the IEEEACM International conference on Computer-aided design*, pages 422–429. IEEE Computer Society, May 2005.

[92] Bo Liu, Francisco V Fernández, and Georges G E Gielen. Efficient and accurate statistical analog yield optimization and variation-aware circuit sizing based on computational intelligence techniques. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(6):793–805, June 2011.

[93] Honghuang Lin and Peng Li. Circuit performance classification with active learning guided sampling for support vector machines. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 34(9):1467–1480, 2015.

[94] Liuxi Qian, Zhaori Bi, Dian Zhou, and Xuan Zeng. Automated technology migration methodology for mixed-signal circuit based on multistart optimization framework. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 23(11):2595–2605, December 2014.

[95] Lucas C Severo and Alessandro Girardi. A methodology for the automatic design of operational amplifiers including yield optimization. *26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, 2013.

[96] Dimitri P Bertsekas. *Nonlinear Programming.* Athena Scientific, Cambridge, MA., 1999.

[97] Dimitri P Bertsekas, Angelia Nedic, and Asuman Ozdaglar. *Convex Analysis and Optimization.* Athena Scientific., Belmont, MA., 2003.

[98] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory.* A consise introduction. Princeton university press, Princeton, NJ, 2012.

[99] R Rutenbar. Simulated annealing algorithms: an overview. *Circuits and Devices Magazine*, 1989.

[100] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[101] Ilya Sutskever. *Training Recurrent Neural Networks.* PhD thesis, University of Toronto, Toronto, January 2013.

[102] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, February 2011.

[103] Geoffrey Hinton, Nitrish Srivastava, and Kevin Swersky. Neural networks for machine learning coursera course, slide 29 of lecture 6a, overview of mini-batch gradient descent. *University of Toronto CSC321 Class notes*, 2016.

[104] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Neural information processing systems (NIPS)*, pages 2933–2941, 2014.

[105] Yann N Dauphin, Harm de Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. *arXiv.org*, February 2015.

[106] Diederik Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. *arXiv.org*, December 2014.

[107] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods. *arXiv.org*, November 2013.

[108] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, and Andrew Y Ng. Large scale distributed deep networks. *Neural information processing systems (NIPS)*, pages 1223–1231, 2012.

[109] Harold Szu and Ralph Hartley. Fast simulated annealing. *Physics letters A*, 122(3-4):157–162, June 1987.

[110] S Kirkpatrick, C D Gelatt, and M P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[111] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. *Metaheuristics for Multiobjective Optimisation*, pages 1–32, November 2003.

[112] K Deb, A Pratap, S Agarwal, and T Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[113] G M Morris, D S Goodsell, R S Halliday, and R Huey. Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 1998.

[114] J Horn, N Nafpliotis, and D E Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. *First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, 1:82–87, 1994.

[115] Om Prakash Agrawal. A general formulation and solution scheme for fractional optimal control problems. *Nonlinear Dynamics*, 38(1-4):323–337, 2004.

[116] A M Bloch and P E Croach. *Reduction of Euler Lagrange problems for constrained variational problems and relation with optimal control problems*, volume 3. IEEE, 1994.

[117] Dmitry S Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning using a numerical Hamilton-Jacobi-Bellman solver and an adaptive mesh refinement. *The International Journal of Robotics Research*, 35(5):570–584, October 2015.

[118] Rainer Buckdahn and Tianyang Nie. Generalized Hamilton–Jacobi–Bellman equations with Dirichlet boundary condition and stochastic exit time optimal control problem. *SIAM Journal on Control and Optimization*, 54(2):602–631, March 2016.

[119] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, March 2002.

[120] C G Sørensen, T Bak, and R N Jørgensen. Mission planner for agricultural robotics. *Proc AgEng*, 2004.

[121] Fernando Alfredo Auat Cheein and Ricardo Carelli. Agricultural Robotics: Unmanned Robotic Service Units in Agricultural Tasks. *IEEE Industrial Electronics Magazine*, 7(3):48–58, 2013.

[122] Fardin Abdi Taghi Abad, Marco Caccamo, and Brett Robbins. A fault resilient architecture for distributed cyber-physical systems. *2012 IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2012)*, pages 222–231, 2012.

[123] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129, January 2002.

[124] S M LaValle and J J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. *Citeseer*, 2000.

[125] Lydia Kavraki, Petr Svestka, Jean Latombe, and Mark Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[126] V Boor, M H Overmars, and A F van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. *International Conference on Robotics and Automation*, 2:1018–1023 vol.2, 1999.

[127] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):0278364915577958–921, May 2015.

[128] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime Motion Planning using the RRT*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011.

[129] Pierre Bonami, Alberto Olivares, and Ernesto Staffetti. Energy-optimal multi-goal motion planning for planar robot manipulators. *Journal of Optimization Theory and Applications*, 163(1):80–104, January 2014.

