Input Space Partitioning and Other Heuristics for Minimizing the Number of Corner Simulations During Design Verification

by

Oleg Oleynikov

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Department of Electrical and Computer Engineering

University of Alberta

Abstract

The continuing reduction in the feature sizes of the latest CMOS (Complementary Metal-Oxide-Semiconductor) technologies allow faster, more compact, and more energy-efficient integrated circuits (ICs). On the downside, the performance of each transistor becomes harder and harder to characterise accurately as smaller transistors are more affected by smaller errors during production process. This tendency makes it more difficult for IC designers to perform an important part of the production flow – design verification (DV) – to ensure their circuits will always behave as required by the specifications, and thus ensure a satisfactory yield of the product. The traditional way of doing DV, corner analysis, requires simulating the circuit for random combinations of expected device parameters (i.e., corners) that affect circuit behaviour. When transistors were larger and more predictable, it was sufficient to simulate them at a smaller number of corners, ranging from five to up to, at worst, several dozen. Now, that number can be greater than one thousand. The corners are tested in a simulator, like Simulation Program with Integrated Circuit Emphasis (SPICE), and it takes significant processing power and time to simulate all the corners as required by DV, significantly extending the time for design iterations and IC production. However, it is not strictly required to simulate all the corners. IC designers really only require the worst-case corner, the corner at which the characteristic is the closest to failing the specifications. If it is possible to locate that worst-case corner before every corner has been simulated, a significant amount of time and resources can be saved. Surrogate function modelling techniques, like Gaussian Process Regression (GPR), provide relatively cheap estimates of function values at a set of test points based on the observations from a set of training points. In addition to the estimates, GPR also provides uncertainties in the estimates, which allows judging the confidence of the resulting Gaussian Process Model (GPM) in deciding if the current known maximum is the global maximum. This easily translates to corner analysis, by representing the characteristic of a circuit as a function dependent on the combination of inputs (corners). A previous student, Michael Shoniker, took the first steps in this problem in his Master of Science thesis. This thesis builds on his work by overcoming some weaknesses of Shoniker's approach, analysing the benchmark datasets, and analysing own generated datasets to learn what output behaviours make circuits difficult to verify.

Acknowledgements

This work is the result of a research supervised by Dr. Bruce Cockburn and Dr. Jie Han. I am incredibly grateful to them for all their help and support throughout the time it took to conduct this research. Special thanks also go to Manish Rana, whose deep knowledge in the subject and the ability to explain the relevant concepts in an accessible language were invaluable to my own understanding of the problem, and Solido Design Automation for their continued cooperation in the project. I would also like to thank my parents for their moral support and everyone who has shared any knowledge with me.

The research was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada (NSERC STPGP 447513 or Project No. RES0018685).

Table of Contents

List of Tables

List of Figures

List of Acronyms

BLUP	Best Linear Unbiased Prediction
BTI	Bias Temperature Instability
CCC	Central Composite Circumscribed Design
CCD	Central Composite Design
CCF	Central Composite Faced Design
CCI	Central Composite Inscribed Design
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
DACE	Design and Analysis of Computer Experiments
DLL	Delay Locked Loop
DoE	Design of Experiment
DV	Design Verification
FF	Fast-fast Process Corner
FS	Fast-slow Process Corner
GND	Ground Voltage
GP	Gaussian Process
GPM	Gaussian Process Model
GPR	Gaussian Process Regression
HCI	Hot Carrier Injection
IC	Integrated Circuit
MAP	Maximum-A-Posteriori Estimate

MCMC	Markov Chain Monte Carlo
MOS	Metal-Oxide-Semiconductor
MSE	Mean Square Error
NMOS	N-channel Metal-Oxide-Semiconductor
PMOS	P-channel Metal-Oxide-Semiconductor
PVT	Process Voltage Temperature
RBF	Radial Basis Function
RSM	Response Surface Model/Methodology
SBS	Sequential Backward Selection
SED	SED Systems (Saskatoon)
SF	Slow-fast Process Corner
SFS	Sequential Forward Selection
SPICE	Simulation Program with Integrated Circuit Emphasis
SR-L1	Sparse Regression L1-norm
SRAM	Static Random-Access Memory
SS	Slow-slow Process Corner
SUS	Subset Simulation
TT	Typical-typical Process Corner
V_{DD}	Supply Voltage Level
VLSI	Very-Large-Scale Integration
WCSS	Within Cluster Sum of Squares

Chapter 1: Introduction

1.1 Introduction

The electrical and computer engineering industry has been growing extremely fast for dozens of years (Moore's Law [1]). Much of that growth is explained by the continued ability of semiconductor process engineers to decrease the size of transistors, allowing the number of transistors per economical chip to be doubled roughly every 18 months, following Moore's Law. Using smaller feature size technologies not only allows for faster switching performance in each particular transistor, it also allows more of them to be packed into integrated circuits (ICs) of the same area and for roughly the same cost in high production volumes. The downside to this, however, is the fact that when shrinking their size, the actual performance of transistors becomes more and more difficult to control precisely [2]. In other words, the most recent technologies are more prone to variations caused by the production process and the statistics of the dopant atoms in the critical regions of the transistors, and consequently it is more difficult to assure the correct performance of all of the manufactured circuits in the chips. Therefore, the problem of accounting for process variations in integrated circuits is becoming increasingly important in circuit design.

To test an IC design for correctness with respect to the intended behaviour specifications, it is standard procedure to perform a series of simulation tests on the circuit over sets of representative operating conditions, so-called corner analysis or design verification (DV). These tests produce results that must comply with the specifications for the design. With the latest technologies, DV could require verification for thousands of corners [3][4]. As such, there would be great benefit to developing a way to save time on this important part of integrated circuit development flow.

In theory, DV does not necessarily require simulating each corner, however. The primary task is to ensure that the circuit complies with the performance specifications, which really means it is sufficient to only simulate the corner with the worst performance – the worst-case corner. This observation underlines the potential benefits of being able to find the worst-case corner for every specified behaviour. Simulating each corner, usually in a simulator like Simulation Program with Integrated Circuit Emphasis (SPICE)[5] can take a significantly long time. In general, DV does

not possess any a priori knowledge of the circuit under analysis: the only information supplied to the DV algorithm is the set of corners at which the designer expects correct performance from every function representing key circuit output characteristics, such as the propagation delays of a signal going from one input of the circuit to each of the outputs. Thus, the problem of selecting a set of worst-case corners can be viewed as the problem of optimising an expensive unknown (black-box) function over a common-sampled discrete domain. The problem of optimising expensive arbitrary functions has long been a topic of research, with most approaches looking at this problem from two perspectives: exploration and exploitation[6]. With exploration, the primary goal is to learn the larger scale shape of the function to build a satisfactory model that can then be used as the basis for taking exploitation actions. With exploitation, the focus is shifted to finding the optimum over the most likely input regions, identified with the help of the function model. If the model is good, then the found optimum should indeed be the best-case or worst-case estimate. However, if the function model is found to be poor, then we must attempt to improve the model. Maintaining a balance between exploration and exploitation is critical to successful optimisation of arbitrary functions.

This thesis project is a continuation of the work of a previous MSc student Michael Shoniker [7]. That work made great use of Gaussian Process Regression (GPR), a general function modelling technique that provides both an estimation of the function value at any point in the input domain, as well as a measure of the uncertainty of the predicted function value. Having this model of the function under analysis allows us to construct algorithms that can take into account the confidence of estimated function values to decide whether a suspected function maximum can be confidently concluded to be the true global function maximum. The approach to optimisation is split into three separate stages that try to strike good balance between exploration and exploitation: the initial exploration phase, selection of corners to simulate next, and requirements for termination. For almost all cases, only a fraction of the total number of input corners will actually be selected for simulation. The information given by the Gaussian Process Model (GPM) that is built on the basis of the simulated corners is often enough to decide either that (a) a corner should be selected for simulation, or (b) it is unlikely to be the global optimum, and thus can be safely ignored. Applying this to the DV process would help to reduce the time from design to manufacture, as the time to simulate process corners is usually the major factor in the time delay cost of design verification.

The rest of this thesis is structured as follows. Chapter 2 summarises the work done by Michael Shoniker, and reviews the relevant theory behind the concepts used in this project. Chapter 3 takes a deeper look into the circuit datasets provided to us by our industrial partner and outlines some key problems with analysing them in the context of this project. Chapter 4 details the exploration of just how sure we, as researchers, can be of the expected performance of our developed algorithm and gives some thoughts on how to compensate for the downsides to existing approaches; the chapter also explores an alternative convergence procedure. Chapter 5 attempts to learn how the search for the function maximum can be improved by adopting a divide-and-conquer approach, providing an alternative view on exploration and next corner selection. Chapter 6 reviews and summarises our insight into what function characteristics tend to make a dataset difficult to analyse. Finally, Chapter 7 summarises the thesis research and proposes directions for future work.

Chapter 2: Background and Overview

2.1 Background on Process Variations

2.1.1 Sources of Variation

Variations in the properties of semiconductor devices must be carefully considered when designing IC systems so that the yield of manufactured devices will be acceptably high. The variations can affect every aspect of performance, most importantly, those related to timing and power consumption. There are several kinds of variation, the most important being the process (transistor), voltage and temperature variations. These variations are often called PVT variations by IC designers.

Process variations arise inevitably from the production techniques used to manufacture CMOS (Complementary Metal-Oxide-Semiconductor) transistors and other semiconductor components (e.g., capacitors, resistances, and interconnecting conductors)[3]. Variations are introduced for the gate oxide thickness[8], the length of the channel, statistically inevitable fluctuations in doping concentrations of the channel region under the gate[9], line-edge and line-width roughness[10], and variations in the overall quality of the wafer[11]. Process variations affect the threshold voltage required for a transistor to switch from the isolated to the conductive state, the speed of formation of the conductive channel (and thus the switching speed), and the effects of parasitic resistances and capacitances.

Voltage variations arise from device variations in the power supply (V_{DD}) circuits. A higher voltage means faster transistor switching, but also higher power consumption. A lower voltage not only makes the circuits slower, it also increases the chance of failure (not switching) for transistors in the circuit. Finally, temperature variations affect the mobility of charge carriers and hence the conductivity of transistors, with lower temperatures generally resulting in faster performance.

With time, transistor properties degrade and aging processes, like Hot Carrier Injection (HCI) and Bias Temperature Instability (BTI), have bigger deleterious effects[12]. With the shrinking linewidths sizes in the newer semiconductor processes, interconnect variations are also becoming more noticeable in the performance of designs[13].

Finally, the design of an IC itself can make variations in signal timing have significant effects on the performance of circuits. For example, flip-flop behaviour is very reliant on the timing of the clock signal with respect to control and data input signals. For example, the clock arriving too early or too late might result in setup and hold time violations. In a circuit that contains multiple flip-flops, internal race conditions can cause drastic changes in output behaviour as a result of PVT variations.

2.1.2 Effects of PVT Variations

PVT variations can have profound effects on the behaviour and performance of ICs. The wider distributions in semiconductor device properties mean that not only the designers should be more mindful of the challenges that arise when optimising for power, clock, and interconnect distributions, but the modelling of the devices themselves becomes more and more complex[14].

In general, wider distributions also tend to reduce our confidence that each manufactured instance of a design will have the correct behaviour. For example, when looking at the statistical timing analysis of circuit blocks, larger parameter variations cause less certainty in determining the critical path of a signal[15]. This requires that the circuit must be designed to function correctly with worst-case PVT variations in mind.

Sometimes due to variation effects established designs become less desirable in complex systems. For example, the Static Random-Access Memory (SRAM) cell, a high-speed volatile memory design that is widely used in ICs, is defined by the stability of operations. With increasing variability and decreasing power supply levels, the classical 6T design[16], an SRAM cell built using six transistors, becomes less and less stable against noise and other transients in the surrounding interconnect. An SRAM must be extremely reliable as millions of them are used in each chip. Therefore, degrading reliability means new robust designs need to be developed. For example, a more stable 8T-SRAM design[17] has gained favour over the traditional 6T-SRAM design. The 8T-SRAM scales to small technologies much better than the 6T-SRAM, and thus it is becoming widely used in industry.

Integrated circuit designers have developed a variety of techniques for evaluating the effects of process variations on circuit behaviour. Some of these techniques, ranging from simple to complex verification methodologies, will be described next.

2.2 Background on Design Verification

2.2.1 Early Design Verification

Up until the last several semiconductor generations, to ensure effective design verification against PVT variations, it was sufficient to verify circuit behaviour at the nominal and extreme (but still acceptable) conditions, usually SS, SF, TT, FS, and FF, where S (for slow), F (for fast), and T (for typical) denote the relative switching speeds of the NMOS or PMOS transistors (N-channel and P-channel Metal-Oxide-Semiconductor, respectively)[18]. The assumption was that if the circuit passes the TT condition test as well as the edge cases (SS, SF, FS, and FF), then the circuit behaviour of manufactured devices would correspond to the specifications at all other legal conditions. While it was recognised that such models do not give the most accurate estimates of performance over all possible PVT conditions, they still could be used to ensure satisfactory yield at production. These strategies were widely used in practice as the technologies were predictable enough that even the most rigorous algorithms would require only a few more verification simulations [19].

2.2.2 The Need for More Variation-aware Design Techniques

With further shrinking of transistor sizes, however, it was noticed that the effects of PVT variations became more significant [20][21][22][23]. The latest technologies also require a relatively large number of corners to be simulated to test completely [24]. One reason is that over several generations, the absolute effect of process variations introduced by manufacturing stages contributes more variation to the device performance in relative terms. So, for example, the thickness of the gate oxide can vary by the thickness of several atom layers. However, as the thickness of the gate oxide is itself only roughly a dozen atoms thick, the relative effect of the thickness variation is big. Another example is that the number of doping atoms has also decreased significantly with the smaller transistor channel volumes in the newest technologies, and so the inevitable statistical fluctuations in the exact number of doping atoms in the channel region makes

a bigger difference. As such, with time grew the need for more precise statistical modelling of circuit response[25].

2.2.2.1 Monte-Carlo Analysis

The most reliable and straightforward way to ensure the correctness of a circuit at all PVT conditions is to perform a Monte-Carlo analysis[26]. Monte-Carlo analysis generates a huge number of random corners and simulates each of them to receive a response. In its most expensive form, the parameters of each device are varied randomly over distributions. From a sufficiently large number of simulations, bell curve distributions are constructed and the likelihood that a response variable will lie within under a certain range is estimated.

With more sources of parameter variations, however, the number of corners to be simulated with Monte-Carlo analysis grows very fast. More complex designs require more optimized power supply, resulting in multiple power supplies, each of which has its own variation. IC designers sometimes have a choice in which type of transistors to use, prioritising speed of switching or power consumption. Finally, increased variability of separate parameters makes the general performance distributions less linear. Thus, more simplified and less expensive approaches to DV are used.

2.2.2.2 Design of Experiments

Design of Experiments (DoE) is a method for choosing the most informative set of input conditions of a process over a space of possible operating conditions so that the performance of the process can be assessed and then optimised [27]. In this project's problem, the theory of DoE can be applied to select the set of PVT corners at which a design needs to be verified. The datasets used for analysis in this work were themselves generated from full-factorial DoE designs, by our industrial collaborator, SED Systems (Saskatoon).

The main goal of DoE is to quantify both the effects of each input (factor) on the output (observation or response) as well as the effects of pairwise and high-order interactions among these inputs. The most reliable way to estimate these effects is to perform a full-factorial design. A full-factorial design sweeps through every possible combination of input parameters, given the finite

sets of values that each input parameter can assume. As such, the size of a full-factorial design (i.e., the number of corners to test) is equal to the product of the number of possible values that each input can adopt. So, for example, if there are n dimensions, and each one can take one of k_i values, then the total size of a full-factorial design will be equal to $\prod_{i=1}^{n} k_i$ corners. Clearly, the size of a full-factorial design grows exponentially in the number n of independent parameters.

Consider an experiment with three factors A, B and C, which take on one of two possible values throughout the experiment. There are various ways of choosing those two values for an input that might have three or more possible levels of discrete values, or a continuous range of values. Following established convention[27], the high value and the low value of each factor (or input) are denoted simply as + and -, respectively. Then, a full factorial experiment can be summarised as shown in Table 2.1. Note that the entries in the Response Value column use a special DoE notation, rather than a standard algebraic notation. For example, (1) denotes the response when all the inputs are at their low level, while ac means that factors A and C are at their high levels, while B is at its low level.

A	B	C	Response Value
-	-	-	(1)
+	-	-	a
-	+	-	b
-	-	+	c
+	+	-	ab
+	-	+	ac
-	+	+	bc
+	+	+	abc

Table 2.1 2^3 Full-Factorial Design for Three Factors

Then, to estimate the effect on the response of each factor and all possible interactions among the factors, a linear combination of responses needs to be taken. For example, to estimate the overall effect of factor A, all that needs to be done is to subtract the average response of all the observations where A was at its low level (denoted as $\bar{y}_{A^-}$ in the equation below) from the average of all the observations where A was at its high level (denoted as $\bar{y}_{A^+}$ in the equation below). The same logic applies to factors B and C. Mathematically, the effect of A is

$$A = \bar{y}_{A^+} - \bar{y}_{A^-} = \frac{a + ab + ac + abc}{4} - \frac{(1) + b + c + bc}{4} \qquad (2.1)$$

To estimate the effect of two-factor interactions, for example AB, all that is needed is to subtract the average of the responses for the runs where A and B were at different levels (denoted $\bar{y}_{A\oplus B}$ in the equation below) from those responses where A and B were at the same level (denoted $\bar{y}_{A\Leftrightarrow B}$ in the equation below).

$$AB = \bar{y}_{A\Leftrightarrow B} - \bar{y}_{A\oplus B} = \frac{abc + ab + c + (1)}{4} - \frac{bc + b + ac + a}{4} \tag{2.2}$$

Finally, the tri-factor interaction is defined as the difference between the AB interaction at different levels of C.

$$ABC = \bar{y}_{AB_{C^+}} - \bar{y}_{AB_{C^-}} = \frac{abc + c - ac - bc}{4} - \frac{(1) + ab - a - b}{4} \tag{2.3}$$

A partial-factorial design, on the other hand, does not require that its corners assume every possible value but instead considers just enough corners to give an idea about the contribution of each input dimension to the value of the function under analysis. The result is a more compact design, but one that cannot distinguish between certain combinations of effects. For example, consider the partial-factorial design in Table 2.3. The design is generated using the key I=ABC. The key (also called the defining relation) is the combination of factors that is used to generate the partial-factorial design. The choice of + and – is particularly useful here as using the algebraic multiplication operation on these signs will produce a "key" column. Table 2.2 demonstrates this rule. The key is then used to extract the combinations of factors A, B and C that result in a + in the column of Table 2.2 for the key. So, for example, with key I=ABC, Table 2.3 is produced for the partial-factorial design.

A	B	C	AB	AC	BC	ABC
-	-	-	+	+	+	-
+	-	-	-	-	+	+
-	+	-	-	+	-	+
-	-	+	+	-	-	+
+	+	-	+	-	-	-
+	-	+	-	+	-	-
-	+	+	-	-	+	-
+	+	+	+	+	+	+

Table 2.2 Factorial Effects of a 2^3 Full-Factorial Design for Three Factors

A	B	C	Response Value
+	-	-	a
-	+	-	b
-	-	+	c
+	+	+	abc

Table 2.3 2^{3-1} Partial-factorial Design with Key I=ABC

This design is compact but has disadvantages. For example, to calculate the effect of factor A, following the procedure described in the preceding paragraphs, one needs to calculate the following

$$A = \frac{a + abc}{2} - \frac{b + c}{2} = \frac{1}{2}(a - b - c + abc) \tag{2.4}$$

Then, to calculate the effect of the two-factor interaction BC, the formula is

$$BC = \frac{abc + a}{2} - \frac{b + c}{2} = \frac{1}{2}(a - b - c + abc) \tag{2.5}$$

which is exactly the same effect as A. So, in essence, by performing these calculations, the designer really gets information on the combined effects of A and BC, that is A + BC, meaning that for the key I=ABC, the effects of A and BC are aliased. And, indeed, Table 2.2 shows that the values in the columns A and BC have the same values in every row where the column ABC is positive. For a more in-depth discussion of partial-factorial designs, refer to [27].

Full-factorial and partial-factorial designs, however, do not have to be defined over only two levels. By generating samples that adopt one of a number of intermediate values along one or more input dimensions, DoE analysis makes possible a more complicated and also likely more accurate model of a function. This leads to p^k designs, where $p \geq 2$ is the number of levels in an input dimension. Note that for p^k designs the number of corners grows extremely fast with the number p of extra levels and the number k of extra dimensions. To correctly represent a strongly non-linear function, thousands of simulations would likely be required, greatly increasing the time required for design verification. This problem has a solution, presented in the previous work on the subject, and described in Section 2.3.

A central composite design (CCD) is often used to construct quadratic response surface models[28]. In addition to the points selected by a full factorial or a fractional factorial design, it also includes a collection of "star points", corners that are located either at the centre of the faces

of the partial or full factorial hypercube design (thus generating a central composite faced design (CCF)) or equidistantly from each other and the factorial hypercube on a sphere encircling the factorial hypercube design. The number of star points is equal to the number of faces of the hypercube, or twice the number of factors. Designers differentiate between the circumscribed (CCC) and the inscribed (CCI) central composite designs. The difference between them is that CCC goes beyond the defined boundaries of the experiment, while CCI stays within them. Also, the hypercube of the factorial design occupies the entire defined space in a CCC design, whereas in a CCI design it is constructed so that the encircling sphere is within the boundaries. The star points provide information about the curvature of the response functions, which is used to build a quadratic response surface model (RSM). Figure 2.1 illustrates the structures of CCC, CCF and CCI designs on a two-dimensional example.

