Automated Transient Input Stimuli Generation for Analog Circuits

Seyed Nematollah Ahmadyan, *Student Member, IEEE* and Shobha Vasudevan, *Senior Member, IEEE*

***Abstract*— We present an automated directed random input stimulus generation algorithm with high coverage for nonlinear analog circuits. Our methodology is able to generate input stimuli to meet two kinds of objectives: (i) to reach user-defined goal regions and (ii) increased coverage of state space. The principal benefit of our approach is that it can provide directed input stimulus generation, as opposed to the randomly generated input stimulus by Monte Carlo-based methods. The methodology introduces Multiple Objective Rapidly-exploring Random Trees (MORRTs), which add a bias and a feedback loop to the standard RRT algorithm. The biasing is provided by a statistical inference algorithm. Simultaneous biasing towards goal regions and coverage is possible in MORRT to a user-defined extent. Our methodology generates several input stimuli that are concentrated in the *goals* or relevant operating regions, while providing high coverage of the state space. We demonstrate the efficiency and scalability of our approach on high-dimensional analog case studies.**

***Index Terms*— Directed input stimulus generation, Rapidly-exploring Random Trees, Nonlinear analog circuits.**

I. Introduction

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]. The analog IC 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 complexity of the circuits has increased significantly beyond that of the hand-crafted, isolated analog circuits of the past. Traditionally, the practice in pre-silicon and post-silicon analog validation has been to use input stimuli manually generated by the designer of the circuit[1]. However, given the scale and complexity of the analog components used in modern devices, that process is inefficient, expensive, error-prone, and frequently misleading in predicting unknown behaviors, worst case corners and stressing weak components of the circuit. Hence, automated input stimulus generation for analog circuits has been identified as a critical need in the analog design and validation process [2].

In this work, we present an *automatic directed input stimulus generation technique* 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.

[1]We use the terms *input stimuli* and *test cases* interchangeably in this paper.

A. Motivation

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 are 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). The iPhone-6 has 27 chips, 20 of which are analog/mixed signal. [3]. 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 [4] [5]. 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.

Our approach is based on *Rapidly-exploring Random Trees (RRTs)*. The RRT algorithm is a random tree simulation, which, in contrast to the random walk simulation of Monte Carlo methods, is able to branch from any previously simulated state. The RRT 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 RRT 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 [6], 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 [7], 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 [7] we used a clustering algorithm, in this work, we infer these goal and coverage distributions automatically using Variational Bayesian inference (VBI)[8], 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.

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 $199\times$ more iterations and $188\times$ 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 [9]. 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 [10].

In [7] 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 [7] 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 qgenerate 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 [7]) that show the efficacy, efficiency and scale of the MORRT.

II. Related Work

We refer our readers to [11] for an introductory tutorial and to [12] for a review of the classic works. Researchers have focused on generating post-silicon tests for nonparametric testing [13] [14] [15] [16], and parametric fault models [17]. Some techniques use learning algorithms to identify bad regions [18][17].

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 [19] [20][21][22][23].

The RRT algorithm was originally developed in robotic motion planning [24]. In the classic RRT, the growth of RRTs is locally, but not globally, optimal. Several techniques have tried to address that issue [23][20] [25]. In [19], the authors propose to introduce LTL properties into RRT to verify safety

properties of hybrid systems for falsification. [23] use RRT to generate counter-examples in analog and hybrid systems.

Sample generation in the state space is one of the major issues in test generation. Dang et al. use RRT with coverage measurement to improve coverage quality and use RRT to verify hybrid systems [23].

Our advantage over the current state of the art random testing algorithms such as Markov Chain Monte Carlo [26] methods (such as Gibbs sampling [26] or Metropolis-Hastings algorithm [26]) is that our approach is a directed testing approach that allows us to control the simulation targets [7] as well as providing high coverage. As a result, we generate input stimuli with higher quality. Our algorithm can be fine tuned for generating goal-oriented input or coverage-driven input stimuli. Markov-Chain Monte Carlo doesn't provide these degrees of control over the simulation. On the other hand, similar to Monte Carlo Markov Chain, our algorithm avoids doubling-back on the states that we have already visited [26].

The MORRT algorithm is an *incomplete* search method. As a result, we do not provide an exhaustive search of the state space. For formal verification, many papers have addressed exhaustive safety verification through formal verification[27], reachability analysis [28][29] and simulation-aided verification [30][31][32]. The reachability analysis algorithm can determine if the reachable set of the circuit intersects with the unsafe or the goal region. Formal methods are sound, rigorous and provide complete guarantee of the result. In comparison to formal methods, MORRT is efficient, highly scalable and can generate input stimuli which can be used to falsify the circuit [6].

III. 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 [33]. 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 V-D Figure 4).

Figure 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 IV-C) 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 VI-C. 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 V-C). 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 V-D).

