Accelerated Verification of Integrated Circuits Against the Effects of Process, Voltage and Temperature Variations

by

Michael Shoniker

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

in

Integrated Circuits & Systems

Department of Electrical and Computer Engineering
University of Alberta

Edmonton, Alberta
Fall 2015

Abstract

As Moore's Law continues to drive the advancement of new complementary metal-oxide semiconductor (CMOS) technology generations toward feature sizes in the sub 10 nm regime, the role of process, voltage and temperature (PVT) variations have become increasingly important when designing integrated circuits. very large scale integration (VLSI) designers are challenged to get millions or even billions of transistors to reliably operate over a wide range of possible operating conditions. This is not an easy task. Variations in transistor properties due to the inherent statistical variability of the fabrication process, power supply voltages, and operating temperatures all determine the conditions that affect the behaviour and/or output of an integrated circuit (IC). IC designers commonly model PVT variations as a combination of discretized values that represent the full range of expected operating conditions and are referred to as PVT corners. Designing such a large system of transistors, while ensuring that a high yield of devices will operate correctly, requires designers to consider all possible combinations of PVT corners to determine the worst-case performance of a circuit during the verification process. Traditional verification methods require SPICE-level circuit simulations for every possible PVT corner to find the worst-case performance of a circuit, which has become both time consuming and computationally expensive as modern designs can have several thousands of PVT corners. The PVT verification problem can be treated as a mathematical optimization problem. This thesis proposes a verification method that uses a machine learning model and heuristics to determine worst-case PVT corners without the need for simulating all possible combinations. The Rapid PVT Verification (RPV) algorithm that was developed employs a Gaussian process model (GPM) to extract information generated by PVT simulations to create a model of a given circuit's output behaviour. Heuristics are used to further exploit the information provided by the GPM to determine if the global optimum (worst-case) has been found by the algorithm to a certain degree of confidence.

Acknowledgements

I owe a debt of gratitude to my co-supervisors, Dr. Bruce Cockburn and Dr. Jie Han. Their patience and invaluable guidance were crucial to the writing of this thesis. Working with you both was a wonderful learning experience. I would also like to thank Dr. Duncan Elliot, Dr. Witold Pedrycz, Nancy Minderman, Russell Dodd, Jinghang Liang, David Sloan and all my colleagues in the Electrical & Computer Engineering department at the University of Alberta. A special thanks to Dr. Trent McConaghy and everyone at Solido Design Automation for their advice and for providing circuit datasets.

This research was supported by an Engage Grant and by a Strategic Project Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). I would also like to thank the Faculty of Graduate Studies for the Queen Elizabeth II Graduate Scholarship award and also the Electrical & Computer Engineering department for nominating me.

"With ideas it is like with dizzy heights you climb: At first they cause you discomfort and you are anxious to get down, distrustful of your own powers; but soon the remoteness of the turmoil of life and the inspiring influence of the altitude calm your blood; your step gets firm and sure and you begin to look - for dizzier heights."
Nikola Tesla

"Research is what I'm doing when I don't know what I'm doing."
Wernher von Braun

To my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

CCD central composite design

CMOS complementary metal-oxide semiconductor

CPU central processing unit

DE differential evolution

EA evolutionary algorithms

EGO Efficient Global Optimization

GPEME Gaussian process surrogate model assisted evolutionary algorithm for medium-scale computationally expensive optimization problems

GPM Gaussian process model

IC integrated circuit

ISC infill sampling criterion

LER line edge roughness

LVR latent variable regression

NMOS n-channel metal-oxide semiconductor

PCO poly crystal orientation

PMOS p-channel metal-oxide semiconductor

PVT process, voltage and temperature

QCQP quadratically constrained quadratic programming

RDF random dopant fluctuation

RPV Rapid PVT Verification

RSM response surface model

SAEA surrogate model assisted evolutionary algorithm

SDP semidefinite programming

VCO voltage-controlled oscillator

VLSI very large scale integration

Chapter 1

Introduction

How are integrated circuit (IC) designers able to create microprocessors with billions of reliable transistors in the nanometer regime, where minimum feature sizes are currently near 9 nm? This is not a straightforward task for designers and requires new challenges to be overcome with every new technology generation. To follow the road map set out by Moore's Law, classical transistor scaling techniques came from Dennard scaling, where the oxide thickness (T_{ox}), transistor length (L_g) and transistor width (W) were all scaled by a constant factor ($1/k$). However, classic Dennard scaling techniques ended with at the 130-nm technology node [1]. In the subsequent technologies simply scaling the transistor feature sizes introduced performance degrading issues such as off-state current leakage, gate leakage current, and increased channel doping due to impurity scattering [1, 2, 6].

For designers to push past this challenge, new enhancements were added to transistor design. Mechanical strain was introduced for the 90-nm and 65-nm nodes. For n-channel metal-oxide semiconductor (NMOS), strain was introduced by adding a high-stress Si_3N_4 cap layer over the transistor. On the other hand, p-channel metal-oxide semiconductor (PMOS) strain was achieved by replacing the traditional source/drain region with embedded $SiGe$ [7]. The addition of strain increased the mobility of the minority charge carriers in the channel region and thus increased the output drive current, thus improving the performance of both the NMOS and PMOS transistors [7]. High-k gate dielectric and metal gate (HiK-MG) technology was introduced to further enhance the strain effectiveness in the 45-nm and 32-nm nodes [6]. HiK-MG improved drive current performance and significantly reduced the gate leakage current [6]. These new enhancers allowed for the challenges of degraded performance to be overcome and technology scaling to continue.

Another serious challenge for very large scale integration (VLSI) designers is to get millions or even billions of transistors to reliably operate in all specified operating conditions. Over the past several decades, technology scaling has improved the performance and density of complementary metal-oxide semiconductor (CMOS) ICs [2]. Figure 1.1 shows that at the recent 32-nm technology node there are on the order of billions of transistors per central processing unit (CPU). Designing such a large system of transistors while ensuring

that a high yield of devices will operate correctly over a wide range of possible operating conditions is not an easy task. The precise values of those process properties cannot be exactly controlled given the unavoidable statistical variability of nanoscale semiconductor device fabrication [8]. Those operating conditions could include operating temperature range, power supply voltage range, and the ranges of possible key transistor properties such as threshold voltage and transconductance. For convenience transistor properties are often described with only one effective speed property, where "fast" transistors have a lower threshold voltage and "slow" transistors have a higher threshold voltage.

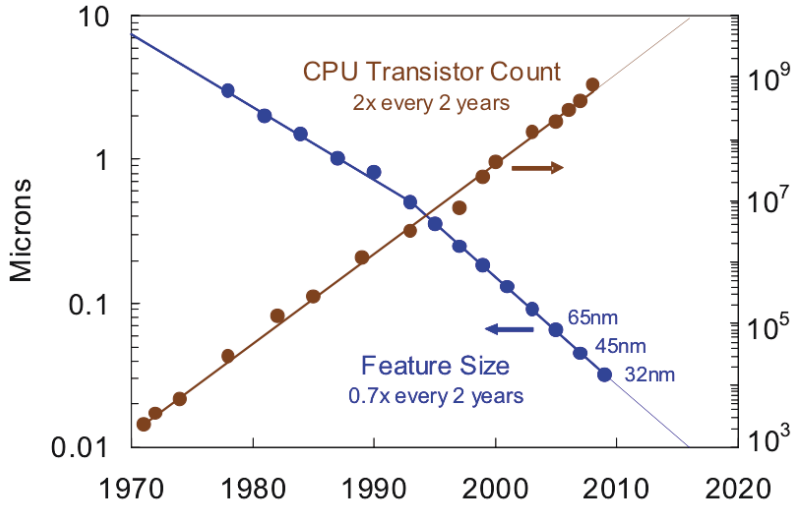


Figure 1.1: Moore's Law: The transistor count of CPU integrated circuits has increased by $2X$ and the minimum transistor feature size has decreased by $0.7X$ every two years [1].

Ensuring correct circuit operation in the presence of inevitable process variations has always been a critical aspect of CMOS design [2]. Process variations can adversely affect performance and power consumption, which can ultimately reduce the chip yield (that is, the fraction of manufactured chips that operate within specification). A source of variation that was historically of minor impact, but has emerged as a serious challenge because of the nanoscale dimensions of present transistors, is random dopant fluctuation (RDF) [2]. RDF alters the transistor's characteristics, notably the important threshold voltage. RDF has a larger impact on advanced technologies because the total number of dopant atoms in the channel region is far fewer than before. Figure 1.2 shows that minimum-sized transistors in the present technology nodes have fewer than 100 dopant atoms in the channel region. The precise number of dopant atoms that get placed into each channel cannot be controlled exactly because of the statistical nature of processing steps, such as dopant atom implantation. Still the VLSI designers must create ICs with billions of transistors that all operate reliably in the presence of such process variations.

The process, voltage and temperature (PVT) variations determine the conditions that affect the internal behaviour and output characteristics of an IC. IC designers commonly

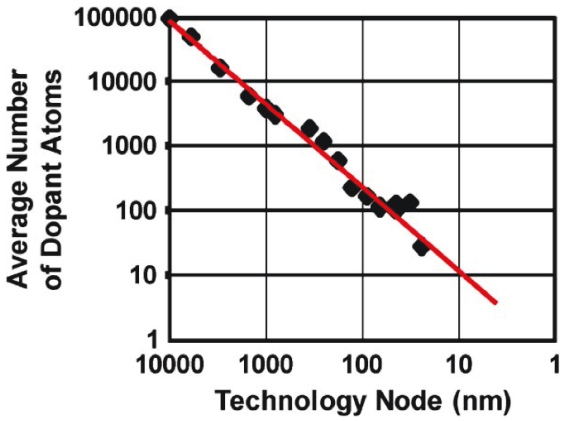


Figure 1.2: Reduction in the number of dopant atoms in the transistor channel [2].

model PVT variations as PVT corners. Each PVT corner is a combination of discretized values that represents one possibility among the full range of expected operating conditions. A common PVT corner parameter is the typical or model set value. The model set values we will use follow the standard two-letter naming convention. The first letter refers to the characteristics of the NMOS transistors and the second letter refers to the characteristics of the PMOS transistors. In this convention there are three characteristic values; typical, fast and slow. These characteristics refer to the relative mobilitities of the minority charge carriers in the transistor's channel region. Note that fast indicates higher carrier mobility than normal, slow indicates lower carrier mobility than normal, and typical is considered normal carrier mobility. This results in five possible model set combinations: typical-typical (TT), fast-fast (FF), slow-slow (SS), fast-slow (FS), and slow-fast (SF). Other PVT corner parameters can account for variations in one (or more) supply voltages and variations in the average die operating temperature.

Traditionally, only a few PVT corners were considered when verifying the behaviour of a chip design: model set {FF, SS} * operating voltage {min V, max V} * operating temperature {min T, max T} = 8 [9]. However, modern processes have tighter voltage and timing margins and they require more intermediate values as well as an increased number of PVT variables. A recent example of this is a voltage-controlled oscillator (VCO) circuit at the 28-nm node that has 15 model set values, 3 values for temperature, and 5 values for each of its three voltage variables, for a total of 3375 PVT corners [9]. To verify that this circuit will operate within specifications for all of these PVT corner conditions, up to 3375 accurate computer simulations of the circuit designs must be run. For this VCO circuit, a single PVT corner would take the standard HSPICE circuit simulator 70 seconds to simulate and therefore 66 hours would be required to simulate all 3375 corners [9].

An important goal of PVT verification is to find the worst-case performance/output value over all possible PVT corners. IC designers tackle this problem with different ap-

proaches. The most straightforward approach is a full factorial experiment [3]. The full factorial experiment simply simulates all possible PVT corners. This approach is guaranteed to find the worst-case PVT corner(s); however, it can be very time consuming and consequently in large VLSI designs the simulation time can be too large [3]. Frequently, designers desire to expedite the verification process and seek less costly alternatives to the full factorial approach.

The designer "best guess approach" consists of simulating only a fraction of all possible PVT corners. The PVT corners that are simulated are selected using the designer's best guess at worst-case PVT corners based on prior experience and expertise [3]. This could lead to an inadequate selection of PVT corners and overly optimistic belief in circuit behaviour. This could mean that under certain conditions the circuit might not fall within the required specifications. If not discovered before chip production, the problem could greatly reduce the yield or ultimately cause dangerous failures in the field. Although the "best guess approach" has the advantage of reduced simulation time, it sacrifices confidence in the robustness of the design.

Another interesting approach is sensitivity analysis via linear modeling. In this type of approach each PVT variable is adjusted one-at-a-time over a suitable range of values while all other variables are held constant and the circuit is simulated for each variation [3]. A linear response surface model (RSM) is constructed from the observed behaviour of the circuit's output. The RSM is used to predict worst-case PVT corners for simulation. This approach requires relatively few simulations, which makes it fast, but it may have poor reliability. Reliability may be poor due to falsely assuming a linear response to the PVT variables and by assuming there are no interactions between variables [3].

The goal of PVT verification can alternatively be viewed as a mathematical optimization problem. From this viewpoint the input domain would consist of PVT corners, $\vec{x}_i \in X$, where $\vec{x}_i = [x_{i1}, x_{i2}, \ldots, x_{in}]$ and $n \geq 1$ is the number of PVT variables. Therefore the problem of finding the worst-case PVT corner could be viewed as finding the global maximum of the simulated output value, $f(\vec{x}_i) \in Y$, over the input domain X. The new aim of the verification problem is to find the global optimum and not get stuck on local optima. Figure 1.3 illustrates PVT verification cast as an optimization problem over a domain, in this case, the very simple one-dimensional domain of a temperature parameter x. The x-axis shows all possible values of the PVT variable while the y-axis shows the output values (dots) of simulated PVT corners and the predicted output values (curve) of unsimulated PVT corners.

In many engineering optimization problems, the number of computationally expensive function evaluations is severely limited by time or cost [10]. This has become an expanding field of interest called simulation-based optimization [4]. Simulation-based optimization addresses functions that are not expressed with closed-form analytical equations, but that can only be evaluated with so-called "black-box" computer simulations. Finding the worst-

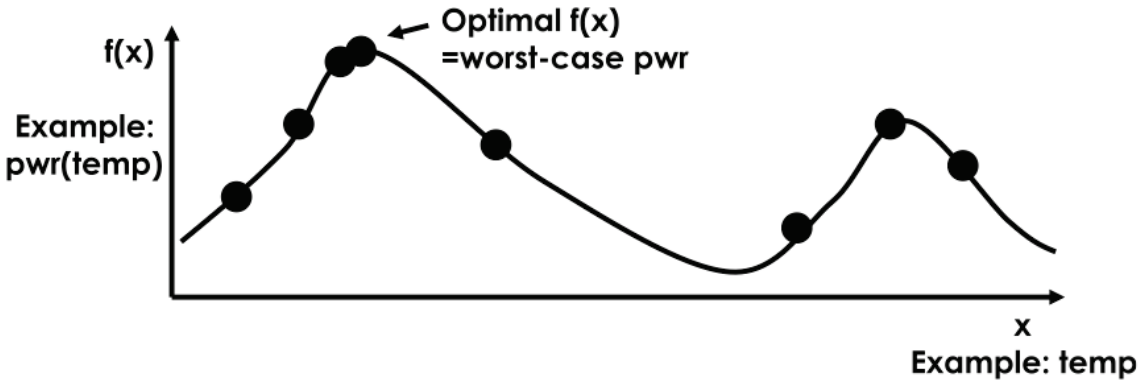


Figure 1.3: PVT verification cast as a global optimization problem [3]

case PVT corner of a given circuit's output is a black-box optimization problem. Black-box functions can be noisy or discontinuous (i.e., rough or non-smooth) and/or require a long time to compute [4].

Figure 1.4 shows an example black-box function: the fuel economy of an automobile as a function of the vehicle's final drive ratio. Finding the optimal final drive ratio that yields the highest fuel economy poses several challenges due to the noisy and discontinuous behaviour of the function. Traditional gradient-based optimization techniques may inaccurately estimate the gradient due to the non-smooth nature of the function [10]. This can lead gradient-based algorithms to potentially miss the maximum. Discontinuities may also cause algorithms (i.e., hill climbing) to find local maxima, but prevent them from finding the global maxima.

A method is proposed in [10] to address these problems through the use of approximations and machine learning techniques. To address these problems, [10] fits a response surface to data collected by evaluating the black-box function at a few points. These surfaces (function approximations) are then used for function optimization. The particular optimization algorithm that we investigated is known as Efficient Global Optimization (EGO) [10]. In Chapter 2 the EGO algorithm will be reviewed in detail, including the use of kriging models as an approximation method that is able to provide estimates of local uncertainty.

The objective of machine learning techniques is to extract generalized knowledge directly from the data. Machine learning is concerned typically with programs or algorithms that optimize a performance criterion using known/sampled data. When we have a model of a certain problem, that includes adjustable parameters, learning is described as optimizing the values of those parameters with the use of training data [11]. Training data is a collection of data points that are known (i.e., the output for each input combination in the collection is known). Errors or uncertainties might also be associated with the training data points.

There are two main types of machine learning: unsupervised learning, which is commonly used for clustering, and supervised learning, which is commonly used for classification or regression. Both regression and classification are supervised learning problems, where an

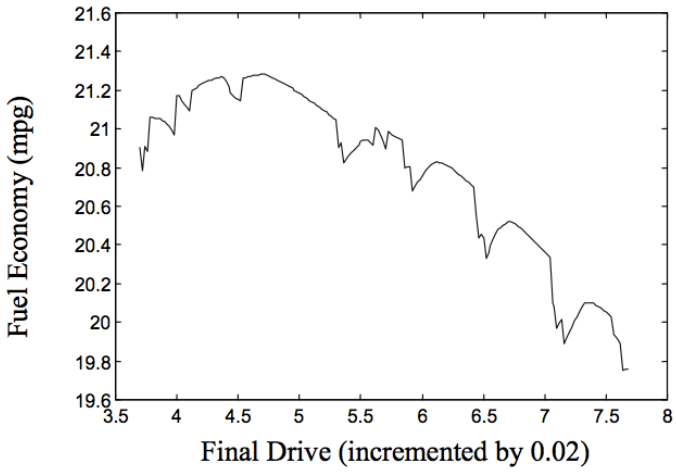


Figure 1.4: Example of a simulation-based design problem [4].

algorithm is tasked with mapping an input, from a domain X, to an output, in a domain Y. Classification refers to the problem of mapping an input to an output domain that consists of a finite set of discrete values (e.g., [1,2,3]). Regression refers to the mapping of an input to a continuous output domain (e.g., $Y \in \Re$).

Global optimization is an active research area and there are many different available optimization techniques [4]. This thesis shall explore and experiment with a novel Rapid PVT verification optimization method. This method will employ a machine-learning model of the circuit's output and use different heuristics to exploit the information provided from the model. A machine-learning regression model can use information from simulated PVT corners to make predictions about unsimulated PVT corners. Information from the model can also be used to determine if a global optimum has been found with a certain degree of confidence. The goal of this proposed method is threefold: (i) reliably locate the global optimum, (ii) simulate the fewest number of PVT corners, and (iii) compute quickly (i.e., minimize the computational expense).

This thesis is composed of six chapters. Chapter 2 provides background on global optimization problems and describes current algorithms and frameworks. The machine learning model to be used will be explained and justified as an appropriate choice for the PVT verification problem. Chapter 3 gives an overview of the initial Rapid PVT Verification (RPV) algorithm that focuses on a single output and reports initial reliability measurements. In Chapter 4 many alternative heuristics are evaluated experimentally in simulation to see if improvements over the initial algorithm can be made. Chapter 5 extends the RPV algorithm to evaluate multiple outputs simultaneously and experimentally evaluates the benefits of concurrent vs. sequential searching methodologies. Finally, Chapter 6 concludes the thesis with a summary of results and provides suggestions for future work in this area.

Chapter 2

Background

This chapter is organized into three sections. In Section 2.1 the Efficient Global Optimization (EGO) algorithm is reviewed in detail. Section 2.2 introduces Gaussian process models (GPMs) and their application to machine learning. Section 2.3 will review related work.

2.1 The Efficient Global Optimization Algorithm

The optimization algorithm known as EGO was introduced by Schonlau, Welch and Jones [10]. It was developed as an efficient iterative optimization strategy that constructed and updated models of expensive-to-evaluate functions. Since function evaluations are assumed to be costly in terms of time and/or other costs, the algorithm fits a response surface model of the function to data collected by evaluating the function at the smallest possible number of selected points. The response surface model is then used for optimization. The response surface model used has been reported to be especially good at modeling nonlinear multimodal functions, the types of functions commonly found in many engineering problems [10]. This approach has proven to be efficient and effective for many optimization problems [4]. The basic outline of EGO can be summarized in the following five steps.

1. Obtain an initial sample of the function using a space-filling design of experiments method.
2. Fit a kriging model to the function.
3. Select a sampling criterion that maximizes an expected improvement function to determine where to sample next.
4. Sample the point(s) of interest.
5. If the expected improvement function has become sufficiently small, then stop; otherwise return to step 2.

There are a wide variety of optimization algorithms that use global search methods based on statistical models. These types of optimization algorithms are classified as Bayesian analysis algorithms [4]. Bayes' theorem, from probability theory, set the foundation for Bayesian statistical inference [12]. Statistical inference can be used to draw conclusions from known sampled data to unsampled data. Bayesian analysis is a statistical method which attempts to estimate parameters of an unknown function or distribution based on sampled observations. This type of inference uses interpolation from known samples to make approximations. Since these approximations are not certain, there is an associated uncertainty for each predicted unsampled data point. For a more comprehensive review of Bayesian methods the reader is referred to Iversen [12]. The EGO algorithm uses a Bayesian method called a Kriging model. Kriging models, which are also known as GPMs in the machine learing field, will be covered in detail in the next section [4].

To illustrate the search strategy of EGO, a one-dimensional example is shown in 2.1. The sinusoidal dashed line represents the unknown objective function that is to be minimized. The kriging approximation of the function is shown by a solid line and is generated based on the sampled points shown as circles. The curve at the bottom of the plot represents the infill sampling criterion (i.e., the next point selection rule) that has been superimposed and normalized to allow comparisons between iterations. The maximum of the infill sampling criterion function is the point that is to be sampled next.

Subsequent function evaluations are chosen based on the infill sampling criterion (ISC). EGO chooses to simulate the point where the ISC is maximized, meaning that this point is most likely to improve the accuracy of the model and/or discover a better function value than previously found. In this example, EGO creates a kriging model of the function after the initial sample of four points is evaluated Figure 2.1 (a). The resulting approximation is a poor fit compared to the true function, but the ISC leads the algorithm to sample regions of high uncertainty. After the evaluation of two more points the approximation closely resembles the true function for the right side Figure 2.1 (b). After another two function evaluations, Figure 2.1 (c), the ISC has determined another region of high uncertainty and places greater emphasis on it rather than on an area of local optimization (where $7.5 < x > 8$) of the function. By the sixth iteration, Figure 2.1 (d), EGO has explored both local optima and has found the global optimization accurately.