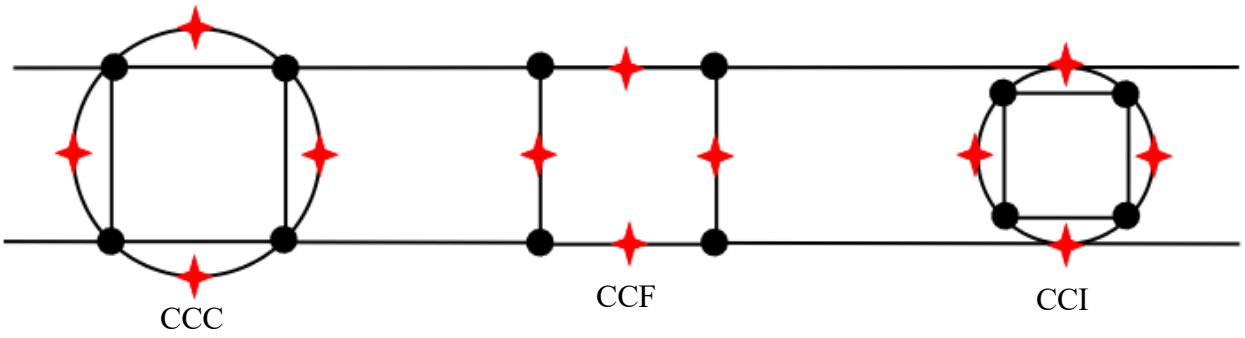


Figure 2.1 Illustration of Central Composite Designs for the Two-Dimensional Case

Yet another interesting design that will be used later in the thesis project is the Plackett-Burman design. This design minimizes the variance of observations for every input dimension with the smallest number of samples[29]. Specifically, the number of samples in the design is always a multiple of four. A Plackett-Burman design of size m can cover between m-4 and m-1 input dimensions. Such compactness requires the assumption that the effect of interactions between factors (that is, input dimensions) is weak or negligible. However, as an initial assumption concerning the dataset for the GPM, this assumption is as good as any. As such, a Plackett-Burman design has the potential to be very helpful at this stage of the algorithm. A representation of a Plackett-Burman design is provided in Table 2.4.

In Table 2.4, the rows represent samples in the design, and columns represent the extreme values (+ for high and - for low) that the input dimension attains in each sample. In a Plackett-Burman design, every pair of high and low levels (that is ++, +-, -+, and --) is repeated the same

number of times for m-1 samples, the one sample left out having every input dimension at the high level. Also, any subset of columns is itself a valid Plackett-Burman design. Removing rows 9 through 12 (as well as removing between four and seven columns) from Table 2.4 will represent a Plackett-Burman design for a 4- to 7-factor experiment, depending on the number of columns removed.

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11
1	+	-	+	+	+	-	-	-	+	-	-
2	-	+	+	+	-	-	-	+	-	-	+
3	+	+	+	-	-	-	+	-	-	+	-
4	+	+	-	-	-	+	-	-	+	-	+
5	+	-	-	-	+	-	-	+	-	+	+
6	-	-	-	+	-	-	+	-	+	+	+
7	-	-	+	-	-	+	-	+	+	+	-
8	-	+	-	-	+	-	+	+	+	-	-
9	+	-	-	+	-	+	+	+	-	-	-
10	-	-	+	-	+	+	+	-	-	-	+
11	-	+	-	+	+	+	-	-	-	+	-
12	+	+	+	+	+	+	+	+	+	+	+

Table 2.4 Plackett-Burman Design for up to 11 Input Dimensions, m=12

In this work, the industrial circuit datasets supplied to us by our sponsor (Solido Design Automation) are full-factorial with respect to a sampled input space. At the same time, these datasets completely define the search space for the algorithm, and the performance of the algorithm is defined by how many points from that search space can be safely omitted from being selected for precise simulation when searching for the global maximum value. Sometimes, it will be necessary to construct a DoE design on top of the search space defined by another DoE design in order to get a first understanding of a subset of the original input space. The exact procedures for that will be described in the relevant later sections of this thesis.

2.3 Review of Previous Work

2.3.1 Discussion of the Framework Setup

Design of Experiment techniques, briefly described in Section 2.2.2.2, are intended to obtain an informative set of function values as the basis for an initial function model. Other factors (e.g., foundry specifications) provide an input grid with $p_i \geq 2$ of levels along each i-th PVT

dimension, located on the defined step size along the dimension. The choice of the input grid can have important implications on function modelling.

Note that for conventional PVT corner simulations, the input domain of the functions is represented as discrete, rather than continuous. The values not sampled by the input grid are assumed to be easily interpolated by, for example, a simple linear or quadratic Response Surface Model (RSM)[28]. However, this is not always a correct assumption. An unlucky design could miss a high value region, or introduce a sudden perceived discontinuity in the modelled function, something simple linear or quadratic RSMs cannot interpolate accurately. As most of the datasets provided for the project are from unknown circuits, the datasets must be assumed to be error-free samples of the circuit output functions and that the full-factorial design is a perfectly sampled representation of those functions. The commercial tool provided by our industrial partner, Solido Design Automation, allows the designer to specify their own DoE for the FastPVT design verification, which has the same functionality explored in this project, and the reader will understandably want to see comparisons with FastPVT. Therefore, this assumption is valid for application in the industry. Lastly, it can be noted that the algorithms presented in this work do not need to be constrained by an input grid as Gaussian Process Regression can give an estimate of the value and uncertainty in the value for any set of inputs, even ones that are off the defined grid. While going off a given input grid is not a goal of the research, it could be an interesting direction for future work.

An additional constraint on the research problem is that each function is a-priori assumed to be an arbitrary, expensive-to-evaluate black-box function. This means that nothing is known about it or its internal structure, and each evaluation at any input point requires significantly more time than it takes to construct the mathematical model of the function. This also means that the primary goal is then to minimise the number of samples (that is, function evaluations) of the black-box function. As such the primary metric of each version of the algorithm is the speedup, defined as the total number of points in the input domain divided by the number of samples required to declare a point as the global optimum.

As a simplifying approximation, any operation during the modelling and optimisation of a black-box function can be categorised as either an exploration action or an exploitation action. Exploration means that the focus is placed on increasing the knowledge of the larger-scale

properties of the function, such as establishing the general range of possible function values or narrowing down the likely optimum regions. Exploitation means that the amassed knowledge is used to place the computational focus on the most promising region. So, for example, once the confidence threshold in a certain subregion of the search space is reached, the exploitative action would be to focus on exploring this subregion. As the algorithm proceeds, there is less and less distinction between these two notions. Clearly ensuring a good balance between exploration and exploitation is critical to successful optimisation.

Another way to look at the differences between exploration and exploitation is to say that exploration involves unbiased sampling over less well-known input regions (at least initially), whereas exploitation is biased sampling where the bias is controlled by heuristic rules. With this point of view, an interesting discussion can be had on the trade-off between randomness and determinism. On the one hand, deterministic solutions will lead to consistent and reproducible, possibly even better, results. For example, as will be explored in this thesis, having a deterministic solution to the initial exploration stage often leads to faster and in most cases more accurate convergence to the global maximum (see Section 5.1.2). On the other hand, incorporating some randomness allows traps created by deterministic rules to be avoided. So, as will be demonstrated, while a deterministic initial exploration phase leads to faster and more accurate convergence most of the times, the other times the results are unacceptably worse than the true global maximum. And as effective exploitation depends on the result of the knowledge gained from exploration, randomness indirectly affects that part of optimisation as well.

Finally, while optimising the cost of building mathematical models is not a concern of this project, there are still a few points to consider. First, it might be possible to take advantage of the computational capability available to run the algorithms. Taking advantage of multithreading, for example, might be beneficial to simulating several points or optimising several subregions at once, while taking the same total processing time as for a serial implementation. Another thing to consider would be how well the assumptions work together, whether some of the new ones might contradict or exclude the assumptions that were made earlier. Such insights can form the basis for developing new heuristics. However, having many heuristics might make termination of the algorithm less optimal, for example if there are no clearly defined rules on when one heuristic should be applicable over another. To limit such downsides, it would be beneficial to develop

heuristics that are applied based on the gathered information on the dataset during the course of execution.

2.3.2 Overview of Gaussian Process Regression

Gaussian Process Regression (GPR), also known as kriging, is a regression (function fitting) technique that makes use of Gaussian Processes (GPs). A Gaussian Process is defined to be a statistical distribution over a number of samples where each sample is represented by a Gaussian random variable and any finite number of the samples has a jointly Gaussian distribution [30]. A D-dimensional GP has samples $\boldsymbol{X} \in \mathbb{R}^D$ where $D > 1$. Kriging was originally developed for the purposes of mining engineering, with the primary task being to approximate how much material (gold ore, in the original problem) was present in an area based on the a-priori spatial distribution of a few known samples in the area[31]. The description of GP in Section 2.3.2.2 is valid for the implementation of the python package sckit-learn, version 0.18.0 and above. The version of the package used in the project is 0.16.1, and is described in Section 2.3.2.3. The difference between the two implementations is primarily in the way the variances of the predictions are estimated, and are explained in more detail towards the end of their respective sections.

2.3.2.1 General Overview of GPR

A GP is completely defined by the mean and covariance functions, which together represent the target function $f(\boldsymbol{X})$. The mean function $\mu(\boldsymbol{X})$ is usually 0, and the covariance function is generated from the so-called kernel function of the Gaussian Process Model (GPM). The mean $\mu()$ and covariance $k()$ functions, respectively, are defined as follows:

$$\mu(\boldsymbol{X}) = E[f(\boldsymbol{X})] \tag{2.6}$$

$$k(\boldsymbol{X}, \boldsymbol{X}') = E\big[\big(f(\boldsymbol{X}) - \mu(\boldsymbol{X})\big)\big(f(\boldsymbol{X}') - \mu(\boldsymbol{X}')\big)\big] \tag{2.7}$$

where $E[f(\boldsymbol{X})]$ denotes the expected values of the Gaussian process $f(\boldsymbol{X}) \sim GP(\mu(\boldsymbol{X}), k(\boldsymbol{X}, \boldsymbol{X}'))$ where $\boldsymbol{X} \in \mathbb{R}^D$ and $\boldsymbol{X}' \in \mathbb{R}^D$ are any two input points.

For discrete datasets, covariance functions are replaced with covariance matrices Σ whose entries correspond to the values of the covariance function for pairs of points. Covariance functions

themselves are generated from the kernel functions by changing the varying parameters of the kernel (also called hyperparameters). Kernel functions specify the prior distributions over the training points, and the posterior distributions over the test points, with information available from the training points. Essentially, what this means is that for each test point, we can draw the distribution, with mean and variance, of the random variable corresponding to the optimised function value at nearby input points. This information can be used not only to predict function values, but it also gives an easy way to estimate the confidence in each prediction (see Figure 2.2, generated using scikit-learn 0.16.1).

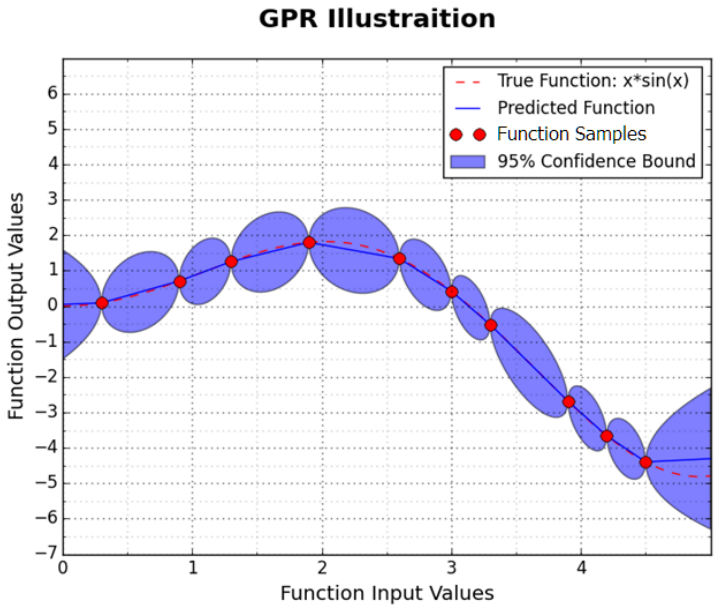


Figure 2.2 Demonstration of Gaussian Process Regression for the Function $f(x) = x*\sin(x)$

In Figure 2.2, the red dots are the training points (known function values or function samples) that are used to construct the GPM. The shaded blue region shows the 95% likelihood region of the values of the posterior distributions (functions) generated from the absolute exponential kernel based on the observed values of the training points that go through the training points.

The absolute exponential kernel function was used to construct Figure 2.2. This kernel defines functions that expect discontinuities or sudden changes in values and instantaneous slopes of the tangents. Contrast to this with the Radial Basis Function (RBF) kernel, which generates infinitely differentiable (i.e., smooth) functions. Some other examples of kernels are the rational

quadratic and the squared sine exponential kernels. Figure 2.3 (generated in scikit-learn 0.18.1) illustrates the differences between the four kernels.

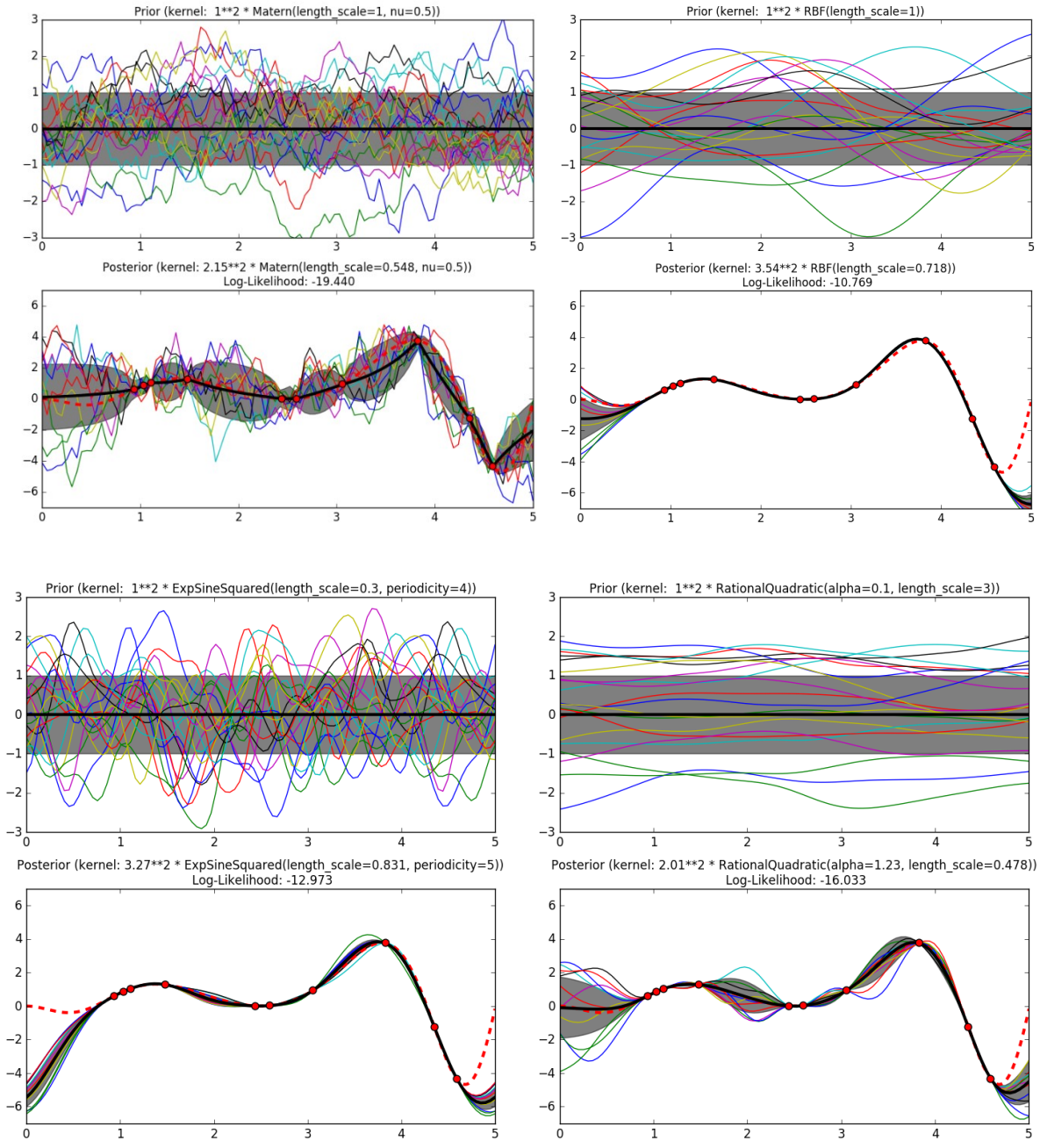


Figure 2.3 Comparison of Different Kernels for GPR

The mathematical definitions of the kernels are the following[32]. The RBF kernel is defined as

$$k(x_i, x_j) = \exp\left(-\frac{1}{2} d\left(\frac{x_i}{l}, \frac{x_j}{l}\right)^2\right) \tag{2.8}$$

where x_i and x_j are the input values of any two points, $d()$ is the function of distance between the points, and l is the length scale that defines the extent of interaction between two points. The absolute exponential kernel is defined similarly to RBF, except it only takes the distance between the points, rather than the square of the distance

$$k(x_i, x_j) = \exp\left(-\frac{1}{2} d\left(\frac{x_i}{l}, \frac{x_j}{l}\right)\right) \tag{2.9}$$

The rational quadratic kernel is defined as follows

$$k(x_i, x_j) = \left(1 + \frac{d(x_i, x_j)^2}{2\alpha l^2}\right)^{-\alpha} \tag{2.10}$$

where the new parameter α, called the scale mixture, defines how much effect the distance between these two points should influence the predictions and uncertainties for the points. Finally, the squared sine exponential kernel is defined as

$$k(x_i, x_j) = \exp\left(-2\left(\frac{\sin\left(\frac{\pi}{p} * d(x_i, x_j)\right)}{l}\right)^2\right) \tag{2.11}$$

where π is the irrational constant and p is the periodicity parameter that defines how soon the pattern of the output function will start repeating.

The top left part of Figure 2.3 shows how regression is performed using the absolute exponential kernel (which is obtained through setting the hyperparameter ν=0.5 for the Matern kernel [30]) on the function $f(x) = x * \sin((x - 2.5)^2)$. The top row shows 20 samples from the prior (uninformed) distributions. From those samples, it can be seen what kind of functions the GPM will use to model $f(x)$. The bottom row shows the final results of the regression. The thick solid black line is the predicted mean function $\mu(\boldsymbol{X})$ (compared to the thick dashed red line of the true function to be predicted), and the thin lines are the posterior (informed) functions. The top right part of Figure 2.3 shows the same information produced by applying the RBF kernel. Notice

how the samples of that kernel are much smoother than those of the absolute exponential. As the function to be modelled is itself a smooth one, RBF gives a smaller uncertainty for its prediction compared to that of the absolute exponential even if the mean predicted functions are similarly accurate in their predictions of the true function. The squared sine exponential and the rational quadratic kernels (bottom left and right, respectively) also show smooth predictions but provide more uncertainty and, in the case of squared sine exponential, less accurate predictions. It is interesting to note that for this function for this collection of samples and unrestricted ranges of the length scale and the periodicity, the squared sine exponential produces the exact same predictions and uncertainties (as well as the same posterior samples) as the RBF kernel. It is unwise to have unrestricted hyperparameters as it takes a long time to finalise the model.

2.3.2.2 Bayesian approach

The approach implemented in scikit-learn 0.18.0 and above follows that of Gaussian Process for Machine Learning[30]. According to that approach, a function is modelled in the form $y = f(\boldsymbol{x}) + \varepsilon$, where $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w}$ is the function model, $\boldsymbol{w}$ is the weight vector, meaning the function model is represented as a linear combination of its inputs, and $\varepsilon \sim N(0, \sigma_n^2)$ is the Gaussian independent and identically distributed error with zero mean and variance σ_n^2. Given the set of input points X, and the weights $\boldsymbol{w}$, the likelihood of observing the number n outputs $\boldsymbol{y}$ for a joint Gaussian process is the product of the likelihood of observing a separate instance y_i of $\boldsymbol{y}$.

$$p(\boldsymbol{y}|X, \boldsymbol{w}) = \prod_{i=1}^{n} p(y_i|X, \boldsymbol{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{(y_i - \boldsymbol{x}_i^T\boldsymbol{w})^2}{2\sigma_n^2}\right)$$

$$p(\boldsymbol{y}|X, \boldsymbol{w}) = \frac{1}{(2\pi\sigma_n^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma_n^2}|\boldsymbol{y} - X^T\boldsymbol{w}|^2\right) = N(X^T\boldsymbol{w}, \sigma_n^2 I) \qquad (2.12)$$

where I is the identity matrix and $|\cdot|$ is the L^2 norm (the Euclidean length of a vector). Thus, the total likelihood is also Gaussian distributed.

In Bayesian models, we learn the likelihood of an event based on observations (posterior likelihood) by first defining a prior likelihood (the best guess for the likelihood of the event) and

the likelihood of observing the targets given the inputs and the weights (in this case, Equation 2.12). Then, the following formula is applied.

$$posterior = \frac{likelihood * prior}{marginal\ likelihood}$$

which translates to regression as

$$p(\boldsymbol{w}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|X)} \tag{2.13}$$

essentially defining how likely it is that the weights $\boldsymbol{w}$ are to give the right linear coefficients for the $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{w}$ model based on the observed targets $\boldsymbol{y}$. The prior (uninformed) distribution of the weights $\boldsymbol{w}$, $p(\boldsymbol{w})$, is usually taken as a zero-mean Gaussian distribution $\boldsymbol{w} \sim N(\boldsymbol{0}, \Sigma_p)$, where Σ_p is the prior covariance matrix, which is generated from the kernel (see Figure 2.3 and the related discussion). The marginal likelihood is marginalization of the likelihood over the weights $\boldsymbol{w}$ to give the likelihood independent of the weights' probability

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})d\boldsymbol{w} \tag{2.14}$$

Then, the distribution of the model at a test point $\boldsymbol{x}_*$ given the observations $\boldsymbol{y}$ at training points X is given by

$$p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = \int p(f_*|\boldsymbol{x}_*, \boldsymbol{w})\, p(\boldsymbol{w}|\boldsymbol{y}, X)d\boldsymbol{w}$$

$$p(f_*|\boldsymbol{x}_*, X, \boldsymbol{y}) = N\left(\frac{1}{\sigma_n^2}\boldsymbol{x}_*^T\left(\frac{1}{\sigma_n^2}XX^T + \Sigma_p^{-1}\right)^{-1}X\boldsymbol{y}, \boldsymbol{x}_*^T\left(\frac{1}{\sigma_n^2}XX^T + \Sigma_p^{-1}\right)^{-1}\boldsymbol{x}_*\right) \tag{2.15}$$

is also Gaussian distributed with the mean and variance as specified in Equation 2.15.