IV. Preliminaries

A. Models for Nonlinear Systems

A nonlinear time-variant circuit is modeled as a differential algebraic equations (DAEs) through modified nodal analysis (MNA) [33] 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 [33]. A nonlinear analog circuit is described by an n-dimensional differential algebraic equation[2]:

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

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

$$F(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t), t) = 0 \tag{1}$$

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 1 determine the trajectory of the circuit in the interval $t \in [t_1\ t_2]$. 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

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

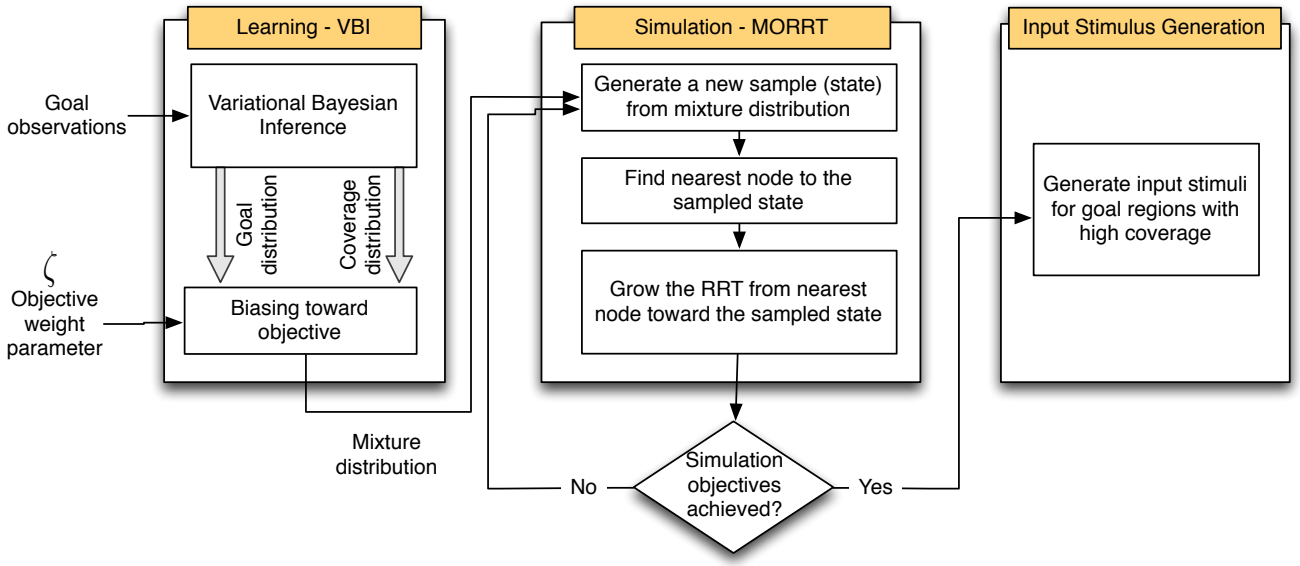


Fig. 1: Framework of our directed input stimulus generation technique (Section III)

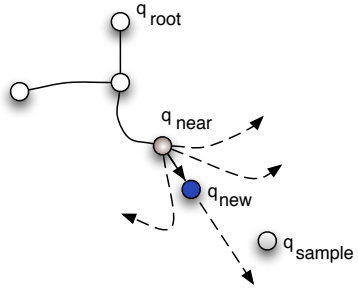


Fig. 2: Growth of RRT through addition of a new node sampled from the state space.

inputs (in analog circuits) and piecewise continuous inputs (like analog interfaces such as DAC circuits). Equation 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 1.

B. Rapidly-exploring Random Trees

We briefly describe the RRT algorithm presented in [24]. The RRT is a tree data structure. The tree is initialized through fixing of its root at a specified state[3] in the state space $\mathbb{S}$. The tree is then *grown* incrementally through 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 $\mathbf{u}$.

[3]Throughout this paper, *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**

Algorithm 1 describes the growth of the tree $\mathbb{G}$ in the classic RRT algorithm [24]. At every iteration, the RRT algorithm generates a random state q_{sample} **uniformly distributed** in the state space. For every new generated state q_{sample}, the RRT algorithm will find a 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 [23][20] 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 trajectory is determined, the tree will be expanded from q_{near} toward q_{new} through 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 2 shows the growth of the RRT tree toward a given sample node. The RRT algorithm will terminate after a fixed number of iterations $MAX - ITER$.

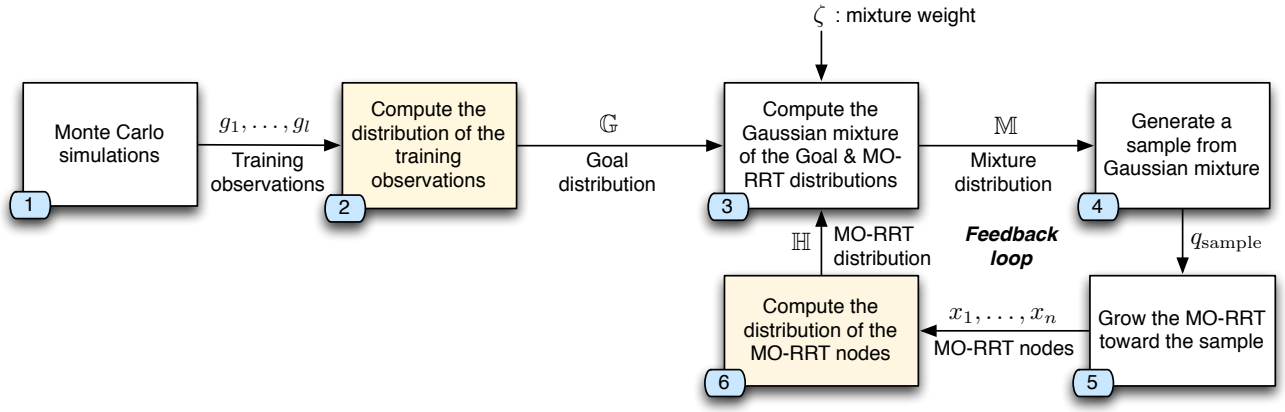


Fig. 3: Detailed block diagram of the learning phase of the Multi-Objective RRT algorithm. First, we identify the goal distribution (block 1 and 2). We grow the MORRT by sampling states from the mixture distribution. We feed the MORRT states back to the learning algorithm to update the mixture distribution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