From this quick example it is clear that EGO does not follow a traditional gradient-based search path that could become stuck at local optima [4]. Points of interest may be found anywhere in the design space depending on where the ISC is the highest. This shows how EGO uses a global search strategy that is more robust than simple local search methods, such as gradient-based searches that can get stuck in local optima. However this robustness does come at a cost. The process of generating kriging models and locating the maximum of the ISC does have significant overhead compared to gradient-based methods [4]. However, gradient-based methods usually require a relatively large number of function

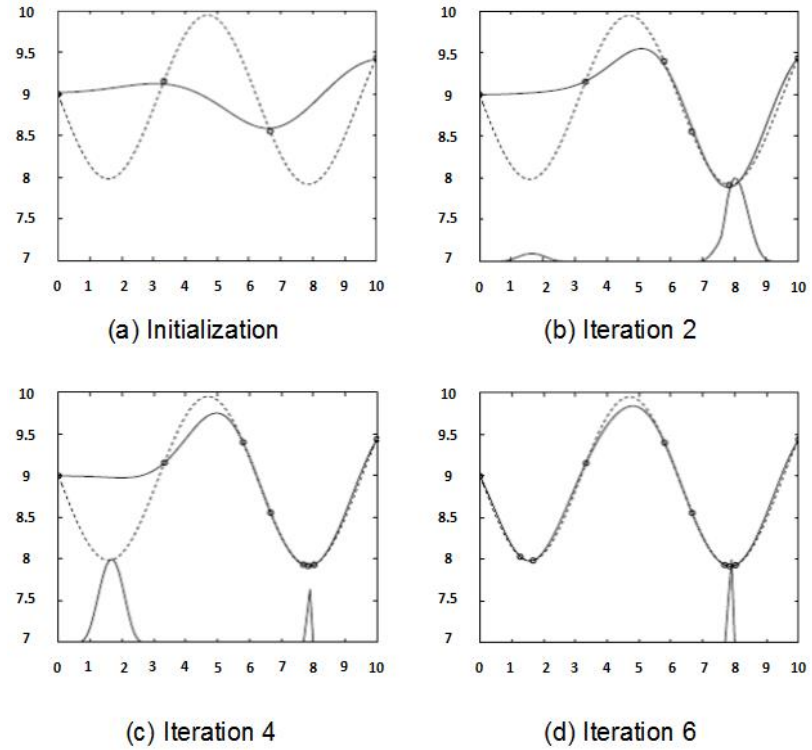


Figure 2.1: Demonstration of the EGO algorithm. The dashed curve is the unknown function, solid curve is the approximation, and the circles are the sampled points. The curve at the bottom represents the infill sampling criterion [4]

.

evaluations before converging on a solution [4]. The trade-off EGO makes does have a larger overhead, but during each iteration as much information is used as possible to identify where to evaluate the function next. This allows the algorithm to find good solutions in fewer iterations. This makes the EGO algorithm well suited for problems that have expensive-to-evaluate functions.

2.2 Gaussian Process Model

In Bayesian analysis the standard linear regression model takes the form of

$$y(x) \;=\; \sum_{i=1}^{N_B} w_i f_i(x) \;+\; \epsilon.$$

In this model, every $f_i(x)$ is a linear or nonlinear function of x and is referred to as a basis function. Each of the N_B basis functions has an unknown coefficient, w_i. During model fitting (i.e., training) the w_i and $f_i(x)$ terms are determined by optimizing criteria such as minimizing mean-squared error on the training data [13]. The residual ϵ is assumed to be independent and identically normally distributed with mean zero and variance σ_n^2

$$\epsilon \;\sim N(0, \sigma_n^2).$$

In this standard linear regression model the error distributions remain constant throughout the entire input domain (i.e., $\epsilon \;\sim N(0, \sigma_n^2)\ \forall x$), even for data points that have already been sampled. Conceptually this does not make sense for a computer-calculated deterministic function because any lack of fit of the model will be entirely modeling error (incomplete regression terms) and not measurement error or noise [10]. Since these error terms are only a collection of left out w_i and $f_i(x)$ terms, we can view the error term as a function of x, $\epsilon(x)$ [13]. If x_i and x_j are two points that are close together, then the errors $\epsilon(x_i)$ and $\epsilon(x_j)$ should also be close. It no longer seems reasonable to assume that $\epsilon(x_i)$ and $\epsilon(x_j)$ are independent. Instead, it would be reasonable to assume that these error terms are correlated, and that the correlation is higher when x_i and x_j are closer and lower when the two points are further apart.

The main difference between the standard linear regression model and a Gaussian Process model is that the GPM assumes that if $y(x)$ is continuous then $\epsilon(x)$ is also continuous since it is the difference between $y(x)$ and the regression terms [10]. The GPM does not assume that errors are independent, but rather that the correlation between errors are related to the distance between corresponding points. The GPM does not use Euclidean distance, since this distance measurement weights all variables equally. A special weighted distance formula is used instead, as shown below

$$d(x_i, x_j) \;=\; \sum_{h=1}^{n} \theta_h |x_{i,h} - x_{j,h}|^{p_h} \quad (\theta_h \geq 0, p_h \in [1, 2]). \tag{2.1}$$

With this distance function, the correlation between $\epsilon(x_i)$ and $\epsilon(x_j)$ is

$$Corr[\epsilon(x_i), \epsilon(x_j)] \;=\; exp[-d(x_i, x_j)]. \tag{2.2}$$

The correlation function, as defined in (2.1) and (2.2), proves to have intuitive properties. As two points x_i and x_j get closer, their distance approaches zero, and therefore their correlation goes to one. Similarly, when two points move far apart their correlation will approach zero. The distance function also has parameters, n dimensional vectors θ and p, that characterize the activity and smoothness of input variables. The θ_h parameter in the distance function (2.1) can be interpreted as an activity measurement of the variable x_h. This means that when variable x_h is active, small values of $|x_{i,h} - x_{j,h}|$ may lead to large differences in the function values at $y(x_i)$ and $y(x_j)$. In cases such as this, a large value of θ_h can be equivalent to a large distance and hence low correlation. The p_h parameter relates to the smoothness of the function in the coordinate direction h. For example $p_h \;=\; 2$, corresponds to relatively smooth functions and values of 1 correspond to less smooth functions [10]. These parameters are determined using maximum-likelihood estimation on the training data [3].

Modeling the correlation in this way is so powerful that the GPM can replace regression terms with a constant value [10]. This yields the GPM in the form of

$$y(x) \;=\; \mu + \epsilon(x),$$

where μ is the mean of a stochastic process (i.e., mean of the regression terms), ϵ is $Gaussian(0, \sigma^2)$, and that correlation is defined by Equations (2.1) and (2.2). The variance, σ^2, is represented by a correlation matrix $\mathbf{R}$. Every entry in $\mathbf{R}$ is defined as $\mathbf{R}_{ij} \;=\; Corr[\epsilon(x_i), \epsilon(x_j)]$.

Let us see how the GPM can be used as a predictor for unknown points. Let $\mathbf{r}$ denote the k-vector of correlations between the error term at an unknown point, x^*, and the k error terms at previously sampled points. Using (2.1) and (2.2), we find that element i of $\mathbf{r}$ is $\mathbf{r}_i(x^*) \;\equiv\; Corr[\epsilon(x^*), \epsilon(x_i)]$. Also, let $\mathbf{1}$ denote a k-vector of ones. This allows the GPM to predict $y(x^*)$ as

$$\hat{y}(x^*) \;=\; \mu + \mathbf{r}^T\mathbf{R}^{-1}(y - \mathbf{1}\mu).$$

The GPM can also be used to report the associated uncertainty of a predicted output value. The GPM is able to report an estimate of variance as given by

$$s^2(x) \;=\; \sigma^2[1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r} + (1 - \mathbf{1}^T\mathbf{R}^{-1}\mathbf{r})^2/(\mathbf{1}^T\mathbf{R}^{-1}\mathbf{1})].$$

For any further details we refer the reader to [10] and [13].

2.3 Related Work

2.3.1 Response Surface Models of Process Variations

Sengupta et al. proposed a statistical methodology of monitoring process variations during IC verification in [14]. Nardi et al. investigated process variations and found that as process dimensions scaled down, process variability tends to increase [15]. Increased process variability can have a non-negligible affect on a circuit's performance margins [15]. The observed random process variations reflected a stochastic spread of a given circuit's performance measurements [15, 16]. Sengupta et al. introduce a statistical methodology to determine the worst-case combination of process parameters (parameters such as oxide layer thickness, NMOS and PMOS threshold voltages, polysilicon thickness, etc.), referred to as process corners, for a set of circuit performance measurements [14]. The proposed methodology uses quadratic functions of the process parameters to estimate RSMs over the process parameter space. The RSMs are used to identify process corners that yield worst-case circuit performances as indicated by maximum/minimum values of the RSMs.

The RSM takes the form

$$y_k = a_k + b_k^T \mathbf{x} + \mathbf{x}^T B_k \mathbf{x}, \qquad k = 1, 2, ...m$$

for the k^{th} output of the possible m outputs. In this model a_k is a constant term, b_k and B_k are matrices, and $\mathbf{x}$ is a vector of n process parameters.

The proposed methodology was evaluated using digital standard cells and typical communication analog/radio frequency circuits in 130 nm technology. The methodology was able to reduce verification time of circuit designs by orders of magnitude [14]. As circuit complexity increases, statistical simulations for larger circuits can sometimes become prohibitive [14]. The proposed methodology uses statistical device models efficiently in order to determine worst-case process corners [14].

2.3.2 Beyond Low-Order Statistical Response Surfaces

This subsection reviews the work presented in [17]. Singhee et al. developed the SiLVR algorithm to address increasing dimensionality, large variations, and nonlinearity found in generating response surface models of circuit behaviour in the presence of process variations in the 90 to 45 nm technology nodes. The number of sources of process variations, e.g. random dopant fluctuation (RDF), line edge roughness (LER), poly crystal orientation (PCO), etc., for even a simple digital circuit, such as a flip-flop, can be over fifty [18]. Larger analog cells can easily have the number of sources of variations in the hundreds and therefore require response surface models to have large dimensionality [17]. Some variation sources can have a relatively large effect [17]. For example, the variation source RDF, at the 70nm node, was investigated to have a standard deviation of the threshold voltage (V_t) that can be 10% of the nominal V_t [19]. At the 25 nm node, the standard deviation of the threshold voltage can be up to 21% of the nominal V_t [20].

Not all variation scenarios are well modeled by low-order (i.e., linear or quadratic) response surfaces [17]. Linear RSMs, as described in [21,22], are effective only when variations are small enough to allow for a linear approximation [17]. Initial quadratic RSMs proposed in [23], as well as improved quadratic RSMs proposed in [24, 25] cannot always capture the nonlinearity seen in the presence of large process variations [17].

The SiLVR algorithm uses latent variable regression (LVR) techniques as described in [26] to reduce dimensionality of the variation sources. The LVR techniques iteratively extract the next most important statistical variable (i.e., latent variable) until the error from the response surfaces generated with only extracted variables falls within a reasonable level [17]. SiLVR employs a feedforward neural network to generate arbitrarily nonlinear statistical response surfaces. SiLVR is an efficient statistical response surface modeling method that can handle large and nonlinear effects from process variations [17]. SiLVR provides a basis for future work on nonlinear yield optimization strategies [17].

2.3.3 Efficient Design-Specific Worst-Case Corner Extraction

This subsection reviews the work developed in [27]. Statistical analysis is an integral tool for nanoscale integrated circuit (IC) design [27]. RSMs can be used efficiently to predict performance distribution and/or parametric yield in the presence of both inter-die and intra-die variations for ICs [28, 29]. To capture the large-scale variations that are observed in IC technologies 45 nm and below, quadratic or even strongly nonlinear RSMs are required to improve modeling accuracy [17, 24, 25, 30, 31]. It is important for CAD tools to predict yield values, but also to provide additional information that helps the circuit designers to improve the IC design [27]. Worst-case corner extraction aims to identify the unique process conditions that cause a given circuit to operate outside of specifications [14, 23]. By determining the worst-case corners, the designers can run simulations on these corners to determine the reason for the circuit's performance failure and develop an appropriate solution to improve robustness [27].

Extracting the worst-case corners using RSMs is not a simple task. Using a quadratic RSM results in a non-convex quadratically constrained quadratic programming (QCQP) problem for extracting the worst-case corners [14, 23]. Zhang et al. propose a method that converts the non-convex QCQP problem to a convex semidefinite programming (SDP) problem that can be solved more easily. The approach is derived from the Lagrange duality theory of nonlinear optimization in [32]. Zhang et al. exploit the unique case that the QCQP formulated during worst-case corner extraction only contains a single quadratic constraint [27]. In this special case, the dual form of the QCQP is a convex SDP [27]. The proposed method can efficiently and robustly find the worst-case corners with global convergence [27].

2.3.4 Gaussian Process Surrogate Model Assisted Evolutionary Algorithm for Medium Scale Expensive Optimization Problems

This subsection reviews the work presented in [33]. There are many real-world optimization problems that require expensive-to-compute simulations for unknown function evaluations [10,34,35]. Traditional gradient-based mathematical optimization methods cannot be applied directly to these problems since analytic formulations are unknown [33]. Similarly, evolutionary algorithmss (EAs) cannot directly solve these types of problems either since EAs require a large number of function evaluations and therefore unrealistic to perform [33]. Using a surrogate model assisted evolutionary algorithm (SAEA) is one approach for dealing with expensive-to-compute optimization problems. The SAEA uses a surrogate model to replace the expensive-to-compute function evaluations. The computational cost can be reduced significantly by using a surrogate model to approximate an unknown function rather than performing expensive function evaluations [33].

Many current SAEAs focus on small-scale expensive optimization problems (problems with less than 20 variables), such as the ones proposed in [10,36,37]. Real-world applications such as integrated circuit design problems can have around 20 to 50 design variables [38,39]. The method proposed by Liu et al. focuses on expensive optimization problems of 2050 variables (medium-scale problems) [33]. Other medium-scale SAEA methods were proposed by [35,40,41]. In [41], Zhou et al. uses a Gaussian process model (GPM) with probability of improvement prescreening [42] as a global surrogate model and a Lamarckian learning process as a local surrogate model to accelerate an EA. Liu et al. propose a new SAEA method called Gaussian process surrogate model assisted evolutionary algorithm for medium-scale computationally expensive optimization problems (GPEME). The GPEME uses Sammon mapping [43], a dimension reduction technique, to lower the dimensionality of the GPM. The GPEME iteratively searches for promising candidate points using a differential evolution (DE) algorithm as described in [44]. Experimental results on benchmark problems show that the GPEME performs comparably to other state-of-the-art SAEAs, namely [35,40,41], while requiring 12% to 50% fewer exact function evaluations [33].

Chapter 3

Initial Algorithm Methodology

3.1 Examine Initial Circuit Dataset

The Rapid PVT Verification (RPV) algorithm was developed and evaluated using pre-computed simulation data results (preferably full factorial). A full factorial search requires circuit responses to be simulated for all PVT corners of interest and stored in a data set. When used in its intended scenario, RPV would be used to select which simulation(s) would be necessary to perform on the basis of past simulation results. The data sets we used were prepared using Cadence/Spectre simulation tools and provided to us by Solido Design Automation. These data sets represent nine typical circuits and they have different input and output characteristics. Some of these data sets contain all full-factorial PVT combinations while the other data sets contain only a subset of all PVT combinations (i.e., they are partial factorial). A list of the circuit data sets is shown in Table 3.1. In these nine different circuits the number of PVT corners ranges from 120 to 1800 and they involve up to 10 PVT parameters. Each parameter spans a small finite domain of possible values.

To illustrate the input and output domains, we will first consider the *shift register* circuit. This circuit has seven PVT parameters (NMOS model set, PMOS model set,

Table 3.1: Initial Circuit Dataset

Circuit	Data Set Type	Total Corners	PVT Parameters	Outputs
shift_reg	Full-Factorial	1080	7	3
buffer_chain	Full-Factorial	1800	10	6
bitcell	Full-Factorial	120	5	2
mux	Partial-Factorial	120	8	7
charge_pump1	Partial-Factorial	216	8	5
charge_pump2	Partial-Factorial	324	8	5
sense_amp1	Partial-Factorial	120	10	7
bias_gen	Partial-Factorial	120	3	10
opamp1	Partial-Factorial	120	6	1

Temperature, V_{in_ac}, V_{VCC}, V_{VDD}, and V_{REF}) and three different output functions (*delay*, *fall time*, and *rise time* of one circuit output signal). Figure 3.1 shows a scatter plot of the *delay* output values for each individual PVT parameter. The x-axis of each subplot indicates a domain of possible PVT parameter values and the y-axis shows the possible range of values for the *delay* output at each PVT parameter.

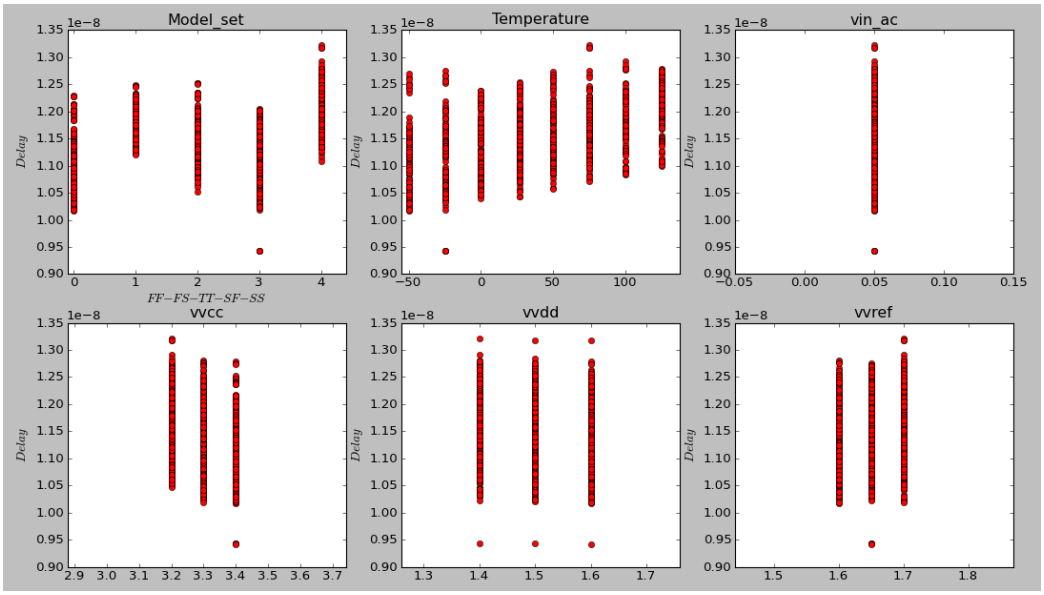


Figure 3.1: The *delay* output range of the *shift register* circuit for each PVT parameter.

Figure 3.1 shows the multidimensional nature of our optimization problem. There are no obvious trends for individual PVT parameter values. The delay output can have a large range of values for each PVT parameter. The search for minima or maxima in the output functions must consider the input parameter values in combination.

To get a better visualization of the *delay* output behaviour we observe a 3D plot in Figure 3.2. In this plot we vary two PVT parameters, *temperature* and $Vvdd$, while holding all of the other parameters constant. The blue dots correspond to the *delay* output being measured at a given PVT corner. The blue curves are artificially added in to represent a possible response surface curvature.

3.2 Initial Algorithm

This proposed methodology will utilize the Efficient Global Optimization (EGO) framework to develop a search algorithm to solve the PVT optimization problem efficiently and reliably. An efficient solution is defined as an algorithm that on average requires far fewer circuit simulations than a full factorial search. This so-called Rapid PVT Verification (RPV) algorithm will iteratively learn Gaussian Process models of the initially unknown circuit output functions. A Gaussian Process regression model is constructed using a standard algorithm to exploit the increasing amount of information from all previous rounds of PVT

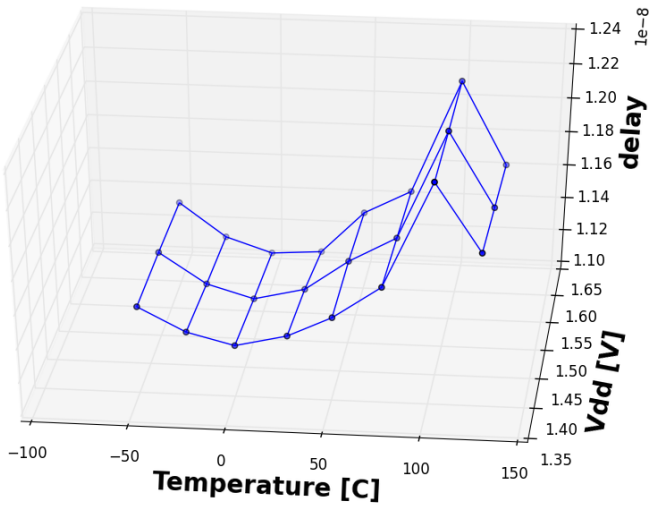


Figure 3.2: The *delay* output of the *shift register* circuit is measured over a range of temperatures and supply voltages.

simulations. The model can be used to predict output values of corners not yet simulated. A major contribution of our work is to propose and then refine heuristics that refer to the current GPMs and decide which unsimulated PVT corners should be selected next to simulate. The three main steps of the EGO approach include: (1) choosing the initial training set (the set of PVT corners that are simulated at the start to construct the first GPM), (2) choosing the next one or more training corners to simulate next, and (3) deciding when to stop the search for the function maximum. The method follows a looping pattern as illustrated by Figure 3.3. The cycle begins by updating the GPM(s) and then selecting the next PVT corners to simulate. This cycle is repeated until the stopping conditions have been reached.

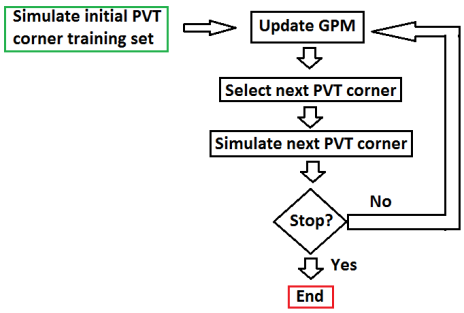


Figure 3.3: Rapid PVT Verification (RPV) flowchart.

Once the GPM has been updated (i.e., fit or trained) to the simulated PVT corners,

the GPM can be evaluated at the unknown PVT corners to generate predictions based on interpolations. By supplying a set of initial PVT corners along with their simulated output values (i.e., the initial training set), the GPM fits a response surface to the simulated PVT corners and can generate predictions for the unsimulated PVT corners. Figure 3.4 shows an example of a GPM for a simple function ($f(x) = x\ sin(x)$). In the figure five simulated and hence "error-free" points are indicated by red dots. The GPM uses these simulated points to generate a model or approximation of the function, shown as a solid blue line. The shaded blue region shows the expected range of actual function values between the observed points.

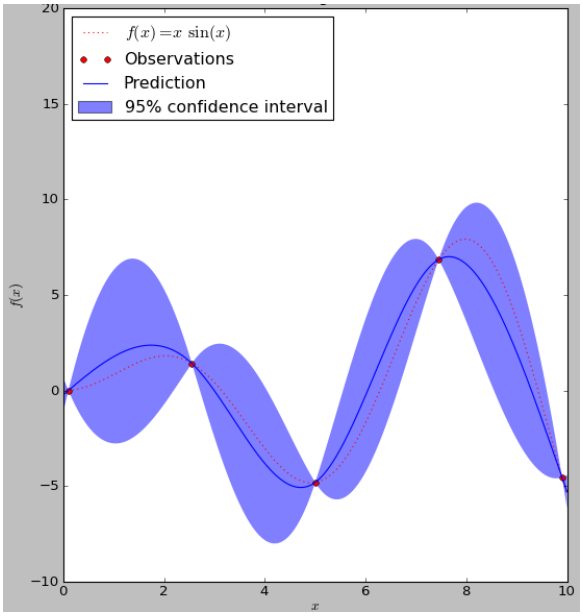


Figure 3.4: Example of a Gaussian Process Model for $f(x) = x \times sin(x)$ constructed over the domain $0 \leq x \leq 10$ using five error-free observations.