In all figures in Figure 2.3, the shaded region corresponds to the 68.3% likelihood region, which can also be called the 1-sigma confidence region. The mean functions are linear combinations of the priors or posteriors, which are samples of the kernel. For a continuous functional optimisation, it can be said that the regression task of a GPM is to find the best linear combination of the posterior functions. A better linear combination is assumed to have a larger log likelihood. Also note that the parameters, like length_scale, changed between the top and bottom

figures. The parameter length_scale governs to the length scale of the kernel, and defines the extent of interaction between the uncertainties of any two points. Having an overly small length scale tends to give myopic estimates, meaning that the uncertainty rises too quickly with the distance between the test point and the closest training point. Conversely, having an overly big length scale would mean that there is too little effect of the distance between test and training points on the uncertainties, and as a result the uncertainties are too similar for every test point. Thus, optimising the length scale is a very important part of regression. The process of regression goes on until the log likelihood does not rise any more. The log likelihood in this case is a measure of how much the model is confident in its current selection of hyperparameters, and is the natural logarithm of the likelihood of a set of parameters. For a set of hyperparameters θ given observations X, the likelihood is defined as

$$L(\theta|Y,X) = P(Y|\theta,X) \tag{2.16}$$

where $P(Y|\theta,X)$ is the probability of having observations Y given the chosen hyperparameters θ and inputs X. From another point of view, the likelihood (and consequently the log likelihood) is a measure of the validity of the hyperparameters θ, given by the evidence Y. For GPR, the log likelihood is given by

$$\log P(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^{\mathrm{T}}(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(X,X) + \sigma_n^2 I| - \frac{n}{2}\log(2\pi) \tag{2.17}$$

where $\boldsymbol{y}$ is the observed function values (training samples), $K(X,X)$ is the covariance (also known as Gram) matrix, generated from the covariance (kernel) function with set hyperparameters for the input values of the observed values X, σ_n^2 is the square of the allowed noise level, and n is the number of observations (training samples)[30]. The covariance matrix $K(X,X)$ is calculated as

$$K(X,X) = \boldsymbol{\Phi}(\mathrm{X})^{\mathrm{T}}\Sigma_p\boldsymbol{\Phi}(\mathrm{X}) \tag{2.18}$$

where Σ_p is the prior covariance matrix and $\boldsymbol{\Phi}(\mathrm{X})$ is the collection of projected inputs X, the projection function being defined as the kernel with the set θ of hyperparameters, and each entry of $K(X,X)$ is equal to

$$k(\boldsymbol{x},\boldsymbol{x}_*) = \varphi(\boldsymbol{x})^{\mathrm{T}}\Sigma_p\varphi(\boldsymbol{x}_*) \tag{2.19}$$

To produce the mean estimates and the variance, the covariance function with the hyperparameters that lead to the highest log likelihood is used to calculate the following for each test point $\boldsymbol{x}_*$:

$$\overline{f_*} = K(X, \boldsymbol{x}_*)^{\mathrm{T}}(K(X,X) + \sigma_n^2 I)^{-1}\boldsymbol{f} \tag{2.20}$$

$$\mathbb{V}[f_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(X, \boldsymbol{x}_*)^{\mathrm{T}}(K(X,X) + \sigma_n^2 I)^{-1}K(X, \boldsymbol{x}_*) \tag{2.21}$$

here $\overline{f_*}$ is the predicted mean value at $\boldsymbol{x}_*$, as before**,** $\boldsymbol{f}$ is the true values of the training points, $\mathbb{V}[f_*]$ is the predicted variance (also equal to the square of the predicted standard deviation at $\boldsymbol{x}_*$), and σ_n is the allowed noise in the model. The square of σ_n is multiplied by the identity matrix I to have the same dimensions as $K(X,X)$, adding the square of the noise magnitude to the main diagonal of the covariance matrix to model noisy observations.

2.3.2.3 Best Linear Unbiased Prediction

In the implementation of scikit-learn 0.16.1, the BLUP for every point is calculated instead[33], adapted from the MATLAB software package DACE (Design and Analysis of Computer Experiments)[34]. Best linear unbiased prediction means that the predicted mean of the function is a linear combination of the posterior functions, the expected difference between the predicted values and the true values is 0 (i.e., the prediction is unbiased), and the selected linear combination of the posterior functions minimises the variance (mean squared error) of the predictions, so it is the best fit. The conditions give a constrained minimisation problem to find the linear coefficients $a(X)$ of the linear combination:

$$a(X_*) = argmin_{a(X)}E[(G(X) - a(X)^T y)^2]$$

$$s.t.\, E[G(X) - a(X)^T y] = 0 \tag{2.22}$$

for the multi-objective function to be predicted $G(X)$ and the observed values (i.e., training samples) y. The error in the predictions is equal to

$$\hat{y}(X) - y(X) = a(X)^T y - y(X) = a(X)^T(F\beta + Z) - (f^T\beta + z)$$

$$\hat{y}(X) - y(X) = a(X)^T Z - z + (F^T a(X) - f)^T\beta \tag{2.23}$$

where F is the set of projections of the training points into the kernel space, f is the projection of the single test point under consideration into the kernel space (both of them together are equivalent to $\mathbf{\Phi}(\mathrm{X})$ in Section 2.3.2.1), β is the set of linear coefficients for the projections, and Z is the collection of regression errors z at every test point. For the unbiased predictor, $F^T a(X) - f = 0$. And so, the mean squared error (MSE) in the predictions (which for unbiased predictors is equal to the variance) is calculated as

$$\sigma(X) = E\left[\left(\hat{y}(X) - y(X)\right)^2\right] = E[(a(X)^T Z - z)^2] = E[z^2 + a(X)^T Z Z^T a(X) - 2a(X)^T Z z]$$

$$\sigma(X) = \sigma^2(1 + a(X)^T R a(X) - 2a(X)^T r) = \sigma^2(1 + a(X)^T(Ra(X) - 2r)) \tag{2.24}$$

where σ is the MSE of the predictions for the training points (i.e., the process variance), and R is the symmetric correlation matrix (equivalent to the noisy covariance matrix $K(X,X) + \sigma_n^2 I$ in Section 2.3.2.2) with entries r (equivalent to $K(X, \boldsymbol{x}_*)$ in Section 2.3.2.2) that define the similarity of the predictions and the errors between two different points in the input space.

The goal of the BLUP is to minimize the MSE of the predictions subject to the constraints in Equation 2.22. For that, the method of Lagrange multipliers is used and the Lagrangian function is constructed

$$L(a(X), \lambda) = \sigma^2(1 + a(X)^T R a(X) - 2a(X)^T r) - \lambda^T(F^T a(X) - f) \tag{2.25}$$

where λ is a Lagrange multiplier. Then, the gradient is $L'(a(X), \lambda) = 2\sigma^2(Ra(X) - r) - F\lambda$, and the system of equations is constructed

$$\begin{cases} 2\sigma^2(Ra(X) - r) - F\lambda = 0 \\ F^T a(X) - f = 0 \end{cases} \tag{2.26}$$

Substituting $\tilde{\lambda} = -\frac{\lambda}{2\sigma^2}$ into Equation 2.26 leads to the following matrix

$$\begin{bmatrix} R & F \\ F^T & 0 \end{bmatrix} \begin{bmatrix} a(X) \\ \tilde{\lambda} \end{bmatrix} = \begin{bmatrix} r \\ f(X) \end{bmatrix} \tag{2.27}$$

and finally, we solve for $\tilde{\lambda}$ and $a(X)$ to get

$$\tilde{\lambda} = (F^T R^{-1} F)^{-1}(F^T R^{-1} r - f) \tag{2.28}$$

$$a(X) = R^{-1}\left(r - F\tilde{\lambda}\right) \tag{2.29}$$

Having $a(X)$, and remembering that R and R^{-1} are symmetric, we can substitute $a(X)$ to find the expression for the mean function of the prediction

$$\hat{y}(X) = a(X)^T y = \left(r - F\tilde{\lambda}\right)^T R^{-1} y$$

$$\hat{y}(X) = r^T R^{-1} y - (F^T R^{-1} r - f)^T (F^T R^{-1} F)^{-1} F^T R^{-1} y \tag{2.30}$$

Note that, as R is equivalent to $K(X, X) + \sigma_n^2 I$, and from Equation 2.23, $Ra(X) = r - F\tilde{\lambda}$. r is also equivalent to $K(X, \boldsymbol{x}_*)$ in Section 2.3.2.2, so we arrive at a somewhat similar, but different expression as Equation 2.20 for the mean prediction. Finally, substituting Equation 2.29 into Equation 2.24, we get

$$\sigma(X) = \sigma^2\left(1 + a(X)^T (Ra(X) - 2r)\right) = \sigma^2\left(1 + \left(F\tilde{\lambda} - r\right)^T R^{-1}\left(F\tilde{\lambda} - r\right)\right)$$

$$\sigma(X) = \sigma^2\left(1 + \left(F\tilde{\lambda} - r\right)^T R^{-1}\left(F\tilde{\lambda} - r\right)\right) = \sigma^2\left(1 + \tilde{\lambda}^T F^T R^{-1} F\tilde{\lambda} - r^T R^{-1} r\right) \tag{2.31}$$

Substituting $u = F^T R^{-1} r - f(X)$ into Equation 2.31 to keep consistent with [33] and [34], we get the following for the MSE

$$\sigma(X) = \sigma^2(1 + u^T (F^T R^{-1} F)^{-1} u - r^T R^{-1} r) \tag{2.32}$$

The process variance σ^2 is found from the generalized least squares fit analysis and is equal to

$$\sigma^2 = \frac{1}{m}(y - F\beta^*)^T R^{-1}(y - F\beta^*) \tag{2.33}$$

where m is the number of samples and β^* is the same as in Equation 2.19, and is equal to $(F^T R^{-1} F)^{-1}(F^T R^{-1} y)$. That expression for β^* also appears in Equation 2.30, which signifies that the mean prediction of the BLUP approach is the same as that of linear regression in the kernel space.

The difference between the Bayesian approach of conditioning the prior distributions on the observations in Section 2.3.2.2 and the BLUP approach in this section is primarily in the fact that the BLUP estimate for the variance (MSE) explicitly depends on the values of the training points y, while the Bayesian approach depends less directly through conditioning the covariance matrix on the observations. As such, the variance is more constrained by the observations for

BLUP, and, if the GPM is not confident in the mean function, the variances calculated in that way are smaller. Since this project was a continuation of a previous work, it was decided to keep using the 0.16.1 version of the scikit-learn package to expand on the heuristics developed there. The previous work is summarized in Section 2.3.3.

2.3.3 The Iterative Framework of Michael Shoniker

For this project, it was found empirically that the absolute exponential kernel worked the best when using the scikit-learn 0.16.1 implementation of kriging[33]. This might be explained by the fact that the datasets provided are discrete, and thus are likely to appear irregular to the model, with discontinuities and sudden changes in values. For greater generality, the absolute exponential kernel is used, even if the underlying function is smooth. This would mean that, for example, the RBF kernel would lead to overly confident predictions. Additionally, the expressive capacity of the absolute exponential kernel is objectively higher than that of RBF. RBF will always try to predict smooth functions, which may result in overly complex models which might change a lot from iteration to iteration. In addition to choosing the kernel, the previous work by Michael Shoniker, developed a set of rules and heuristics empirically using the same set of benchmark circuits used in this thesis. These rules saved an average of 78.9% of the simulations before being confident in finding the true global maximum[7]. Shoniker's algorithm is reviewed below.

Following the framework described in [35], the problem is partitioned into three parts: (1) selection of the initial training set, (2) selection of points to simulate in the next iteration of the algorithm, and (3) the termination decision. The purpose of the initial training set is to provide a good starting point (that is, a good initial model of the function) for the algorithm. A good initial training set should be large enough to be a representative sample to direct early exploration of the input domain, but should also be compact enough that there is still significant opportunity to save on the number of points to be simulated before the algorithm is terminated. After a series of tests, Shoniker decided that the initial training set should be constructed the following way. The total size of the initial training set is equal to $m = max(0.01*N, 2*n)$, where N is the total number of points in the total dataset, and n is the number of input dimensions. The initial design includes the mean point (the central point) of the input space, and then adds to it a set containing every extreme point. An extreme point is one that has at least one of its input values equal to an extreme (i.e.,

smallest or biggest) value that can be assumed for the input dimension. If the number of extreme points is bigger than m-1, then m-1 points out of all of the extreme points are selected at random for the initial training set. If, however, there are fewer than m-1 extreme points, then all extreme points are selected for the initial training set, and more points are selected at random from the rest to raise the total number of points in the initial training set to m.

The second step, next corner selection, is executed as follows. With each algorithm iteration after selecting the initial training set, a GPM is constructed, and a mean predicted value $\hat{y}$ and a standard deviation value σ are generated for each test (unsimulated) point. These values are used to construct scatter plots, where the standard deviation values are plotted against the corresponding mean predicted values (see Figure 2.5). From each such plot, the upper convex hull is constructed, and every point that lies on the hull is selected as one of the training points for the next round of simulations. The reason the points on the upper convex hull are selected is that it was found that these points are most likely to become the true maximum (because they have large predicted values and predicted errors), and by selecting multiple points on the hull the selection heuristic ensures diversity in the search and counteracts against too much greed in the search algorithm.

The third and final step of the algorithm is the termination decision. If each training point is indeed a Gaussian random variable, then the probability that the variable assumes a value bigger than $\hat{y} + k*\sigma$ is described by the single-tailed cumulative probability $\mathrm{Q}(k)$ of the Gaussian distribution with mean of $\hat{y}$ and standard deviation σ[36]. The plot of the Q-function is provided in Figure 2.5. Thus, if we have a known current simulated maximum value of $\hat{y}_{\max?}$ and for a certain test point it is true that, for example, $\hat{y}_{\max?} = \hat{y}_{\text{test}} + 3*\sigma_{\text{test}}$, then there is a $1 - \mathrm{Q}(3) = 0.99865$ probability that the test point has a smaller value than $\hat{y}_{\max?}$. Then, it was decided that the algorithm has a 3-sigma confidence that the current maximum is the true maximum if, for every test point, it is true that $\hat{y}_{\max?} \geq \hat{y}_{\text{test}} + 3*\sigma_{\text{test}}$. Graphically, this is represented if every test point on the scatter plot lies below the straight line going from the current simulated maximum with slope -k, the line representing the k-sigma confidence of termination to the true maximum (see Figure 2.4).

In Figure 2.4, an intermediate state in the search is shown for the delay output function of the shift_reg dataset. The figure shows a scatter plot each point's predicted value versus the corresponding uncertainty. The straight lines starting at the left of the figure are the termination

lines, with the magnitude of the slope being equal to the sigma-confidence level (i.e., the coefficient factor of the σ value). The termination lines go from the current known maximum, and the dataset has not terminated even to 1-sigma confidence level as not every point is located below the top straight line. As soon as every point is predicted to be below a termination line with slope -*k*, it is said that the *k*-sigma confidence has been reached and the algorithm is terminated for that sigma level. The red point in the red circle between the 3-sigma and the 4-sigma termination lines is the prediction of the true global maximum for this particular circuit. On the figure, the global maximum is predicted to have the value of approximately 1.22e-8 and uncertainty of approximately 1.5e-10 units, when the true value is 1.32e-8, so GPM misses that prediction by approximately 7 sigma at that stage of the algorithm.

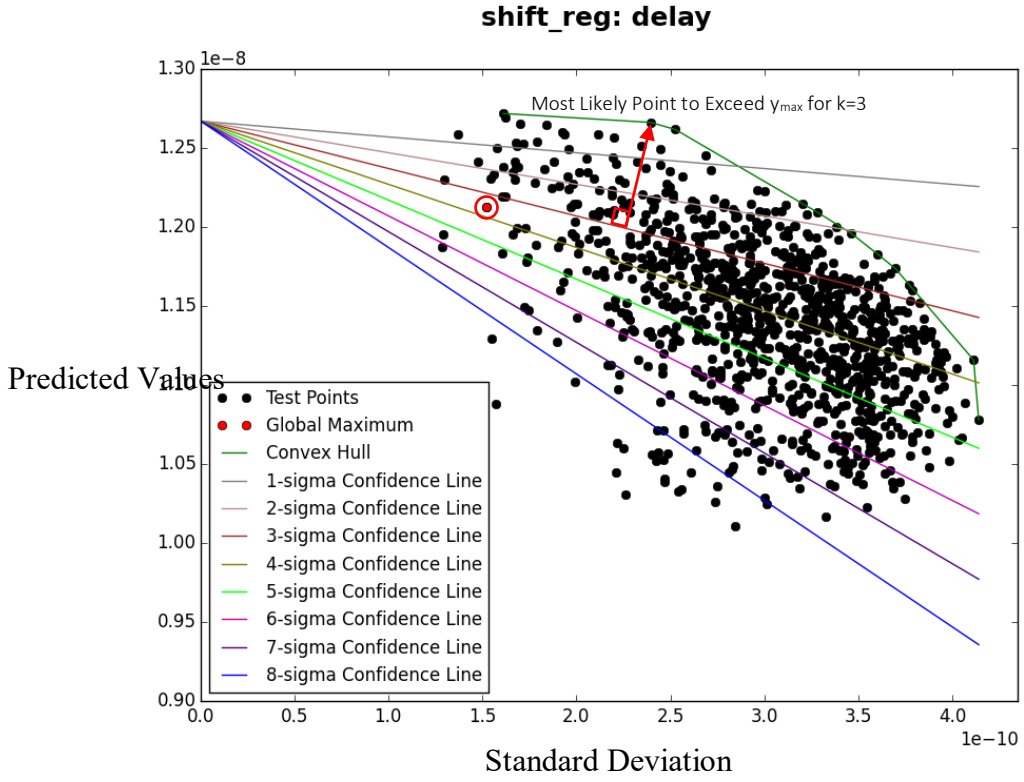


Figure 2.4 Demonstration of the Convergence Criteria for Different k-sigma Confidences

The green multi-segment line enveloping all the points at the top right is the upper convex hull. It goes from the point with the highest predicted value to the point with the highest predicted uncertainty. The perpendicular red line going from the 3-sigma termination line points to the corner farthest from the termination line. This is the worst-case corner for the 3-sigma termination rule, and in an earlier version of Shoniker's algorithm it would be the only point selected for next step

simulation. After Shoniker tried several such termination rules (6-sigma and 9-sigma), he found that taking all the points from convex hull makes termination faster and more accurate, possibly because the corners on the upper convex hull essentially account for every such termination rule, from 0-sigma up to an arbitrarily-high-sigma rule. After selecting the next corners for simulation and after building a new GPM that accounts for the newly selected corners, it is desirable that all the corners move towards the lower left corner of the figure, to accelerate termination. This, however, is not always the case. It is sometimes possible that selecting the corners on the convex hull significantly changes the understanding of the dataset by the model, which often produces drastically different predicted values and increases uncertainties.

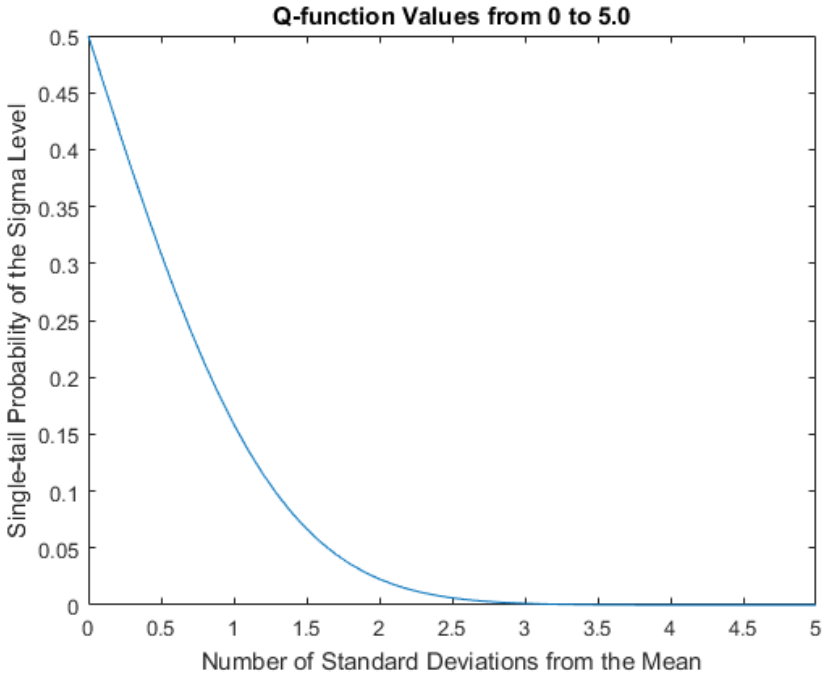


Figure 2.5 Illustration of the Q-function for Sigma Levels from 0 to 5.0

Shoniker found that it was useful to make certain modifications, however, as the iterative algorithm would often converge to an incorrect identification of the global maximum, getting stuck in a local optimum. The selected modifications were to increase the uncertainty in the predictions for function values, by introducing a so-called three-step amplification procedure and the application of a heuristic boosting factor. The need for a three-step amplification procedure arose from noting that GPR would often underestimate the value of points near the current maximum. After conducting a number of experiments, Shoniker decided that every point within one

manhattan step away in the input space from the current maximum would have its uncertainty (as predicted by the latest GPM) amplified by 25%, and every point two manhattan steps away from the current maximum would have its uncertainty amplified by 15%. To take a manhattan step means to change one and only one value of the inputs to a value right next to the former value on the defined grid. The difference between the old and the new input corners is then defined as on manhattan step, regardless of the Euclidean distance in the input space. This would also serve as an incentive for the algorithm to more thoroughly explore the region close to the current maximum to make it less likely that a true global function maximum will be missed.

The need for the global boosting factor arose from observing that for certain datasets (e.g., circuit shift_reg) many more points would not be within a certain number of standard deviations from their true values. In other words, GPMs systematically underestimate the uncertainty for the predicted functions. So, for example, it would be expected that the true value of a point would lie within $\pm 3*\sigma$ of the actual function value approximately 99.73% of the time. For shift_reg outputs, however, that was not the case, and so it was decided to empirically boost the uncertainty for all points to bring the fraction of estimations being within the true values to the expected rates. A strategy was adopted to modify a well-known 10-fold cross-validation technique[37] to find by how many standard deviations exactly predictions from GPR miss their true values. Ten cross-validation folds would be constructed and for each cross-validation run, 9 of the folds would serve as the training set and the last fold would be used as the test set. Then, for each point in the test set it would be recorded by how many standard deviations the mean estimate missed the true value. After each cross-validation run is complete, the largest miss is divided by 3, and the resulting ratio becomes the boosting factor. Finally, as it was noted that at the beginning of this procedure the fraction of the corners within $\pm 3*\sigma$ of the true value would still be less than expected, the boosting factor is multiplied by a further factor of 1.25 if less than 25% of the corners in the dataset have been simulated. The boosting factor is then multiplied by the uncertainty estimation of each test point to produce the final σ_{test_final}. The complete expression for σ_{test_final} is

$$\sigma_{test_{final}} = \begin{cases} 1.25 * boost\ factor\ *\ \sigma_{test}, & N_{corners\ simulated} \leq 0.25 * N_{corners\ total} \\ boost\ factor\ *\ \sigma_{test}, & N_{corners\ simulated} > 0.25 * N_{corners\ total} \end{cases} \quad (2.34)$$

With the higher uncertainty, the algorithm takes several more iterations to terminate.