C. Variational Bayesian Inference

In this section, we describe the *variational Bayesian inference* (VBI) algorithm [8][4]. The VBI algorithm infers a mixture distribution of the given set of samples. 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 who's its 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

$$\sum_{i=1}^{K} \pi_i \, \mathcal{N}(\mu_i, \Lambda_i^{-1}) \tag{2}$$

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) 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 $k = 1 \ldots K$. 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 $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$. 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 [8] 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

$$p(\mathbf{Z}|\pi) = \prod_{i=1}^{n} \prod_{j=1}^{K} \pi_j^{\mathbf{z}_{ij}}. \tag{3}$$

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

$$p(\mathbf{X}|\mathbf{Z}, \mu, \mathbf{\Lambda}) = \prod_{i=1}^{n} \prod_{j=1}^{K} \mathcal{N}(\mathbf{x}_i|\mu_j, \mathbf{\Lambda}_j^{-1})^{\mathbf{z}_{ij}}, \tag{4}$$

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* [8] distribution for mixture weights π:

$$p(\pi) = Dir(\pi|\alpha_0) = C(\alpha_0) \prod_{i=1}^{K} \pi_i^{\alpha_0 - 1} \tag{5}$$

where $C(\alpha_0)$ is the normalization constant for Dirichlet distribution [8]. For mean and precision, we assume a *Gaussian-Wishart* [8] prior distribution, as follows:

$$p(\mu, \mathbf{\Lambda}) = p(\mu|\mathbf{\Lambda}) p(\mathbf{\Lambda}) \tag{6}$$

$$= \prod_{i=1}^{K} \mathcal{N}(\mu_i|\mathbf{m}_0, (\beta_0 \mathbf{\Lambda}_i)^{-1}) \mathcal{W}(\mathbf{\Lambda}_i|\mathbf{W}_0, v_0), \tag{7}$$

In [8], 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

$$p(\mathbf{X}, \mathbf{Z}, \pi, \mu, \mathbf{\Lambda}) = p(\mathbf{X}|\mathbf{Z}, \mu, \mathbf{\Lambda}) p(\mathbf{Z}|\pi) p(\pi) p(\mu|\mathbf{\Lambda}) p(\mathbf{\Lambda}) \tag{8}$$

Where $\mathbf{X}$ is the set of samples and $\mathbf{Z}$ is the latent variable.

[4]We only provide the inputs and assumptions that we used in this algorithm, and we refer interested readers to [8] for more details on the algorithm.

We assume that variational distribution factorizes between the latent variables and parameters such that

$$q(\mathbf{Z}, \pi, \mu, \mathbf{\Lambda}) = q(\mathbf{Z})q(\pi, \mu, \mathbf{\Lambda}) \tag{9}$$

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). Further details of this technique can be found in [8]. 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 [8]. 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 [8], 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 [8]. Finally, the VBI algorithm can be implemented online. As a result, the algorithm can compute and update the sample distribution incrementally [34].

V. Proposed Directed Input Stimulus Generation Algorithm: Multi-Objective RRT

The details of our Multi-Objective RRT algorithm are explained in Algorithm 2. 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:

$$\mathbb{M} = (1 - \zeta) \times \mathbb{H} + \zeta \times \mathbb{G}. \tag{10}$$

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.

Algorithm 2 Multi-Objective RRT algorithm

1: ζ: 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

For simplicity, first we explain the case where $\zeta = 1$ and the algorithm is purely goal-oriented, as in [7]. In this case, the goal states $\{g_1, \dots, g_l\}$ 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 $\{g_1, \dots, g_l\}$ (Algorithm 2, line 6). We determine the Gaussian mixture distribution of the goal states $\mathbb{G}$ using the VBI algorithm (Algorithm 2, line 7).

In the simulation phase (Algorithm 2, 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 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 2, 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 V-D).

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 2, 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 2, line 16).

Each state $x_1, \ldots, x_n$ in the MORRT is 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 2, 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.

A. Inferring the Goal and Coverage Distributions

We utilize the *variational Bayesian inference (VBI)* (Section IV-C), [8] algorithm to determine the distribution of the goal states and visited states (MORRT states) in the learning phase of the MORRT algorithm (Figure 3). In the learning phase, we compute the distribution of the goal states from the training data $\{g_1, \ldots, g_l\}$ (Figure 3, 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 3, block 4).

1) 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 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 states does not have to be a solution of the circuit or even reachable. MORRT will find an input stimuli 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). 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:

$$\mathbb{G}(\boldsymbol{\mu}_{\mathbb{G}}, \boldsymbol{\Lambda}_{\mathbb{G}}) = \sum_{i=1}^{K_{\mathbb{G}}} \boldsymbol{\pi}_{\mathbb{G}_i} \; \mathcal{N}(\boldsymbol{\mu}_{\mathbb{G}_i}, \boldsymbol{\Lambda}_{\mathbb{G}_i}^{-1}) \tag{11}$$

2) Infer the Coverage Distribution $\mathbb{H}$: As the MORRT algorithm explores the reachable state space, it provides 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):

$$\mathbb{H}(\boldsymbol{\mu}_{\mathbb{H}}, \boldsymbol{\Lambda}_{\mathbb{H}}, \boldsymbol{\pi}_{\mathbb{H}}) = \sum_{i=1}^{K_{\mathbb{H}}} \boldsymbol{\pi}_{\mathbb{H}_i} \; \mathcal{N}(\boldsymbol{\mu}_{\mathbb{H}_i}, \boldsymbol{\Lambda}_{\mathbb{H}_i}^{-1}) \tag{12}$$

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.