A great benefit of using a GPM to model output functions in our optimization problem is that the model not only generates a prediction ($\hat{Y}$), it also generates an associated estimated variance (σ^2) for each $\hat{Y}$. The GPM exploits the relative distances in the domain between the training and predicted corners to report an estimated uncertainty or error (as discussed in Chapter 2). By taking the square root of the σ^2 we are able to estimate the standard deviation ($Stddev$) of each $\hat{Y}$. This in turn allows us to estimate $Normal$-distribution confidence intervals. Following the EGO algorithm, we can use the GPM to balance the priorities between sampling where estimated maxima occur ($max(\hat{Y})$) and where the estimated prediction error is high ($max(Stddev)$).

The framework of the new algorithm follows the EGO algorithm as shown in Figure 3.5. The algorithm starts by selecting samples of the input domain and calculating/simulating the function's output at those points to create an initial data sample (i.e., a training set). The initial data sample set is used to construct a GPM. The GPM allows for the ap-

proximation of unknown points of the output function. The next point selection rule is a heuristic that identifies the points that are most likely to benefit the model. These next points are then calculated/simulated and then added to the set of known sample points. If the stopping criterion is met the algorithm will end; otherwise it will enter a loop that will update the GPM and select new points until the stopping criterion is met. This framework has three main subproblems: initial training set, next corner(s) selection, and stopping criterion. The subproblems are solved using experimentally-tuned heuristics. The remainder of this section will explain these three topics in detail.

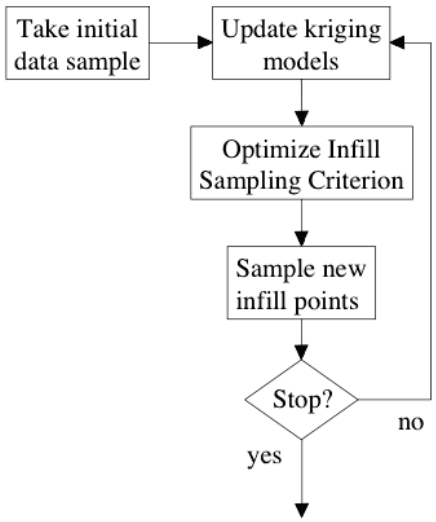


Figure 3.5: Flowchart of the EGO algorithm [4].

3.2.1 Selecting the Initial Training Set

The first step of this algorithm is to select an initial training set of corners to be simulated to train the first GPM. Selecting the initial set involves two issues: how many corners should be selected and what heuristic rule should be used to select those corners.

Deciding exactly how many PVT corners to simulate as the initial training set does not have a straightforward answer. We only want enough corners to provide a good basis for our initial GPM. Selecting too big an initial training set could be wasting time simulating corners that may not benefit the model or more rapidly advance the algorithm to the end goal of reliably finding the maxima.

Selecting an initial training set is the classic design of experiment problem [12]. There are many experimental designs ranging from classical experiment designs, such as central composite design, to space filling experimental designs like orthogonal Latin hypercube design. To select the initial training set we opt to use a modified central composite design (CCD). The CCD requires fewer than 3^n points to compute a 2^{nd} order response surface by using the center points along with axial points to estimate the curvature of the surface.

Suppose a circuit has n input PVT corner parameters, where each PVT parameter may have several possible values, for a total of M PVT combinations (i.e. M possible corners). A traditional CCD is a combination of 2^n factorial points (edge points), $2 \times n$ star (axial) points, and a center point, as shown in the three-dimensional example in Figure 3.6 [5]. The factorial points refer to the 2^n possible combinations of maximum and minimum PVT parameter values, which are also referred to a edge or extreme-case PVT corners. However, a $2^n + 2n + 1$ design produces a large training set for $n \geq 5$. As a starting point for the algorithm, we used a heuristic and selected a training set size of n^2 corners. This number of corners will be much less than the total number of possible corners, that is $n^2 << M$, but should provide enough information of a circuit's function behavior to have a good starting point for the algorithm.

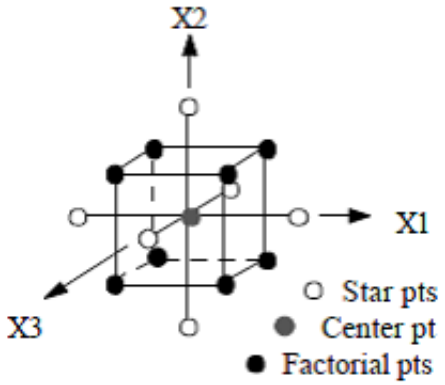


Figure 3.6: Central composite design, a classical experimental design [5]

The modified CCD that we developed selects one center point PVT corner (i.e. the PVT corner that is comprised of PVT parameter median values) and $n^2 - 1$ edge PVT corners (i.e., factorial points). Since there are 2^n possible edge PVT corners we initially select $n^2 - 1$ edge corners at random. The performance implications of choosing fewer than $n^2 - 1$ edge corners will be explored experimentally in subsequent sections.

For example the shift register circuit has this set of PVT parameters: nMOS modelset, pMOS modelset, Temp, Vin_ac, Vvcc, Vvdd, Vvref. The default method to select input vectors, a.k.a. corners, is to create two arrays; one for the minimum parameter values and one for the maximum parameter values. The shift register circuit has the following minimum values [0, 0, -50, 0.05, 3.2, 1.4, 1.6] and the following maximum values [2, 2, 125, 0.05, 3.4, 1.6, 1.7] for each of the PVT parameters, respectively.

These minimum and maximum values are referred to as extreme values. All possible combinations of these extreme values construct the set of edge-case corners or factorial points. Identifying edge-case corners allows us to describe the bounds of our input domain. In the case of the shift register circuit there are 64 edge-case corners.

3.2.2 Next Corner Selection

The next step of the algorithm is to select the infill sampling criterion (i.e., the next corner selection rule). The infill sampling criterion seeks to reduce the error of the current function model while searching the input domain for the most likely potential maxima corners to simulate next. The selection of the next PVT corner(s) to be simulated is based on the predictions of the GPM. Similar to the EGO algorithm, the choice of next sample is based on the model's estimated predictions ($\hat{Y}$) as well as the model's estimated error ($Stddev$) for those predictions.

Again we are faced with the decision of how many corners should be chosen by the infill criterion to simulate. Since we assume that training a GPM is computationally inexpensive compared to simulating a PVT corner, we believe it is reasonable to iteratively train GPMs and simulate corners one corner at a time. It makes intuitive sense to select only one corner per iteration so that we are able to exploit the newly available simulation information as soon as possible and therefore allow the selection decision to be made on the most up-to-date data. However, selecting multiple corners distributed about the input domain at each iteration may yield more information about the output function. Several different approaches of next corner selection will be explored in the next chapter.

We want our infill sampling criterion to maintain a balance between estimated predicted values and estimated error. Simply selecting a corner with the highest predicted value, $c_{next} = argmax(\hat{Y}(c))$, may leave us blind to regions of the predicted outputs that have a large estimated error. Following [45], we have found that the worst-case combinations of $\hat{Y}$ and $Stddev$ can be captured and understood using a convex hull plot.

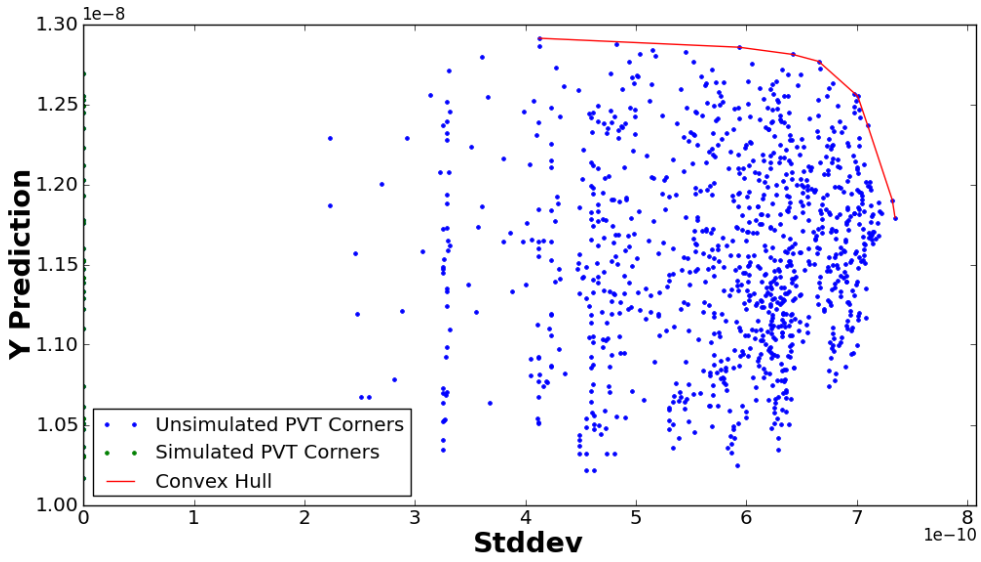


Figure 3.7: Convex hull of the output *delay* of the shift register circuit after 35 PVT corners have been simulated and used to construct a Gaussian Process model.

The GPM gives approximations for the output function for all unsimulated (unknown) corners, $\hat{Y}(c)$, and their associated estimated standard deviations, $Stddev(c)$. In Figure 3.7, these values along with the known corners are plotted to give an intuitive view of the GPM after 35 PVT corners have been simulated for the delay output of the shift register circuit. Please note the y-axis of Figure 3.7 is $\hat{Y}(c)$ and the x-axis is $Stddev(c)$. The points along the vertical y-axis (i.e., at $x = 0$) correspond to the 35 simulated corners, which have $Stddev = 0$ and were used to construct the GPM. A convex hull is shown as a red curve. Intuitively, the convex hull is the set of "worst-case" points that could potentially be maxima of Y [45]. More formally, the convex hull is the set of corners ($c_{covexhull}$) for all values of w such that $c_{convexhull} = argmax(\hat{Y}(c) + w \times Stddev(c)),\ w \in [0, \infty]$.

The PVT corners that are found on the convex hull are the PVT corners most likely to be a simulated maximum for all possible choices of the parameter w [45]. The value selected for w determines how much weight is placed on the estimated error of an unsimulated corner compared to the predicted value of the corner. This allows the search algorithm to search areas with large predicted values and also explore areas of the input domain that have large values of uncertainty.

Selecting the optimal value of w is not straightforward. It may even be desirable for the algorithm to have a dynamic value of w, which changes as the algorithm progresses. Starting the algorithm with a high value of w would allow the algorithm to explore areas of the input domain that are sparsely sampled and potentially reveal regions where a possible maxima could occur. As the algorithm progresses, it could switch to a smaller value of w and focus on regions that have high predicted output values. In other words, the algorithm could start out by exploring the input domain to identify regions where potential maxima could occur and then shift focus to those regions.

As a starting point for the RPV algorithm we will use another heuristic and set the coefficient w to a value of 3, which will not dynamically change during the searching process. The next training corner will therefore be the rule $c_{next} = argmax(\hat{Y}(c) + 3 \times Stddev(c))$. Later in this chapter we will examine the results of experiments that use different choices of the w value. In the next chapter we will explore alternatives that consider more complex improvements of the heuristic.

3.2.3 Stopping Condition

Following the EGO framework, the last subproblem is the algorithm's stopping criterion. The difficulty with this type of problem is that even if we find the global maxima, we cannot be certain that the maxima we have found so far is indeed the global maxima. To determine a reliable stopping condition there needs to be a high level of confidence supporting it from the model. There is of course a major problem with stopping the search too soon. Finding a local optima, and not the true worst-case PVT corner, could have dire consequences if the yield of sellable chips becomes very low because of PVT variations, or if the yield can

only be improved by respinning the integrated circuit.

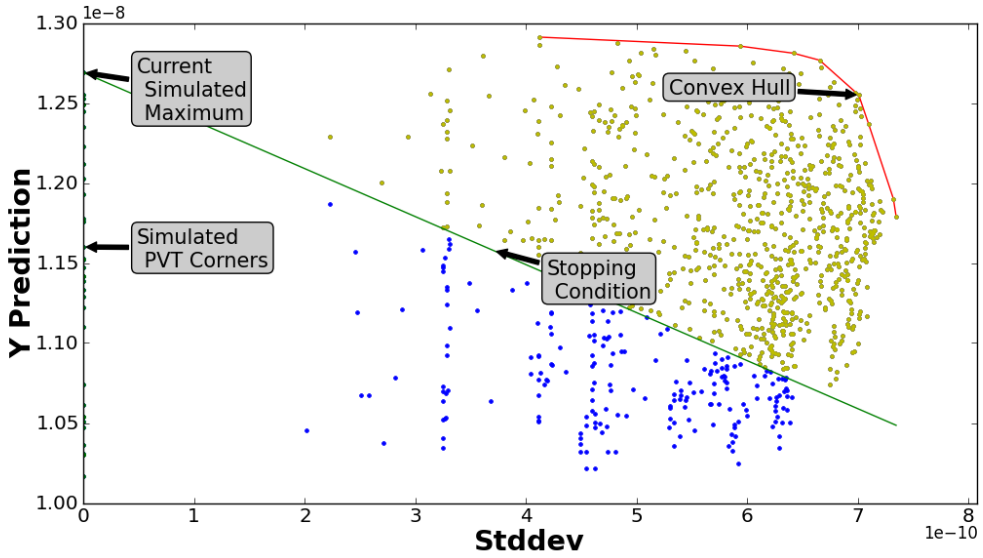


Figure 3.8: Visualization of the stopping condition of the *delay* output of the shift register circuit after 35 PVT corners have been simulated

With the GPM's ability to generate a prediction for the standard deviation ($Stddev(c)$) for every unsimulated corner ($\hat{Y}(c)$), we are able to estimate normal distribution confidence intervals and establish reasonable bounds for all predictions ($\hat{Y}(c)$). Assuming a normal distribution, the actual $Y(c)$ should fall within 2 standard deviations of the $\hat{Y}(c)$ prediction with 95.4% (i.e. 2-$Sigma$) confidence. Similarly, $Y(c)$ should fall within a 3 standard deviations of the $\hat{Y}(c)$ prediction with 99.7% (i.e. 3-$Sigma$) confidence.

Therefore we can impose a stopping criterion based on confidence levels. To do this, the algorithm uses the rule

$$Current\ maximum\ found\ so\ far\ >\ \hat{Y}(c) + n \times Stddev(c),\ \ for\ all\ unsimulated\ c$$

where n indicates the desired confidence level with respect to a normal distribution. To visualize this in terms of a stopping condition, Figure 3.8 plots $\hat{Y}(c)$ versus the standard deviation ($Stddev(c)$) of the *delay* output of the shift register circuit and indicates the stopping rule as a green line. The unsimulated PVT corners that are above this line are considered to be possible maxima. Once all PVT corners fall below the line the algorithm is able to terminate with a n-$Sigma$ level of confidence.

Figure 3.9 shows many different n-$Sigma$ confidence levels. For any selected value of n stopping criterion, all unsimulated PVT corners must be moved below the corresponding confidence line as a result of simulating more PVT corners and learning more accurate GPMs of the unknown circuit output function. The search for the global maximum terminates when the last unsimulated corner falls below the stopping condition line.

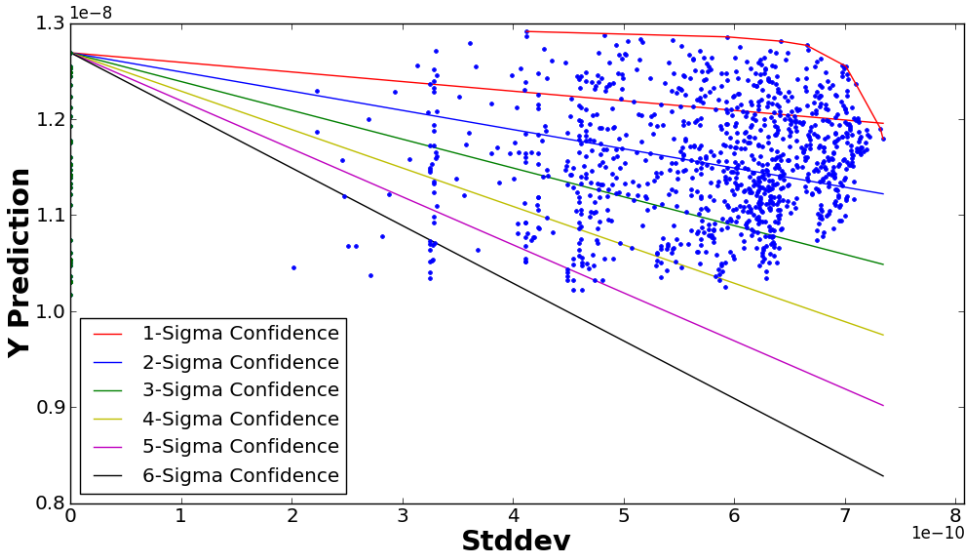
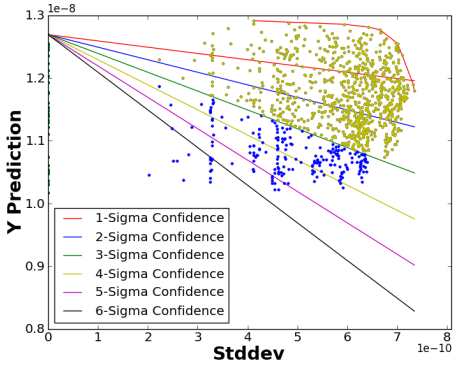


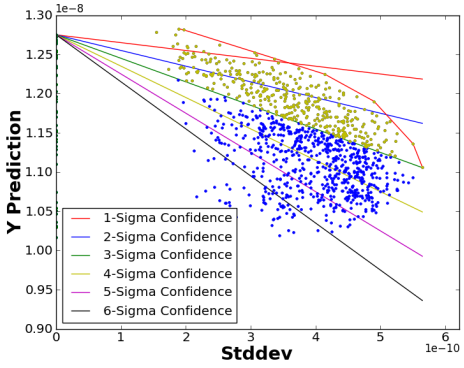
Figure 3.9: Visualization of six *n-Sigma* stopping conditions for the *delay* output of the shift register circuit after 35 PVT corners have been simulated.

We demonstrate how confidence levels are reached by looking at the progression of a GPM for the *delay* output of the register circuit at four instances. Figure 3.10 shows the progression of a GPM's confidence level after 35, 50, 70, and 100 PVT corners have been simulated. From Figure 3.10(c) we can see that the GPM has reached 2-*Sigma* confidence after 70 PVT corners have been simulated, and from Figure 3.10(d) we can see that the GPM has reached 3-*Sigma* confidence after 100 PVT corners have been simulated.

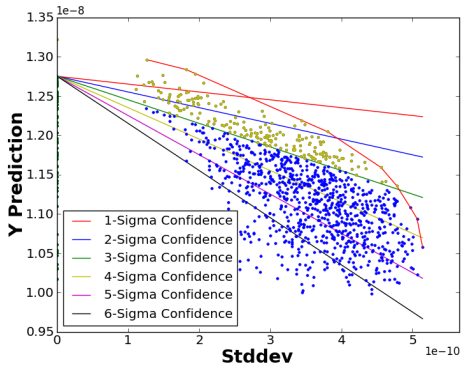
In the demonstration of Figure 3.10, the global maximum for the *delay* output was found after 67 PVT corners had been simulated. In this case, having a stopping condition of 2-*Sigma* ($n = 2$) would have been sufficient. Since the primary goal of this algorithm is reliability, an initial stopping criterion of $n = 3$ will be chosen to allow for a 99.7% confidence interval and should provide a reasonable level of reliability. This of course assumes that the predicted corners are indeed Gaussian distributed about the actual simulated corners and that the estimated deviations are near ideal standard deviations of those distributions.



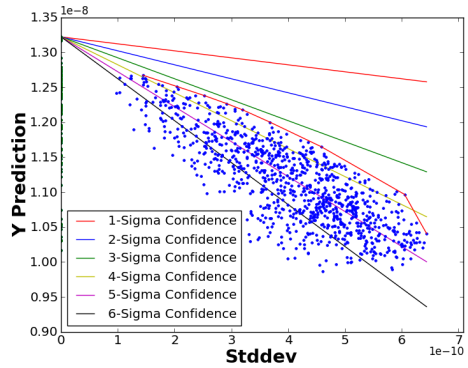
(a) GPM after 35 simulations



(b) GPM after 50 simulations



(c) GPM after 70 simulations



(d) GPM after 100 simulations

Figure 3.10: Progression of a GPM's confidence level of the *delay* output for the shift register circuit

3.3 Experimental Results of the Initial Algorithm

In this section we will investigate how well the initial RPV algorithm works on the nine-circuit dataset described at the beginning of the chapter.

In brief review, the current algorithm is set up as follows. The initial training set selects n^2 PVT corners, where n is the number of PVT parameters. This selection consists of one center point PVT corner and $n^2 - 1$ edge PVT corners according to the CCD. The next training corner is selected by the rule $c_{next} = argmax(\hat{Y}(c) + 3 \times Stddev(c))$. The algorithm terminates when the $Current\ maximum\ found\ so\ far\ > \hat{Y}(c) + n \times Stddev(c),\ for\ all\ unsimulated\ c.$

3.3.1 Initial Algorithm Results From Initial Dataset

In this experiment we will evaluate the RPV algorithm with the nine circuit data sets. Currently RPV only considers finding the maxima of one output at a time, so each circuit output will be considered in turn. We will record the total number of PVT corners in the data set, the initial training set size, how many PVT corners are required to be simulated to reach the 3-Sigma confidence, the speed-up of the algorithm (measured by the total number of corners divided by the number of corners required to reach the 3-Sigma stopping condition), and also the computational time taken (not including the time taken for corner simulation). These experiments were run on a desktop computer equipped with a 3.2-GHz Intel i5-3470 processor and 10 GB of DDR3 SDRAM.

Speed up is a measure of the efficiency of the RPV algorithm. We define speed up to be the total number (full factorial) of PVT corners for a given circuit divided by the number of PVT simulations required to reach 3-$Sigma$ confidence for a given output (i.e. Total Corners column divided by the Simulation 3 Sigma Reached column). In this type of measurement we desire a larger speed up as it reflects the goal of simulating fewer PVT corners for verification as opposed to the brute force approach of simulating every possible PVT corner. From Table 3.2 we can see the best speed up found was 16.22× and the worst was 1.03×. The average speed up was found to be 5.37× with a standard deviation of 4.54.

These initial results appear to be promising for the RPV algorithm. The global maximum was found for every output and the average speed up was 5.37×. This indicates that on average the RPV algorithm requires approximately one-fifth of the number of circuit simulations to verify a circuit compared to a full factorial approach.

In the trial of the MUX *qsgn* output we find the worst speed up and possibly the worst-case set of conditions for the algorithm. Although the global maximum was found on the first simulation, the algorithm required the simulation of 116 corners out of the available 120 to reach 3-$Sigma$ confidence. This indicates that the worst-case set of conditions for the RPV algorithm occurs when the total number of corners is in the range of only a few hundred and the GPM s confidence for the output function is slow to converge.