[130] Ryan Luna, Morteza Lahijanian, Mark Moll, and Lydia E Kavraki. Asymptotically optimal stochastic motion planning with temporal goals. In *Algorithmic Foundations of Robotics XI*, pages 335–352. Springer International Publishing, Cham, 2015.

[131] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: The driftless case. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2368–2375, 2015.

[132] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[133] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[134] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[135] James J Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, pages 995–1001. IEEE, 2000.

[136] Seyed Nematollah Ahmadyan. Duplex optimization toolbox: https://github.com/ahmadyan/Duplex. *Github code repository*, 2016.

[137] Seyed Nematollah Ahmadyan. Directed test genearation tool: https://github.com/ahmadyan/RRT. *Github code repository*, 2016.

[138] Seyed Nematollah Ahmadyan. Reachability analysis tool https://github.com/ahmadyan/Reachability. *Github code repository*, 2016.

[139] Seyed Nematollah Ahmadyan. Analog benchmarks: https://github.com/ahmadyan/analog. *Github code repository*, 2016.

[140] Seyed Nematollah Ahmadyan. Urbana SAT Solver https://github.com/ahmadyan/Urbana. *Github code repository*, 2016.

[141] Seyed Nematollah Ahmadyan. Rapidly-exploring random forests https://github.com/ahmadyan/RRT/tree/master/Myrkwood/Myrkwood. *Github code repository*, 2016.

[142] Seyed Nematollah Ahmadyan. Capacitated Selfish Replication Game: https://github.com/ahmadyan/Capacitated-Selfish-Replication-Game. *Github code repository*, 2016.

[143] Haralampos-G Stratigopoulos and Sedat Sunter. Fast Monte Carlo-based estimation of analog parametric test metrics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 33(12):1977–1990, December 2014.

[144] Rasit Onur Topaloglu. Early, accurate and fast yield estimation through Monte Carlo-alternative probabilistic behavioral analog system simulations. *24th IEEE VLSI Test Symposium*, pages 6 pp.–142, 2006.

[145] Seyed Nematollah Ahmadyan, Jayanand Asok Kumar, and Shobha Vasudevan. Runtime verification of nonlinear analog circuits using incremental time-augmented RRT algorithm. *DATE 2013*, pages 1–6, December 2012.

[146] A Shkolnik, M Walter, and R Tedrake. Reachability-guided sampling for planning under differential constraints. *International Conference on Robotics and Automation*, January 2009.

[147] Eric Thiémard. An algorithm to compute bounds for the star discrepancy. *Journal of Complexity*, 17(4):850–880, December 2001.

[148] Hassan K. Khalil. *Nonlinear Systems (3rd Edition)*. Prentice Hall, 3 edition, December 2001.

[149] Francesca Mazzia and Cecilia Magherini. Test set for initial value problem solvers, release 2.4. Technical Report 4-2008, Department of Mathematics, University of Bari, Italy, February 2008.

[150] E Hairer and G Wanner. *Solving ordinary differential equations II: stiff and differential-algebraic problems*. Springer-Verlag, April 1996.

[151] Thao Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, Verimag, Institut National Polytechnique de Grenoble, May 2006.

[152] John Havlicek, Dana Fisman, and Cindy Eisner. Basic results on the semantics of Accellera PSL 1.1 foundation language. *Accellera Technical Report, IBM Haifa Research Lab*, April 2004.

[153] S Gupta, B H Krogh, and R A Rutenbar. Towards formal verification of analog designs. In *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pages 210–217, 2004.

[154] Paul Pedersen. Multivariate Sturm theory. *Applied algebra, algebraic algorithms and error-correcting codes*, 539(Chapter 30):318–332, June 1991.

[155] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, pages 153–162, January 1987.

[156] J Comba and B Naylor. Conversion of binary space partitioning trees to boundary representation. In *Proceedings of Theory and Practice of Geometric*, January 1996.

[157] Gene H. Golub and Charles F. van der Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)(3rd Edition)*. The Johns Hopkins University Press, 3rd edition, October 1996.

[158] Robert G Bartle. The elements of integration and Lebesgue measure. John Wiley & Sons Inc, New York, 1995.

[159] Seyed Nematollah Ahmadyan, Suriyaprakash Natarajan, and Shobha Vasudevan. Every test makes a difference: compressing analog tests to decrease production costs. In *2016 21st Asia and South Pacific Design and Automation Conference (ASP-DAC)*, pages 539–544, Macau, China, 2016. IEEE.

[160] J F Parker and D Ray. A 1.6-GHz CMOS PLL with on-chip loop filter. *IEEE Journal of Solid-State Circuits*, 33(3):337–343, March 1998.

[161] Kyoohyun Lim, Chan-Hong Park, Dal-Soo Kim, and Beomsup Kim. A low-noise phase-locked loop design by loop bandwidth optimization. *IEEE Journal of Solid-State Circuits*, 35(6):807–815, 2000.

[162] Vladimir N Vapnik. *Statistical Learning Theory.* Wiley-Interscience, 1998.