B. Biasing towards Objectives

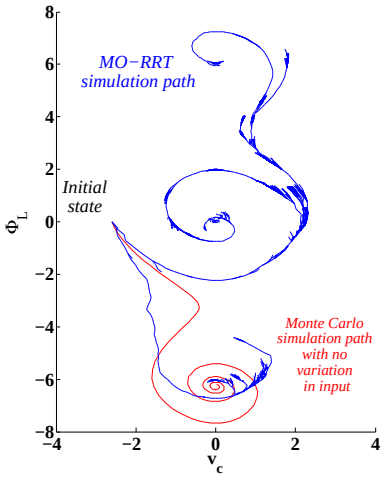
The most important part of the MORRT algorithm is generation of the biased states (Figure 3 - block 3). In our algorithm, we bias the states toward the distribution of the goal region or the reachable space according to the parameter ζ. The goal distribution and the coverage distribution are the mixture Gaussian distribution $\mathbb{G}$ and the $\mathbb{H}$ determined by the VBI algorithm (Equations 11 and 12). The biased mixture distribution is a mixture of the goal and coverage distributions proportional to the weight parameter ζ:

$$\mathbb{M}(\mathbf{x}) = (1-\zeta)\times\mathbb{H}(\boldsymbol{\mu}_{\mathbb{H}}, \boldsymbol{\Lambda}_{\mathbb{H}}, \boldsymbol{\pi}_{\mathbb{H}})+\zeta\times\mathbb{G}(\boldsymbol{\mu}_{\mathbb{G}}, \boldsymbol{\Lambda}_{\mathbb{G}}, \boldsymbol{\pi}_{\mathbb{G}}). \tag{13}$$

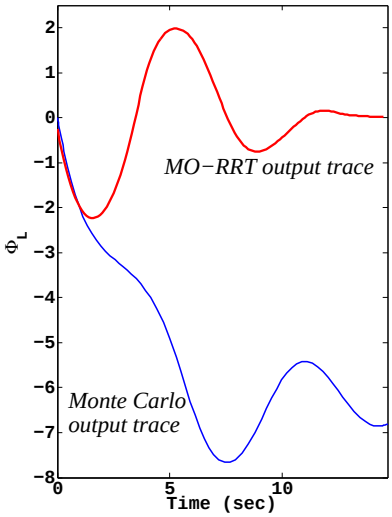
As a result of our weight mixture, if the user specifies $\zeta = 0$, then the biased distribution $\mathbb{M}$ is the same as the coverage distribution, and our algorithm is completely coverage-driven. Similarly, if the user specifies $\zeta = 1$, then the biased distribution $\mathbb{M}$ is equal to 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.

C. MO-RRT based circuit simulation

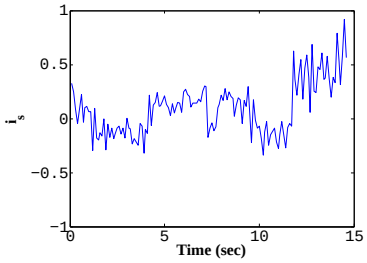
We grow the MORRT in the simulation phase of the algorithm to explore the state space. At every iteration, we generate a state from the mixture distribution of the goal and coverage distributions. We find the nearest node in the



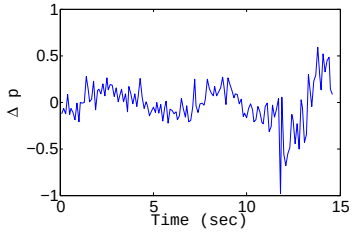
(a) The state space of the Josephson circuit. The MORRT algorithm explores very different paths from the nominal simulation. Our algorithm is able to excite behaviors of the circuit that are normally very hard to test.



(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.

Fig. 4: An example of the input stimuli generated for the Josephson circuit (Section VI-A) from the MORRT. Generating input stimulus for Josephson circuit is difficult using conventional Monte Carlo methods.

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 circuits 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.

D. 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 6c.

Figure 4a shows the state space of the Josephson circuit (explained in Section VI-A) where the initial state is selected at state $(-2.6, 0)$. If we use nominal inputs, the circuit will end up in the state at equilibrium state $(0, -6)$, which results in the output trace shown in Figure 4b. However, if we use Multi-Objective RRT, we can explore alternative goal regions and obtain different results (Figure 4a). 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 4c and 4d) and report them to the user as input stimuli.

VI. 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 1) for the netlist. Next, we construct the MORRT data structure and grow the RRT in the state space of the circuit according to Algorithm 2. 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 VI-A); and a CMOS opamp circuit 8-dimensional state space (Section VI-D). In 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.

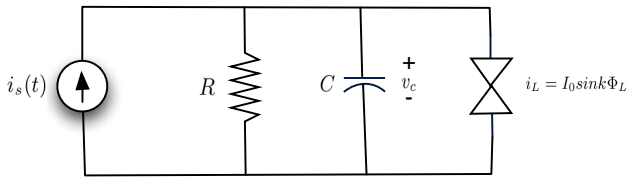


Fig. 5: Josephson junction circuit.