Also from looking at Table 3.2 we note that out of the 46 output trials, the maximum was

Table 3.2: Initial RPV Algorithm Results

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Shift Reg	delay	1080	49	66	93	11.61	2.40
Shift Reg	fall_time	1080	49	159	242	4.46	10.62
Shift Reg	rise_time	1080	49	127	127	8.50	2.73
Buffer Chain	Tf4_5	1800	100	19	111	16.22	0.57
Buffer Chain	Tr4_5	1800	100	19	114	15.79	0.73
Buffer Chain	avg_slew	1800	100	19	113	15.93	0.69
Buffer Chain	avgdly4_5	1800	100	19	112	16.07	0.63
Buffer Chain	fslew	1800	100	19	123	14.63	1.24
Buffer Chain	rslew	1800	100	19	118	15.25	0.97
Bit Cell	blwm	120	25	15	28	4.29	0.04
Bit Cell	blwm_mv	120	25	15	28	4.29	0.04
MUX	qfinal	120	64	7	65	1.85	0.02
MUX	qinit	120	64	2	101	1.19	0.98
MUX	qsgn	120	64	1	116	1.03	0.68
MUX	qtran	120	64	56	76	1.58	0.13
MUX	qtran0	120	64	56	76	1.58	0.13
MUX	setup_time	120	64	4	68	1.76	0.04
MUX	t_ref	120	64	8	65	1.85	0.02
Charge Pump 1	boostcr	216	64	58	66	3.27	0.05
Charge Pump 1	eq_error	216	64	10	65	3.32	0.02
Charge Pump 1	holdcrd	216	64	10	66	3.27	0.05
Charge Pump 1	holdcru	216	64	10	65	3.32	0.02
Charge Pump 1	ovdrive	216	64	14	67	3.22	0.07
Charge Pump 2	boostcr	324	64	28	67	4.84	0.08
Charge Pump 2	eq_error	324	64	28	69	4.70	0.07
Charge Pump 2	holdcrd	324	64	13	67	4.84	0.07
Charge Pump 2	holdcru	324	64	13	67	4.84	0.07
Charge Pump 2	ovdrive	324	64	71	74	4.38	0.26
Sense Amp	SAspeed	120	100	3	101	1.19	0.04
Sense Amp	glitch_senout	120	100	5	101	1.19	0.04
Sense Amp	maxout	120	100	12	104	1.15	0.06
Sense Amp	offset	120	100	6	101	1.19	0.04
Sense Amp	rslt	120	100	7	101	1.19	0.04
Sense Amp	sen_dip	120	100	12	104	1.15	0.06
Sense Amp	sen_dip_pctg	120	100	2	112	1.07	0.19
Bias Gen	bgr_m51_v145	120	9	2	24	5.00	0.17
Bias Gen	bgr_m51_v150	120	9	2	23	5.22	0.14
Bias Gen	bgr_m51_v155	120	9	2	20	6.00	0.09
Bias Gen	bgr_m51_v180	120	9	2	20	6.00	0.08
Bias Gen	bgr_m51_v195	120	9	2	21	5.71	0.10
Bias Gen	bgr_m51_v25	120	9	2	20	6.00	0.09
Bias Gen	bgr_m51_v27	120	9	2	20	6.00	0.08
Bias Gen	bgr_m51_v30	120	9	2	20	6.00	0.08
Bias Gen	bgr_m51_v33	120	9	2	20	6.00	0.09
Bias Gen	bgr_m51_v36	120	9	2	20	6.00	0.08
Op Amp	dc_gain	120	36	2	39	3.08	0.02

found within the initial training set 42 times. This suggests that the remaining simulations after the initial training set are being used to bring confidence in the GPM's predictions of the output function up to the 3-Sigma level. We can also notice that for cases where the total number of corners is fewer than 200, that smaller speed ups are observed, namely the MUX and Sense Amp circuits. In these cases the large number of input PVT parameters causes the initial training set to be large compared to the total number of corners. These two points raise the question, is the initial training set too large? In the case of the Sense Amp the initial training set size is comparable to the total number of corners, which suggests that the algorithm is not benefiting from the efficiency of the GPM's learning process.

In the next section we will investigate the impact of smaller initial training set sizes in an attempt to optimize the benefit of the GPM's learning process and increase the overall speed up.

3.3.2 Effect of the Initial Training Set Size

In this section we will use the same experiment as described before, but in this case we will vary the initial training set size. We will implement three new trials in which the initial training set sizes will be $\frac{3}{4}n^2$, $\frac{1}{2}n^2$, and $\frac{1}{4}n^2$. No other parameters will be changed from the RPV algorithm as described in the previous section.

With the original n^2 large initial training set, all trials were able to successfully find the global maximum by the time the algorithm declared termination with 3-$Sigma$ confidence. In Tables 3.3, 3.4, and 3.5 we can see there are times when the RPV algorithm failed to find the global maximum. In the $\frac{3}{4}n^2$ trial there were four termination failures for two of the circuits, Shift Reg and MUX. In the $\frac{1}{2}n^2$ trial there was only a single termination failure, for the Shift Reg circuit. The $\frac{1}{4}n^2$ trial suffered five termination failures for three different circuits, Shift Reg, MUX and Charge Pump 2.

This experiment is interesting because even though there are failures, it indicates that smaller initial training set sizes can allow for better speed ups. The speed up for the $\frac{3}{4}n^2$ trial ranged from 1.20× to 20.22×. The speed up for the $\frac{1}{2}n^2$ trial ranged from 1.32× to 29.03×. The speed up for the $\frac{1}{4}n^2$ trial ranged from 1.40× to 43.90×.

We find that sometimes it fails to find the global max and we wonder if it's due to the initial training corner selection. This leads to the next round of experimentation where we randomize the training corner selection and observe the effects of all four training set sizes.

Table 3.3: Initial RPV Algorithm Results for the $\frac{3}{4}n^2$ Initial Training Set Size

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Shift Reg	delay	1080	36	71	71	15.21	0.82
Shift Reg	fall_time	1080	36	Fail	138	7.83	3.59
Shift Reg	rise_time	1080	36	Fail	297	3.64	20.72
Buffer Chain	Tf4_5	1800	75	20	90	20.00	0.55
Buffer Chain	Tr4_5	1800	75	20	92	19.57	0.64
Buffer Chain	avg_slew	1800	75	20	91	19.78	0.59
Buffer Chain	avgdly4_5	1800	75	20	89	20.22	0.51
Buffer Chain	fslew	1800	75	20	106	16.98	1.19
Buffer Chain	rslew	1800	75	20	101	17.82	0.99
Bit Cell	blwm	120	18	16	23	5.22	0.07
Bit Cell	blwm_mv	120	18	16	23	5.22	0.07
MUX	qfinal	120	48	7	49	2.45	0.02
MUX	qinit	120	48	2	99	1.21	1.25
MUX	qsgn	120	48	1	49	2.45	0.01
MUX	qtran	120	48	Fail	66	1.82	0.17
MUX	qtran0	120	48	Fail	66	1.82	0.17
MUX	setup_time	120	48	4	57	2.11	0.08
MUX	t_ref	120	48	8	49	2.45	0.02
Charge Pump 1	boostcr	216	48	50	52	4.15	0.08
Charge Pump 1	eq_error	216	48	10	50	4.32	0.04
Charge Pump 1	holdcrd	216	48	10	52	4.15	0.08
Charge Pump 1	holdcru	216	48	10	50	4.32	0.03
Charge Pump 1	ovdrive	216	48	14	51	4.24	0.06
Charge Pump 2	boostcr	324	48	28	52	6.23	0.09
Charge Pump 2	eq_error	324	48	28	56	5.79	0.08
Charge Pump 2	holdcrd	324	48	13	55	5.89	0.14
Charge Pump 2	holdcru	324	48	13	54	6.00	0.12
Charge Pump 2	ovdrive	324	48	57	60	5.40	0.26
Sense Amp	SAspeed	120	75	3	76	1.58	0.03
Sense Amp	glitch_senout	120	75	5	78	1.54	0.08
Sense Amp	maxout	120	75	12	84	1.43	0.11
Sense Amp	offset	120	75	6	76	1.58	0.03
Sense Amp	rslt	120	75	7	76	1.58	0.03
Sense Amp	sen_dip	120	75	12	84	1.43	0.11
Sense Amp	sen_dip_pctg	120	75	2	100	1.20	0.32
Bias Gen	bgr_m51_v145	120	6	2	19	6.32	0.12
Bias Gen	bgr_m51_v150	120	6	2	19	6.32	0.11
Bias Gen	bgr_m51_v155	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v180	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v195	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v25	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v27	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v30	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v33	120	6	2	17	7.06	0.08
Bias Gen	bgr_m51_v36	120	6	2	17	7.06	0.08
Op Amp	dc_gain	120	27	2	34	3.53	0.07

Table 3.4: Initial RPV Algorithm Results for the $\frac{1}{2}n^2$ Initial Training Set Size

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Shift Reg	delay	1080	24	74	107	10.09	1.97
Shift Reg	fall_time	1080	24	Fail	90	12.00	1.65
Shift Reg	rise_time	1080	24	67	246	4.39	10.32
Buffer Chain	Tf4_5	1800	50	20	62	29.03	0.33
Buffer Chain	Tr4_5	1800	50	20	64	28.13	0.52
Buffer Chain	avg_slew	1800	50	20	64	28.13	0.40
Buffer Chain	avgdly4_5	1800	50	20	63	28.57	0.37
Buffer Chain	fslew	1800	50	20	89	20.22	1.22
Buffer Chain	rslew	1800	50	20	67	26.87	0.67
Bit Cell	blwm	120	12	17	17	7.06	0.07
Bit Cell	blwm_mv	120	12	17	17	7.06	0.07
MUX	qfinal	120	32	7	37	3.24	0.04
MUX	qinit	120	32	2	91	1.32	1.21
MUX	qsgn	120	32	1	33	3.64	0.01
MUX	qtran	120	32	41	41	2.93	0.07
MUX	qtran0	120	32	41	41	2.93	0.07
MUX	setup_time	120	32	4	44	2.73	0.09
MUX	t_ref	120	32	8	37	3.24	0.07
Charge Pump 1	boostcr	216	32	34	36	6.00	0.07
Charge Pump 1	eq_error	216	32	10	35	6.17	0.05
Charge Pump 1	holdcrd	216	32	10	38	5.68	0.09
Charge Pump 1	holdcru	216	32	10	34	6.35	0.03
Charge Pump 1	ovdrive	216	32	14	37	5.84	0.08
Charge Pump 2	boostcr	324	32	28	42	7.71	0.17
Charge Pump 2	eq_error	324	32	28	45	7.20	0.12
Charge Pump 2	holdcrd	324	32	13	42	7.71	0.16
Charge Pump 2	holdcru	324	32	13	42	7.71	0.17
Charge Pump 2	ovdrive	324	32	38	47	6.89	0.26
Sense Amp	SAspeed	120	50	3	53	2.26	0.06
Sense Amp	glitch_senout	120	50	5	58	2.07	0.15
Sense Amp	maxout	120	50	12	66	1.82	0.14
Sense Amp	offset	120	50	6	51	2.35	0.02
Sense Amp	rslt	120	50	7	51	2.35	0.02
Sense Amp	sen_dip	120	50	12	66	1.82	0.15
Sense Amp	sen_dip_pctg	120	50	2	90	1.33	0.42
Bias Gen	bgr_m51_v145	120	4	2	13	9.23	0.08
Bias Gen	bgr_m51_v150	120	4	2	13	9.23	0.08
Bias Gen	bgr_m51_v155	120	4	2	14	8.57	0.09
Bias Gen	bgr_m51_v180	120	4	2	15	8.00	0.08
Bias Gen	bgr_m51_v195	120	4	2	15	8.00	0.08
Bias Gen	bgr_m51_v25	120	4	2	14	8.57	0.09
Bias Gen	bgr_m51_v27	120	4	2	15	8.00	0.08
Bias Gen	bgr_m51_v30	120	4	2	15	8.00	0.08
Bias Gen	bgr_m51_v33	120	4	2	15	8.00	0.08
Bias Gen	bgr_m51_v36	120	4	2	15	8.00	0.08
Op Amp	dc_gain	120	18	2	29	4.14	0.09

Table 3.5: Initial RPV Algorithm Results for the $\frac{1}{4}n^2$ Initial Training Set Size

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Shift Reg	delay	1080	12	75	100	10.80	1.94
Shift Reg	fall_time	1080	12	Fail	57	18.95	0.83
Shift Reg	rise_time	1080	12	Fail	390	2.77	39.44
Buffer Chain	Tf4_5	1800	25	20	43	41.86	0.39
Buffer Chain	Tr4_5	1800	25	20	42	42.86	0.37
Buffer Chain	avg_slew	1800	25	20	43	41.86	0.40
Buffer Chain	avgdly4_5	1800	25	20	41	43.90	0.35
Buffer Chain	fslew	1800	25	20	59	30.51	0.79
Buffer Chain	rslew	1800	25	20	47	38.30	0.66
Bit Cell	blwm	120	6	7	14	8.57	0.08
Bit Cell	blwm_mv	120	6	7	14	8.57	0.09
MUX	qfinal	120	16	7	22	5.45	0.03
MUX	qinit	120	16	2	86	1.40	1.15
MUX	qsgn	120	16	1	17	7.06	0.01
MUX	qtran	120	16	Fail	36	3.33	0.13
MUX	qtran0	120	16	Fail	36	3.33	0.13
MUX	setup_time	120	16	4	31	3.87	0.19
MUX	t_ref	120	16	8	25	4.80	0.12
Charge Pump 1	boostcr	216	16	22	24	9.00	0.12
Charge Pump 1	eq_error	216	16	10	23	9.39	0.09
Charge Pump 1	holdcrd	216	16	10	26	8.31	0.14
Charge Pump 1	holdcru	216	16	10	24	9.00	0.11
Charge Pump 1	ovdrive	216	16	14	27	8.00	0.16
Charge Pump 2	boostcr	324	16	23	29	11.17	0.18
Charge Pump 2	eq_error	324	16	Fail	25	12.96	0.07
Charge Pump 2	holdcrd	324	16	13	26	12.46	0.15
Charge Pump 2	holdcru	324	16	13	28	11.57	0.17
Charge Pump 2	ovdrive	324	16	21	27	12.00	0.17
Sense Amp	SAspeed	120	25	3	28	4.29	0.04
Sense Amp	glitch_senout	120	25	5	41	2.93	0.23
Sense Amp	maxout	120	25	12	46	2.61	0.14
Sense Amp	offset	120	25	6	30	4.00	0.07
Sense Amp	rslt	120	25	7	30	4.00	0.03
Sense Amp	sen_dip	120	25	12	46	2.61	0.14
Sense Amp	sen_dip_pctg	120	25	2	77	1.56	0.44
Bias Gen	bgr_m51_v145	120	2	2	14	8.57	0.10
Bias Gen	bgr_m51_v150	120	2	2	14	8.57	0.10
Bias Gen	bgr_m51_v155	120	2	2	14	8.57	0.10
Bias Gen	bgr_m51_v180	120	2	2	14	8.57	0.10
Bias Gen	bgr_m51_v195	120	2	2	15	8.00	0.10
Bias Gen	bgr_m51_v25	120	2	2	14	8.57	0.09
Bias Gen	bgr_m51_v27	120	2	2	15	8.00	0.10
Bias Gen	bgr_m51_v30	120	2	2	15	8.00	0.10
Bias Gen	bgr_m51_v33	120	2	2	14	8.57	0.09
Bias Gen	bgr_m51_v36	120	2	2	15	8.00	0.10
Op Amp	dc_gain	120	9	2	25	4.80	0.09

3.3.3 Experiment with Randomized Training Sets of Varying Size

The previous experiments showed varying results when we augmented the initial training set size. This indicates that not only the initial training set size, but which corners are selected next are important to consider. To explore the impact of the initial training set on the performance of the RPV algorithm we will experiment with randomized selections of initial training corners. In the previous section we saw that the Shift Reg circuit seems to be a difficult case for the algorithm to solve. The Shift Reg has three outputs, one of which seems to be easy and the other two seem to be more difficult for the RPV algorithm.

In this section we will further experiment with the Shift Reg circuit with the aim of determining the reliability of the RPV algorithm with randomized initial training sets. To investigate this we decide to perform separate experiments for four different initial training set sizes, n^2, $\frac{3}{4}n^2$, $\frac{1}{2}n^2$, and $\frac{1}{4}n^2$. For each training set size we will perform 100 trials in which the corners selected for the initial training set will be randomized. We will ensure that each trial has a unique initial training set of PVT corners.

Termination failures from previous experiments may indicate that terminating the algorithm when the 3-*Sigma* confidence level is reached may not be a strict enough stopping criterion. Alternatively, perhaps the GPM's predictions are not as accurate as we have assumed and will be examined in Chapter 4. For these experiments each trial will record when the global maximum was found as well as when Sigma confidence levels from 3 to 10 are reached for the Shift Reg circuit outputs.

Since it would be cumbersome and unintuitive to analyze many tables of results for these trials directly, we will instead view plots of averaged results. In these plots we want to capture the probability of finding the global max when a certain Sigma confidence level is reached for each circuit output. Since these plots are also complex to analyze, we will view the results from only the n^2 initial training set size first. In Figure 3.11 the x-axis indicates how many PVT corner simulations are needed to reach the 3 to 10-*Sigma* confidence level. The y-axis measures the probability of finding the global maximum as a percentage of the number of trials that successfully found the global maximum by the time that each *Sigma* level was reached (i.e., if the true maximum is found 90 times out of 100, this would correspond to a 90% probability of finding the maximum for that confidence level).

In Figure 3.11 we can see the results of the n^2 initial training set size experiment. Each output is designated by a different line style: *delay* is a dashed line, *fall time* is a dotted line, and *rise time* is a solid line. As we can see from the plot, by the time the GPM reaches 3-Sigma confidence, the output's maximum is found 78% of the time (78 successful trials out of 100) for the *delay* output, 74% for *fall time*, and 44% for *rise time*. These probabilities are much lower than expected from a truly Gaussian model that has 3-*Sigma* confidence. We can see that for the algorithm to always find the true maximum for the *delay* output, a confidence level of 4-*Sigma* is required and that the *fall time* output requires 5-*Sigma*. We can also see that the *rise time* output requires 10-*Sigma* confidence to successfully find

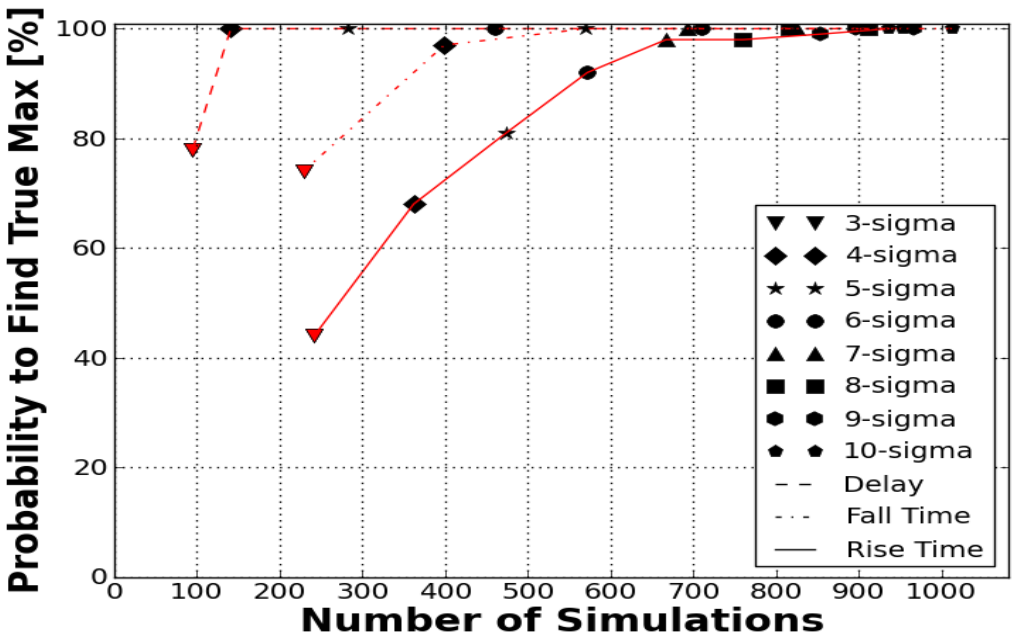


Figure 3.11: Measure of the RPV algorithm's reliability using randomized trials and averaging the results for an initial training set size of n^2.

the global maximum in all 100 trials.

Ideally when the GPM has 3-*Sigma* confidence, it should successfully find the true maximum with a probability of 99.7%. To understand these discrepancies from observed probability values and theoretical ideal values, we must acknowledge that the *Sigma* confidence level is only an estimation that is based on the GPM's own approximation of predicted standard deviations of output predictions. This will be a topic of great importance in the next chapter when we instigate ways to improve the algorithm.

To further explore the observed reliability of the algorithm, the results for all initial training set sizes are displayed in Figure 3.12. This plot is an extension of Figure 3.11. All conventions are the same as the previous plot as each output is designated by different line styles, but now the results from each training set size are indicated by the line color. The n^2 experiment uses red lines, the $\frac{3}{4}n^2$ experiment uses green lines, the $\frac{1}{2}n^2$ experiment uses blue lines, and the $\frac{1}{4}n^2$ experiment uses yellow lines.

Many interesting observations can be made from this plot. The experiments with the *delay* and *rise time* outputs seem to produce very similar results for all sizes of the training sets. In the case of the *fall time* output it seems that similar results are found for the n^2, $\frac{3}{4}n^2$, and $\frac{1}{2}n^2$ trials, but the $\frac{1}{4}n^2$ trial seems perform much worse. It seems that the *fall time* $\frac{1}{4}n^2$ trials are able to reach *Sigma* confidence levels after approximately the same number of simulations as the other training set size trials, but this training set size has decreased probability of finding the maximum for the 3-*Sigma* and 4-*Sigma* confidence

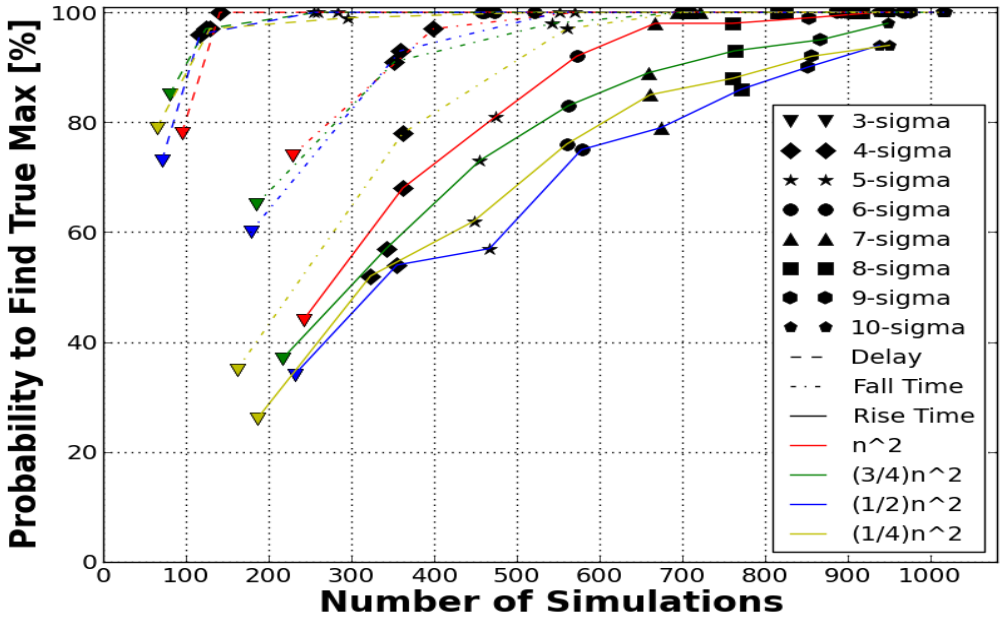


Figure 3.12: Measure of the RPV algorithm's reliability using randomized trials and averaging the results for an initial training set size of n^2, $\frac{3}{4}n^2$, $\frac{1}{2}n^2$, and $\frac{1}{4}n^2$.

levels. It is also interesting to note that for the *rise time* output the $\frac{3}{4}n^2$, $\frac{1}{2}n^2$, and $\frac{1}{4}n^2$ trials fail to find the global maximum by the time 10-*Sigma* confidence is reached. Clearly the GPM's reported 10-*Sigma* confidence does not reflect an ideal 10-*Sigma* level.