As doing cross-validation is time and resource consuming, it would be done only occasionally, after successive 10% increments of the total set simulated. It was noted that the boosting factor grows roughly linearly, and so it was decided to linearly interpolate the factor in-between cross-validation procedures. Finally, to perform cross-validation immediately after selecting the initial training set, the minimal number of points in the training set was bounded from below at 10 points, and so the size of the initial training set became $m = max(0.01*N, 2*n, 10)$. This allowed a significant increase in accuracy of the predictions made by the algorithm, at the cost of lower speedups. The results of the final algorithm by Shoniker are presented in Table 3.2.

Before starting new work on the project, we included one small modification. For many datasets, one or more input dimensions had only one unique value. In this work, these dimensions were removed, as they would not be of any benefit to the accuracy of predictions, and they would introduce unnecessary work for the algorithm. One significant factor that would change with this is the number of points in the initial training set. This way, the number of ways to choose the initial training set is slightly reduced. Since selection of the initial training set is the biggest source of variance in the reported results, the results are thus more stable with the modification.

This change, however, adds an additional point of consideration. A dataset used in Shoniker's work, mux, lost half of its input dimensions, going from eight to four. Therefore, the size of the initial training set dropped from sixteen to ten. An unwelcome consequence from this was losing approximately 30% in accuracy of termination in two of the output functions, qtran and qtran0. This cast doubts on whether it was smart to remove the static dimensions. Looking at this from the other perspective, however, is it smart to leave those dimensions in? If such static dimensions should be allowed, then the selection of the initial training set according to Shoniker's selection procedure would not depend on the structure of the dataset in the input space. This would make the size of the initial training set rather arbitrary, which would introduce additional problems for the algorithm, particularly in the number of corners it takes to simulate to terminate to an answer.

It is worth noting that the two output functions that lost in accuracy have the exact same structure, and, in addition to that, have three non-adjacent values, which are the maximums, that are 161 times bigger than the next biggest value. Such spikes, which are not even located on a ridge, are impossible to predict, and, in the worst case, the only sure way to catch them is to perform

the exhaustive full-factorial simulation. All three values could potentially be selected for the initial training set, and thus without removing the static dimension the algorithm is much "luckier". This luck, however, is different from the intentional randomness introduced in the flow of the algorithm, such as the selection of the initial training set. This randomness in Shoniker's algorithm was intended to provide more diversity to the results of applying the algorithm, as well as give a higher probability to not use a wrong model for functions that are not well modelled by the GPM. With the outputs of the mux dataset, simulating more points in the unbiased stage makes it closer to pure luck. As such luck is the only way that could help in the situations when an output has an isolated spike, it was decided to remove mux dataset from analysis in this work as including it would distort the true performance of the algorithms presented in this work. No algorithm can deterministically find a truly worst-case isolated spike in the output function. The dataset will be analysed separately in Chapter 3. Isolated spikes are an interesting structure in some datasets and, unfortunately, this is a valid problem that IC designers have to deal with.

2.4 Related Work

2.4.1 Similar Research

Related work was done by Dr. Xin Li's group at Carnegie Melon University (Dr. Li is currently working in Duke University). This work concerns statistical modelling[38][39][40] and incorporating the models into production flow to optimise production[41][42]. Although this work is not directly applicable to this project, it could indicate several directions for future work, and could demonstrate how these concepts are applied in production flow.

Some of this work is extremely close to the subject of this project, and thus it will be reviewed in detail. For example, in [38], a process is described that translates the problem of extracting the worst-case corners by formulating it as quadratically constrained quadratic programming with the help of a quadratic response surface model based on a DoE design, and then converts it to a convex semi-definite programming problem for extra efficiency. This approach works if a quadratic response surface model is an accurate model for the circuit, which is not always the case. Also, for our problem, the DoE design is the only input.

In [39], a way is presented to approximate the performance of circuits that are highly influenced by variability as the linear combination of a set of basis functions. The coefficients for the linear regression come from Maximum A Posteriori (MAP) estimates based on the (prior Laplacian-distributed) errors of the regression model. The error distribution is then itself approximated as a Gaussian kernel density distribution. As a result, the modelling is more precise than the SR-L1 methodology (sparse regression method with L_1-norm) on which the approach is based. This approach is interesting in its use of kernels that are linear combinations of Gaussian distributions. Some of these ideas can be used to build more suitable kernels for GPR based on the initial training set; however, building a good representation would require a large fraction of the small datasets, and thus it is inapplicable for this project.

In [40], the authors suggest an approach to model the response of circuits as linear surfaces at subdivided patches of the grid, essentially performing piecewise linear approximation of the surface. As a result, they get an accurate representation of the response function. This is applied to two common circuits, a 6T SRAM cell and a two-stage operational amplifier, and the results are within a few percent of the true response functions. However, these also require a thousand pre-selected corners to build, which is not always possible in this project. Additionally, if the bitcell and opamp1 datasets provided for this project are indeed functions of an SRAM cell and an operational amplifier, then the tested circuits in [40] are not so difficult to model, as Shoniker's algorithm has no trouble with any of the functions in the two datasets. Finally, an error of several percent is still too imprecise for the purposes of this project.

2.4.2 Other Function Optimisation Techniques

There are several widely accepted function optimisation techniques, like gradient descent[43], hill climbing[44], and simulated annealing[45]. Like all general function optimisation techniques, they are not guaranteed to find the global optimum, and are vulnerable to getting stuck in a local optimum.

For example, gradient descent looks at the gradient (e.g., the slope) of the cost function at the present point, and selects the next sample in the direction of the biggest change (steepest descent). The risk here is that the final answer for the lowest point of the cost function is very dependent on the starting point from which the descent started. Some modifications were proposed

to increase the probability of not getting stuck in a local minimum. For example, a momentum term, which takes into account the previous values of the cost function, can be included. Another option, multi-start strategies, avoid local optima by starting the search at different points. These, however, still do not guarantee convergence to the global minimum for any reasonable time. Hill climbing is similar to gradient descent, except that it optimises in one direction at a time, and has the same problems as gradient descent.

Simulated annealing attempts to increase the probability of finding the global maximum, but it still cannot be guaranteed to do so, and may only find a local maximum. The possible changes in the state of the system at a point in time for simulated annealing are affected by its temperature parameter. The temperature parameter is lowered in steps as the simulation proceeds and affects the probability that a higher cost next state will be selected instead of a lower cost state. The process starts at a random reference point, and then a random sample is selected, the distance in the input space between the reference point and the sample depends on the temperature. At the beginning when the temperature is high, the next sample is allowed to be far from the reference, whereas at the end when the temperature is low, the selection is less likely to explore regions far from the current candidate. Then, the sample is evaluated. If the value of the cost function at the sample is higher than reference, the sample is selected as the new reference. If it is not higher, the sample can still be chosen as the new reference, with probability that exponentially decays with decreasing temperature. This allows for more thorough exploration phase at the beginning of the algorithm. However, the maximum found with simulated annealing is still vulnerable to converging to a local optimum. Simulated annealing struggles with flat functions, of which we have found several in our industrial benchmarks.

2.4.3 Formal Verification

Formal verification is application of logical operations to exhaustively prove that a system corresponds to a set of constraints. Formal verification can be for both software[46] and hardware[47][48][49][50] systems. These methods are exhaustive, but also take a long time to develop the mathematical tools, and are generally circuit specific[51], and cannot be applied to arbitrary black-box functions. As this project requires a universal method of finding the worst-case estimation of a black box function, none of the formal verification strategies can be applied.

2.4.4 Rare-Event Failure Estimation

In cases when extreme reliability is required, a designer needs to apply so-called rare-event failure estimation techniques. Typically, such requirements would be applicable either to components that are used in huge numbers, like SRAM cells[52], or to complex systems for the purposes of verification[53]. A common approach here is subset simulation (SUS), where corners are chosen iteratively, selected by Markov Chain Monte Carlo (MCMC)[54], in order to calculate the probability of failure in a given subset[55]. Then, a subset of the previously selected subset of corners is selected, and the probability of failure within that subset is estimated again, giving the conditional probability of failure for the current subset. The eventual probability of failure is equal to the product of the conditional probabilities of failure for each subset.

The rare-event failure estimation approach could potentially be applied here if a high-sigma termination threshold is defined, and the search space is regenerated after each convergence. This could prove to be an attractive area of further research.

2.4.5 Applications in Other Fields of Engineering

The function optimisation problem considered in this thesis does not necessarily need to be applied only to the design verification of integrated circuits. In fact, process optimisation is a very important problem for a number of engineering disciplines, for example process optimisation in chemical engineering [56][57][58][59][60], where the exact proportions of reactants, as well as the process conditions, like pressure and temperature, need to be optimised subject to constraints for the maximum yield of the desired product.

Chapter 3: Deeper Look into Benchmark Datasets

3.1 Shapes of Datasets

The datasets supplied to us by Solido Design Automation provided a variety of functions, supposedly from real industrial circuits, for us to explore using different candidate algorithms for minimal worst-case corner selection. We do not know how the datasets were produced, and we can only guess at the characteristics that the functions model from the names of the datasets and the functions. Table 3.1 summarises the output functions available to us.

Dataset	Input	Unique Values	Output	Min, Max, Mean, and Median Values
bitcell	Model_set	ff, fs, sf, ss, tt	blwm	0.2241, 0.5775, 0.385958333333, 0.3862
	HI	0.9, 1.0, 1.1		
	LO	0	blwm_mv	224.1108, 577.5151, 385.956115, 386.1763
	Temperature	-25, -50, 0, 100, 125, 25.0, 50, 75		
charge_pump1	Model_set	ff, ss, tt	boostcr	0.7993, 0.8132, 0.805446759259, 0.8052
	Temperature	-25, -40.0, 0, 100, 125, 25, 50, 75		
	gnd	0	eq_error	0.000897, 0.0774, 0.017937125, 0.011055
	hvnn_fs_vnds	-40	holdcrd	0.917, 0.9347, 0.92617962963, 0.9262
	vcc	2.4, 2.5, 2.6		
	vcce	1.5	holdcru	0.9229, 0.938, 0.930111111111, 0.93005
	vin	26		
	vv3	2.4, 2.5, 2.6	ovdrive	1.712, 2.031, 1.88417592593, 1.8855
charge_pump2	Model_set	ff, ss, tt	boostcr	0.7945, 0.815, 0.804158641975, 0.8044
	Temperature	-25, -40, 0, 25, 50, 75		
	gnd	0	eq_error	-0.001558, 0.1231, 0.0158857746914, 0.0069705
	hvnn_fs_vnds	-40	holdcrd	0.9148, 0.9372, 0.925291975309, 0.92495
	vcc	2, 2.5, 3		
	vcce	1.4, 1.6	holdcru	0.9207, 0.9426, 0.930122222222, 0.92995
	vin	26	ovdrive	1.535, 2.142, 1.8972345679, 1.912
	vv3	2.4, 2.5, 2.6		
sense_amp1	Model_set	ff, fs, sf, ss, tt	SAspeed	3.476866e-11, 1.370766e-10, 6.147855625e-11, 5.548009e-11
	Temperature	-25, -50, 0, 100, 125.0, 25, 50, 75	glitch_senout	0.003463692, 0.1749941, 0.061537111925, 0.038181035
	bidirection_flag	0	maxout	0.540016, 0.5408317, 0.5402601125, 0.540239
	k	0.9	offset	-0.001171875, 0.008422852, 0.001993560785, 0.001785278
	preext_flag	0		
	saebgn	1.00E-11		
	saewid	7.9e-10, 8.1e-10, 8e-10	rslt	-1.080105e-05, 0.0002046455, 1.4537917e-05, -5.1428055e-06
	slwrt	2.00E-11	sen_dip	0.540016, 0.5408317, 0.5402601125, 0.540239
	vs	0	sen_dip_pctg	0.3990759, 0.3999823, 0.399711, 0.3997345

Dataset	Input	Unique Values	Output	Min, Max, Mean, and Median Values
bias_gen	Model_set	ff, ss, tt	bgr_m51_v145	1.143, 1.198, 1.17404166667, 1.1745
			bgr_m51_v150	1.153, 1.198, 1.17566666667, 1.1745
			bgr_m51_v155	1.158, 1.199, 1.17641666667, 1.1745
	Temperature	-25, -50, 0, 100, 125, 25.0, 50, 75	bgr_m51_v180	1.16, 1.201, 1.177, 1.1745
			bgr_m51_v195	1.16, 1.201, 1.17729166667, 1.175
			bgr_m51_v25	1.16, 1.202, 1.177875, 1.1755
			bgr_m51_v27	1.161, 1.203, 1.17816666667, 1.176
	vcc_ext	2.8, 2.9, 3.0, 3.1, 3.2	bgr_m51_v30	1.161, 1.203, 1.17841666667, 1.176
			bgr_m51_v33	1.161, 1.204, 1.178875, 1.1765
			bgr_m51_v36	1.162, 1.204, 1.17929166667, 1.177
opamp1	Model_set	ff, ss, tt	dc_gain	32.7, 35.16, 34.0946666667, 34.125
	Temperature	-25, -50, 0, 100, 125, 25.0, 50, 75		
	psweep	1		
	vcc	2		
	vm	0.005		
	vref	1.23, 1.24, 1.25, 1.26, 1.27		
shift_reg	Model_set	ff, fs, sf, ss, tt	delay	9.421e-09, 1.322e-08, 1.15065518519e-08, 1.1515e-08
	Temperature	-25, -50, 0, 100, 125, 27.0, 50, 75		
	vin_ac	0.05	fall_time	2.004e-10, 4.25e-09, 1.33582814815e-09, 1.097e-09
	vvcc	3.2, 3.3, 3.4		
	vvdd	1.4, 1.5, 1.6	rise_time	3.672e-10, 4.032e-09, 1.08265601852e-09, 8.8495e-10
	vvref	1.6, 1.65, 1.7		
buffer_chain	Model_set	ff, fs, sf, ss, tt	Tf4_5	3.332e-11, 1.635e-10, 6.78230183435e-11, 6.119e-11
	Temperature	-25, -50, 0, 100, 125, 25, 50, 75	Tr4_5	2.984e-11, 1.519e-10, 6.4019688716e-11, 5.792e-11
	cl	1.8E-015, 1.9E-015, 2.1E-015, 2.2E-015, 2E-015		
	in_slew	1.1E-010, 1E-010, 9E-011	avg_slew	30.0797, 114.8954, 56.5462939411, 52.2514
	myvdd	0.9, 1, 1.1		
	tp	0.00000002	avgdly4_5	31.5795, 157.7366, 65.9213255142, 60.1182
	tstart	0.000000001		
	tstep	1.00E-12	fslew	2.7e-11, 1.061e-10, 5.18013618677e-11, 4.758e-11
	tstop	0.0000001	rslew	3.316e-11, 1.237e-10, 6.12911839911e-11, 5.599e-11

Table 3.1 Information about the Inputs and Outputs of the Available Datasets

Table 3.1 summarises what inputs the datasets have, the possible input values, and gives brief information about the outputs. It is interesting to note from the output information that there is sometimes a significant difference between the mean and the median values of the output function. One might expect the distribution to be close to being uniform or Gaussian, meaning there will not be a big difference between the mean and the median. However, if the median is significantly lower than the mean, this indicates that the average of the function is skewed by the higher values, potentially indicating the presence of significant excursions or spikes in the responses of the function. Looking at the difference between the extremal values also might give interesting insights as a function that has a small difference between the minimum and the

maximum values (i.e., relatively flat functions) might make it difficult for the GPM to properly model all the small variations, which might be significantly affected by noise at the time of measurement.

The results achieved by Shoniker's final algorithm for the 4-sigma termination rule are provided in Table 3.2. The table shows results for 4-sigma confidence bound for a number of test circuits provided by our industrial partner Solido Design Automation. The average speedup is 4.74x, with the speedup defined as

$$Speedup = \frac{Total\ number\ of\ input\ points}{Points\ evaluated\ before\ finding\ the\ max} \tag{3.1}$$

Circuit Name	Type of Set	Initial Training Set Size	Corners to Find F_{max}	Corners Until Termination for each output	Min., Max. and Mean Speed-Ups
shift_reg	Full Factorial	14 of 1080	84-306	478, 988, 1047	1.03, 2.25, 1.46
buffer_chain	Full Factorial	20 of 1799	29-45	159, 104, 134, 99, 365, 121	4.93, 18.18, 13.32
bitcell	Full Factorial	10 of 120	12	26, 26	4.61, 4.61, 4.61
charge_pump1	Full Factorial	16 of 216	4-23	38, 41, 43, 45, 50	4.32, 5.68, 5.02
charge_pump2	Full Factorial	16 of 324	8-23	53, 56, 61, 68, 71	4.56, 6.11, 5.31
sense_amp	Full Factorial	20 of 120	1-7	34, 51, 81, 37, 35, 81, 114	1.05, 3.53, 2.37
bias_gen	Full Factorial	10 of 120	1	40, 41, 38, 36, 37, 38, 37, 37, 37, 37	2.93, 3.33, 3.18
op_amp	Full Factorial	12 of 120	1	46	2.61, 2.61, 2.61

Table 3.2 Results of Michael Shoniker's Original Algorithm

Each of the functions in Table 3.2 terminated with 100% accuracy. Accuracy is defined as the fraction of runs of the algorithm for which the true global maximum was found. So, for example, if in a test of 100 runs of the algorithm on an output function, 95 runs terminate to the correct worst-case corner (meaning that 5 times out of 100 an incorrect worst-case corner was chosen), the accuracy of the algorithm on this function is 95%. 100 trials were performed for every entry in Table 3.2. In each trial, a different initial training set is selected according to Michael Shoniker's randomized design (see Section 2.3.3). The results are averaged to produce the values in the column "Corners Until Termination for each output".

Some of the functions were very easy to search for the maxium value, while others turned out to be a challenge. Of course, in order to understand how to improve the algorithm, it is important to first understand why some of the functions are more difficult than the others, and then try to come up with a way to generalise and leverage that insight. Looking at the ordered histogram

distribution of the output values of the most challenging functions might provide some insight into this problem.

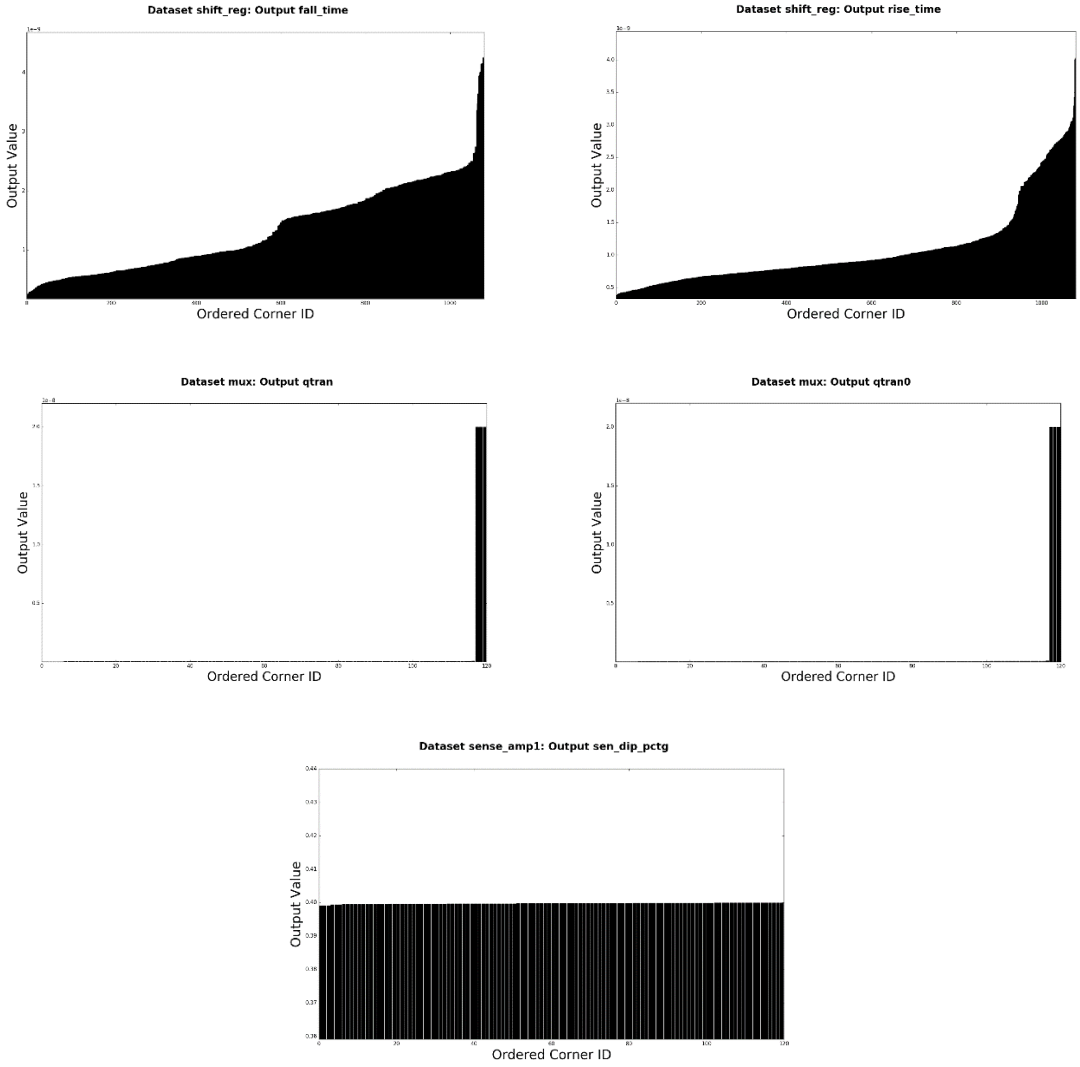


Figure 3.1 Shapes of Select Output Functions

The histograms in Figure 3.1 are for the hard fall_time and rise_time functions of the shift_reg benchmark dataset, the outputs of the mux dataset with three sudden spikes, and the very flat sen_dip_pctg function of the sense_amp1 benchmark. It seems that Michael Shoniker's algorithm had difficulty with at least two types of functions: ones that have sudden spikes, and ones that are very flat. Each of the types affects performance of the algorithm in a different metric of performance of the algorithm: spikes decrease the probability of finding the true global maximum, while flat regions require almost every point in the region to be simulated. And while it is undesirable to have a spike in the dataset, it is still often possible to model the underlying

function that produced the spike as it depends on the combination of the input parameters. Modelling flat surfaces causes GPM additional challenges for reasons that are explored in Section 3.2.