A. Effects of ζ on Growth of the MORRT

The Josephson junction circuit is shown in Figure 5. The Josephson junction is a time-invariant nonlinear inductor governed by Equation 14. As before, we set $\Delta t = 0.1$. We executed the classic RRT algorithm for 3,000 iterations. We executed 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.

$$i_L = I_0 \times sin(k\Phi_L) \tag{14}$$

Therefore, the differential system for the circuit in Figure 5 is

$$\dot{v}_c = \frac{1}{C}(\frac{1}{R}v_c - I_0 sin(k\Phi_L) + i_s(t)) \tag{15}$$

$$\dot{\Phi}_L = v_c \tag{16}$$

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 6 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 6, 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 6 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 VI-C, we observed that setting $\zeta = 0.5$ yielded the best trade-off between coverage and concentration of tests around the goal regions.

B. 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 VI-A. The initial state was selected at state $(-2.6, 0)$ (similar to Section V-D Figure 4a). 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.
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.

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

TABLE I: Performance comparison of MORRT vs M.C.

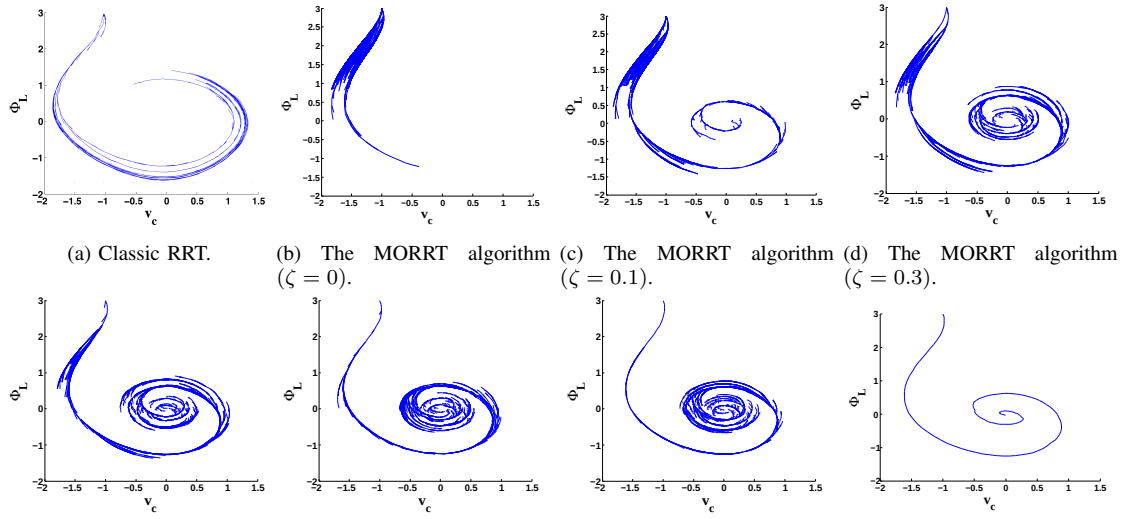
Table I shows the result of the performance comparison. Given the same number of iterations, Monte Carlo was 22.6% faster than our algorithm. However, MC didn't find any goal input stimuli, whereas our algorithm was able to generate 487 input stimuli directed toward the goal region. This experiment demonstrates 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 $199\times$ more iterations and $188\times$ more time to generate 30 goal input stimuli, as compared to our directed approach.

C. 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 equi-distributed 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 [10][23] to compute the discrepancy. The star discrepancy has previously been used by [23] 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.



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

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

Fig. 6: Exploring the state space of a Josephson junction circuit using the classic RRT and MORRT. Figure 6a shows the classic RRT algorithm; for the given number of iterations (3,000), the algorithm did not converge. Figures 6b to 6g show the various MORRT results for different increasing values of ζ. Finally, Figure 6h 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).

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

$$D(P, I) \equiv \Big|\frac{A(P, I)}{k} - Vol(I)\Big| \tag{17}$$

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 [10][23] of the state set P in the box B is defined as

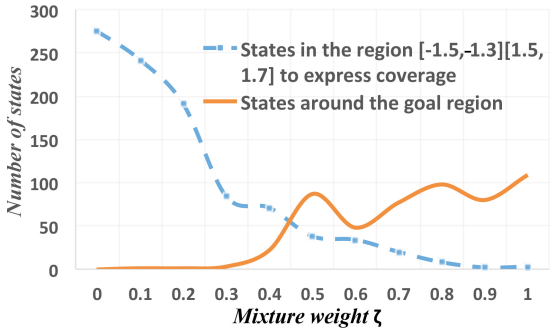
$$D^*(P, B) \equiv \sup_{i \in I} D(P, i). \tag{18}$$

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

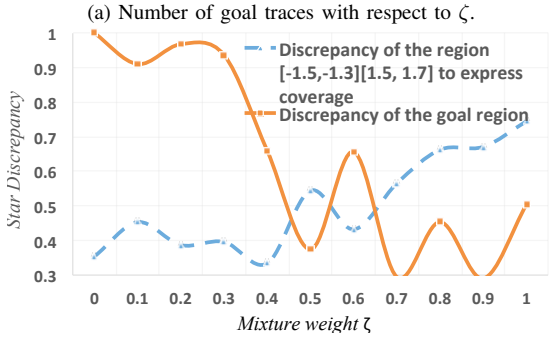
The range of the star discrepancy is the set $(0, 1]$, where low discrepancy means a more uniform set and a higher discrepancy indicates greater nonuniformity. In general, generating a low-discrepancy random sequence is very difficult. We estimated the coverage of the MORRT algorithm with respect to 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 7 shows the discrepancy and goal-orientedness results of the MORRT algorithm. Figure 7a 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 7a, as 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 7a are confirmed by the results in Figure 6. The increase in the number of states around the origin indicates that our algorithm is more goal-oriented as ζ increases. Figure 7a clearly shows the correlation between the mixture weight ζ and the number of goal traces. Figure 7b 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 7b 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 7, 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.