It is interesting that different initial training set sizes produce similar results. This indicates that using initial training sets smaller than n^2 is reasonable. We also notice that reported *Sigma* confidence levels do not match the ideal *Sigma* levels. This is an indication that the GPM's estimation of standard deviations is not always accurate. In the next chapter we will look at methods to improve the accuracy of the GPM's *Sigma* confidence levels.

Chapter 4

Improving the Single Output Method

In this chapter we will focus on improving the Rapid PVT Verification (RPV) method applied to finding the maximum of a single output function. We will explore new heuristics that make it more reliable, efficient and fast. The algorithm needs to be reliable in the sense that upon termination we can be confident that the true worst-case PVT corner has been found. We also want to make sure that the algorithm reaches termination in the fewest possible number of PVT corner simulations in order to minimize simulation expense and keep the algorithm efficient. As we experiment with new ideas to improve the RPV algorithm, we will make sure that the algorithm's computational complexity is kept minimal to allow for fast execution.

4.1 Experimenting With Smaller Initial Training Sets

In the previous chapter we used a modified CCD to select the initial training corners. The initial training set size or sample size, Q, was n^2, where n is the number of parameters. This sample size may not be the best choice as it does not consider the total number, M, of PVT corners. With our current data sets it seems that many circuits have $n > 5$ and $M < 200$. In such cases, $Q = n^2$ can be a large percentage of the total training corners available. For example, the Sense Amp circuit has 10 input parameters and only 120 available PVT corners. Choosing an initial sample size of n^2 corners results in simulating 100 of the possible 120 corners. This does not give the RPV method much scope to increase efficiency of the verification process. Conversely, a situation where n is very small, but M is very large, is possible (e.g., a case with few PVT parameters, but each PVT parameter has a large range of possible values). In such cases, choosing n^2 initial training corners may provide too small of a sample size and result in inaccurate modeling by the GPM.

In an attempt to balance the training set sizes based on values of M and n, we developed the heuristic rule $Q = max(0.01M, 2n)$. This rule should allow for smaller training set sizes and account for large and small values for both M and n. The goal of reducing the initial

sample size is to reduce unnecessary simulations and allow the algorithm to benefit from the likely advantage of having more directed choice on the corners that were selected earlier for simulation.

Previously our heuristic for selecting the initial training set was based on the CCD with a sample size of $Q = n^2$. The initial training set would be composed of one average-case PVT corner (a corner with the average/median value for each PVT parameter) and $(n^2 - 1)$ extreme PVT corners chosen at random. Our new proposed method will again select one average-case PVT corner, but will now select the remaining $max(0.01M, 2n) - 1$ edge corners in a space filling method. This space filling method searches the edge-case corners and selects the one that is at the greatest distance from any other corner that is already included in the training set. After adding the average-case corner to the training set, the method will select the corners with all maximum values. The method will then measure the distances of each edge-case corner to the closest corner in the training set. The method will add the edge-case corner with the largest distance away from those corners already in the present the training set.

Since different PVT parameters can be measured in different units, we decided to use a Manhattan distance expressed in dimensionless grid units to measure distances. To find these Manhattan distances we first find all possible values of each input PVT parameter $(x_1, x_2, ..., x_n)$. We then sort them in ascending order. We then give each value a relative position (i.e., 0, 1, 2, etc.). We use this relative grid position to compare different PVT corners. Distance is measured by the difference of each PVT parameter's relative grid position. For example, let x_1 be from the set $\{-50, -25, 0, 25, 50\}$. We map the values of x_1 to relative positions $0, 1, 2, 3, 4$. Let corner A be $[x_1 = -50, x_2, ..., x_n]$ and corner B be $[x'_1 = 0, x'_2, ..., x'_n]$. Assume that $x_2, ..., x_n$ are the same. We would determine the distance between corner A and B to be $|x'_1 - x_1| + |x'_2 - x_2| + ... + |x'_n - x_n| = |2 - 0| = 2$ grid units.

To examine the reliability of the new training set size we will perform an experiment with randomized training sets, similar to the experiment in the previous section, using the Shift Reg circuit. In this experiment we will perform 100 trials for each output of the Shift Reg circuit in which the corners selected for the initial training set will be randomized. We will ensure that each trial has a unique initial training set of PVT corners. Each trial will record when the global maximum was found as well as when *Sigma* confidence levels from 3 to 10 are reached. We will view the averaged results in Figure 4.1. In this figure the x-axis indicates how many PVT corner simulations are needed to reach the 3 to 10-*Sigma* confidence levels. The y-axis measures the probability of finding the global maximum as a percentage of the number of trials that successfully found the global maximum by the time that each *Sigma* level was reached.

We can compare Figure 4.1 to the previous $Q = n^2$ training set size experiment from Figure 3.11. In the case of the *delay* output we can see that the 5-*Sigma* confidence level is reached after ≈200 simulations with the smaller training set, whereas with the larger

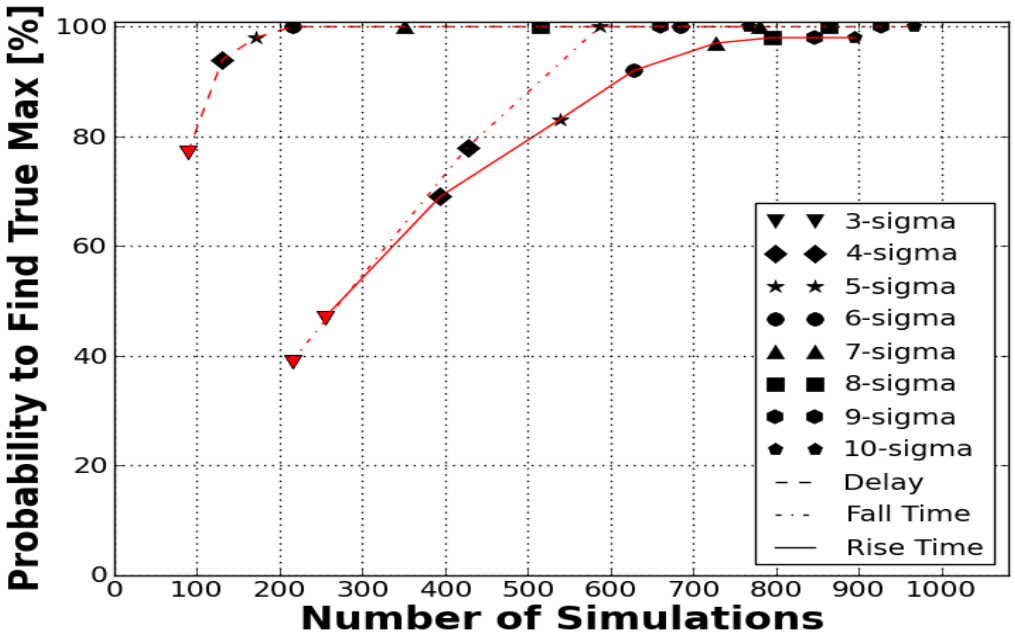


Figure 4.1: Measure of the RPV algorithm's reliability using randomized trials and averaging the results (axis labels still need to be changed).

training set it does not reach 5-*Sigma* until ≈300 simulations. There seem to be many cases where the smaller training set size achieves faster convergence to given target *Sigma* confidence levels.

We expected that the probability of finding the global maximum would be increased for each confidence level because the algorithm would spend more simulations actively searching for the maximum rather than exploring the input domain more broadly to create a larger initial training set. It seems that for confidence levels 3 and 4-*Sigma* for all outputs that the probability of finding the global maximum decreases with smaller training set sizes. However this may be due to the present next corner selection method. We will explore alternative next corner selection methods in the upcoming sections.

In Figure 4.1 a big surprise occurs with the *rise time* output. Two out of the one hundred trials fail to find the global maximum by the time the GPM declares 10-*Sigma* confidence. The probability of finding the global maximum seems to steadily increase between Sigma confidence levels until 8-*Sigma* is reached. Once 8-*Sigma* is reached the algorithm seems to not increase any closer to 100% the probability of finding the global maximum, even with many more simulations. In the next section we will investigate why this plateauing phenomenon occurs.

4.2 Introducing an Amplification Factor for Uncertainty Near Known Maxima

In the previous section we reported an unexpected result. In our 100 randomized initial training corner set experiment we observed that in two trials, the RPV method failed to find the global maximum of the *rise time* output of the Shift Register circuit, even when $10 - Sigma$ confidence was declared. This is very unexpected since the 10-Sigma threshold should be very hard to reach, and reaching it should be a reliable indicator of termination with the correct function maximum. This discrepancy should help us identify a weakness in our RPV method.

Since we kept detailed records for each trial in the experiment, we were able to recreate the GPM for the *rise time* output and observe what happened using a convex hull plot. In Figure 4.2 we can view the state of the GPM before it terminates. As seen in previous convex hull plots, the x-axis is the estimated standard deviation of each prediction ($Stddev(c)$) and the y-axis is the predicted output value ($\hat{Y}(c)$) of each PVT corner. Note that the corners on the vertical y-axis (where $Stddev = 0$) are the corners that have been simulated and that presently are the training set for the GPM. The highest corner on the vertical y-axis is the current maximum value that has been simulated. The downward sloping line that extends from the current maximum represents the 10-$Sigma$ confidence level. Once all of the unsimulated PVT corners are below this line 10-$Sigma$ confidence will be reached.

At the situation shown in Figure 4.2, 895 PVT corners of the possible 1080 of the *rise time* output have been simulated and there are very few corners remaining above the $10 - Sigma$ confidence level. From our initial observations about this trial we found that the corner, which the method believed to be the global maximum, was only 1 Manhattan distance step away from the true global maximum. This intrigued us and lead us to highlight PVT corners that are within 3 Manhattan steps away from the current global maximum. PVT corners that are 1 Manhattan step away are highlighted with a red dot, corners that are 2 Manhattan steps away are highlighted with a blue dot, and corners that are 3 Manhattan step away are highlighted with a yellow dot. We also mark the global maximum corner with a black diamond. The second largest maximum is marked with a black square and the third largest maximum is marked with a black triangle. From this convex hull plot we can see that the second largest maximum has been simulated and is the GPM's current maximum. Both the first and third largest maxima are only 1 Manhattan step away from the GPM's current maximum, but they have not yet been identified by the algorithm as potential maxima.

From these observations it seems that the GPM has the unfortunate tendancy to underestimate the uncertainty of it's predictions near local maxima. To offset this weakness we investigated using an amplification factor for the GPM's reported $Stddev$. This amplification factor will artificially increase the $Stddev$ value for PVT corners near the algorithm's current maximum. To determine the best strategy for using an uncertainty amplification

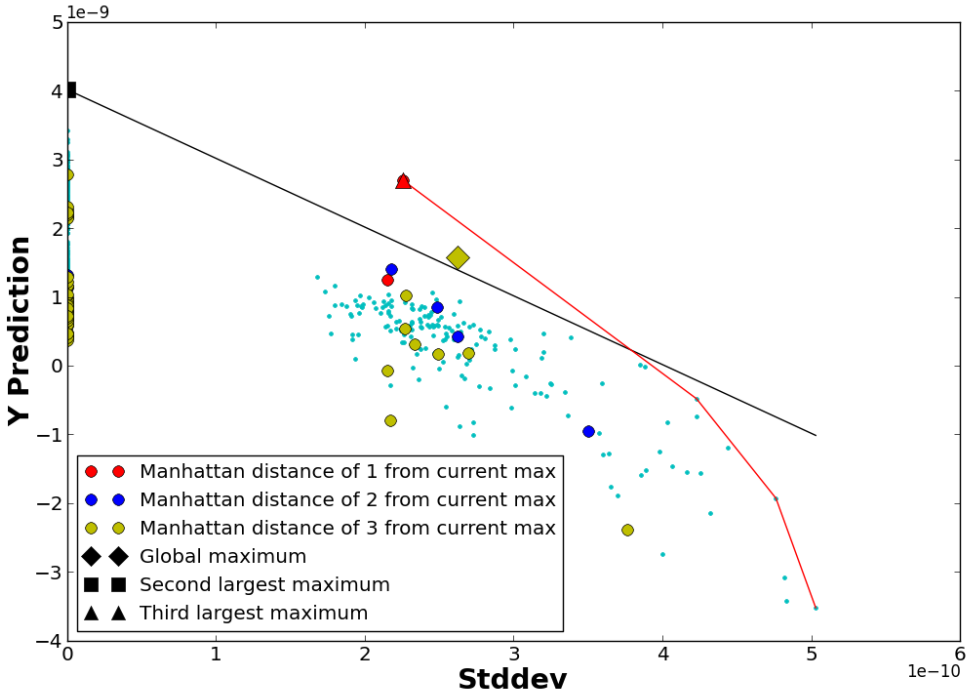


Figure 4.2: GPM of the *rise time* output after 895 PVT corner simulations.

factor, we performed experiments to determine which approach works best.

In this experiment we will examine three variations of uncertainty amplification using the 100 randomized initial training sets experiment with the same three outputs of the Shift Register circuit. The first will apply the amplification factor to *Stddev* of PVT corners that are up to three Manhattan steps away. The second will apply the amplification factor to PVT corners that are up to two Manhattan steps away. The third will apply the amplification factor to PVT corners that are only one Manhattan step away. We will use a heuristic that PVT corners that are one Manhattan step away will have their *Stddev* increased by 25%. Similarly, PVT corners that are two Manhattan steps away will have their *Stddev* increased by 15% and PVT corners that are three Manhattan steps away will have their *Stddev* increased by 5%.

In Figure 4.3 we can see the results of the 3-step amplification factor. In this variation of the amplification factor, the experiment took 120879.396 seconds (33.58 hours) to complete with an average of 402.931 seconds per single trial completion (there were 100 trials performed for each of the three circuit outputs, for a total of 300 trials run). The 3-step amplification factor method is able to reach 99% probability of finding the global maximum for the *delay* output by the $4 - Sigma$ confidence level, which is higher than the other two uncertainty amplification methods at the 4-Sigma confidence level. The results for the *fall time* output do not seem to vary much from the results of the other methods in this

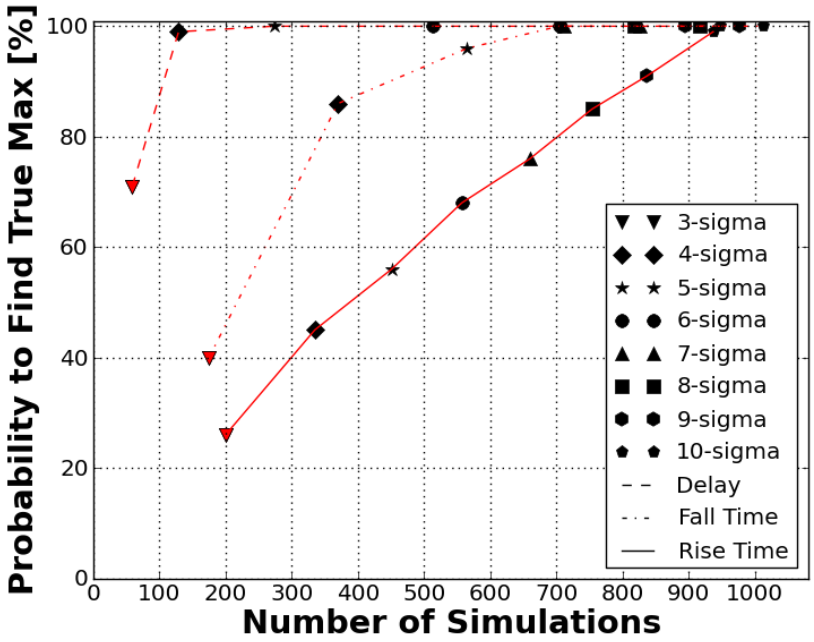


Figure 4.3: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for the 3-step uncertainty amplification factor.

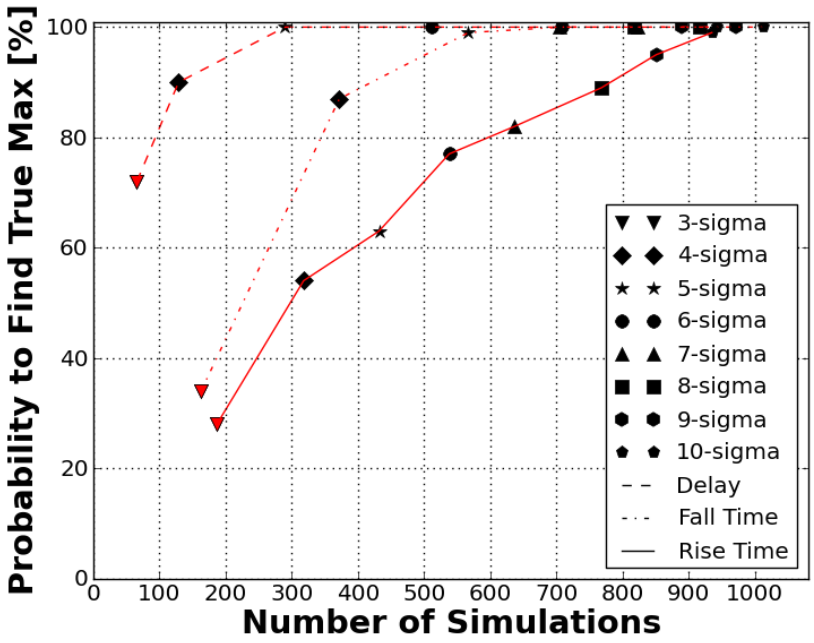


Figure 4.4: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for the 2-step uncertainty amplification factor.

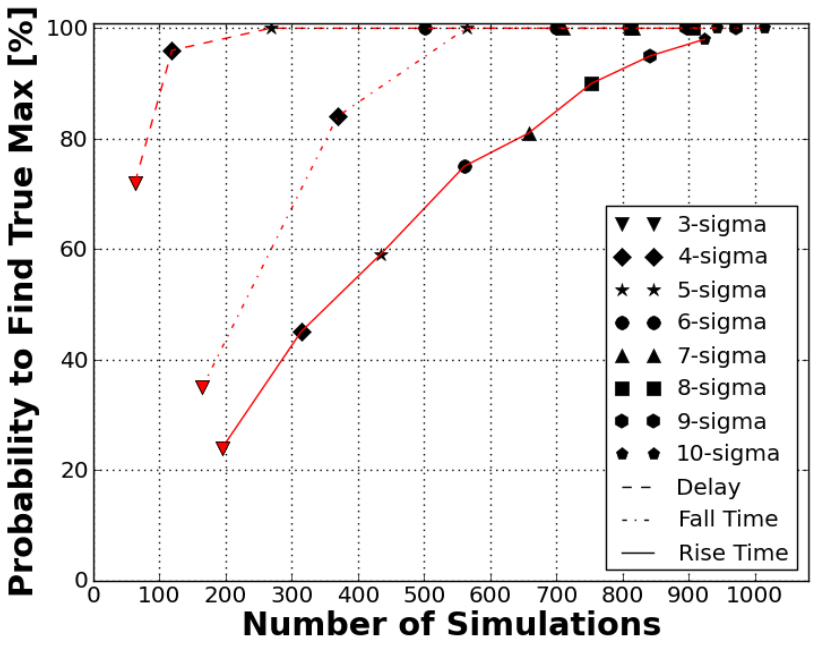


Figure 4.5: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for the 1-step uncertainty amplification factor.

experiment. We can see that the 3-step amplification method is able to reach 100% probability of finding the true maximum for the *rise time* output, while previously our algorithm had trials in which the global maximum was overlooked due to a near local maximum.

Figure 4.4 shows the results of the 2-step amplification factor which artificially increases the *stddev* value for any PVT corners that are 2 Manhattan steps away from the current maximum. This experiment took 118425.707 seconds (32.90 hours) to complete with an average of 394.752 seconds per single trial completion. By 4-Sigma confidence, the RPV algorithm is able to reach a 100% probability of finding the true maximum for the *delay* output and 98% for the *fall time* output (which reaches 100% by 5-Sigma). By the 10-*Sigma* confidence level the RPV algorithm is able to reach 100% probability of finding the true maximum for the *rise time* output.

The third and last variation of the uncertainty amplification factor experiment results can be seen in Figure 4.5. In this variation only PVT corners that are 1 Manhattan step away from the current maximum have their *Stddev* value increased by the amplification factor. This experiment took 116903.492 seconds (32.47 hours) to complete with an average of 389.678 seconds per trial completion. For the *delay* and *fall time* outputs, the RPV algorithm was able to reach 100% probability of finding the true maximum by 4-*Sigma* confidence. For the *rise time* output, the RPV algorithm reached 98% probability of finding the true maximum by the 10-Sigma confidence level. This means that in 2 of the 100 trials, for the *rise time* output the true global maximum was not found.

From these three experiments we can see that an uncertainty amplification factor can improve the algorithm's probability of finding the global maximum by the time of termination. The 1-step method does not seem to provide the desired increase in probability as the other two methods do. Although the 3-step method does see an increase in probability of finding the true maximum for the *delay* output than the other variations, it does take the longest to execute. Since the 2-step amplification factor is able to find the global maximum for all 100 trials of each of the circuit's outputs and is able to execute faster than the 3-step method, we will utilize this 2-step method in out RPV algorithm.

4.3 Experimenting with the Next Corner Selection Rule

In this section we want to explore several options for selecting the next corner(s) to be simulated. The current algorithm selects one corner at a time to be simulated using the rule $c_{convexhull} = argmax(\hat{Y}(c) + w \times Stddev(c))$ with $w = 3$. The parameter w could reasonably hold any suitable value in the range $(0, \infty)$ so it may be best to try multiple values in this range. We will experiment with different w values and look at methods that choose more than one corner at a time.

We will use the randomized initial training set experiment for the Shift Register circuit when determining if the algorithm's performance can be increased by changing the w parameter. For this experiment the algorithm will also use the 2-step uncertainty amplification factor as described in the previous section. In this experiment we will observe the effects of increasing the w parameter to 6 and also 9. The results are plotted below.

In Figure 4.6 the experiment was run with $w = 6$ and took 82173.267 seconds (22.83 hours) to complete. Figure 4.7 shows the results of the experiment with $w = 9$ that took 86279.171 seconds (23.97 hours) to complete.

From the previous section we saw that with the 2-step uncertainty amplification factor and $w = 3$ the algorithm was able to reach 100% probability of finding the true maximum for the *delay* output by 5-*Sigma* confidence; similarly, $fall\ time$ reached 100% probability by 5-*Sigma* confidence as well and *rise time* reached 100% probability by the 10-*Sigma* confidence level. From Figure 4.6 we can see that with $w = 6$, the algorithm reached 100% probability of finding the true maximum for the *delay* output by 4-*Sigma* confidence, the $fall\ time$ output by 6-Sigma and the *rise time* output by 9-*Sigma*. From observation it seems that with $w = 6$ that almost every confidence level has a higher probability of finding the true global maximum than with $w = 3$. It seems that for the performance for the more difficult output, *risetime*, there is significant improvement. For example, at the 5-*Sigma* confidence level, a correct termination probability of 85% is reached with $w = 6$ compared to 62% with $w = 3$.