3.2 Overfitting and Underfitting

In the cases when a GPM is used to model what looks like statistical noise, the model still tries to ensure that the mean function goes through the training points exactly, resulting in irregular functions that tend to change a lot from one GPM generation stage of the algorithm to the next. This could be seen as a case of overfitting[61]. Overfitting is when the model is too complex to be useful for learning. In the case of neural networks, for example, extreme overfitting essentially means memorizing the entire training set, while simultaneously being rather poor at trying to interpolate between or extrapolate training samples to predict unknown samples. In the context of this project, overfitting might cause the predictions and uncertainties for test points to change rapidly from one stage of the algorithm to another. This is undesirable as it introduces even bigger uncertainty in selecting corners for simulation.

A common way to counter overfitting is to account for noise in the predictions. Then, the model is allowed to fit less precisely, essentially creating a soft margin [62] for regression, thus allowing for simpler function shapes. In scikit-learn 0.16.1, the software package used for this thesis, this is achieved by setting the nugget parameter of the gaussian_process object to a bigger value. Testing this on the most challenging dataset (shift_reg) yields the results in Table 3.3. The value for the nugget was based on the average value of the three functions in the shift_reg dataset, which is 4.64E-09.

Nugget	Average Corners	Average Speedup	Accuracy
4.64E-11	815.97	1.32	100
4.64E-10	862.57	1.25	100
4.64E-09	803.60	1.34	100
4.64E-08	846.07	1.28	100

Table 3.3 Results of Allowing Noisy Predictions in GPM for the shift_reg Dataset

As can be seen from Table 3.3, there is no significant difference between noiseless predictions and noisy predictions, which suggests that overfitting is not a problem. This is

supported by closely observing performance on the most difficult functions at every stage. From those observations, it looks like the model performance on the datasets were more likely due to underfitting. Underfitting is the opposite problem to overfitting – the model is not expressive enough to be used for accurate function regression. For example, if the task is to model a sinusoid, but the model can only produce a linear function, then the resulting prediction will simply be a flat 0 function, that is, the mean of the sinusoid. With some functions, a GPM is unable to properly model the functions due to, for example, the poor fit of the kernel function. In these cases, the GPM just produces a 0 mean function and the uncertainties based on the proximity to the training points.

While the predicted values of the functions for problematic output functions, like rise_time, are not 0, they are still often more than 100 times smaller than the respective uncertainties. In contrast to that, in the well-behaved delay function from the same shift_reg dataset, the values of the predictions at test points are very rarely less than 100 times bigger than the respective uncertainties. While it would be inappropriate to conclude that underfitting is the bigger problem for the rise_time function, the eventual conclusion is the same: for such functions, optimisation is largely shifted into the exploration phase as the next corners selection procedure will be dominated by the value of the uncertainties, and these points are likely to be those that are in the farthest regions from the trained points. This also means that the algorithm will take longer to converge than it should. Unfortunately, we could find no simple and reliable heuristic to prevent that without sacrificing accuracy.

3.3 Flat Functions

An additional problem introduced by functions that have maxima hidden in “flat” regions is that we can not be sure that the sampling method that generates the input to the algorithm for selecting worst-case corners will contain the true worst-case corner. For example, consider the functions of a phase detector circuit described in Chapter 6. All of them are rather flat and, having generated a finer dataset, we can see how the sampling of the input space affects what can be selected as the worst-case corner.

Number of Corners for the Sampling Method for phase_det_down	Worst-Case Value for phase_det_down down_rise_time	Worst-Case Value for phase_det_down down_fall_time	Number of Corners for the Sampling Method for phase_det_up	Worst-Case Value for phase_det_up up_rise_time	Worst-Case Value for phase_det_up up_fall_time
Total Sampling (9450 Corners)	1.00877856305E-11	9.29257005458E-12	**Total Sampling (9450 Corners)**	1.02241882614E-11	9.32792900107E-12
Partial Sampling 1 (1944 Corners)	1.00877451023E-11	9.29257005458E-12	**Partial Sampling 1 (1944 Corners)**	1.02221401286E-11	9.13006671489E-12
Partial Sampling 2 (6300 Corners)	1.00877451023E-11	9.29257005458E-12	**Partial Sampling 2 (6300 Corners)**	1.02238923073E-11	9.19881168078E-12
Partial Sampling 3 (1800 Corners)	1.00877451023E-11	9.15371593837E-12	**Partial Sampling 3 (1800 Corners)**	1.02238923073E-11	9.32792900107E-12
Partial Sampling 4 (4050 Corners)	1.00877451023E-11	9.15371593837E-12	**Partial Sampling 4 (4050 Corners)**	1.02241882614E-11	9.32792900107E-12
Partial Sampling 5 (1400 Corners)	1.00877451023E-11	9.29257005458E-12	**Partial Sampling 5 (1400 Corners)**	1.02238923073E-11	9.32792900107E-12
Partial Sampling 6 (504 Corners)	1.00877451023E-11	9.29257005458E-12	**Partial Sampling 6 (504 Corners)**	1.02221401286E-11	9.19881168078E-12

Table 3.4 Summary of Several Sampling Procedures of Functions Generated by a Phase Detector Circuit

Table 3.4 and Table 3.5 list some of the sampling procedures of the four functions, and the respective global maximums. The main observation from Table 3.4 is that no matter the sampling procedure, there is always a risk of not sampling the true global maximum of the continuous response function. The Total Sampling procedure is itself a full-factorial design of 9450 corners in each case, and selecting sub-samples, even if to bring the number of corners to a reasonable level more often than not misses that maximum. Even selecting over 4000 corners still does not capture the biggest value of one of the four functions, and selecting 6300 corners misses both of the maximum points in the phase_det_up dataset.

This problem has been noted as a fundamental problem of the corner analysis[63]. Choosing a discrete combination of parameters always carries the risk of underestimating the variance of responses, and the only truly reliable choice is a long-running and expensive Monte-Carlo simulation. However, as we did not have any information on the provided circuit, and only had a limited time to analyse our own custom circuits, we must assume that the input space given to us is a comprehensive sample of the function that must be optimised.

Sampling Procedures for phase_det_down			Sampling Procedures for phase_det_up		
Sampling Procedure Name	Input Names	Unique Input Values	Sampling Procedure Name	Input Names	Unique Input Values
Total Sampling	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)	Total Sampling	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 3.75e-10, 5e-10, 6.25e-10, 7.5e-10, 8.75e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -8.75e-10, -7.5e-10, -6.25e-10, -5e-10, -3.75e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)		Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)		Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
Partial Sampling 1	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)	Partial Sampling 1	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 5e-10, 7.5e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -7.5e-10, -5e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 4e-11, 5e-11)		Fall Time of Adjustable Clock	(3e-11, 4e-11, 5. e-11)
	Rise Time of Adjustable Clock	(3e-11, 4e-11, 5e-11)		Rise Time of Adjustable Clock	(3e-11, 4e-11, 5. e-11)
	Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
Partial Sampling 2	Temperature	(-25.0, 25.0, 50.0, 100.0)	Partial Sampling 2	Temperature	(-25.0, 25.0, 50.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 3.75e-10, 5e-10, 6.25e-10, 7.5e-10, 8.75e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -8.75e-10, -7.5e-10, -6.25e-10, -5e-10, -3.75e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)		Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4.0e-11, 4.5e-11, 5e-11)		Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
Partial Sampling 3	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)	Partial Sampling 3	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 6.25e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -6.25e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5 e-11)		Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5. e-11)		Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 4 e-11)		Rise Time of Reference Clock	(3e-11, 4e-11)
Partial Sampling 4	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)	Partial Sampling 4	Temperature	(-25.0, 0.0, 25.0, 50.0, 75.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 6.25e-10, 1 e-09)		Delay of Adjustable Clock	(-1e-09, -6.25e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5. e-11)		Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5 e-11)		Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
Partial Sampling 5	Temperature	(-25.0, 100.0)	Partial Sampling 5	Temperature	(-25.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 3.75e-10, 5e-10, 6.25e-10, 7.5e-10, 8.75e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -8.75e-10, -7.5e-10, -6.25e-10, -5e-10, -3.75e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)		Fall Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)		Rise Time of Adjustable Clock	(3e-11, 3.5e-11, 4e-11, 4.5e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 4e-11)
Partial Sampling 6	Temperature	(-25.0, 100.0)	Partial Sampling 6	Temperature	(-25.0, 100.0)
	Delay of Adjustable Clock	(2.5e-10, 3.75e-10, 5e-10, 6.25e-10, 7.5e-10, 8.75e-10, 1e-09)		Delay of Adjustable Clock	(-1e-09, -8.75e-10, -7.5e-10, -6.25e-10, -5e-10, -3.75e-10, -2.5e-10)
	Fall Time of Adjustable Clock	(3e-11, 5e-11)		Fall Time of Adjustable Clock	(3e-11, 5e-11)
	Rise Time of Adjustable Clock	(3e-11, 5e-11)		Rise Time of Adjustable Clock	(3e-11, 5e-11)
	Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Fall Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)
	Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)		Rise Time of Reference Clock	(3e-11, 3.5e-11, 4e-11)

Table 3.5 Unique Values of the Input Dimensions for the Different Sampling Methods of the Phase Detector Functions

3.4 Spikes and Ridges

In some functions (like rise_time), we observe ridges, where adjacent input points have relatively high values compared to the surrounding points. In fact, a ridge is a region that is flat along one (or a small number) of dimensions having significantly larger output values than the rest of the dataset. The three by far highest values of the rise_time function have input values (Model_set, Temperature, vin_ac, vvcc, vvdd, vvref) of (sf, -25, 0.05, 3.4, 1.4, 1.65), (sf, -25, 0.05, 3.4, 1.5, 1.65), and (sf, -25, 0.05, 3.4, 1.6, 1.65), the only difference being vvdd parameter. From Shoniker's observations, it was noted that GPM tends to significantly underestimate the values of the function in the neighbourhood of the current known maximum. The proposed solution was to introduce a three-step amplification factor for the predicted uncertainties near the current maximum. This observation can explain why ridges would give problems to the current setup.

Naturally, it would be beneficial to somehow detect if ridges exist in the dataset and to make suitable adjustments in the search strategy. Unfortunately, there is no good solution to doing that. The only way would be to try and guess that the algorithm has encountered a ridge is to sample the immediate neighbours of a point. This would mean that many more points are selected for next step simulation, which is inefficient and costly for the functions which do not cause problems for the GPM. Even restricting this additional sampling only to the cases when a new maximum is known can have significant adverse effects on the performance.

We investigated a method to look for ridges as follows. Whenever a new maximum or a new minimum is found, every point within one manhattan step from the current known maximum is selected for sampling. After each of those corners is simulated, their output values are compared to the current known maximum. If the values at one of those corners are within 10% of the current known maximum, relative to the distance between the current known minimum and the current known maximum of the function, the corner is marked as a ridge. Since with one manhattan step the input values of the corners differ only in one dimension, the unsimulated corners in that entire dimension are selected for simulation at the next stage.

Since many points are selected in this way, it only makes sense to do so for big datasets, like shift_reg. There was, however, no significant benefit observed. When running the ridge

heuristics with the global boosting factor, no benefit is observed at all for shift_reg dataset, which includes the only functions that could potentially benefit from this method. The average respective speedup and accuracy is 1.28x and 100% for the 4-sigma termination rule. Excluding calculating the boosting factor produces speedup of 2.40x and accuracy of 97.17%. While the accuracy is better than the accuracy of 95.33% that was observed in Shoniker's work for the 4-sigma termination rule without the global boosting factor, and the speedup is impressive, the accuracy is still too far below the target accuracy 99.86%, and so the method was decided to be inapplicable to the project. The low accuracy could be explained by the fact that stumbling upon a point on a ridge is rather random in the first place, so the ridge heuristic is only as reliable as the next corner selection procedure is reliable at finding a spike.

3.5 Conclusions

This chapter explored the structure of the provided circuit datasets and identified some of the problems that were encountered when applying Shoniker's best algorithm. It was noted that the biggest challenges are provided by the functions that have either sudden irregularities (e.g., spikes), or, conversely, are flat. Both provide problems of overfitting and underfitting.

The problem of overfitting was analysed on the example of the provided datasets. It was found that allowing softer margins for regression, which often helps with overfitting, did not give any significant benefit. This points to the problem for the challenging functions being studied (the outputs of the shift_reg dataset) being closer to underfitting, and closer observing states of simulation supports that.

The problems of flat functions are primarily caused by the fact that it is very difficult to model them, and find the true global maximum. Indeed, even the choice of the sampling procedure introduces a significant problem of missing the true worst-case corner, as studied on the custom dataset. As most datasets provided for us are of unknown origin, there is not much we can do about this problem, so we will have to assume that the datasets contain the true worst-case corner for the respective functions.

It was noticed that spikes in datasets are often located close to one another, producing something close to ridges or *n*-dimensional plateaus. When trying to account for such structures in

the function, specifically looking for ridges does not seem to help, possibly because locating ridges is a matter of luck in the first place.

The general conclusion from this chapter is that there is no simple solution to the problems experienced by Shoniker's algorithm. The best we can do is to try to narrow the regions of locating the spikes by attempting to prune away regions that are not likely to have such spikes. For example, partitioning the input space could increase the chance to encounter a spike or a ridge and thus make algorithms terminate faster and more accurately, while also allowing to prune away unlikely regions to focus computational effort on the promising regions. The partitioning approach will be explored in more detail in Chapter 5.

Chapter 4: Pruning Singular Points

4.1 The Problem of Termination

The framework of the algorithm that searches for the worst-case corners of an arbitrary black-box function can be separated into three subproblems: (1) initial exploration of the specified function of the given circuit, (2) selection of the next corners to simulate, and (3) termination of the search for the function maximum. This chapter explores the effects of the choice of different termination heuristics on the performance of the Shoniker's algorithm for selecting a minimal set of worst-case corners.

4.1.1 Premature Termination

The goal of the algorithm is to find the global function maximum in the fewest number of PVT corner simulations. A search that terminates too fast should raise suspicion about the accuracy of that termination decision. For example, output rise_time of circuit shift_reg was sometimes found to converge to a candidate corner for the global maximum with the 10-sigma confidence heuristic having selected as few as 17 corners (speedup of 63.5x). In many of those cases, the found corner did not correspond to the actual worst-case corner. In general, rise_time was found to take fewer than 100 simulations to converge to an incorrect corner in approximately 3% of the cases, even with algorithm improvements like the boosting factor for the predicted error produced by the GPM. Such a high error rate in termination is not acceptable and an improved heuristic needs to be introduced to avoid these problems.

Having collected the results of simulations for 2744 of such termination errors for the rise_time output of shift_reg, it was found that the most common reason for premature termination on the wrong corner is that sometimes the output values of the corners that lie on the convex hull in an iteration of the algorithm are significantly higher than the previous known maximum. In Figure 4.1, this effect is demonstrated. The small blue symbols (the 26 corners, in this case) at the left along the vertical axis are the ones that were used to construct the present GPM, and the big green dots are the true values of the points lying on the convex hull (black points on the lower right). The red point below the convex hull is the estimate of the true global maximum by the GPR.

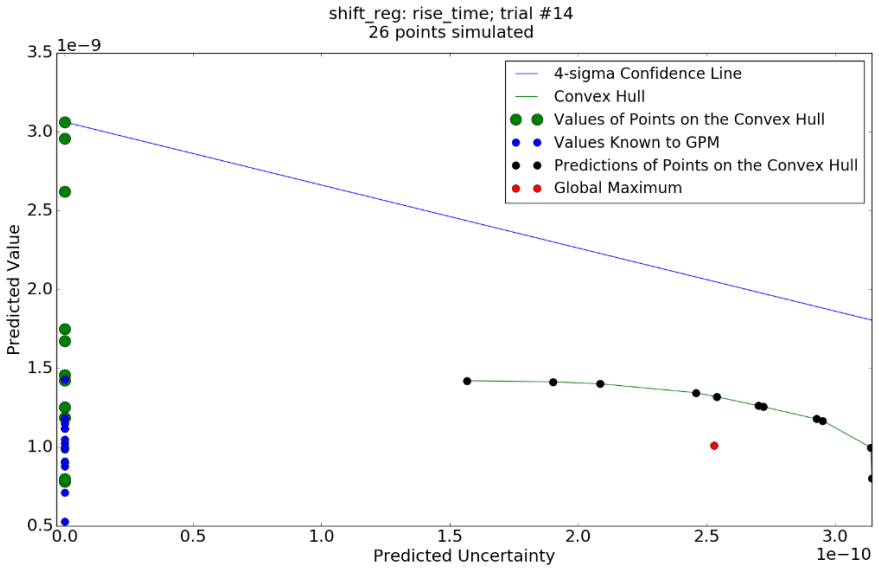


Figure 4.1 Undesired Dislocation of the 4-Sigma Confidence Line

However, since the points on the convex hull are not part of the training set that was used to construct the present GPM, the resulting function value predictions and uncertainties can be greatly underestimated. In the case of Figure 4.1, the highest value known by the GPM (the "old" maximum, the highest blue symbol on the figure) is more than twice as small as the highest value among the points selected for the next iteration (the "new" maximum, the highest green symbol from which the confidence line extends). Had the confidence line extended from the highest trained value, the algorithm would not have terminated at that stage. Figure 4.2 summarises by how much the "new" maximum is bigger than the "old" maximum.

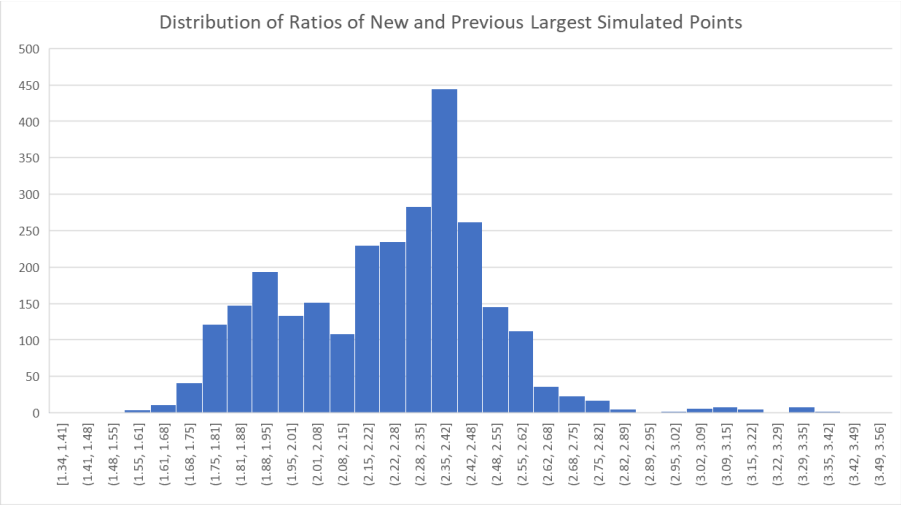


Figure 4.2 Distribution of Ratios of New and Old Maxima

Figure 4.2 was constructed by running 90000 trials of Shoniker's algorithm on the rise_time output of the shift_reg dataset. As the problem of early termination was being investigated, each trial only continued either until termination was declared or until 100 corners had been selected. If the algorithm did indeed terminate with fewer than 100 corners, it was recorded whether or not the global maximum was found, as well as the biggest value on the convex hull and of the points that were the basis for the GPM were recorded. Some of those cases (71 out of 90000) did indeed terminate with the true global maximum; however, even they had the same problems described in Figure 4.2. The 71 cases just happened to select the true global maximum in the few corners that they looked at out of good luck.

From Figure 4.2 then, we note that most of the time the new maximum is more than twice as big as the old maximum, with the smallest such ratio being equal to 1.34. This can barely be seen on Figure 4.2, as there was only one such case, compared to nearly 450 cases of the ratio being between 2.35 and 2.42. To account for this information, a new heuristic was developed. If the new maximum is 33% bigger than the old maximum, the search is not allowed to terminate, and so the next iteration of the algorithm is started unconditionally. While this heuristic would likely not allow early termination at the true global maximum, such cases would only account for 0.04% of all cases, according to the trials performed to generate Figure 4.2. Running this heuristic on the rise_time function, the algorithm terminates with 4-sigma confidence after simulating an average of 991.70 points out of 1080, with a termination accuracy of 99.82% compared to 96.95% without the heuristic. As can be seen, it takes a similar number of simulations to terminate with this heuristic, compared to the times when there was no premature termination. One interesting thing to note from these tests is that terminating with 4-sigma confidence produces the termination accuracy that should be expected from the 3-sigma confidence termination rule. This phenomenon will be further explored in Section 4.1.2.

The results of applying this heuristic on every dataset are summarised below in Table 4.1. As can be seen, the results are not significantly different from the ones provided in Table 3.2. The gains are explained by the reduction in the number of input dimensions, as described at the end of Section 2.1.4, and the termination accuracies are approximately what should be expected from the 3-sigma termination rule. Note, however, that for these results the k=4 termination rule was used, and therefore the accuracy would be expected to be higher, following the assumptions of Michael

Shoniker's work. In reality, however, the k-sigma rule leads to termination failure statistics that are rather different from the Q(k) single-tailed distribution function. The likely reason for this is explored in Section 4.1.2. The average speedup across all datasets is 4.99x, and the average termination accuracy is 99.94%. Thanks to the reduced size of the initial training set, more simulations were saved, while the accuracy remained at an acceptable level, compared to the results of applying Shoniker's algorithm.

Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bias_gen	37.29	3.22	100
bitcell	26.16	4.59	100
buffer_chain	171.25	13.48	99.83
charge_pump1	35.29	6.21	100
charge_pump2	58.038	5.72	100
opamp1	47.31	2.54	100
sense_amp1	59.81	2.60	100
shift_reg	818.59	1.55	99.67
Average Speedup	4.99	**Average Accuracy**	99.94

Table 4.1 Results of Applying a Safeguard Against Premature Termination

4.1.2 Understanding the Meaning of the σ-threshold

In Shoniker's thesis, the 3-sigma confidence level was taken to be the benchmark metric for evaluating the performance of the various versions of the algorithm for minimal worst-case corner selection. The reasoning behind this decision was that, for Gaussian random variables, that would provide the expected termination accuracy of 99.865%, as described by the Q-function that represents the tail probability of a Gaussian distribution for a certain standard deviation σ (σ = 3 in this case). That is, the algorithm would be expected to fail to find the global worst case corner in only one out of 740 runs, which, according to our industrial partner, would be an acceptable error rate for such tasks in the industry. However, there are some conceptual flaws with this expectation. Note that for GPR each test point, for which a prediction is made, is considered to be an independent (from other test points) Gaussian random variable, the above reasoning would be valid if we only have one unsimulated point below the 3-sigma confidence threshold. If there are, however, multiple points below the threshold, the total expected accuracy $E[acc]$ would be the product of factors where each factor is 1 minus the Q-function for the σ levels of each predicted test point. That is, the expected total accuracy would actually be equal to

$$E[acc] = \prod_{i=0}^{n}\big(1 - Q(\sigma_i)\big) = \prod_{i=0}^{n}\varphi(\sigma_i) = \varphi\big(\sigma_{eff}\big) \tag{4.1}$$

for the number n of unsimulated points and the function $\varphi(\sigma_i) = 1 - Q(\sigma_i)$ providing the complement of the Q-function. It is clearer now that using the same 3-sigma confidence threshold as before will lead to overly optimistic expectations since there is almost always more than one point under the confidence line at the time of termination for almost every function under analysis. Moreover, having as many points as possible below the confidence threshold at the time of termination is the entire point of the algorithm. As such, a more in-depth analysis of what we mean by confidence of termination is needed.

Fortunately, due to the roughly exponentially decaying nature of the Q-function, the effect of multiple equally confident corners is not overly significant. The σ_i-confidence level of the corner i is calculated according to the expression:

$$\sigma_i = \frac{current\ known\ maximum - estimated\ value_i}{uncertainty_i} \tag{4.2}$$

Performing some easy algebraic transformations to show the effective σ_{eff}-confidence level of the number u of unsimulated points having the same σ_i-confidence level, and to show the required σ_i-confidence level for u points to have the effective σ_{eff}-confidence level, the following expressions are obtained:

$$\varphi\big(\sigma_{eff}\big) = \prod_{i=0}^{u}\varphi(\sigma_i) = \varphi(\sigma_i)^u$$

$$\ln\Big(\varphi\big(\sigma_{eff}\big)\Big) = u * \ln\big(\varphi(\sigma_i)\big)$$

$$\sigma_{eff} = \varphi^{-1}\big(\exp\big(u * \ln\big(\varphi(\sigma_i)\big)\big)\big) \tag{4.3}$$

$$\sigma_i = \varphi^{-1}\Big(\exp\Big(\ln\Big(\varphi\big(\sigma_{eff}\big)\Big)/u\Big)\Big) \tag{4.4}$$

Thus the number $u > 1$ of unsimulated points that together produce the same σ_i-confidence as one point at the σ_{eff}-confidence level is

$$u = \frac{\ln\Big(\varphi\big(\sigma_{eff}\big)\Big)}{\ln\big(\varphi(\sigma_i)\big)} \tag{4.5}$$

This means that, for example, to achieve a target $\sigma_{eff} = 3$, we can have up to 42 points at $\sigma_i = 4$, and the number of corners required to make an effect on σ_{eff} for higher σ_i grows very fast. Figure 4.3 illustrates this.

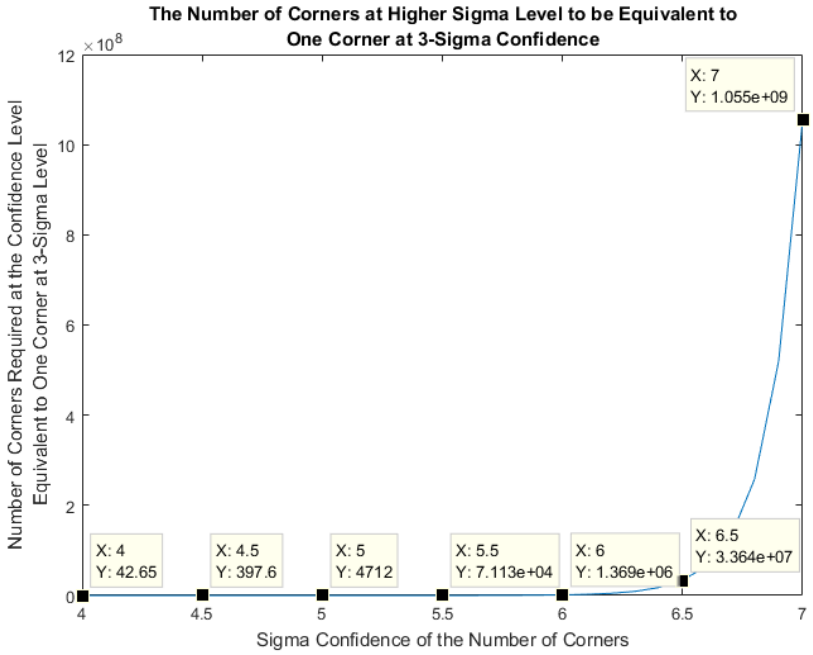


Figure 4.3 The Number of Unsimulated Corners at Confidence σ_i Required to Have the Same Confidence as One Corner at 3-Sigma Level

Since not every point will lie on the same sigma threshold at the time of termination, with most points having significantly higher individual sigma confidences, in Shoniker's final algorithm it was decided that simply increasing the termination threshold by 1 would be sufficient to approach the expected level of accuracy. As such, all of the results in Chapter 4 are reported for termination criterion at the 4-sigma confidence level.

4.1.3 Terminating Execution on a Point-by-Point Basis

An interesting way to decrease the number of simulated points that it takes to terminate at a global maximum is to look at when it is safe to exclude certain points from consideration when looking for the global maximum. This subsection describes several ways to go about that.

The ideas presented below are somewhat similar to the three-step amplification factor for the predicted uncertainty in the function estimates in Shoniker's algorithm, but instead of next

corner selection, it is applied to the termination stage of the framework. The purpose of the three-step amplification factor was to encourage exploration near the current candidate maximum, to ensure that the true global maximum is not missed just because the current known one is close and is good enough. The purpose of these new heuristics, however, is to take advantage of the predicted values and uncertainties at an earlier stage of simulation. As will be demonstrated, to achieve the desired accuracy, it will be necessary to look at a collection of points within a radius of a candidate for termination to be able to make a decision on whether or not it will be safe to do so.

4.1.3.1 The Naïve Algorithm

As previously described, GPR sees every test point for which it is required to make a prediction as being independent from all other predicted test points. For each one of the test points, then, the algorithm produces the estimated value $\hat{y}$ and uncertainty σ in the estimation of the value at the test point. This leads to an idea that certain points can be said, with high enough confidence, to be very unlikely to be bigger than the global maximum, and thus these points can be safely excluded (that is, pruned away) from consideration at an earlier stage of the simulation.

Following the calculations in Section 4.1.2, those equations can be reformulated with u being the number of points in a dataset. This is a somewhat pessimistic estimation of σ_{eff} as it assumes that every point will be considered for termination (i.e., below the σ_{eff} termination threshold), and a significant fraction of points from the dataset may already have been selected for simulation, rather than being involved in the termination decision. Additionally, many points will be at a level of confidence significantly higher than σ_{eff} at the time of termination. However, it is better to have more safety if the points are going to be excluded completely, especially if the GPM cannot model the function accurately, so these considerations provide reasonable σ_{eff}. The effective σ_{eff}-confidence levels are shown in Table 4.2

Size of Set	Datasets of this Size	σ_{eff}
120	bitcell, sense_amp1, bias_gen, op_amp	4.2384
216	charge_pump1	4.3685
324	charge_pump2	4.4563
1080	shift_reg	4.708
1799	buffer_chain	4.811

Table 4.2 Dependence of σ_{eff} on the Number of Corners in a Dataset to Produce the Target 3-Sigma Confidence

As expected, the heuristic provided a significant reduction in the number of simulated points required to reach termination on the global maximum, however with lower overall accuracy. Table 4.3 summarises the results of using this procedure.

Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bias_gen	32.65	3.68	100.00
bitcell	21.12	5.68	100.00
buffer_chain	116.61	16.78	99.17
charge_pump1	29.71	7.38	99.60
charge_pump2	46.51	7.06	98.40
opamp1	42.93	2.80	100.00
sense_amp1	48.47	3.07	98.14
shift_reg	632.44	2.21	93.33
Total Average Speedup	6.08	**Total Average Accuracy**	98.58

Table 4.3 Results of Applying Pointwise Termination of Datasets

The average speedup this way is 6.08x, and the average termination accuracy is 98.58%. This accuracy is not acceptable for an important application such as the finding of the worst-case corner for the purposes of design verification, and therefore developing a more reliable termination rule is required.

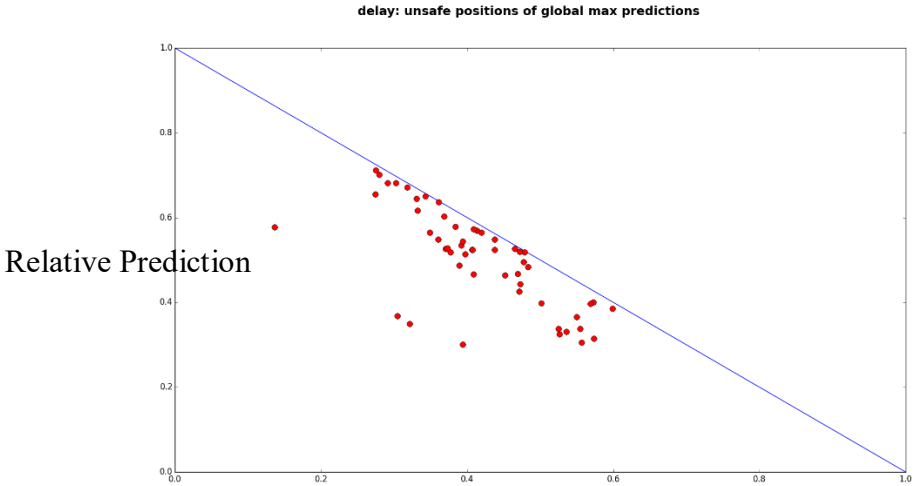
4.1.3.2 Experiments with a Stricter Termination Threshold

While reducing the average number of required simulations for convergence was a welcome improvement, the resulting reduction in termination accuracy meant that the heuristic needed more work. Stricter termination rules were therefore considered.

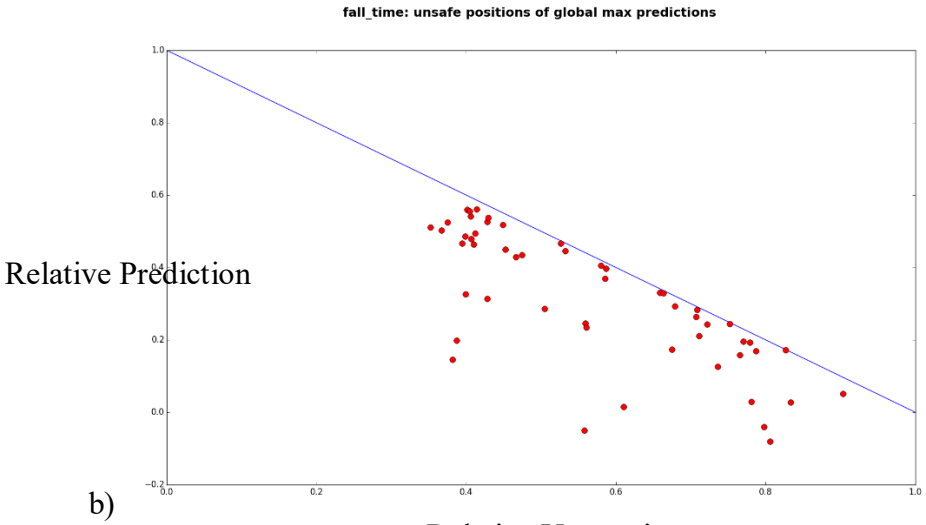
For the termination rule described in Section 4.1.3.1 to yield the best accuracy, the actual global maximum should never fall below the termination threshold. As the shift_reg dataset that had by far the biggest difficulty with termination accuracy, that dataset was studied carefully for insights when developing a stricter and hopefully more accurate termination rule.

An initial idea was to have a curved termination threshold, rather than the straight one defined by the sigma confidence equation $y = y_{max?} - k * \sigma_{pred}$. It was noted that often when the predictions and uncertainties for the unsimulated true maximum would place that point below the termination threshold, the point would be located in a small region just below the confidence threshold. It was decided to try to define a certain “dangerous zone”, where it would not be safe to prune away points, to eliminate the possibility of wrongly terminating the global maximum.

Essentially, this would make the termination threshold slightly stricter for a range of predicted uncertainties, the “dangerous zone”, while outside of it the threshold would remain at the defined sigma level. Performing a comprehensive visual analysis, however, proved that such an approach would be infeasible as the dangerous zone could essentially include the entire scatter area (see Figure 4.4), offering not even a theoretical benefit in the number of simulated corners to convergence for any dataset under analysis.



a)



b)

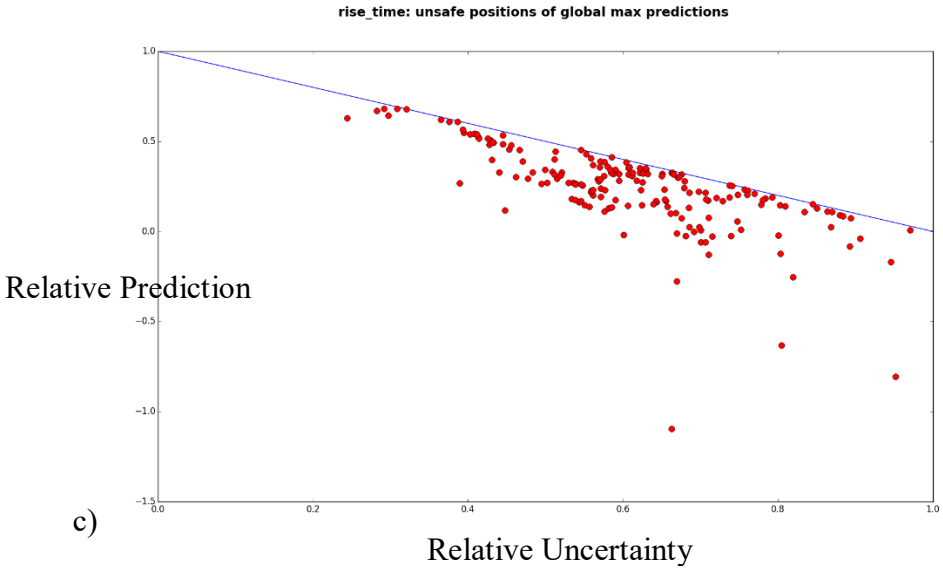


Figure 4.4 Unsafe Positions of the Global Max Relative to the σ_{eff} Confidence Line

In Figure 4.4 the vertical and horizontal positions are calculated by

$$\hat{y}_{relative} = 1 - \frac{y_{max?} - \hat{y}_{global\ max}}{y_{max?} - \hat{y}_{\sigma_{max}}} \tag{4.6}$$

$$\sigma_{relative} = \sigma_{global\ max}/\sigma_{max} \tag{4.7}$$

respectively, for k=4.708, where $y_{max?}$ is current known maximum for that stage of simulation where the global maximum was predicted to be pruned away, σ_{max} is the highest uncertainty at that stage of simulation, $\hat{y}_{\sigma_{max}}$ is the predicted value of the point with the highest uncertainty at that stage of simulation, and $\hat{y}_{global\ max}$ is the predicted value of the true global maximum for the function. In the figures, the main information is that the predictions of the global worst-case corner are often much lower than the σ-confidence termination line and that the true global maximum can sometimes be located much lower than the termination line. This makes it impossible to produce a reasonable curved termination threshold that would allow to easily prune away unsimulated corners.

Another way to approach the termination problem was to consider changing the parameters of the termination inequality $\hat{y} + k_{pointwise} * \sigma \leq \text{scale} * y_{max?}$, where $k_{pointwise}$ is a different sigma threshold, and scale is a factor that would place the benchmark value below the current known maximum value $y_{max?}$ relative to the current known minimum value. Mathematically,

$$\text{scale} = \frac{\hat{y}_{\sigma_{max}} + k_{pointwise} * \sigma_{max}}{y_{max?}} \tag{4.8}$$

where $y_{max?}$ is the current known maximum value, σ_{max} is the highest uncertainty of a test point at that stage of the simulation, and $\hat{y}_{\sigma_{max}}$ is the mean estimate of the point with σ_{max}.

To explore which parameter values in the new termination rule would give the best trade-off between complexity and termination accuracy, a series of a series of experiments were performed for the outputs of shift_reg dataset. Whenever the point with the true global maximum was predicted to be below the $k_{pointwise}$-sigma confidence threshold, the state of the simulation (namely, the $\hat{y}$ and σ values for every test point) was recorded for further analysis. This information allowed for the creation of scatter plots showing the position of the global maximum on the convex hull scatter plots relative to the $k_{pointwise}$-sigma confidence line (see Figure 4.4).

The desirable set of parameters would be a low $k_{pointwise}$ and a high "scale", with a higher $k_{pointwise}$ being more favourable than a low scale. The combinations of the parameters that caused the point of the global maximum to be above the termination line were sought. Among all the tested combinations, $\text{scale} = 0.85$ and $k_{pointwise} = 8.5$ was selected as the most appropriate (see Figure 4.5), having affordable rates of failure. This means that it is assumed safe to terminate every corner that is predicted to be below the 8.5-sigma confidence line stretching from 85% of the value of the current known maximum. Unfortunately, even these conditions turned out to be too strict to produce any improvement on the original results. The results are provided in Table 4.4 combine this threshold with the combined confidence threshold of 4-sigma for all points. That is, the search terminates if every point is below the 4-sigma confidence threshold.

Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bias_gen	37.15	3.23	100.00
bitcell	25.87	4.64	100.00
buffer_chain	166.54	13.26	100.00
charge_pump1	34.96	6.27	100.00
charge_pump2	57.03	5.82	99.00
opamp1	47.29	2.54	100.0
sense_amp1	58.36	2.62	99.71
shift_reg	813.47	1.55	99.33
Total Average Speedup	4.99	**Total Average Accuracy**	99.76

Table 4.4 Results of Stricter Threshold Simulations

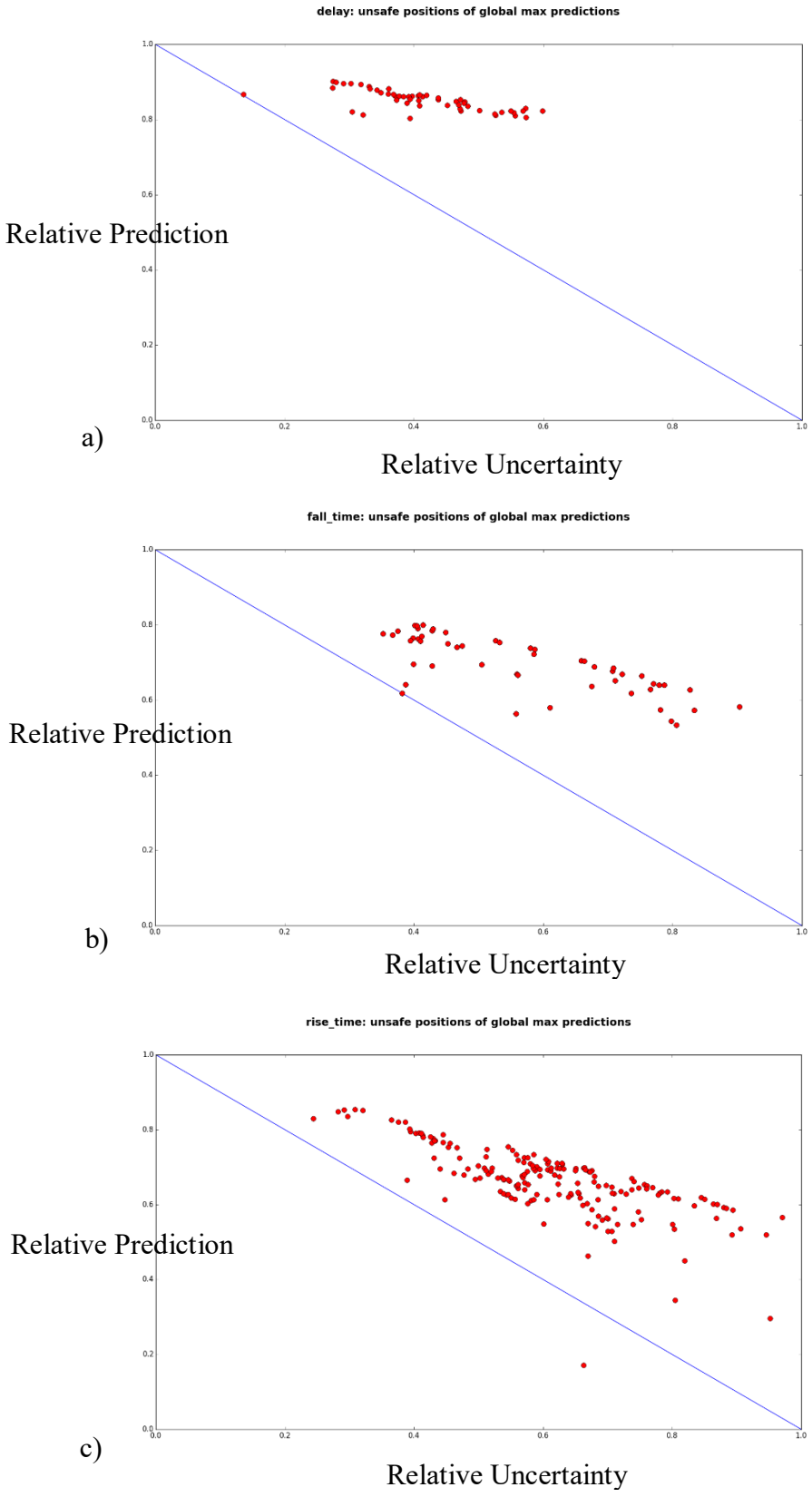


Figure 4.5 Relative Positions of Global Maxima Estimates for scale=0.85 and $k_{pointwise}$=8.5

With the average speedup of 4.99x, and 99.76%, there is no noticeable benefit compared to the results produced by Shoniker's original algorithm, and thus there appears to be no point in exploring pointwise termination in this way. Section 4.1.3.3, however, describes another approach to pointwise termination that considers the predictions and the uncertainties of neighbouring points.