D. 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 8, 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 $0.18um$ library. We sat $VDD = -VSS = 0.9v$. Each test was applied to the V_{in} signal. The output of the opamp was saturating at $0.2v$ and $-0.8v$, respectively.



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



(b) Star discrepancy with respect to ζ.

Fig. 7: Effects of ζ on discrepancy and number of states in MORRT.

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 8). 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 be completed 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 [6]. 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 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.

1) Generating functional tests: We generated a test that would saturate the output of the opamp circuit. We manually generated random vector where $v_o = 0.2v$ and $v_o = -0.8v$ 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 (Fig. 3 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

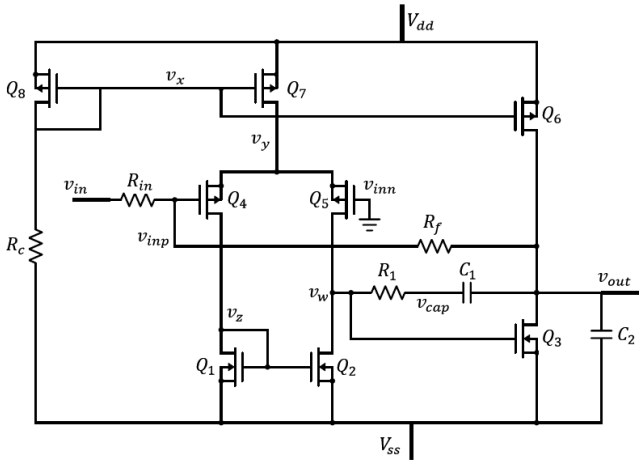
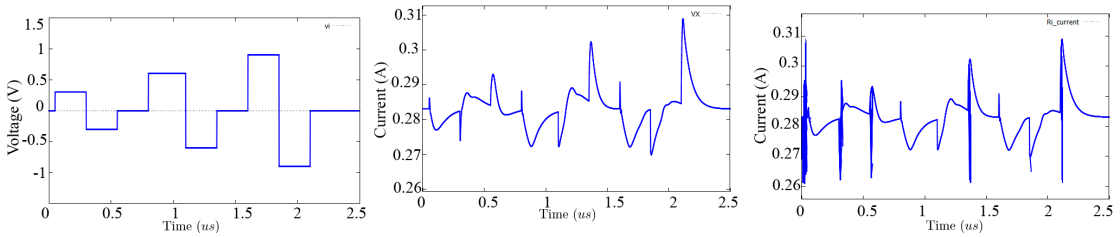


Fig. 8: Schematic of the opamp circuit.

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 10a 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 length of the input stimuli generated by MO-RRT are very compact and efficient.

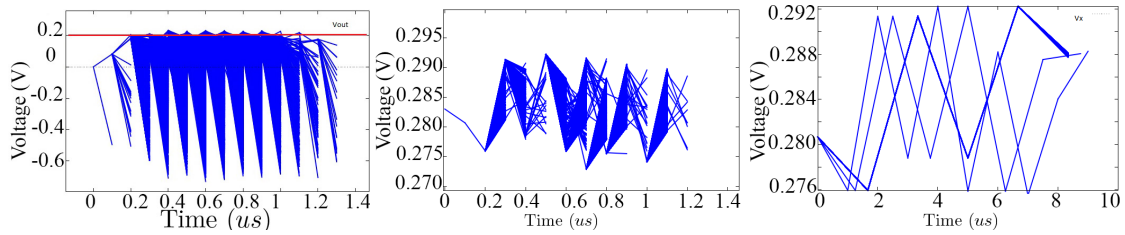
2) Generating stress tests: We used our algorithm to generate stress tests for different components on the circuit. For stress testing the components, initially we computed a "profile" for each node in the circuit. We applied the input signal v_{square} as shown in the Figure 9a to compute the minimum and maximum value of the current through the resistor. Figure 9b shows the voltage v_x and the current through resistor $R_1 = 2.1k\Omega$. 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 $i_{R_1} = 0.6mA$. 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 $i_{R_1} = 0.6mA$ as the goal region and generated tests that would reach that region. Figure 9c 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, where as the manually generated test by the designer requires $200us$ 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.

3) 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 10a shows the MORRT G_1 that is used to generate tests for saturating the output voltage. Similarly, we computed



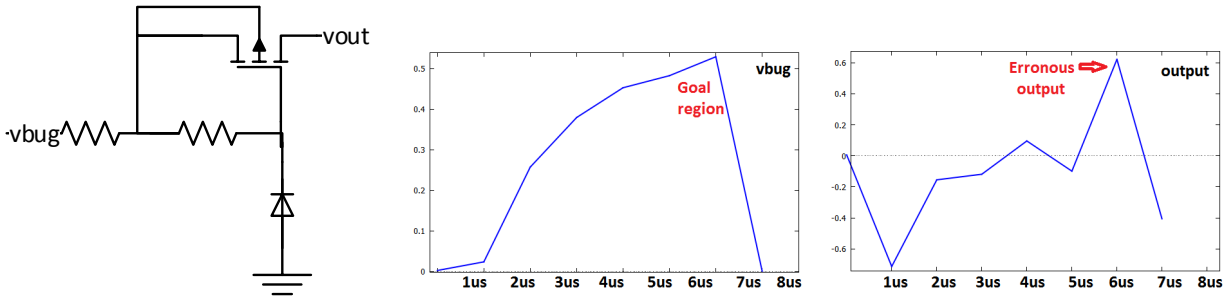
(a) The input signal to generate signal profile (b) The current profile of the resistor R_1 (c) The MORRT algorithm generates a stress tests for resistor R_1.