From Figure 4.7 we see that with $w = 9$, the algorithm reaches 100% probability of finding the true maximum for the *delay* output by 4-*Sigma* confidence and for the $fall\ time$ output by 6-Sigma. However, for the *rise time* output, the algorithm is only able to reach

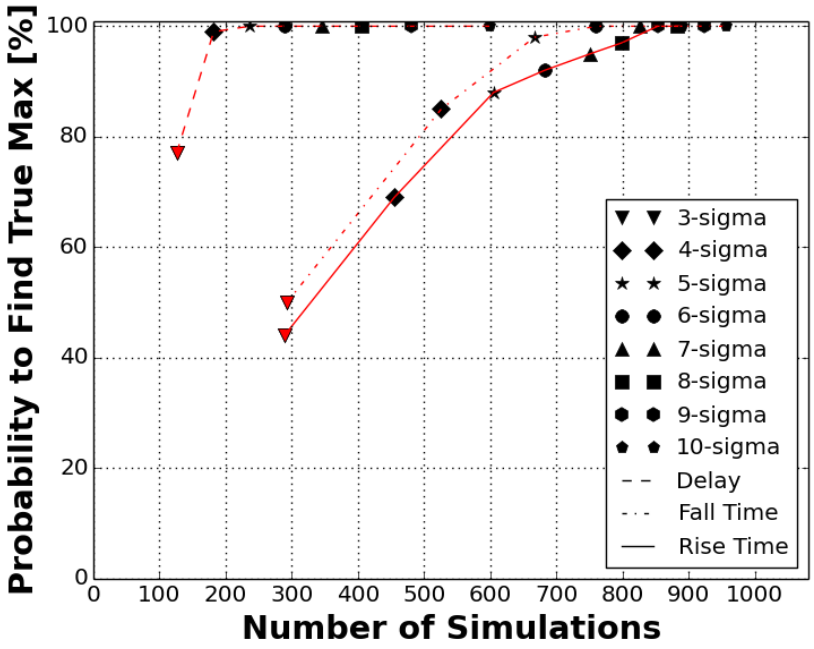


Figure 4.6: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for $w = 6$.

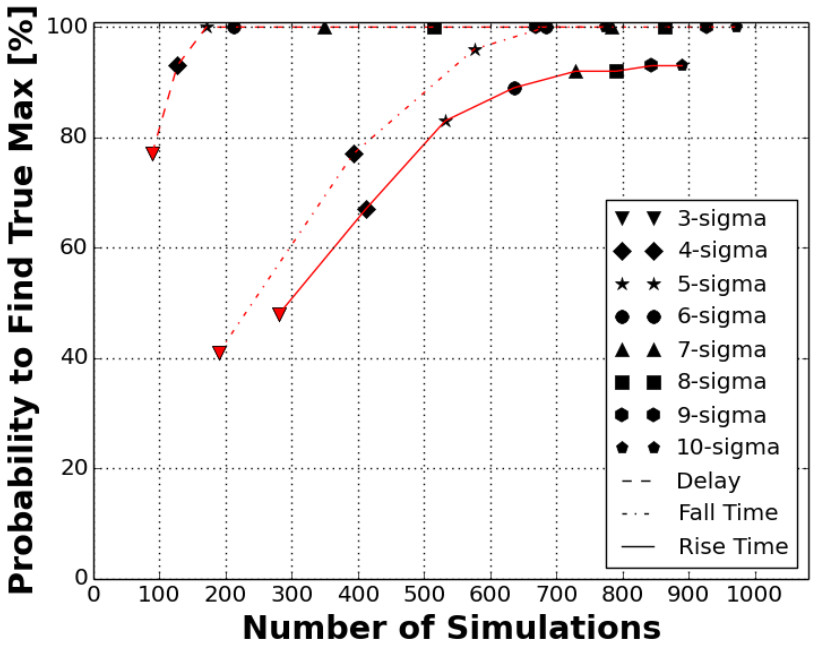


Figure 4.7: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for $w = 9$.

93% probability of finding the true maximum by the 10-Sigma confidence level.

From this *shift register* circuit we see that selecting $w = 6$ produces the best performance for these three outputs. However, this may not be the case for other circuits. We must consider a more general approach that does not depend on a user-determined w parameter. One approach would be to simulate all of the corners on the convex hull instead of choosing only one corner to simulate at each iteration. By simulating all of the corners on the convex hull we are able to focus on a relatively small set of corners that have the greatest potential of being a maxima. This approach may also allow the RPV algorithm to terminate with less computational time because intermediate calculations (i.e., generating GPMs and distance measurements) are not being done after each individual simulation. We will use this next corner selection method in the randomized initial training corner experiment to see how it performs.

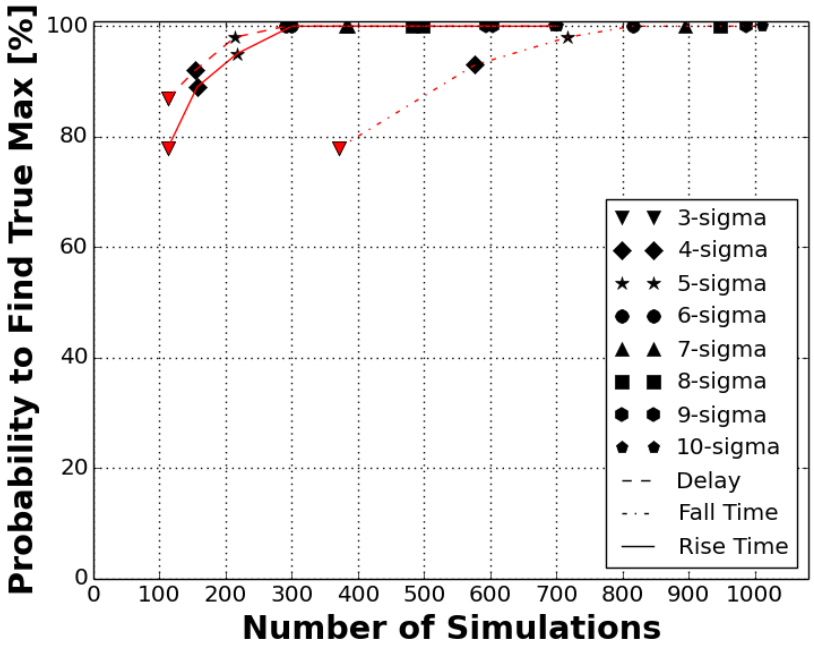


Figure 4.8: Probability that the RPV algorithm finds the global maximum using 100 randomized trials and averaging the results for simulating all corners on the convex hull.

In Figure 4.8 we can see that the algorithm's probability of finding the global maximum reaches 100% for all three outputs by the 6-*Sigma* confidence level. With the previous best result, $w = 6$, the algorithm takes until the 9-*Sigma* confidence level for all three outputs to reach 100% probability of finding the true maximum. We can also see that by the 3-*Sigma* confidence level the algorithm reaches 78% to find the true maximum for the *rise time* and *fall time* outputs and 87% for the *delay* output. These probabilities for the 3-*Sigma* confidence level are much higher than for any of our previous methods.

From Table 4.1 we can see that the convex hull method took only 4.919 hours to com-

Table 4.1: Experiment Running Times

Next Corner Selection Method	**Running Time (seconds)**	**Running Time (hours)**
$w = 3$	118425.707	32.90
$w = 6$	82173.267	22.83
$w = 9$	90702.285	25.20
Convex Hull	17709.466	4.92

plete, which is significantly faster than all other methods. The convex hull method is 4.6× faster than the $w = 6$ method and is 6.7× faster than the $w = 3$ method.

These results show that selecting multiple corners per iteration has a clear computational time advantage over selecting only one corner per iteration. We have also seen increased probability of finding the global maximum by using the convex hull method. From our experiments so far, we have seen the best overall performance is achieved when the RPV algorithm uses a 2-step uncertainty amplification factor and simulates all corners on the convex hull at each iteration.

4.4 Examine the Predicted Error and Introduce a Correction Factor for the GPM

In this section we will address the accuracy of the GPM's sigma estimation. We have seen from the many previous trials that the GPM's estimation of $Sigma$ values are often far from ideal statistical $Sigma$ values. If the GPM's estimated standard deviations were ideal, we would expect to see 99.7% of predictions within 3 standard deviations.

$$Pr(\hat{y}_i - 3 \times \sigma \leq y_i \geq \hat{y}_i + 3 \times \sigma) \approx 0.9973$$

When the RPV algorithm declares 3-$Sigma$ confidence of finding the maximum value, then we would expect that the global maximum value would be found 99.7% of the time. From our observations, we find that the probability of finding the maximum at the reported 3-$Sigma$ confidence level is much lower than the ideal value.

We will design an experiment to monitor the GPM's reported $Sigma$ levels. In this experiment we will measure the percentage of PVT corners that fall within 3 standard deviations (i.e., 3-$Sigma$) of their predicted output values with different training set sizes and display the results in a plot. The x-axis of the plot will be the percentage of total available PVT corners and the y-axis will report the percentage of PVT corner predictions that fall within the GPM's estimated 3-$Sigma$ level. The experiment will use training set sizes that range from 5% to 95% incrementing by 5% step sizes (i.e., 5%, 10%, 15%,..., 95%) of the total number of PVT corners. For each training size percentage we will create 100

randomly selected training sets and for each training set we will generate a GPM. For each training set we will calculate the percentage of unknown corners that are within the GPM's estimated 3 standard deviations of their predicted output values. We will then average these values for each training set size and plot them. This will be done for all three outputs of the *shift register* circuit.

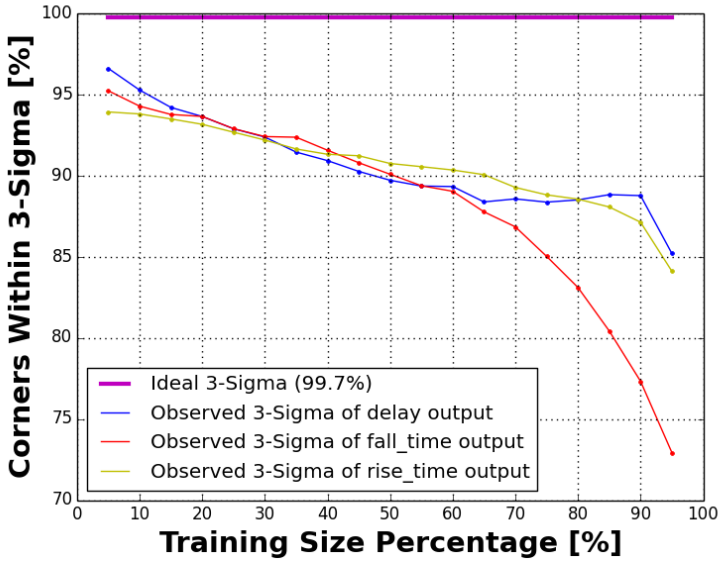


Figure 4.9: GPM's observed 3-*Sigma* of the shift register circuit outputs.

Figure 4.9 shows the ideal 3-*Sigma* level and the observed $3-Sigma$ levels for *delay*, *fall time*, and *rise time* outputs of the *shiftregister* circuit. We can see that the GPM's estimation of standard deviation is not accurate and results in a 3-*Sigma* level that is lower than the ideal. Since the GPM's estimated standard deviation is not accurate we will impose a correction factor, β, to improve the observed *Sigma* levels. This β value will be applied to the GPM's estimated standard deviation in order to reflect near-ideal *Sigma* values.

To calculate β we will use a popular validity test known as k-fold cross validation [46]. In k-fold cross validation the known data (i.e., the training set) is divided into approximately equally sized k subsets [46]. Models are created k times by the machine learning method and each time one of the subsets is left out of the model's training set. The subset not included with training the model is used for evaluation and is referred to as the test set. Normally cross validation is used to calculate the prediction error of each test set, but we will use it to establish a reasonable β value. We will use 10-fold cross validation as it is a commonly accepted standard for k-fold cross validation [46].

During cross validation we want to find a β value that for every test-set PVT corner prediction will fall within these bounds:

$$\hat{y}_i - 3 \times \sigma \times \beta \leq y_i \geq \hat{y}_i + 3 \times \sigma \times \beta$$

To find this β value we first calculate intermediate β values for each prediction using

$$\beta_i = |y_i - \hat{y}_i|/(3 \times \sigma)$$

. We then set β to be the maximum β_i value calculated. If we apply this β value to all the test-sets during cross validation we will find that the percentage of predicted output values fall within the new 3-*Sigma* to be 100%, which is $\approx 99.7\%$.

We will now observe the GPM's reported *Sigma* levels with the new correction factor, β. We will perform the previous experiment from this section in which we select various different training size sets and create 100 randomized training sets to monitor the percentage of GPM predictions that fall within the estimated 3-*Sigma*. The results can be found in Figure 4.10.

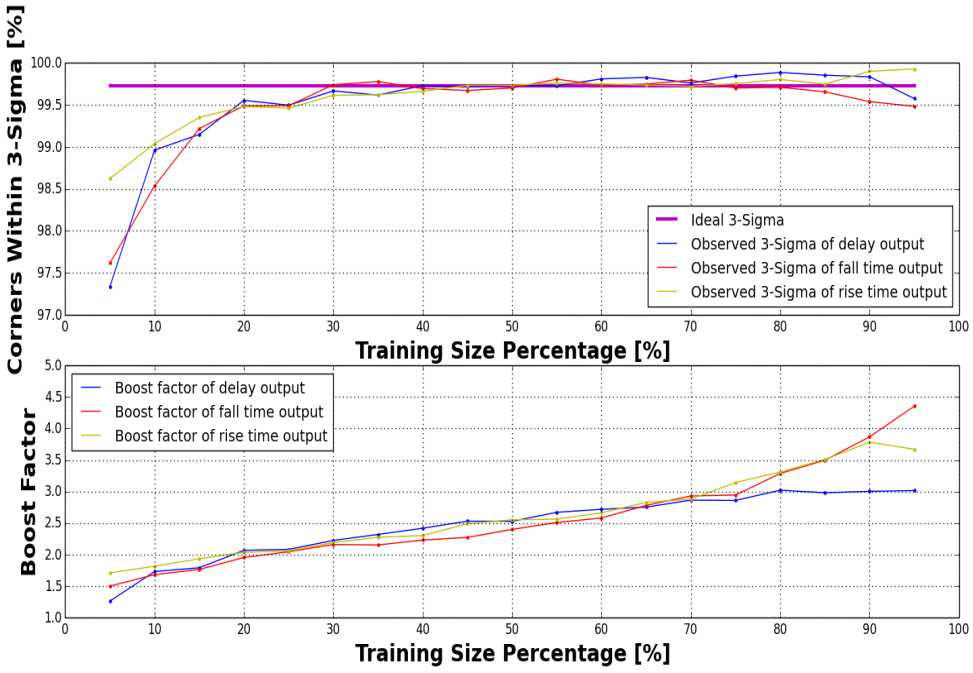


Figure 4.10: GPM's observed 3-*Sigma* with the correction factor of the shift register circuit outputs.

The results shown in Figure 4.10 shows a significant increase in performance and shows near-ideal statistical values for 3-*Sigma*. However, we still observe lower than ideal values for training size sets below 20% of the total number of corners. We will add a heuristic to the way we calculate β to improve these values. We shall artificially increase the β value by 25% until the training set size is larger than 20% of the total number of corners.

The experiment was run a third time and the results are shown in Figure 4.11. We see that the PVT corners within 3-*Sigma* are almost always above 99% for the training set sizes less than 20% of the total corners and maintain a near ideal statistical level for all training set sizes larger than 20%. This heuristic based correction factor for the GPM's

standard deviation provides probabilities that predictions are within $3-Sigma$ that are much closer to ideal statistical values.

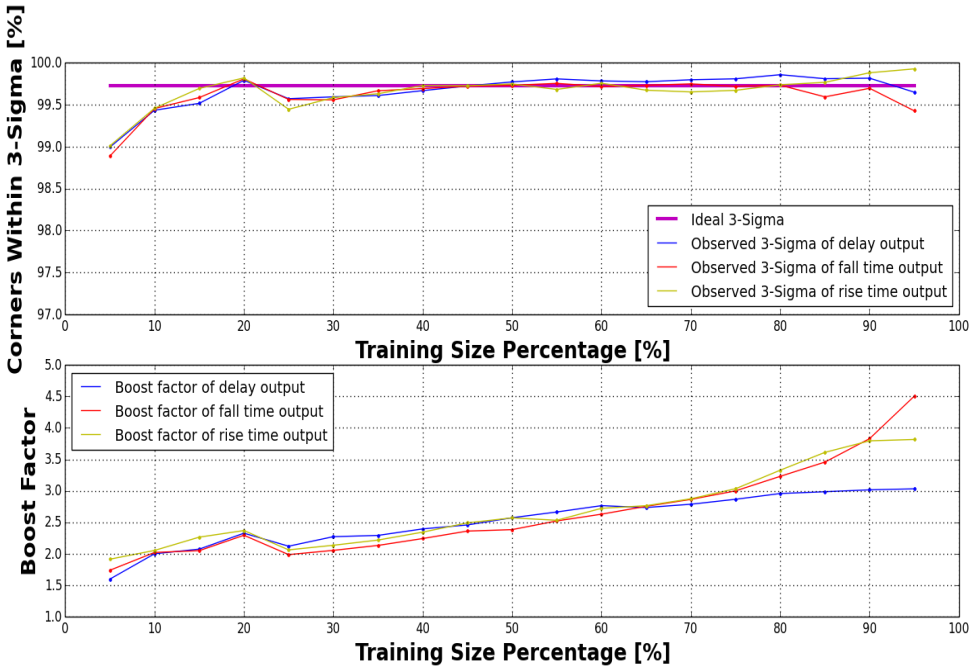


Figure 4.11: GPM's observed $3-Sigma$ with the modified correction factor of the shift register circuit outputs.

Before implementing a correction factor in the RPV algorithm we must consider the significant cost of which it takes to calculate the correction factor. It would not be feasible to perform 10-fold cross validation every iteration of the algorithm since that would roughly increase the computational complexity by $10\times$. We propose a method that only computes the correction factor, β, a maximum of 10 times during the execution of the algorithm. We will calculate β after the initial training set has been simulated and then at every 10% interval of the total number of PVT corners (i.e., calculate β at 10%, 20%,..., 90% of the total number of PVT corners). Since we observed the correction factor to have a very linear nature in Figure 4.10 and 4.11 we will use a linear approximation to update the correction factor during iterations that do not use 10-fold cross validation to calculate the correction factor.

To define the linear approximation of β we will use an example case, as shown in Figure 4.12. At iteration x_i, correction factors β_A and β_B are known and have been calculated for iterations x_A and x_B, respectively. Note that $x_A < x_B < x_i$. We use the linear approximation $\beta_i \approx m \times x_i + \beta_0$. We calculate the slope as $m = (\beta_B - \beta_A)/(x_B - x_A)$, and the y-intercept as $\beta_0 = \beta_B - m \times x_B$. We can now use the linear approximation $\beta_i \approx m(x_i - x_B) + \beta_B$. This approximation is used to update the correction factor during iterations that do not use 10-fold cross validation to calculate β_i. This approximation is

used to reduce the computational complexity of calculating the correction factor.

Another consideration that must be taken into account is the size of the initial training set, Q. We must ensure there are at least 10 PVT corners in the initial training set, since we intend to perform 10-fold cross validation immediately after they have been simulated. Therefore, we will set $Q = max(0.01M, 2n, 10)$.

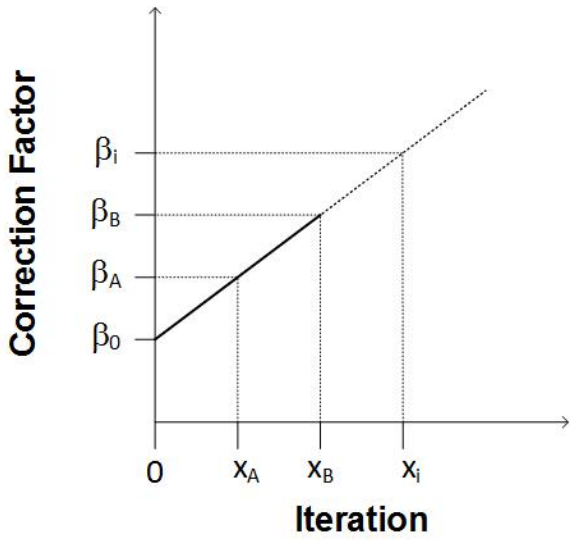


Figure 4.12: Linear approximation of the correction factor β_i.

4.5 Improved Algorithm Performance

In this section we will investigate how well the improved RPV algorithm performs on the nine-circuit dataset described in the previous chapter as well as an alternative ten-circuit dataset.

In brief review, the improved RPV algorithm is set up as follows. The initial training set selects $Q = max(0.01M, 2n, 10)$ PVT corners, where n is the number of PVT parameters and m is the total number of PVT corners. This selection consists of one center point PVT corner and $Q-1$ edge PVT corners according to the CCD. We also use a 2-step uncertainty amplification factor that increases the *Stddev* of PVT corners that are two Manhattan steps away by 25% and PVT corners that are one Manhattan step away by 15%. The algorithm selects all corners on the convex hull to be simulated at each iteration. The algorithm terminates when $Current\ maximum\ found\ >\ all\ predicted\ output\ values\ +\ 3 \times \beta \times standard\ deviation$, where β is the standard deviation correction factor. All experiments will be run on a desktop computer equipped with a 3.2-GHz Intel i5-3470 processor and 10 GB of DDR3-SDRAM.

4.5.1 Improved Algorithm Results from the Initial Dataset

In this experiment we will evaluate the improved RPV algorithm with the nine-circuit dataset. The RPV will consider the maxima of one output at a time so each circuit output will be tested individually. We will record the total number of PVT corners in the data set, the initial training set size, how many PVT corners are required to be simulated to reach the 3-Sigma confidence, the speed-up of the algorithm (measured by total number of corners divided by number of corner required to reach the 3-Sigma stopping condition), and also the computational time taken (not including time taken for corner simulation).

In the previous chapter we defined speed up as the total number (full-factorial) of PVT corners for a given circuit divided by the number of PVT simulations required to reach 3-Sigma confidence for a given output (i.e. Total Corners column divided by Simulation 3 Sigma Reached column). Speed up is used as a measure of efficiency for the RPV algorithm. From Table 4.2 the best speed up found was 23.38× and the worst was 1.02×. The average speed up was found to be 5.91× with a standard deviation of 5.92. There are no cases in which the RPV algorithm fails to find the circuit's output maximum.

Previously, from Table 3.2, we saw the best speed up found was 16.22× and the worst was 1.03× for the initial algorithm. The initial algorithm's average speed up was 5.37× with a standard deviation of 4.54. From these observations we can note that the improved algorithm's best speed up and average speed up has been increased. It is important to note that even with a more strict termination criterion, we observe increased average speed up. If we only improved the initial algorithm by making the termination criterion more strict, we would expect to see a much lower average speed up due to increased number of simulations required to reach termination. The fact we observe an increased average speed up must be due to the effective combination of using a reduced initial training set size and using heuristic-based search methods.