4.1.3.3 Neighbourhood Effects and Hybrid Termination

At this point, it is clear that it is difficult to find a reliable and simple termination criterion where the unsimulated points are considered separately. However, by modifying the pointwise termination rule to one that considers the neighbouring points, we found that a further slight improvement in termination accuracy can in fact be achieved over all previously described algorithms.

Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bias_gen	36.29	3.31	100.00
bitcell	25.45	4.72	100.00
buffer_chain	156.81	13.48	99.50
charge_pump1	33.85	6.48	100.00
charge_pump2	55.60	5.96	99.00
opamp1	46.66	2.57	100.00
sense_amp1	58.21	2.65	99.57
shift_reg	788.12	1.63	98.67
Total Average Speedup	5.10	**Total Average Accuracy**	99.59

Table 4.5 Results of Neighbourhood Termination

In this new termination rule, a point is ruled out as the global maximum only if all the points within a certain radius of the point under consideration are themselves below the confidence boundary. This radius would be calculated as a suitable fraction of the diameter of the dataset under consideration. Here, the diameter is defined to be the biggest distance between any two points in the dataset. To determine the appropriate radius, information was collected on every case where the true global maximum ended up below the 4-sigma confidence line. The 4-sigma confidence line was chosen since now the termination criterion for the entire dataset does not have to pass all 1080 times for shift_reg dataset, and thus $k_{pointwise}$ was assumed to be too strict of a threshold. The fraction value for the radius was swept to determine the lowest fraction of the diameter for which the true global maximum would never be selected for termination, provided

that not every point is under the 4-sigma confidence line. The best fraction was determined to be 0.3 (that is, 30% of the biggest distance between two points in the input space) and results are provided in Table 4.5. The average speedup was 5.10x, and the average accuracy was 99.59%, slightly below the desired accuracy. This proved that $k_{pointwise}$ was not too harsh for radial pointwise termination, and therefore the decision boundary was set to the σ_{eff} threshold again.

Achieving one final improvement in termination accuracy requires determining how many points are in the neighbourhood of the candidate global maximum point under consideration for termination. For the raw input value set for the true global maximum of the outputs in shift_reg dataset, these values were 675, 405, and 540 points, respectively. However, by performing standardisation (removing the mean and reducing the standard deviation from the distribution of the values along each dimension to 1) on the input values before, these numbers drop to 96, 94, and 113 points, respectively, while the fraction of the radius increases to 40% of the diameter of the dataset. From a run of simulations, it was observed that performing standardisation on the input values and applying Shoniker's algorithm did not have any significant negative effect on the accuracy of termination (see Table 4.6), and as such there is no downside to doing this kind of preprocessing. In fact, the buffer_chain dataset had great improvements in termination accuracy thanks to standardisation preprocessing. Consequently, the neighbourhood analysis heuristic was combined with Shoniker's original 4-sigma confidence rule of termination.

Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bitcell	25.76	4.66	100.00
charge_pump1	35.34	6.22	99.80
charge_pump2	58.59	5.71	100.00
sense_amp1	59.99	2.58	100.00
bias_gen	37.32	3.22	100.00
opamp1	48.04	2.50	100.00
shift_reg	820.63	1.53	100.00
buffer_chain	137.65	15.37	100.00
Average Speedup	5.22	Average Accuracy	99.98

Table 4.6 Performance after Applying a Standardisation Preprocessing Step

The combination works as follows. After each simulation, every remaining unsimulated point is tested for termination, with the calculated σ_{eff} (see Table 4.2) and with the radius of the neighbourhood of the point being set to 0.4 of the diameter of the entire dataset. However, if every point is below the 4-sigma confidence threshold, the execution is terminated all the same and the

current known maximum is declared to be the global maximum. The results of this heuristic are provided in Table 4.7.

Dataset	Average Corners to Convergence	Individual Function Speedups	Average Speedup	Average Accuracy
bias_gen	36.81	3.08, 3.02, 3.29, 3.37, 3.34, 3.29, 3.29, 3.32, 3.33, 3.32	3.26	100.00
bitcell	24.47	4.86, 4.92	4.90	100.00
buffer_chain	157.34	12.35, 18.18, 13.74, 18.20, 5.23, 14.33	13.67	100.00
charge_pump1	34.69	7.12, 6.21, 5.44, 7.30, 5.54	6.32	99.80
charge_pump2	55.46	5.08, 5.55, 7.16, 7.08, 5.05	5.98	99.20
opamp1	46.00	2.61	2.61	100.00
sense_amp1	56.26	4.15, 2.90, 1.62, 3.92, 3.96, 1.59, 1.06	2.74	99.43
shift_reg	818.42	2.42, 1.04, 1.11	1.52	99.33
Total Average Speedup	5.13	**Total Average Accuracy**	99.72	

Table 4.7 Results of the Hybrid Termination Heuristic

The average speedup is then 5.13x, and the average termination accuracy is 99.72%. This is mainly due to improvements in the termination of functions in shift_reg dataset. The termination accuracy is at the lower boundary of the accuracy objective, but this is mainly due to the problem of premature termination, as described at the beginning of this chapter

4.2 Chapter Discussion

In this chapter, the subject of what termination means using point-level heuristics was explored. It was found that there are limitations to looking at each test point as a separate case. While that is exactly what GPR attempts to do, such an approach can backfire when analysing a function that perhaps was not the result of a process easily described by a Gaussian Process with a set covariance function.

It becomes apparent that some theoretical pitfalls of Shoniker's work are sometimes balanced out by unexpected benefits. So, for example, if the value of one test corner is greatly underestimated, the value of another test corner might be greatly overestimated. As such, the average effect of errors across many test points works out to approximately what one would expect from a heuristic that should produce results with, for example, 3-sigma confidence. In particular, this could explain why the model developed by Shoniker would produce uncertainties of a

magnitude noticeably smaller than desired by his model, and the need to introduce the global boosting factor to compensate for that as fewer corners remain unsimulated. Indeed, as the number of test points becomes smaller, the boosting factor grows larger, signalling that the GPM is getting more and more inaccurate.

This insight might explain why looking at neighbourhoods of points produces much better results compared to focusing on points on an isolated basis. It could be that GPM predictions have limited regions of effect, meaning that the predictions for the center of the region of effect are more reliable when looking at all the points in such area. Indeed, GPR exploits the correlation between closely located points by tying the predictions of a test point to nearby training points.

The next step taken was to investigate the identification of safely pruned regions, rather than safely pruned points. Perhaps by performing relatively simple preprocessing steps (e.g., partitioning), there might be a good chance of quickly learning which areas can be safely pruned away from the search for the global maximum and thus focus computational effort on the most promising partition. This approach of partitioning also provides a framework for allowing different heuristics used in different partitions. This study is the focus of the next chapter.

Chapter 5: Pruning Groups of Points

5.1 *k*-means Clustering on Input Values

From the results of Section 4.1.2, we learned that one of the Shoniker algorithm's key strengths is to prune away individual points from consideration when searching for the global maximum. With respect to a scatter plot diagram (predicted function value vs. predicted uncertainty), this occurs when a point falls below the sigma confidence line. An extension to this idea would be to prune away entire input regions, containing multiple unsimulated points, from consideration. In this chapter, we explore the effects of partitioning the input space into clusters using a popular clustering technique: *k*-means clustering.

5.1.1 A Straightforward Approach

The first idea that we investigated was to perform *k*-means clustering on the inputs for various assumed numbers of clusters. *k*-means clustering [64] is an unsupervised machine learning technique that attempts to partition an input space into $k > 1$ disjoint subspaces. The number k of subspaces is specified by the user. The fundamental task of *k*-means clustering is to assign each point to one of the k clusters, S_i, with mean $\boldsymbol{\mu_i}$, $1 \leq i \leq k$, so that the total sum of the within cluster sum of squares (WCSS) is minimised. The WCSS is defined as follows:

$$WCSS = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu_i}\|^2 \tag{5.1}$$

The problem of minimising WCSS is NP-hard[65][66], so every practical implementation must use heuristics to partition the set of points into clusters. This can result in irregular boundaries between clusters, and this introduces an additional source of variation when testing the algorithm for the minimal selection of worst-case corners, as will be shown later. The cluster means $\boldsymbol{\mu_i}$ are generated randomly at first. Then, the input points are assigned to clusters based on the proximity to the generated means. The means are then updated to the mean position of the points selected to the cluster, and the clusters are regenerated, resulting in new mean points. This continues until the

assignments have not changed between iterations. Note that this iterative process does not guarantee optimal WCSS.

The experiments showed that there is the potential to have more accurate average predictions for test points for the outputs of the most difficult dataset shift_reg, compared to the "no clustering" case, as illustrated in Figure 5.1.

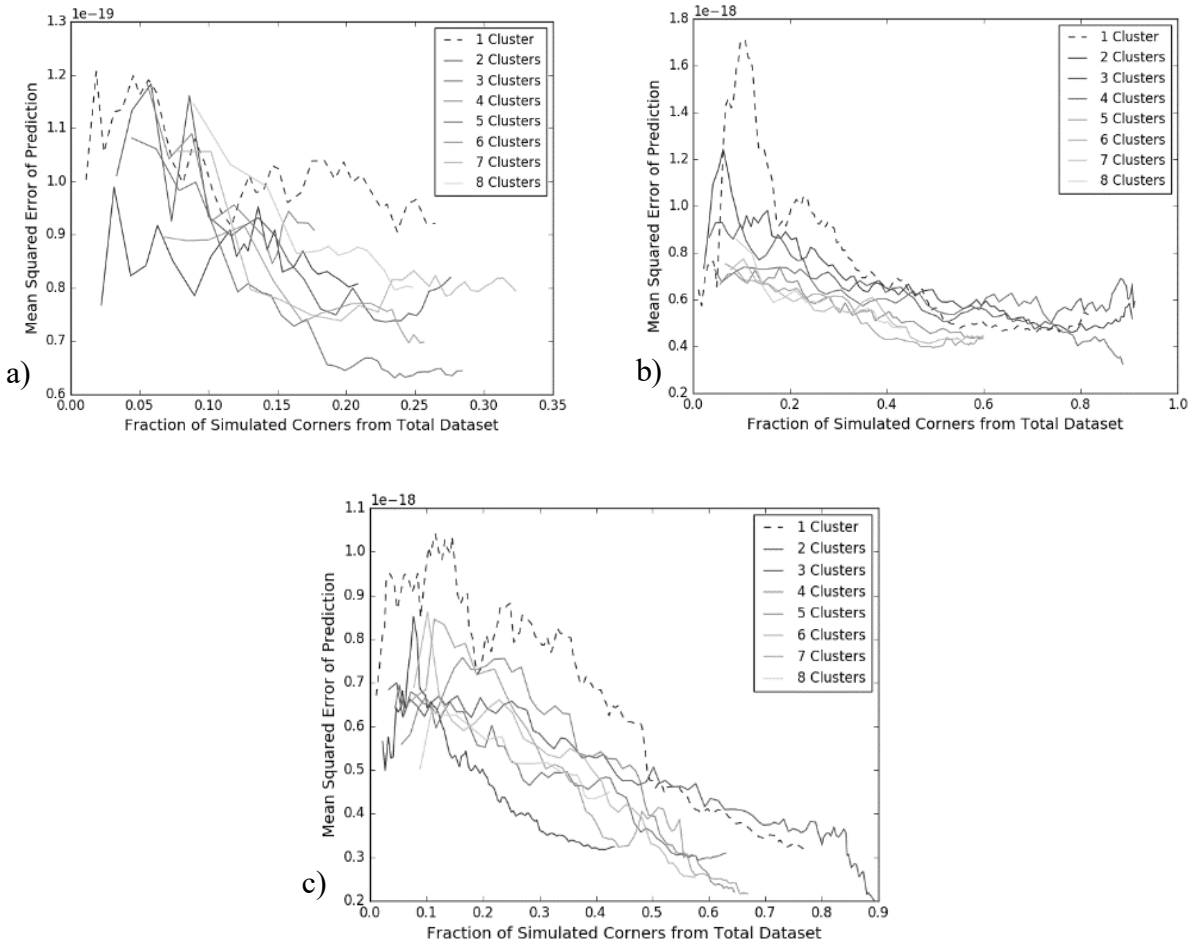


Figure 5.1 Fractions of Simulated Corners vs. Mean Squared Errors of Predictions for the (a) delay, (b) fall_time, and (c) rise_time Outputs of the shift_reg Datasets

Figures 5.1 illustrate the mean squared error of prediction versus the fraction of simulated corners for the corners in the shift_reg dataset for different numbers of clusters. The mean squared error of prediction is calculated as

$$\overline{error} = \frac{1}{N_{test\ points}} \sum_{i=0}^{N_{test\ points}-1} (true\ value_i - \hat{y}_i)^2 \tag{5.2}$$

k-means clustering was performed on the input space of shift_reg, and Shoniker's algorithm was applied to each cluster independently. In the figures, it can be clearly seen that the "no clustering" case (dashed lines) almost universally leads to higher errors from predictions $\hat{\boldsymbol{y}}$ at any stage of simulation, compared to the multiple clustering cases, which suggests the models become more accurate. It is not clear what the optimal number of cluster is, so in the trials, the number of clusters will be swept between 2 and 8 to better understand how performance is influenced by each setup.

Figure 5.2 illustrates how partitioning the input space into a number of clusters (five, in the case of Figure 5.2) can make it easier for the search algorithm to find the global maximum.

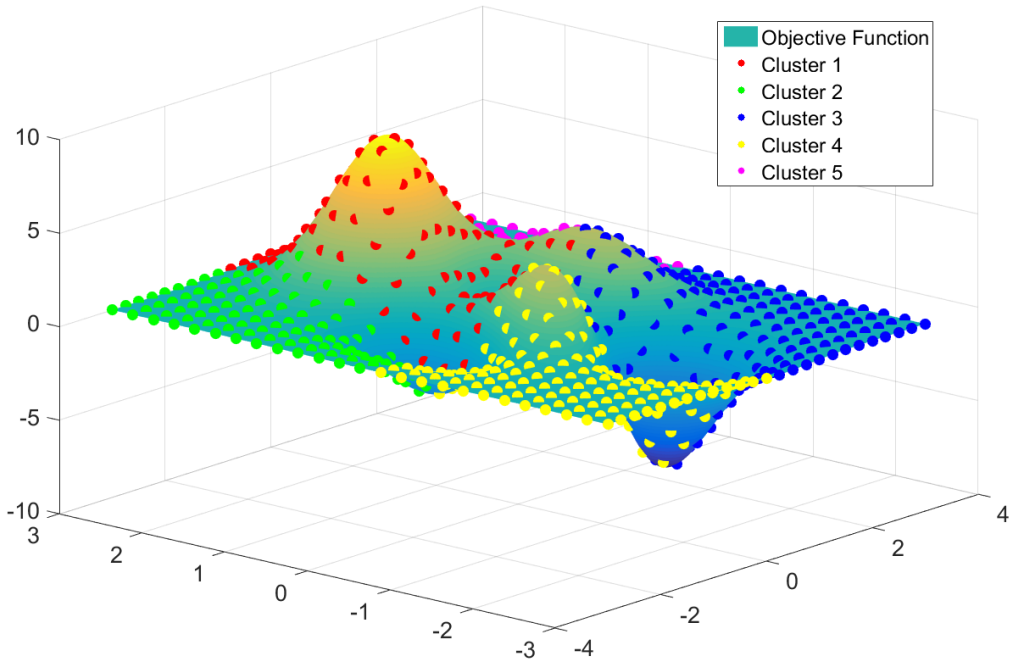


Figure 5.2 Demonstration of the Benefits of Separating the Input Space into Clusters

In Figure 5.2, note that the high peak is completely within Cluster 1, compared to the relatively flat four other clusters. A properly designed cluster-based algorithm would quickly understand that the regions described by Clusters 2 through 5 can be safely pruned away, so the computational effort can then be focused on Cluster 1. Thus, we conjectured that clustering has the potential to improve the performance of the minimal corner selection algorithm.

More applicable to this project are the distributions of the challenging functions fall_time and rise_time of shift_reg (see Figures 5.3 and 5.4, respectively). Before the figures were constructed, a *k*-means clustering procedure was performed on the input space of the shift_reg dataset, with k=4. The figures reveal the distributions of the outputs of each of the four clusters for the two challenging functions rise_time and fall_time. It is clearly seen that one of the clusters (cluster no. 3 in the case of fall_time and cluster no. 0 in the case of rise_time) have the highest output values, and so, hopefully, the algorithm can quickly recognise that the other clusters are not likely to have the global maximum and will prune them away quickly. A potential difficulty could be the problem of underfitting described in Section 3.2; however, we still expect notable improvements, since the predictions are supposed to be much closer to the true values.

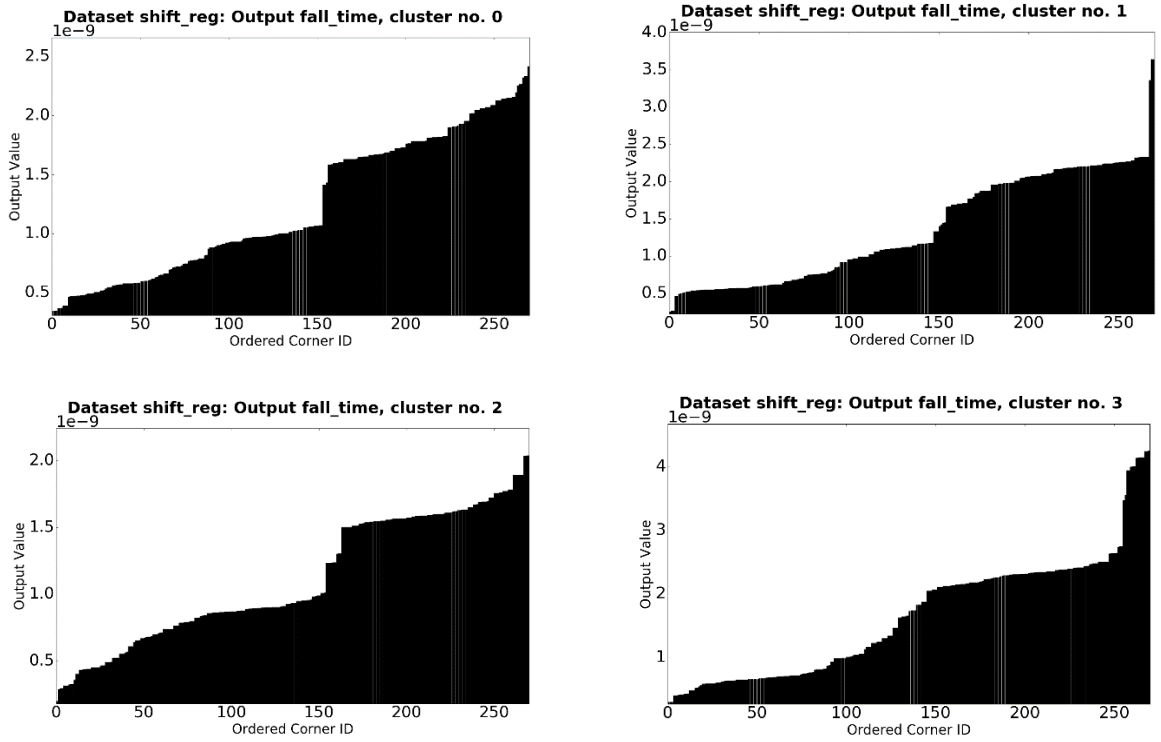


Figure 5.3 Shapes of the Output Distributions of the Four k-means Clusters of the fall_time Function

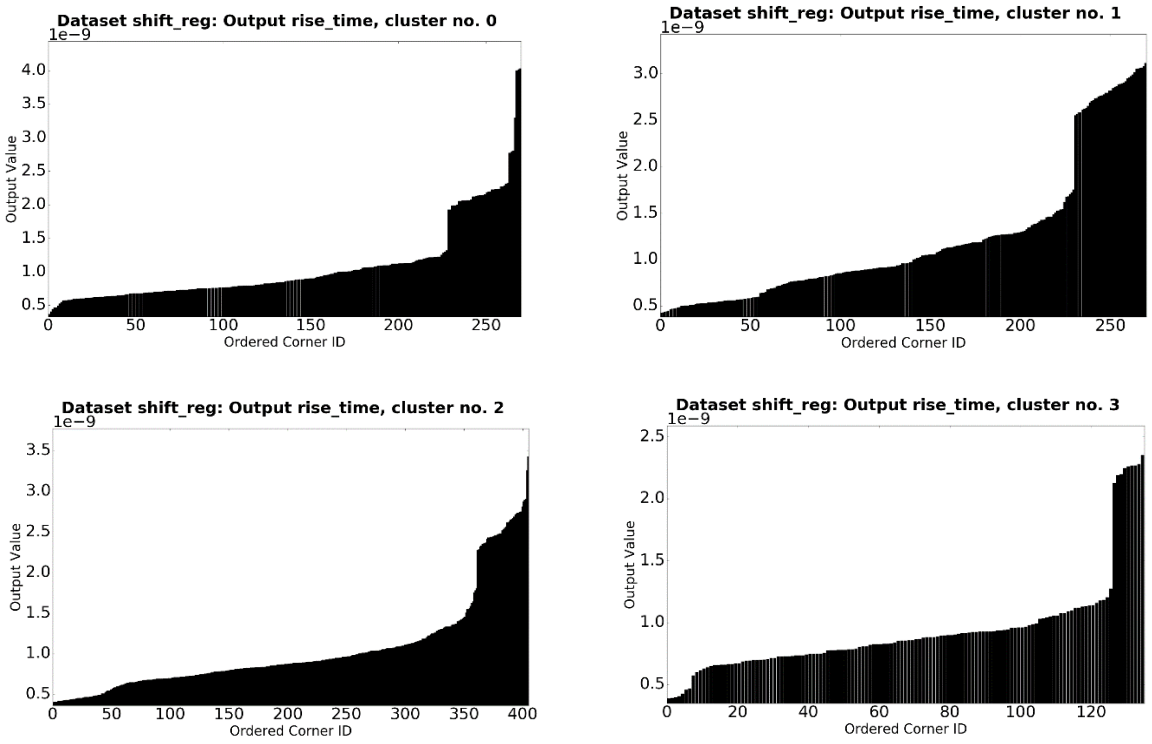


Figure 5.4 Shapes of the Output Distributions of the Four k-means Clusters of the rise_time Function

The proposed procedure is then as follows. Before starting Shoniker's algorithm, *k*-means clustering is performed on the input set, with the number of clusters increased from 1 (no clustering) up to 8. Then, for every cluster, the initial training set is selected according to the established design. No constraints are placed on the form of clusters, so the boundaries between them can be different from one trial to another. Once that is done, each cluster constructs its own GPM, independent of the GPMs of other clusters. Similarly to Section 4.1.3, a cluster can be terminated (or pruned) if every point in the cluster is under the sigma-confidence line defined by the 4-sigma confidence threshold and the maximum value from the corners found in every cluster (i.e., the current known global maximum). Execution for each cluster continues until every cluster has been terminated, and the maximum over all clusters is declared the global maximum. Results of this heuristic are provided in Table 5.1. In Table 5.1, the results in italics with an asterisk denote cases where not every output terminated in all 100 out of the 100 trial runs with the true function maximum. Note that the termination accuracy is rather low for some outputs, particularly rise_time of shift_reg. In general, the termination accuracy seems to drop as the number of clusters rises.