Fig. 9: Generating tests for stressing the resister R_1



(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.

Fig. 10: 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.



(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.

Fig. 11: 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.

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 vise 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 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 $(v_o, v_x) = (0.2, 0.3)$. We grow the MO-RRT toward the goal region $(0.2, 0.3v)$. Figure 10b 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 10c shows the extracted tests from the combined MO-RRT that saturates the output and stresses resistor R_c.

4) 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 bug by adding a small voltage limiter subcircuit to the opamp circuit as shown in Figure 11a. 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 11b and 11c 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.

E. 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-0.18um process, shown in Figure 13a, to validate its functionality and interface. The netlist was extracted from the VCO layout with all the parasitic capacitance and resistors. Figure 13b shows the oscillation of the VCO circuit where $V_{\text{bias}} = 750mv, V_{\text{control}} = 630mv$ and $V_{dd} = 1.8v$. It takes $10.8ns$ for the output to reach its peak-to-peak maximum. The output oscillates between 0.7^v 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 13c shows the result of the experiment. The MORRT successfully generated stimuli that drives 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 0^v 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 prevents the output to become 0. The length of the test was $0.65ns$.

F. Complexity Analysis

In the classic RRT algorithm (and MORRT), the most computationally expensive part of the algorithm is the search for the nearest node in the tree. If the exhaustive search method is used, the complexity of the algorithm is $\mathcal{O}(dn^2)$, where n is the number of states and d is the number of dimensions. The RRT can be optimally constructed using the *KD-Tree* data structure [24], which is very similar to a binary search algorithm for high-dimensional spaces. The KD-tree reduces the complexity of the search operation to $\mathcal{O}(d \log n)$. Therefore, the algorithm has a computational complexity of $\mathcal{O}(dn \times \log n)$.

In our approach, there is a learning phase for identification of relevant regions; We used the VBI algorithm to infer the goal distribution and coverage distribution. In our learning phase, we use l states. The number of learning states is limited and fixed in advance ($l << k$); hence it does not impose any computational complexity on the algorithm. The complexity of generating states from the distribution is $\mathcal{O}(k)$; therefore, the complexity of our algorithm is:

$$\mathcal{O}(l) + \mathcal{O}(k) + \mathcal{O}(dk \times \log k) = \mathcal{O}(dk \times \log k). \qquad (19)$$

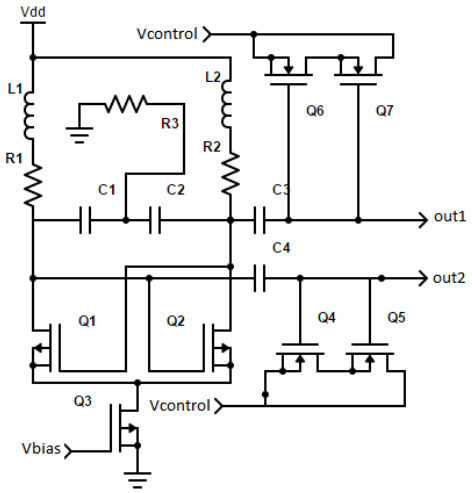


Fig. 12: Schematic of the VCO circuit.

In conclusion, we presented a novel pre-silicon directed input stimulus generation algorithm for nonlinear analog circuits that use a Multi-Objective RRT algorithm. The proposed Multi-Objective MORRT algorithm quickly converges toward the goal regions and provides high coverage. Our algorithm is useful for generating goal-oriented and high coverage input stimulus for analog circuits in order to check their functionality during the design process.

REFERENCES

[1] GBI Research, *Analog Integrated Circuits (IC) Market to 2016*. GBI research group market report No. GBISC029MR, 2012.

[2] *International Technology Roadmap for Semiconductors*. Available online at http://www.itrs.net/Links/2011ITRS/2011Chapters/2011Design.pdf, (accessed May 2014), 2011.

[3] *Teardown.com Analysis: The Apple iPhone 6 and 6 Plus*. Available online at http://www.techinsights.com/teardown.com/apple-iphone-6/, (accessed July 2015), 2015.

[4] H.-G. Stratigopoulos and S. Sunter, "Fast monte carlo-based estimation of analog parametric test metrics," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, pp. 1977–1990, Dec 2014.

[5] R. Topaloglu, "Early, accurate and fast yield estimation through monte carlo-alternative probabilistic behavioral analog system simulations," in *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, pp. 6 pp.–142, April 2006.

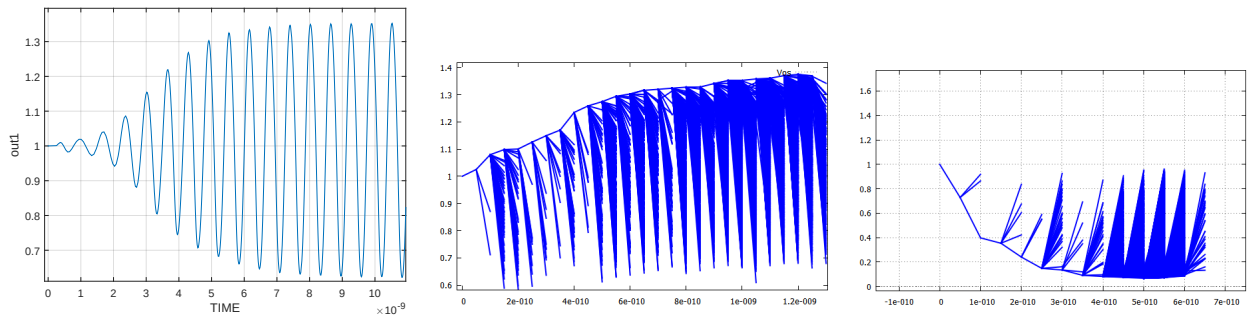
[6] S. N. Ahmadyan and S. Vasudevan, "Runtime verification of nonlinear analog circuits using incremental time-augmented RRT algorithm," *Proceedings of Design, Automation and Test in Europe Conference*, pp. 21–26, March 2013.