4.5.2 Experiment with Randomized Training Sets

We will now examine the reliability of the improved RPV algorithm using the 100 randomized initial training set experiment with the three outputs of the shift register circuit. In this experiment we will generate 100 randomized initial training sets for each circuit output and monitor when the global maximum is found, as well as when 3-$Sigma$ confidence to 10-$Sigma$ confidence levels are reached. A plot will be created to display how many simulations (on average) are required to reach each confidence level (x-axis) and the corresponding probability of finding the global maximum by that confidence level (y-axis). With the introduction of the correction factor we expect to observe near ideal statistical $Sigma$ levels.

Figure 4.13 shows the results of the improved RPV algorithm. This experiment took 41696.161 seconds (11.58 hours) to complete with an average of 138.987 seconds per single trial completion. We observe that by 3-$Sigma$ confidence the RPV algorithm is able to

Table 4.2: Improved RPV Algorithm Results

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Shift Reg	delay	1080	14	172	378	2.86	9.04
Shift Reg	fall_time	1080	14	276	1033	1.05	122.66
Shift Reg	rise_time	1080	14	183	959	1.13	98.93
Buffer Chain	Tf4_5	1800	20	64	85	21.18	2.40
Buffer Chain	Tr4_5	1800	20	60	87	20.69	2.43
Buffer Chain	avg_slew	1800	20	57	82	21.95	2.17
Buffer Chain	avgdly4_5	1800	20	60	77	23.38	2.13
Buffer Chain	fslew	1800	20	86	127	14.17	3.17
Buffer Chain	rslew	1800	20	52	86	20.93	2.36
Bit Cell	blwm	120	10	16	28	4.29	0.31
Bit Cell	blwm_mv	120	10	16	24	5.00	0.30
MUX	qfinal	120	16	10	36	3.33	0.23
MUX	qinit	120	16	2	110	1.09	1.80
MUX	qsgn	120	16	1	18	6.67	0.10
MUX	qtran	120	16	58	58	2.07	0.44
MUX	qtran0	120	16	58	58	2.07	0.42
MUX	setup_time	120	16	3	48	2.50	0.69
MUX	t_ref	120	16	3	33	3.64	0.48
Charge Pump 1	boostcr	216	16	24	35	6.17	0.38
Charge Pump 1	eq_error	216	16	9	35	6.17	0.29
Charge Pump 1	holdcrd	216	16	9	42	5.14	0.39
Charge Pump 1	holdcru	216	16	9	41	5.27	0.37
Charge Pump 1	ovdrive	216	16	12	56	3.86	0.60
Charge Pump 2	boostcr	324	16	23	65	4.98	0.44
Charge Pump 2	eq_error	324	16	39	54	6.00	0.35
Charge Pump 2	holdcrd	324	16	13	43	7.53	0.45
Charge Pump 2	holdcru	324	16	13	40	8.10	0.45
Charge Pump 2	ovdrive	324	16	34	63	5.14	0.52
Sense Amp	SAspeed	120	20	11	27	4.44	0.35
Sense Amp	glitch_senout	120	20	6	47	2.55	0.61
Sense Amp	maxout	120	20	9	49	2.45	0.39
Sense Amp	offset	120	20	8	34	3.53	0.49
Sense Amp	rslt	120	20	10	31	3.87	0.28
Sense Amp	sen_dip	120	20	9	53	2.26	0.47
Sense Amp	sen_dip_pctg	120	20	2	118	1.02	1.32
Bias Gen	bgr_m51_v145	120	10	2	43	2.79	0.51
Bias Gen	bgr_m51_v150	120	10	2	40	3.00	0.31
Bias Gen	bgr_m51_v155	120	10	2	35	3.43	0.29
Bias Gen	bgr_m51_v180	120	10	2	36	3.33	0.29
Bias Gen	bgr_m51_v195	120	10	2	35	3.43	0.28
Bias Gen	bgr_m51_v25	120	10	2	42	2.86	0.31
Bias Gen	bgr_m51_v27	120	10	2	35	3.43	0.30
Bias Gen	bgr_m51_v30	120	10	2	36	3.33	0.29
Bias Gen	bgr_m51_v33	120	10	2	36	3.33	0.30
Bias Gen	bgr_m51_v36	120	10	2	35	3.43	0.29
Op Amp	dc_gain	120	12	2	43	2.79	0.30

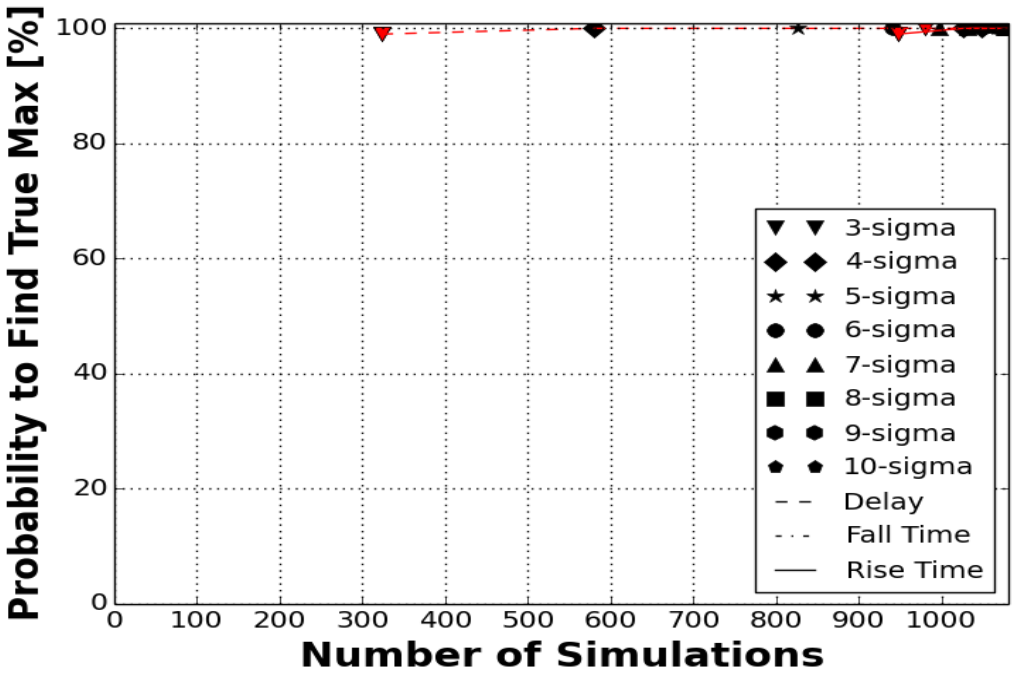


Figure 4.13: Measure of the improved RPV algorithm's reliability using 100 randomized trials and averaging the results.

reach 100% probability of finding of finding the true maximum for the *fall time* output and 99% probability for the *delay* and *rise time* outputs. From the average of the 100 trials for each output, the RPV algorithm is able to reach 3-*Sigma* confidence level after 323 simulations for the *delay* output, 980 simulations for the *fall time* output, and 947 simulations for the *rise time* output.

We note that with the introduction of a standard deviation correction factor, β, that it takes the RPV algorithm more simulated PVT corners to reach the 3-*Sigma* confidence level. For this circuit's three outputs we do see a decreased efficiency (i.e., a decreased speed up). However, the reported probability of finding the true maximum at the 3-*Sigma* confidence level is approximately the same as the ideal statistical value. We see a significant increase in observed reliability of the improved RPV algorithm as compared to the initial algorithm.

4.5.3 Improved Algorithm Results from the Alternative Dataset

An alternative dataset has been provided to us to observe the performance of our final version of the single-output RPV algorithm. Testing the RPV algorithm with this alternative dataset is done to confirm that our RPV algorithm was not designed with an inherent bias towards one dataset. The alternative dataset has 10 circuits and 68 unique outputs.

The RPV algorithm will test each circuit output individually. It will record the total

number of PVT corners for a given circuit, the initial training set size, how many PVT corners are required to be simulated to reach the 3-*Sigma* confidence, the speed-up of the algorithm (measured by total number of corners divided by number of corner required to reach the 3-*Sigma* stopping condition), and also the computational time taken (not including time taken for corner simulation).

From Table 4.3, there are no cases in which the RPV algorithm fails to find a circuit's output maximum. The best speed up observed was 33.09× and the worst was 1.00× (i.e., no speed up). The average speed up was found to be 5.45× with a standard deviation of 6.30. The results from the first dataset reported an average speed up of 5.91× with a standard deviation of 5.92. Table 4.2 also reported the best speed up found was 23.38× and the worst was 1.02× for the first dataset. The results from the new dataset show that the RPV algorithm performs consistently among datasets and does not appear to be biased to only one dataset.

Table 4.3: Improved RPV Algorithm Results for the Alternative Dataset

Circuit	Output	Total Corners	Training Set Size	Simulation Max Found	Simulation 3 Sigma Reached	Speed Up	Time Taken (sec)
Buffer Reference	V1K(DB).Bufref_vss.sp	225	10	18	78	2.88	0.62
Buffer Reference	END1	225	10	7	76	2.96	0.51
Buffer Reference	END2	225	10	7	76	2.96	0.50
Buffer Reference	ERR1	225	10	150	225	1.00	2.76
Buffer Reference	ERRMAX1	225	10	67	89	2.53	0.72
Buffer Reference	ERRMAX2	225	10	3	82	2.74	0.52
Buffer Reference	ERRMIN1	225	10	7	71	3.17	0.38
Buffer Reference	ERRMIN2	225	10	7	73	3.08	0.37
Buffer Reference	IVDD	225	10	18	36	6.25	0.30
Buffer Reference	V1K(DB).Bufref_vdd	225	10	16	26	8.65	0.17
CP Default	bw	225	10	26	65	3.46	0.47
CP Default	gain	225	10	6	40	5.63	0.36
CP Default	fall_time	225	10	5	225	1.00	2.53
CP Default	rise_time	225	10	6	225	1.00	2.87
CP Default	voh	225	10	4	46	4.89	0.31
CP Default	vol	225	10	22	33	6.82	0.33
Current Mirror	dv_degree	315	10	56	259	1.22	3.47
Current Mirror	vmax	315	10	3	37	8.51	0.38
Current Mirror	vmin	315	10	3	37	8.51	0.37
D Flip-Flop	MaxVout	405	10	18	82	4.94	0.47
D Flip-Flop	i_vdd	405	10	8	195	2.08	1.50
D Flip-Flop	pwr	405	10	9	58	6.98	0.34
D Flip-Flop	q_clk_delay_setup	405	10	7	116	3.49	0.73
D Flip-Flop	setup_time	405	10	18	43	9.42	0.27
GMC	ATTEN	1125	11	534	1101	1.02	113.08
GMC	IL	1125	11	5	1122	1.00	132.47
GMC	max_vout	1125	11	46	506	2.22	15.84
GMC	min_vout	1125	11	465	1120	1.00	140.44
GMC	slew	1125	11	386	1078	1.04	126.67
GMC	v1	1125	11	243	828	1.36	54.01
GMC	v2	1125	11	68	1122	1.00	133.43
GMC	vout_pp	1125	11	40	298	3.78	5.32
LSTB	bandwidth	1125	11	36	77	14.61	0.97
LSTB	dc_gain	1125	11	21	78	14.42	1.05
LSTB	gain_margin	1125	11	180	906	1.24	75.82
LSTB	gbw	1125	11	7	55	20.45	0.63
LSTB	idc	1125	11	7	34	33.09	0.46
LSTB	integ1	1125	11	33	85	13.24	1.09
LSTB	phase_margin	1125	11	9	146	7.71	2.62
LSTB	phase_margin1	1125	11	27	47	23.94	0.72
LSTB	test_loop_gain_at_minifreq	1125	11	28	107	10.51	1.24
LSTB	test_phase_margin	1125	11	27	49	22.96	0.71
LSTB	test_phase_margin_freq	1125	11	4	74	15.20	0.90
NDL	idiss	221	10	3	49	4.51	0.36
NDL	out_0_rise	221	10	16	42	5.26	0.31
NDL	out_1_rise	221	10	5	38	5.82	0.28
OP Amp TB	gain	1125	11	455	1020	1.10	90.79
OP Amp TB	overshoot	1125	11	664	1125	1.00	160.37
OP Amp TB	settling_time	1125	11	586	1125	1.00	149.41
OP Amp TB	slew_rate	1125	11	7	422	2.67	9.14
OP Amp TB	v1	1125	11	224	1022	1.10	94.62
OP Amp TB	v2	1125	11	66	538	2.09	16.37
OP Amp TB	vavg	1125	11	430	1124	1.00	133.85
OP Amp TB	vmax	1125	11	46	1107	1.02	120.06
SDF	leakage_power_VBP	405	10	279	405	1.00	9.39
SDF	leakage_power_VDD	405	10	69	253	1.60	3.08
SDF	leakage_power_vbn_i	405	10	23	195	2.08	1.72
SDF	leakage_power_vbp_i	405	10	19	211	1.92	2.26
SDF	leakage_power_vdd_i	405	10	275	405	1.00	8.89
SDF	leakage_power_vss_i	405	10	73	129	3.14	1.05
SDF	maxvout	405	10	18	109	3.72	1.16
SDF	pwr	405	10	19	111	3.65	0.71
SDF	pwr_i	405	10	7	150	2.70	1.02
SDF	q_ck_delay	405	10	7	208	1.95	1.78
SDF	q_slew	405	10	7	282	1.44	3.62
SDF	setup_time	405	10	7	40	10.13	0.19
SDF	total_leakage_power	405	10	15	215	1.88	2.22
Sense Amp	rise_time	405	10	7	29	13.97	0.20

Chapter 5

Extending the Algorithm to Multiple Outputs

When a designer is using SPICE-like circuit level simulation tools, such as the Cadence Virtuoso Spectre Circuit Simulator, it is possible to obtain multiple outputs at once for very little extra cost. Since circuits often have multiple outputs that designers need to verify, as is clear from Table 3.1, it should be an advantage to coordinate the search for the worst-case PVT corners for all outputs at the same time. This way we can maximize the use of all the information acquired by each PVT corner simulation. The verification problem then becomes multiple simultaneous global optimization problems. Thus far we have investigated methods and heuristics to develop a reliable, efficient and fast algorithm, RPV, that is able to determine the worst-case PVT corner of a single output for a given circuit. In this chapter we look at different methods that extend the RPV algorithm to consider finding the worst-case PVT corner for every output of a given multiple-output circuit. Primarily we will consider concurrent search and sequential search methods.

Concurrent search methods create one model for each output and then attempt to converge all models simultaneously. Sequential search methods create models for each output and iteratively attempt to converge one output model at a time. The sequential search method considers only one output model at a time and does not consider the next output model until the present model has reached the termination criterion.

5.1 Concurrent Search Methods and Results

At this point, our RPV algorithm is tailored to single-output problems. To extend it to consider multiple outputs simultaneously we will modify the RPV algorithm to create a GPM for each of the circuit outputs. While considering multiple outputs simultaneously we need to decide how to choose the set of next PVT corners to simulate. Choosing the entire convex hull of one output's model was reasonable for a single output problem, but it may be too computationally expensive if other output models need to be considered. We thus decided to investigate different rules for choosing the next set of corners to simulate.

5.1.1 Concurrent Search Methods

The selection rule for choosing the next set of corners to simulate can have many different approaches. We will examine three types of approaches. One approach is to evenly distribute the search effort over all output models by having a selection rule that picks corners evenly from every output model. A second approach is to focus the search effort on one model by having a selection rule that picks corners from only one output model of interest at a time. A third approach is to asymmetrically distribute the search effort among all output models by having a selection rule that picks more corners from output models of high interest and fewer corners from output models of low interest, where the level of interest is determined using a heuristic.

The asymmetric approach is a blend of the first two approaches: it distributes computational effort over all models, but gives priority to output models with higher levels of interest. The asymmetric distribution approach we will use is based on a geometric series. The output model with the highest interest will receive one half of the computational effort, the output with the second highest interest will receive one quarter of the computational effort, and so on (i.e., computational effort will be distributed as $[\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...]$ in the order of highest interest to lowest interest). The choice of a geometric distribution allows the algorithm to place the majority of computational effort on the output model with the highest level of interest while also allowing some computational effort to be placed on the other models as well.

There are many possible ways of determining which output model should of the highest interest. For example we could gauge the interest level by determining which output model is furthest from convergence (i.e., furthest from termination). We measure how far an output model is to convergence by counting how many unsimulated corners remain above the termination criterion threshold (e.g., shown in Figure 3.7 as yellow dots). Corners that lay above the termination criterion threshold are considered to be potential maxima. The output model with the most potential maxima is considered to have the highest interest level. Therefore, the approaches that focus their search effort will focus on output models that have the largest number of potential maxima (i.e., output models with high levels of interest).

We will examine seven unique concurrent search methods that use different next corner selection methods. All of the concurrent search methods will have the same initial training corner selection method and termination criterion as the improved RPV algorithm. The only difference between the seven concurrent search methods will be the next corner selection rule. The seven unique concurrent search methods are defined below.

Concurrent Method 1

At each iteration the algorithm will focus all of the computational effort on the output model which has the highest interest level.

The corners to be simulated next will be the corners that lie on the convex hull of the

output model with the highest interest.

Concurrent Method 2

At each iteration the algorithm will select m (the number of outputs) PVT corners and evenly distribute this computational effort among all of the non-converged output models.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 3

At each iteration the algorithm will select m PVT corners from only the output model which has the highest interest level.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 4

At each iteration the algorithm will select m PVT corners using the geometric-asymmetric distribution among the non-converged output models. The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 5

At each iteration the algorithm will select $2m$ PVT corners and evenly distribute this computational effort among the non-converged output models.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 6

At each iteration the algorithm will select $2m$ PVT corners from only the output model which has the highest interest level.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 7

At each iteration the algorithm will select $2m$ PVT corners using the geometric-asymmetric distribution among the non-converged output models.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

5.1.2 Concurrent Search Methods: Data Set Results

In this section we will investigate how well the seven concurrent search methods perform on the nine-circuit dataset described in Chapter 3. The multiple-output search algorithm will consider the maxima of each circuit output and report if any output maximas are not

Table 5.1: Speed Up of Concurrent Search Methods for the Nine-Circuit Dataset

Circuit	Method 1	Method 2	Method 3	Method 4	Method 5	Method 6	Method 7
Shift Register	1.01	1.01	1.02	1.04	1.05	1.01	1.04
Buffer Chain	2.97	9.0	3.28	3.25	8.49	3.08	6.62
Bit cell	5.0	6.67	6.67	6.67	5.45	5.45	5.45
MUX	1.00	1.05	1.00	1.05	1.00	1.00	1.05
Charge Pump 1	3.09	3.27	2.67	3.54	2.84	2.51	2.84
Sense Amp 1	1.14	1.02	1.00	1.02	1.00	1.00	1.02
Charge Pump 2	3.77	3.21	3.38	3.56	3.38	3.38	3.06
OP Amp	3.24	3.87	3.64	3.33	2.61	2.73	3.53
Bias Gen	3.16	2.0	2.4	2.0	2.4	1.33	1.71
Average	2.71	**3.46**	2.78	2.83	3.14	2.39	2.92

found by the time of convergence. We will report the observed speed up (measured by the total number of corners divided by the number of corners required to reach the 3-Sigma stopping condition) of the algorithm for each circuit and also calculate an average speed up over all circuits.

Table 5.1 reports the results of the concurrent search methods for the nine-circuit dataset. We observe that all seven of the concurrent search methods were able to find the global maximum of every output for all circuits in the dataset. We observe a range of average speed ups from 2.39× to 3.46× faster than a conventional full-factorial verification approach, which simulates all possible PVT corners.

The observed speed ups seem to be lower than some of observed speed ups measured with the single-output search method as reported in Table 4.2. This is likely due to the fact that the multiple-output method must converge multiple output models instead of only one model. The single-output problem only requires enough corner simulations to converge one output's model. By contrast, the multiple-output problem requires sufficient corner simulations to converge all of the output models of the given circuit.

5.1.3 Concurrent Search Methods: Results from the Alternative Dataset

In this section we will investigate how well the seven concurrent search methods perform on the alternative ten-circuit dataset that was introduced in Chapter 4. The multiple-output search algorithm will consider the maxima of each circuit output and report if any output maximums are not found by the time of convergence. We will report the observed speed up (measured by total number of corners divided by number of corner required to reach the 3-Sigma stopping condition) of the algorithm for each circuit and also calculate an average speed up over all circuits.

Table 5.2 reports the results of the concurrent search methods for the alternative ten-circuit dataset. All seven of the concurrent search methods were able to find the global maximum of every output for all circuits in the dataset. The average speed ups range

Table 5.2: Speed Up of Concurrent Search Methods for the Ten-Circuit Dataset

Circuit	Method 1	Method 2	Method 3	Method 4	Method 5	Method 6	Method 7
Buffer Reference	1.00	1.02	1.02	1.02	1.07	1.07	1.07
CP Default	1.00	1.00	1.02	1.00	1.05	1.05	1.05
Current Mirror	1.88	1.77	2.17	1.04	1.08	1.90	1.13
D Flip-Flop	1.59	1.59	1.09	1.07	1.09	1.01	1.31
GMC	1.00	1.00	1.00	1.00	1.00	1.00	1.00
LSTB	2.98	1.28	2.92	1.38	1.18	1.48	1.20
NDL	4.51	5.97	3.62	4.25	5.53	3.45	5.53
OP Amp TB	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SDF	1.00	1.01	1.01	1.00	1.01	1.01	1.00
Sense Amp	13.97	20.25	20.25	20.25	16.88	16.88	16.88
Average	2.99	**3.59**	3.51	3.30	3.09	2.99	3.02

from 2.99× to 3.59× faster than a full-factorial verification approach. However, these mean values may misrepresent the performance of the concurrent methods. For example, Method 2 reports the highest average speed up of 3.59×, even though, for eight of the ten circuits the speed up is less than 2.00×.

Many of the concurrent search methods report little to no speed up for the ten-circuit dataset in Table 5.2. These unexpectedly low speed ups may be explained from the results of the single-output RPV algorithm results reported in Table 4.3. Table 4.3 shows the individual speed up observed for each output of the ten circuits in the alternative dataset. We note that many circuits report large speed ups for some outputs while also having a one or more outputs that report little or no speed up. For example the $CP\ Default$ circuit has six outputs. Four of the outputs report a speed up between 3.46× to 6.82×. Yet, two of the outputs seem to be difficult for the RPV algorithm and do not show any speed up (1.0×). We also observe that the concurrent multiple-output Method 2 reports no speed up for the $CP\ Default$ circuit.

It would seem that the most difficult output of a given circuit will determine the performance of the multiple-output search method. Therefore, the best speed up that any multiple-output method can yield is determined by the worst speed up of the single-output method for a particular circuit.

5.1.4 Revised Concurrent Search Methods and Results from the Alternative Dataset

In the previous subsection we defined outputs with the highest level of interest as the models that were furthest from convergence. This may have influenced the algorithm to primarily focus on the most difficult outputs of a given circuit. In order to confirm that speed up of the multiple-output method is determined by the worst speed up of the single-output method for a particular circuit, we will investigate the effects of assigning a high level of interest to the output model that is closest to convergence.

In this subsection we will investigate five revised concurrent search methods. These new search methods, which are listed below, are identical to the previously proposed concurrent search methods with one exception. The exception is that we define outputs with the highest level of interest as the output models that are closest to convergence. The goal is to see if how we define levels of interest has an overall effect on the performance of the multiple-output search method.

Concurrent Method 8

At each iteration the algorithm will focus all of the computational effort on only the output model which has the highest interest level.

The corners to be simulated will be the corners that lie on the convex hull of the output model with the highest interest.