[7] S. N. Ahmadyan, J. A. Kumar, and S. Vasudevan, "Goal oriented stimulus generation for analog circuits," *Proceedings of Design Automation Conference (DAC)*, pp. 1018–1023, June 2012.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] A. Shkolnik, M. Walter, and R. Tedrake, "Reachability-guided sampling for planning under differential constraints," in *Proceeding of IEEE International Conference on Robotics and Automation*, pp. 2859–2865, May 2009.

[10] E. Thiemard, "An algorithm to compute bounds for the star discrepancy," *Journal of Complexity*, vol. 17, pp. 850–880, 2001.



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

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

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

Fig. 13: Generating input stimuli for VCO circuit.

[11] L. S. Milor, "A tutorial introduction to research on analog and mixed-signal circuit testing," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 45, no. 10, pp. 1389–1407, 1998.

[12] P. Duhamel and J. Rault, "Automatic test generation techniques for analog circuits and systems: A review," *IEEE Transactions on Circuits and Systems*, vol. 26, no. 7, pp. 411–440, 1979.

[13] R. Voorakaranam and A. Chatterjee, "Test generation for accurate prediction of analog specifications," in *Proceedings of 18th IEEE VLSI Test Symposium*, pp. 137–142, 2000.

[14] A. Abderrahman, B. Kaminska, and E. Cerny, "Optimization-based multifrequency test generation for analog circuits," *Journal of Electronic Testing*, vol. 9, no. 1-2, pp. 59–73, 1996.

[15] F. Liu and S. Ozev, "Statistical test development for analog circuits under high process variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 8, pp. 1465–1477, 2007.

[16] H. G. Stratigopoulos, "Test metrics model for analog test development," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 1116–1128, 2012.

[17] C.-Y. Pan and K.-T. Cheng, "Test generation for linear time-invariant analog circuits," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 5, pp. 554–564, 1999.

[18] P. Mukherjee, G. P. Fang, R. Burt, and P. Li, "Efficient identification of unstable loops in large linear analog integrated circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 9, pp. 1332–1345, 2012.

[19] 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*, vol. 34, pp. 157–182, 2009.

[20] J. Kim and J. Esposito, "Adaptive sample bias for rapidly-exploring random trees with applications to test generation," *Proceedings of American Control Conference*, pp. 1166–1172 vol. 2, June 2005.

[21] M. H. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, vol. 39, no. 12, pp. 1395–1404, 2008.

[22] A. Julius, G. Fainekos, M. Anand, I. Lee, and G. Pappas, "Robust test generation and coverage for hybrid systems," in *Hybrid Systems: Computation and Control* (A. Bemporad, A. Bicchi, and G. Buttazzo, eds.), vol. 4416 of *Lecture Notes in Computer Science*, pp. 329–342, Springer Berlin Heidelberg, 2007.

[23] T. Dang and T. Nahhal, "Coverage-guided test generation for continuous and hybrid systems," *Formal Methods in System Design*, vol. 34, pp. 183–213, 2009.

[24] S. M. LaValle, *Planning Algorithms*. Cambridge, UK: Cambridge University Press, 2006.

[25] S. Proch and P. Mishra, "Directed test generation for hybrid systems," in *Proceedings of 2014 15th International Symposium on Quality Electronic Design (ISQED)*, pp. 156–162, March 2014.

[26] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, "Bayesian Data Analysis, third edition," CRC Press, 2014.

[27] L. Yin, Y. Deng, and P. Li, "Simulation-assisted formal verification of nonlinear mixed-signal circuits with bayesian inference guidance," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, pp. 977–990, July 2013.

[28] S. N. Ahmadyan and S. Vasudevan, "Reachability analysis of nonlinear analog circuits through iterative reachable set reduction," in *Proceedings of the IEEE/ACM International Conference on Design, Automation and Test in Europe (DATE)*, 2013.

[29] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, no. 3, pp. 263–279, 2008.

[30] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid petri nets," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '06, (New York, NY, USA), pp. 275–282, ACM, 2006.

[31] M. J. Holzinger and D. J. Scheeres, "Reachability set subspace computation for nonlinear systems using sampling methods," *IEEE Conference on Decision and Control and European Control Conference*, pp. 7317–7324, Dec. 2011.

[32] W. Tan, U. Topcu, P. Seiler, and G. Balas, "Simulation-aided reachability and local gain analysis for nonlinear dynamical systems," *Decision and Control*, Jan. 2008.

[33] L. T. Pillage, R. A. Rohrer, and C. Visweswariah, *Electronic Circuit and System Simulation Methods*. McGraw-Hill Professional Publishing, 1995.

[34] V. Smidl and A. Quinn, *The Variational Bayes Method in Signal Processing*. Springer, 2006.