Concurrent Method 9

At each iteration the algorithm will select m PVT corners from only the output model which has the highest interest level.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 10

At each iteration the algorithm will select m PVT corners using an asymmetric distribution among non-converged output models. The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 11

At each iteration the algorithm will select $2m$ PVT corners from only the output model which has the highest interest level.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Concurrent Method 12

At each iteration the algorithm will select $2m$ PVT corners using an asymmetric distribution among non-converged output models.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

These five revised concurrent search methods are examined using the ten-circuit dataset. We report the observed speed up for each circuit and also each method's calculated average over all circuits in Table 5.3.

Revised concurrent Methods 8, 9, 10, 11, and 12 can be directly compared to their counterparts, concurrent Methods 1, 3, 4, 6, and 7, respectively. The variations between the revised method average speed ups and the original method average speed ups range

Table 5.3: Speed Up of Revised Concurrent Search Methods for the Ten-Circuit Dataset

Circuit	Method 8	Method 9	Method 10	Method 11	Method 12
Buffer Reference	1.00	1.02	1.02	1.07	1.07
CP Default	1.00	1.02	1.02	1.05	1.05
Current Mirror	3.94	1.23	1.06	1.02	1.15
D-Flip Flop	1.10	1.40	1.05	1.04	1.01
GMC	1.00	1.00	1.00	1.00	1.00
LSTB	2.46	1.22	1.25	1.20	1.18
NDL	3.95	3.62	4.51	3.45	4.25
OP Amp TB	1.00	1.00	1.00	1.01	1.01
SDF	1.00	1.01	1.01	1.01	1.01
Sense Amp	13.97	20.25	20.25	16.88	16.88
Average	3.04	3.28	**3.32**	2.87	2.86

from 0.05 to 0.23. The largest difference is found between Method 3, which reported an average speed up of 3.51×, and Method 9, which reported 3.28×. The speed up difference between these two counter parts is 0.23.

Focusing the computational effort on models that are closest to convergence as opposed to those models that are furthest from convergence does not seem to have a significant effect on the multiple-output problem. This result agrees with the previous observation that the best speed up that the multiple-output method can yield is determined by the worst speed up of the single-output method for a particular circuit.

5.2 Sequential Search Method Experiments and Results

In this section we will investigate sequential search methods. This is a straightforward way to extend the single-output RPV algorithm to multiple outputs. Sequential search methods iteratively use a single-output algorithm to converge one output model at a time and then repeat the process until all output models have converged.

5.2.1 Sequential Search Methods

The sequential search methods will consider only one output model at a time and will not consider another until the current output model has converged. To select which output model should be considered next we will select the output model with the highest level of interest. As described in the previous section, the output model with the highest level of interest is the model with the most unsimulated corners above the termination criterion threshold (i.e., the model furthest from convergence). After the initial training set is selected a GPM will be created for each output. A modified version of the single-output RPV algorithm will search for the maximum of the model with the highest level of interest. Once the current model of interest has reached convergence (i.e., reached the 3-Sigma termination criterion) GPM will be created for all non-converged circuit outputs and the cycle will repeat until all output models have converged.

We will examine three unique sequential search methods that each use different next corner selection approaches. All of the sequential search methods will have the same initial training corner selection method and termination criterion as the improved RPV algorithm. The only difference between the three sequential search methods will be the next corner selection rule. The three unique concurrent search methods are defined below.

Sequential Method 1

At each iteration the algorithm will select the corners on the convex hull of the current output model until that model has determined convergence.

Sequential Method 2

At each iteration the algorithm will select m (the number of outputs) PVT corners of the current output model until that model has determined convergence.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Sequential Method 3

At each iteration the algorithm will select $2m$ (the number of outputs) PVT corners of the current output model until that model has determined convergence.

The corners to be simulated will be selected using the rule $c_{next} = argmax(\hat{Y} + 6 \times stddev)$.

Table 5.4: Speed Up of Sequential Search Methods for the Nine-Circuit Dataset

Circuit	Method 1	Method 2	Method 3
Shift Register	1.03	1.01	1.12
Buffer Chain	10.69	3.18	2.28
Bit cell	4.29	5.45	4.62
MUX	1.01	1.05	1.05
Charge Pump 1	2.63	2.84	2.25
Sense Amp 1	1.02	1.15	1.02
Charge Pump 2	3.31	3.56	3.06
OP Amp	1.02	3.33	3.00
Bias Gen	3.08	2.00	1.71
Average	**3.12**	2.62	2.23

5.2.2 Sequential Search Methods: Data Set Results

In this section we will investigate how well the three sequential search methods perform on the nine-circuit dataset described in Chapter 3. The multiple-output search algorithm will consider the maxima of each circuit output and then report if any output maximums are not found by the time of convergence. We will report the observed speed up (measured by the total number of corners divided by the number of corners required to reach the 3-Sigma stopping condition) of the algorithm for each circuit and also calculate an average speed up over all circuits.

All three of the sequential search methods were able to find the global maximum of every output for all nine circuits in the dataset. From Table 5.4 we observe a range of average speed ups from 2.23× to 3.12× faster than a conventional full factorial verification method. Sequential Method 1 reported the largest average speed up of 3.12×, whereas the largest average speed up for concurrent search methods reported 3.46× for the nine-circuit dataset.

5.2.3 Sequential Search Methods: Results from the Alternative Dataset

In this section we will investigate how well the three sequential search methods perform on the alternative ten-circuit dataset that was introduced in Chapter 4. The multiple-output search algorithm will consider the maxima of each circuit output and report if any output maximums are not found by the time of convergence. We will report the observed speed up (measured by total number of corners divided by number of corner required to reach the 3-Sigma stopping condition) of the algorithm for each circuit and also calculate an average speed up over all circuits.

All three of the sequential search methods were able to find the global maximum of every output for all ten circuits in the dataset. Table 5.5 shows the observed speed ups

Table 5.5: Speed Up of Sequential Search Methods for the Ten-Circuit Dataset

Circuit	Method 1	Method 2	Method 3
Buffer Reference	1.00	1.02	1.07
CP Default	1.00	1.02	1.05
Current Mirror	1.38	1.09	1.15
D-Flip Flop	1.22	1.09	1.35
GMC	1.00	1.00	1.01
LSTB	1.28	1.28	1.15
NDL	5.02	4.02	3.81
OP Amp TB	1.00	1.00	1.01
SDF	1.00	1.01	1.01
Sense Amp	13.97	20.25	16.88
Average	2.79	**3.28**	2.95

for the ten-circuit dataset. We observed a range of average speed ups from 2.79× to 3.28× faster than a conventional full factorial verification method. The largest average speed up reported by the sequential methods for the ten-circuit data set was 3.28×, which is lower than the largest average speed up reported by the concurrent search methods of 3.59×.

5.3 Summary of Multiple-Output Search Methods

At the beginning of this chapter we noted that many circuits have multiple outputs and that measurements for all of these outputs can be taken during a single PVT corner simulation. In an attempt to maximize the use of the information generated during each simulation we proposed to view the verification of all outputs as a multiple simultaneous global optimization problem. We extended the RPV algorithm to consider multiple circuit's outputs using different approaches. The concurrent method approach monitors all output models and attempts to converge all models simultaneously. The sequential search method iteratively attempts to converge one output model at a time.

We explored three different ways of distributing computational effort for the concurrent search methods. The first was to evenly distribute computational effort among all non-converged output models. The second focused all computational effort on a single output model. The last used an asymmetric distribution on all non-converged output models. Altogether we implemented seven different concurrent search methods.

Since sequential methods only focus on one output model at a time, there was no need to distribute computational effort among the multiple models. However, we did explore applying three different amounts of computational effort during the next corner selection step. In the first method we applied a variable amount of computational effort that is determined by the number of PVT corners that fall on the model's convex hull. In the

second method there is always a selection of m (the number of outputs) corners to be simulated. The third method also has a fixed number of corners to be simulated, that is $2m$.

We tested the seven concurrent search methods and three sequential search methods on both of our datasets. For the nine-circuit dataset we found that the average speed ups for the concurrent methods ranged from 2.39× to 3.46× and that the average speed ups for the sequential methods ranged from 2.23× to 3.12×. For the ten-circuit dataset, average speed ups for the concurrent methods ranged from 2.99× to 3.59× and the average speed ups for the sequential methods ranged from 2.79× to 3.28×. For both datasets the method that reported the largest average speed up was the concurrent Method 2. Concurrent Method 2 selects m PVT corners each iteration and evenly distributes this computational effort among all non-converged output models. Overall, concurrent search Method 2 provided larger speed ups over all of the sequential methods. This is likely due to the coordinated search strategy that is used. At every iteration concurrent Method 2 always places some computational effort on any non-converged output model.

We also observed from both the nine-circuit and ten-circuit datasets that some circuits reported little to no speed up. By reviewing the results of the single-output RPV method results, we noticed that circuits reported high speed ups for certain outputs, but much lower speed ups for other outputs. We found that these difficult outputs dominate the overall difficulty of the circuit. The best speed up that the multiple-output search method can yield is limited by the worst speed up of the single-output method for the most difficult output(s) of the given circuit.

Chapter 6

Conclusions and Future Work

In this thesis the problem of integrated circuit verification in the presence of PVT variations was investigated. PVT variations can adversely affect a circuit's performance and behavior, which can cause problems in operation and greatly reduce the overall yield [8]. Circuit designs in modern process technologies can have several thousand PVT corners that need to be verified to ensure that the circuit operates within specifications over all of these conditions [9]. The full-factorial PVT verification method, which simply simulates all possible PVT corners, can be very time consuming and often not feasible to carry out [3]. Other approaches, such as the designer best guess approach, are found to reduce the number of required SPICE level simulations, but may not always find the worst-case PVT corner and may report overly optimistic results [3]. We treated the PVT verification problem as a mathematical optimization problem. Our goal was to develop an algorithm that could reliably locate a global maxima while requiring minimal PVT corner simulations and also be quick to compute. We based our algorithm on the iterative optimization algorithm known as the EGO algorithm, as discussed in Section 2.1. In Chapter 3 we introduced the framework and some initial results for our RPV algorithm. The RPV algorithm employs a Gaussian Process machine-learning model to model a given circuit's output. Heuristics are used to exploit the information provided by the GPM. Information from the GPM is used to determine if the global optimum has been found and to what degree of confidence.

In Chapter 4 we focused on improving the RPV algorithm to make it reliable, efficient, and fast. Reliability is being reasonably confident that upon the algorithm's termination the true worst case PVT corner has been found. The primary way to ensure that our algorithm is reliable is to have a reliable termination criterion. The RPV's termination criterion is based on *Simga* confidence levels that are reported by the GPM's standard deviation estimation. We found that the GPM's estimation of sigma values has a measurable degree of inaccuracy. In Section 4.4 we introduced a standard deviation correction factor, β , that compensates for error in the GPM's overly optimistic sigma estimation. In Section 4.5.2 we thoroughly examined the reliability of the improved termination criterion using a randomized initial training set experiment on the three unique outputs of the $Shift\ Register$

circuit. We observed that when 3-$Sigma$ confidence is declared, the experiment reports 100% probability of finding the global maximum for the $fall\ time$ output and 99% for the *delay* and *rise time* outputs, as shown in Figure 4.13. The GPM's estimated $3-Sigma$ confidence seems to be very close to the expected ideal 3-$Sigma$ confidence levels. The RPV algorithm was tested on both the nine-circuit and ten-circuit datasets over a total of 114 unique outputs and was able to successfully find the global maximum for every output.

For our PVT verification method to be efficient it needs to minimize the number of required SPICE-level PVT corner simulations. We measure efficiency as speed up over the full factorial verification method. Speed up is calculated by the total number of PVT corners divided by the number of PVT corners required to reach the termination criterion. In Section 4.5.1 the RPV algorithm was tested on the nine-circuit dataset. The average speed up found for all 46 outputs was 5.91× and the largest speed up observed was 23.38×. In Section 4.5.3 the RPV algorithm was tested on the ten-circuit dataset. The average speed up found for all 68 outputs was 5.45× and the largest speed up observed was 33.09×.

Computational time is also an important consideration for our verification algorithm. Significant attention to detail was made while creating the RPV algorithm. Decisions were made to reduce computational complexity where ever possible. As an example, we used a linear approximation to calculate the standard deviation correction factor, β, instead of performing 10-fold cross validation at every iteration. Another example would be to select multiple PVT corners to simulate each iteration (e.g., select all corners on the GPM's convex hull) in order to reduce the total number of GPMs built and trained over the course of the algorithm. In Section 4.5 the time from the beginning of the verification process to the time of termination (neglecting PVT simulation time) was monitored for every output of the nine-circuit and ten-circuit dataset. We observed that the RPV's compute time ranged from 0.10 s to 160.37 s on a 3.2-GHz Intel i5-3470 processor. Modern circuit designs can take on the order of minutes for a single PVT corner simulation, such as the VCO in a 28 nm process technology that reports an approximate 70 s per PVT corner simulation [3]. The computational time required to execute the RPV algorithm is minimal compared to the computational time required for a SPICE level PVT corner simulation. The RPV algorithm is relatively fast compared to corner simulation times.

In Chapter 5 we noted that many circuits have multiple outputs and that measurements for all outputs can be taken during a single PVT corner simulation. In an attempt to maximize the use of information generated during each simulation, we extended the RPV algorithm to consider all of a given circuit's outputs simultaneously. The verification problem was then viewed as multiple simultaneous global optimization problems. We investigated many different concurrent and sequential search methods. As stated in Section 5.3, we found that concurrent search Method 2 provided the best observed average speed ups for both the nine-circuit and ten-circuit datasets with 3.46× and 3.59×, respectively. Each iteration of the algorithm the concurrent search Method 2 would select m (the number

of outputs) PVT corners to be simulated and evenly distribute this computational effort among non-converged output models. We also noted that the most difficult output of a given circuit will determine the performance of the multiple-output search method. Therefore, the best speed up that the multiple-output method can yield is determined by the worst speed up of the single-output method for a particular circuit.

6.1 Future Work

We have noted that some outputs, particularly digital outputs, are difficult for the GPM to model. It would be possible to add a new step to the algorithm that monitors if the output function has discrete regions (e.g., digital outputs). This could be done using a discretization algorithm to determine if identifiable clusters (i.e., regions) exist in the output space (e.g., CAIM discretization [47]). If unique clusters are determined to exist, the program could then use a clustering algorithm, such as K-means clustering [46], on known data samples (the number of clusters, k, as determined by the discretization algorithm). The K-means clustering model could then be used to determine expected regions of interest (i.e., predict where regions of possible maxima) exist among unsimulated PVT corners. Then the RPV algorithm could be applied separately to only the regions of interest.

The dimensionality of the GPM may become an issue as the number of PVT parameters increase. Medium-scale problems with 20-50 parameter variables can be computationally expensive for the GPM to compute [3, 33]. Liu et al., [33], proposed using dimension reduction techniques, such as Sammon mapping [43], to lower the dimensionality of the input space of the GPM. The use of dimension reduction techniques for the GPM in the RPV algorithm may be necessary when number of PVT parameters becomes larger than 20. As dimensionality increases it may also be beneficial to explore modeling techniques other than the GPM, such as multilayer perceptron neural networks [48].

Bibliography

[1] K. Kuhn, "Moore's law past 32nm: Future challenges in device scaling," in *Computational Electronics, 2009. IWCE '09. 13th International Workshop on*, May 2009, pp. 1–6.

[2] K. Kuhn, M. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. Ma, A. Maheshwari, and S. Mudanai, "Process technology variation," *Electron Devices, IEEE Transactions on*, vol. 58, no. 8, pp. 2197–2208, Aug 2011.

[3] T. McConaghy, K. Breen, J. Dyck, and A. Gupta, *Variation-Aware Design of Custom Integrated Circuits: A Hands-on Field Guide*. New York, NY : Springer, 2013.

[4] M. J. Sasena, "Flexability and efficiency enhancements for constrained global optimization with kriging approximations," Ph.D. dissertation, University of Michigan, 2002.

[5] T. W. Simpson, "A concept exploration method for product family design," Ph.D. dissertation, Georgia Institute of Technology, 1998.

[6] C. Auth, "45nm high-k + metal gate strain-enhanced cmos transistors," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, Sept 2008, pp. 379–386.

[7] K. Mistry, M. Armstrong, C. Auth, S. Cea, T. Coan, T. Ghani, T. Hoffmann, A. Murthy, J. Sandford, R. Shaheed, K. Zawadzki, K. Zhang, S. Thompson, and M. Bohr, "Delaying forever: Uniaxial strained silicon transistors in a 90nm cmos technology," in *VLSI Technology, 2004. Digest of Technical Papers. 2004 Symposium on*, June 2004, pp. 50–51.

[8] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2011. [Online]. Available: https://books.google.ca/books?id=sv8OQgAACAAJ

[9] D. De Jonghe, E. Maricau, G. Gielen, T. McConaghy, B. Tasic, and H. Stratigopoulos, "Advances in variation-aware modeling, verification, and testing of analog ics," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 1615–1620.

[10] D. Jones, M. Schonlau, and W. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998. [Online]. Available: http://dx.doi.org/10.1023/A%3A1008306431147

[11] E. Alpaydin, *Introduction to Machine Learning, Second Edition*. MIT Press, 2010.

[12] G. R. Iversen, *Bayesian Statistical Inference*. SAGE Publications, Inc., 1984.

[13] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptative computation and machine learning series. University Press Group Limited, 2006. [Online]. Available: https://books.google.ca/books?id=vWtwQgAACAAJ

[14] M. Sengupta, S. Saxena, L. Daldoss, G. Kramer, S. Minehane, and J. Cheng, "Application-specific worst case corners using response surfaces and statistical models," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 9, pp. 1372–1380, Sept 2005.

[15] A. Nardi, A. Neviani, E. Zanoni, and C. Guardiani, "Impact of unrealistic worst case modeling on the performance of vlsi circuits in deep sub-micron cmos technologies," in *Statistical Metrology, 1998. 3rd International Workshop on*, Jun 1998, pp. 42–45.

[16] A. Strojwas, M. Quarantelli, J. Borel, C. Guardiani, G. Nicollini, G. Crisenza, and B. Franzini, "Manufacturability of low power cmos technology solutions," in *Low Power Electronics and Design, 1996., International Symposium on*, Aug 1996, pp. 225–232.

[17] A. Singhee and R. Rutenbar, "Beyond low-order statistical response surfaces: Latent variable regression for efficient, highly nonlinear fitting," in *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE*, June 2007, pp. 256–261.

[18] M. Hane, T. Ikezawa, and T. Ezaki, "Atomistic 3d process/device simulation considering gate line-edge roughness and poly-si random crystal orientation effects [mosfets]," in *Electron Devices Meeting, 2003. IEDM '03 Technical Digest. IEEE International*, Dec 2003, pp. 9.5.1–9.5.4.

[19] T. Ezaki, T. Ikezawa, and M. Hane, "Investigation of realistic dopant fluctuation induced device characteristics variation for sub-100 nm cmos by using atomistic 3d process/device simulator," in *Electron Devices Meeting, 2002. IEDM '02. International*, Dec 2002, pp. 311–314.

[20] D. Frank, Y. Taur, M. Ieong, and H.-S. Wong, "Monte carlo modeling of threshold variation due to dopant fluctuations," in *VLSI Technology, 1999. Digest of Technical Papers. 1999 Symposium on*, June 1999, pp. 169–170.

[21] H. Chang and S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single pert-like traversal," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, Nov 2003, pp. 621–625.

[22] Z. Wang and S. Director, "An efficient yield optimization method using a two step linear approximation of circuit performance," in *European Design and Test Conference, 1994. EDAC, The European Conference on Design Automation. ETC European Test Conference. EUROASIC, The European Event in ASIC Design, Proceedings.*, Feb 1994, pp. 567–571.

[23] A. Dharchoudhury and S. Kang, "Worst-case analysis and optimization of vlsi circuit performances," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 14, no. 4, pp. 481–492, Apr 1995.

[24] X. Li, J. Le, L. Pileggi, and A. Strojwas, "Projection-based performance modeling for inter/intra-die variations," in *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, Nov 2005, pp. 721–727.

[25] Z. Feng and P. Li, "Performance-oriented statistical parameter reduction of parameterized systems via reduced rank regression," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, Nov 2006, pp. 868–875.

[26] V. R. Burnham, A. J. and J. F. MacGregor, "Frameworks for latent variable multivariate regression," *J. of Chemometrics*, vol. 10, no. 1, pp. 31–45, Jan 1996.

[27] H. Zhang, T.-H. Chen, M.-Y. Ting, and X. Li, "Efficient design-specific worst-case corner extraction for integrated circuits," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 386–389.

[28] G. Debyser and G. Gielen, "Efficient analog circuit synthesis with simultaneous yield and robustness optimization," in *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on*, Nov 1998, pp. 308–311.

[29] F. Schenkel, M. Pronath, S. Zizala, R. Schwencker, H. Graeb, and K. Antreich, "Mismatch analysis and direct yield optimization by spec-wise linearization and feasibility-guided search," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 858–863.

[30] X. Li and H. Liu, "Statistical regression for efficient high-dimensional modeling of analog and mixed-signal performance variations," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, June 2008, pp. 38–43.

[31] A. Mitev, M. Marefat, D. Ma, and J. Wang, "Principle hessian direction-based parameter reduction for interconnect networks with process variation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 9, pp. 1337–1347, Sept 2010.

[32] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[33] B. Liu, Q. Zhang, and G. Gielen, "A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 2, pp. 180–192, April 2014.

[34] J. Knowles, "Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 1, pp. 50–66, Feb 2006.

[35] D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 3, pp. 329–355, June 2010.

[36] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, no. 1, pp. 3–12, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00500-003-0328-5

[37] B. Liu, D. Zhao, P. Reynaert, and G. Gielen, "Synthesis of integrated passive components for high-frequency rf ics based on evolutionary computation and machine learning techniques," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 10, pp. 1458–1468, Oct 2011.

[38] D. J. Allstot, K. Choi, and J. Park, *Parasitic-aware optimization of CMOS RF circuits*. Springer Science & Business Media, 2003.

[39] T. A. Milligan, *Modern antenna design*. John Wiley & Sons, 2005.

[40] M. Emmerich, K. Giannakoglou, and B. Naujoks, "Single- and multiobjective evolutionary optimization assisted by gaussian random field metamodels," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 4, pp. 421–439, Aug 2006.

[41] Z. Zhou, Y. S. Ong, P. Nair, A. Keane, and K. Y. Lum, "Combining global and local surrogate models to accelerate evolutionary optimization," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 37, no. 1, pp. 66–76, Jan 2007.

[42] D. R. Jones, "A taxonomy of global optimization methods based on response surfaces," *Journal of global optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[43] J. Sammon, "A nonlinear mapping for data structure analysis," *Computers, IEEE Transactions on*, vol. C-18, no. 5, pp. 401–409, May 1969.

[44] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[45] A. György and L. Kocsis, "Efficient multi-start strategies for local search algorithms," *J. Artif. Int. Res.*, vol. 41, no. 2, pp. 407–444, May 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2051237.2051250

[46] K. Cios, W. Pedrycz, R. Swiniarski, and L. Kurgan, *Data mining : A Knowledge Discovery Approach.* New York, NY : Springer, 2007.

[47] L. Kurgan and K. Cios, "Caim discretization algorithm," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, no. 2, pp. 145–153, Feb 2004.

[48] S. S. Haykin, *Neural networks and learning machines*. Pearson Education Upper Saddle River, 2009, vol. 3.