Dataset	Avg. Speedup No Clustering	Avg. Speedup 2 Clusters	Avg. Speedup 3 Clusters	Avg. Speedup 4 Clusters
shift_reg	*1.33**	*1.61**	*1.81**	*1.89**
buffer_chain	11.11	*9.94**	*9.01**	*8.35**
Big Datasets Average	**6.22**	**5.78**	**5.41**	**5.12**
bitcell	4.7	3.21	2.57	2.11
charge_pump1	6.13	4.74	3.79	3.13
charge_pump2	*5.69**	*4.17**	*3.4**	*2.95**
sense_amp1	*2.01**	*1.89**	1.78	*1.55**
bias_gen	3.21	2.91	2.39	2.03
op_amp	2.59	2.14	2.16	2.07
Small Datasets Average	**3.93**	**3.17**	**2.70**	**2.35**
Overall Average	**4.60**	**3.83**	**3.36**	**3.01**
Dataset	**Avg. Speedup 5 Clusters**	**Avg. Speedup 6 Clusters**	**Avg. Speedup 7 Clusters**	**Avg. Speedup 8 Clusters**
shift_reg	*1.95**	*1.98**	*2.09**	*2.15**
buffer_chain	*7.76**	*7.35**	*6.9**	*6.54**
Big Datasets Average	**4.86**	**4.67**	**4.5**	**4.35**
bitcell	1.79	1.57	1.39	1.25
charge_pump1	2.76	2.48	2.22	2.02
charge_pump2	*2.62**	*2.33**	*2.23**	*2.14**
sense_amp1	1.47	1.38	*1.28**	1.2
bias_gen	1.75	1.53	1.36	1.22
op_amp	1.77	1.55	1.39	1.25
Small Datasets Average	**2.07**	**1.85**	**1.70**	**1.57**
Overall Average	**2.73**	**2.52**	**2.36**	**2.22**

Table 5.1 Average Speedups for Achieving 4-sigma Confidence when Applying Clustering with Cluster Pruning

In addition, only shift_reg seems to benefit from clustering in terms of reducing the number of points simulated to get to convergence. This might be explained by the fact that the procedure for selecting the initial training set has a strictly fixed size that depends on the number of input dimensions and is bounded below by 10 to ensure that 10-fold cross-validation can be performed. For a dataset with many input dimensions, such as sense_amp1, the points to be simulated would be quickly exhausted, and as a result there would be hardly any potential benefit in this procedure. Indeed, the average speedup of small datasets drops noticeably faster than that of big datasets (3.93x to 1.57x vs 6.22x to 4.35x). Circuit buffer_chain is also poorly handled with clustering for the k=4 sigma rule, which might be explained by the fact that the dataset itself is rather "easy" to converge at that confidence level. At higher sigma levels, there seems to be some benefit from a higher number of clusters. This was the case for shift_reg.

One apparent benefit from using clustering is the reduction of the runtime of the algorithm for big and difficult datasets, like shift_reg. The effect is summarised in Table 5.2. The simulations ran in Python 2.7 on an Intel(R) Core(TM) i7-4790 CPU at 3.60 GHz frequency, using the scikit-learn 0.16.1 Python package.

Number of clusters	1	2	3	4
Time to complete, s	100%	~33.7%	~17.8%	~13.0%
Number of clusters	5	6	7	8
Time to complete, s	~10.7%	~9.62%	~8.42%	~4.41%

Table 5.2 Relative Runtime of Algorithm to 10-sigma Convergence

The effect can be explained by the fact that the total complexity of the underlying GPR algorithm is greatly reduced. The computational complexity of GPR is $O(n^3)$, where n is the number of training samples in the model. Using clustering, however, means that each model only needs to process n/k points in the k clusters. The overall complexity is then

$$O((n/k)^3 * k) = O(n^3/k^2) \tag{5.3}$$

So, to summarise, the most immediate benefit of applying clustering is a reduction in the runtime of the algorithm, and so there is possibly better performance (specifically, a lower number of required corner simulations) on big and difficult datasets. The downsides are reduced accuracy and often poor performance for small datasets. Further sections in this chapter will introduce several ways to compensate for these downsides.

5.1.2 A More Structured Initial Training Set

The clustering results presented in Table 5.1 are mixed. On the one hand, there is clearly a decrease in how many simulations it takes for a big and "difficult" dataset like shift_reg to converge to a maximum when increasing the number of clusters. On the other hand, there is also a clear decrease in the termination accuracy when finding the true global maximum, and a notable increase in the required number of simulations to converge for smaller and easier datasets. One possible explanation for both of these downsides is that the initial training set is not well suited for this heuristic. Indeed, as the size of the initial training set is fixed for every cluster, regardless of how many points it contains, the total size of the initial training set can grow to be unacceptable. For example, for sense_amp1, the size of the initial training set is 10 after removing the static dimensions, while the total size of the dataset is 120. This means that, on average, there can be at most twelve clusters before there is no possible room for improvement.

One way to tackle both of these problems is to use a different, more intelligently selected, design for the initial training set. In particular, we obtained good results with the Plackett-Burman design [29] described in Section 2.2.2.2. The procedure then is the same as in Section 5.1.1, with

the exception that now the initial training set for every cluster is selected according to a Plackett-Burman design, rather than the old procedure. The results of the experiments are summarised in Table 5.3.

These results are a clear improvement over those reported in Section 5.1.1 with regards to both the number of points simulated that are required to terminate at the global maximum, and the accuracy of finding the true global maximum. In particular, only the shift_reg run failed to find the true global maximum in all 100 trials out of 100, for every output function. What cannot be seen in Table 5.3, however, is the fact that when using many clusters with Plackett-Burman design, the accuracy for shift_reg drops to almost zero, possibly because the deterministic nature of the design makes it hard to correct for poor assumptions. That is, if the Plackett-Burman design was a bad sample of the input space in a cluster, it will likely be the case for the next trial as well. Another possible explanation is that the design gives a misleading start to the GPM for overly small clusters, and thus it is not advisable to use too many clusters. For up to three clusters, however, the Plackett-Burman design seemed to work well, which can be seen in that the average speed-ups for the big datasets are higher when using two and three clusters, rather than one.

Dataset	**Avg. Speedup No Clustering**	**Avg. Speedup 2 Clusters**	**Avg. Speedup 3 Clusters**	**Avg. Speedup 4 Clusters**
shift_reg	1.31	1.71	1.72	*1.88**
buffer_chain	9.55	13.22	11.2	8.34
Big Datasets Average	**5.43**	**7.47**	**6.46**	**5.11**
bitcell	4.97	3.53	2.45	1.95
charge_pump1	7.17	5.25	4.15	3.42
charge_pump2	6.11	4.33	3.49	2.98
sense_amp	1.99	2.29	2.36	2.31
bias_gen	4.66	3.53	2.84	2.65
op_amp	2.76	2.34	2.19	2.26
Small Datasets Average	**4.61**	**3.55**	**2.91**	**2.60**
Overall Average	**4.82**	**4.53**	**3.80**	**3.22**
Dataset	**Avg. Speedup 5 Clusters**	**Avg. Speedup 6 Clusters**	**Avg. Speedup 7 Clusters**	**Avg. Speedup 8 Clusters**
shift_reg	*1.86**	*1.93**	*1.96**	2.05
buffer_chain	7.71	7.42	6.97	6.96
Big Datasets Average	**4.79**	**4.68**	**4.47**	**4.51**
bitcell	1.84	1.76	1.68	1.62
charge_pump1	2.95	2.59	2.33	2.1
charge_pump2	2.63	2.34	2.24	2.18
sense_amp	1.89	1.65	1.46	1.3
bias_gen	2.26	2.01	1.81	1.64
op_amp	2.2	2.09	1.99	1.88
Small Datasets Average	**2.30**	**2.07**	**1.92**	**1.79**
Overall Average	**2.92**	**2.72**	**2.56**	**2.47**

Table 5.3 Average Speedup for Achieving 4-sigma Confidence using a Plackett-Burman Design for the Initial Training Set

5.1.3 Experiments with Preprocessing

One last bit of improvement concerns the speed of convergence. It was shown in [67] that linear GPR kernels struggle to correctly model step functions like *max*; however, by using even quadratic kernels, the representational accuracy is significantly improved. This leads to considering a preprocessing technique called polynomial features expansion. Now, before performing clustering, the input values are expanded to their polynomial features of degree 2. This includes squared terms for every input and every interaction term (products of two inputs). Otherwise, the process is the same as that described in Section 5.1.2. The results are provided in Table 5.4.

This technique provided major improvements for buffer_chain, and minor improvements for shift_reg, and, again only for a small number of clusters. Small datasets, on the other hand, did not see any benefit. As a result, it can be noted that these procedures benefit only on a case-by-case basis, and so this procedure will not be used for any future heuristics.

Dataset	**Avg. Speedup No Clustering**	**Avg. Speedup 2 Clusters**	**Avg. Speedup 3 Clusters**	**Avg. Speedup 4 Clusters**
shift_reg	1.67	1.75	*1.97**	*2.02**
buffer_chain	16.6	10.7	9.74	8.74
Big Datasets Average	**9.14**	**6.23**	**5.86**	**5.38**
bitcell	3.58	2.35	2.18	2
charge_pump1	5.84	4.5	3.75	3.48
charge_pump2	5.28	3.65	2.98	2.77
sense_amp	2.15	2.03	1.92	1.76
bias_gen	3.66	2.83	2.65	2.6
op_amp	2.18	1.9	2.23	2.15
Small Datasets Average	**3.78**	**2.88**	**2.62**	**2.46**
Overall Average	**5.12**	**3.71**	**3.43**	**3.19**
Dataset	**Avg. Speedup 5 Clusters**	**Avg. Speedup 6 Clusters**	**Avg. Speedup 7 Clusters**	**Avg. Speedup 8 Clusters**
shift_reg	*2.12**	*2.06**	*2.03**	*2.13**
buffer_chain	8.07	7.69	6.64	6.31
Big Datasets Average	**5.1**	**4.88**	**4.34**	**4.22**
bitcell	1.64	1.43	1.28	1.26
charge_pump1	2.91	2.79	2.45	2.29
charge_pump2	2.24	2.01	1.89	1.81
sense_amp	1.68	1.56	1.54	1.52
bias_gen	2.18	1.83	1.78	1.71
op_amp	1.83	1.62	1.65	1.62
Small Datasets Average	**2.08**	**1.87**	**1.77**	**1.70**
Overall Average	**2.83**	**2.62**	**2.41**	**2.33**

Table 5.4 Average Speedup for Achieving 4-sigma Confidence using a Plackett-Burman Design for the Initial Training Set and Quadratic Expansion of Input Features

5.2 Clustering Based on Learned Information about the Function

This section explores procedures that take into account the known values of performed simulations in the hope of arriving at the global maximum faster and more accurately. Several techniques were developed and the results of experiments are presented below.

5.2.1 Clustering on Output Values

As an experiment, it was decided to perform clustering on all outputs. Essentially, this is a test of what would happen were there a way to develop a perfect clustering technique that would separate the biggest output values in all of the datasets. The results give a best-case baseline for clustering. The two different heuristics for selecting the initial training set were tested, and the results are provided in Table 5.5 and Table 5.6. The results show general improvements for all metrics, particularly for a small number of clusters. Having received these promising results, it appeared that the algorithm would benefit from a more intelligently chosen clustering, which will be introduced in the following sections.

2 Clusters				3 Clusters				4 Clusters			
bitcell	34.47	3.48	100.00	**bitcell**	44.77	2.68	100.00	**bitcell**	56.72	2.12	100.00
charge_pump1	47.73	4.70	100.00	**charge_pump1**	53.90	4.01	100.00	**charge_pump1**	63.05	3.43	100.00
charge_pump2	59.12	5.52	100.00	**charge_pump2**	69.73	4.68	100.00	**charge_pump2**	80.73	4.05	99.80
sense_amp1	45.87	2.75	100.00	**sense_amp1**	50.43	2.45	100.00	**sense_amp1**	57.18	2.10	100.00
bias_gen	37.62	3.20	100.00	**bias_gen**	44.84	2.68	100.00	**bias_gen**	56.22	2.13	100.00
opamp1	49.02	2.45	100.00	**opamp1**	52.66	2.28	100.00	**opamp1**	55.32	2.17	100.00
shift_reg	403.01	2.74	97.00	**shift_reg**	248.66	4.40	98.33	**shift_reg**	209.46	5.28	99.00
buffer_chain	170.35	13.47	99.83	**buffer_chain**	163.54	12.48	100.00	**buffer_chain**	151.24	13.05	99.83
Average speedup	4.78	**Average Accuracy**	99.60	**Average speedup**	4.46	**Average Accuracy**	99.79	**Average speedup**	4.29	**Average Accuracy**	99.83

5 Clusters				6 Clusters			
bitcell	65.82	1.82	100.00	**bitcell**	76.02	1.58	100.00
charge_pump1	72.08	3.00	100.00	**charge_pump1**	84.39	2.56	100.00
charge_pump2	87.60	3.72	100.00	**charge_pump2**	97.95	3.32	100.00
sense_amp1	65.40	1.84	100.00	**sense_amp1**	77.51	1.55	100.00
bias_gen	70.73	1.70	100.00	**bias_gen**	76.79	1.56	100.00
opamp1	65.98	1.82	100.00	**opamp1**	76.82	1.56	100.00
shift_reg	179.76	6.27	98.33	**shift_reg**	175.01	6.29	91.67
buffer_chain	159.83	12.07	100.00	**buffer_chain**	171.96	10.96	100.00
Average speedup	4.03	**Average Accuracy**	99.79	**Average speedup**	3.67	**Average Accuracy**	98.96

7 Clusters				8 Clusters			
bitcell	87.13	1.38	100.00	**bitcell**	97.49	1.23	100.00
charge_pump1	95.51	2.26	99.80	**charge_pump1**	106.15	2.04	99.80
charge_pump2	106.24	3.06	100.00	**charge_pump2**	117.09	2.77	100.00
sense_amp1	87.55	1.37	100.00	**sense_amp1**	96.49	1.24	100.00
bias_gen	88.75	1.35	100.00	**bias_gen**	96.73	1.24	100.00
opamp1	88.22	1.36	100.00	**opamp1**	98.31	1.22	100.00
shift_reg	171.77	6.33	98.33	**shift_reg**	164.15	6.64	60.67
buffer_chain	183.09	10.18	100.00	**buffer_chain**	189.23	9.67	100.00
Average speedup	3.41	**Average Accuracy**	99.77	**Average speedup**	3.26	**Average Accuracy**	95.06

Table 5.5 Results of Clustering on Outputs using the Original Training Set Design

2 Clusters				3 Clusters				4 Clusters			
bitcell	36.00	3.33	100.00	**bitcell**	39.26	3.06	100.00	**bitcell**	51.00	2.35	100.00
charge_pump1	40.16	5.57	100.00	**charge_pump1**	48.28	4.49	100.00	**charge_pump1**	53.80	4.10	100.00
charge_pump2	60.50	5.38	100.00	**charge_pump2**	63.05	5.25	100.00	**charge_pump2**	63.22	5.20	100.00
sense_amp1	41.97	3.07	100.00	**sense_amp1**	41.33	2.96	100.00	**sense_amp1**	41.29	2.97	100.00
bias_gen	32.25	3.81	100.00	**bias_gen**	33.18	3.62	100.00	**bias_gen**	39.15	3.08	100.00
opamp1	46.03	2.61	100.00	**opamp1**	54.56	2.20	100.00	**opamp1**	44.46	2.70	100.00
shift_reg	401.86	2.74	100.00	**shift_reg**	251.13	4.38	100.00	**shift_reg**	199.32	5.65	100.00
buffer_chain	172.18	15.40	100.00	**buffer_chain**	169.45	13.33	100.00	**buffer_chain**	162.84	13.50	100.00
Average speedup	5.24	**Average Accuracy**	100.00	**Average speedup**	4.91	**Average Accuracy**	100.00	**Average speedup**	4.94	**Average Accuracy**	100.00

5 Clusters				6 Clusters			
bitcell	51.00	2.35	100.00	**bitcell**	49.00	2.45	100.00
charge_pump1	54.11	4.09	80.00	**charge_pump1**	61.98	3.64	80.00
charge_pump2	78.43	4.20	80.00	**charge_pump2**	82.48	3.98	80.00
sense_amp1	47.74	2.52	100.00	**sense_amp1**	50.71	2.38	100.00
bias_gen	49.46	2.43	100.00	**bias_gen**	51.76	2.33	100.00
opamp1	46.70	2.57	100.00	**opamp1**	50.00	2.40	100.00
shift_reg	171.26	6.59	85.00	**shift_reg**	162.12	7.13	100.00
buffer_chain	140.51	14.13	100.00	**buffer_chain**	144.33	13.36	100.00
Average speedup	4.86	**Average Accuracy**	93.13	**Average speedup**	4.71	**Average Accuracy**	95.00

7 Clusters				8 Clusters			
bitcell	49.00	2.45	100.00	**bitcell**	60.00	2.00	100.00
charge_pump1	67.67	3.26	80.00	**charge_pump1**	74.81	3.00	80.00
charge_pump2	87.56	3.73	60.00	**charge_pump2**	94.10	3.46	80.00
sense_amp1	54.14	2.22	100.00	**sense_amp1**	60.29	2.00	100.00
bias_gen	54.68	2.20	100.00	**bias_gen**	60.00	2.00	100.00
opamp1	53.00	2.26	100.00	**opamp1**	54.00	2.22	100.00
shift_reg	144.39	7.56	100.00	**shift_reg**	158.99	6.91	99.33
buffer_chain	137.04	13.51	100.00	**buffer_chain**	164.68	11.53	100.00
Average speedup	4.65	**Average Accuracy**	92.50	**Average speedup**	4.14	**Average Accuracy**	94.92

Table 5.6 Results of Clustering on Outputs using a Plackett-Burman Design as the Initial Training Set

5.2.2 Clustering Based on the Outputs of Points in the Initial Training Set

From the results in Section 5.2.1 it appears that there is potential benefit to performing a more intelligent partitioning of the input domain for the purposes of finding the global maximum. We considered performing the partitioning in the following way. The points in the initial training set are sorted into several ranks, the number of ranks being equal to the desired number of clusters, as selected by the user. The remaining points are then assigned to clusters according to their proximity to the closest point in the initial training set. Inside each of the clusters formed in this way, an additional initial training set is selected. Since there will not be a lot of variety in how the clusters are formed, the intra-cluster training set is selected according to the randomised procedure proposed by Michael Shoniker. Simulation was performed in the same way as the previous clustering approach otherwise. Up to 5 clusters were considered in this investigation. Within each cluster, an additional set of points was selected to properly sample the cluster. The results in Table 5.7 were produced by adding the sets of points as chosen by Shoniker's procedure for selecting

the initial training set. For Table 5.8, an extra Plackett-Burman design was used in each cluster instead.

The results provided in Table 5.7 and Table 5.8 are rather uneven. As is the case for the naïve clustering approach, the average speed-up performance on the small and easy datasets suffers noticeably, and most of the big datasets do not see any improvement. However, applying either of the heuristics to the dataset custom_sa, which will be described in Section 6, shows remarkable reductions in terms of the number of corners simulated, without sacrificing the accuracy of predictions. Circuit shift_reg also seems to benefit in terms of number of corners that would be simulated, however the accuracy drops below the acceptable level of 99.86%. In general, as the number of clusters increases, the performance appears to get worse, both in the average speedup and in the average termination accuracy.

2 Clusters				3 Clusters			
Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy	Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bitcell	36.91	3.25	100.00	bitcell	44.41	2.70	100.00
charge_pump1	47.29	4.65	100.00	charge_pump1	55.99	3.92	100.00
charge_pump2	71.62	4.64	99.00	charge_pump2	82.27	4.03	99.60
sense_amp1	53.25	2.66	99.86	sense_amp1	54.98	2.41	100.00
bias_gen	42.72	2.81	100.00	bias_gen	48.75	2.46	100.00
opamp1	54.03	2.22	100.00	opamp1	54.45	2.20	100.00
shift_reg	690.73	1.92	94.33	shift_reg	677.19	1.94	94.33
buffer_chain	200.85	11.07	98.83	buffer_chain	201.05	10.44	99.33
Average Speedup	4.15	Average Accuracy	99.00	Average Speedup	3.76	Average Accuracy	99.16
4 Clusters				5 Clusters			
Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy	Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bitcell	51.94	2.31	100.00	bitcell	58.04	2.07	100.00
charge_pump1	65.58	3.32	100.00	charge_pump1	76.09	2.84	100.00
charge_pump2	94.32	3.50	99.20	charge_pump2	107.79	3.03	99.40
sense_amp1	63.47	2.01	100.00	sense_amp1	68.38	1.81	100.00
bias_gen	58.27	2.06	100.00	bias_gen	67.90	1.77	100.00
opamp1	59.79	2.01	100.00	opamp1	65.94	1.82	100.00
shift_reg	628.41	2.11	93.00	shift_reg	628.26	2.08	92.00
buffer_chain	209.24	9.62	99.00	buffer_chain	223.21	9.04	99.50
Average Speedup	3.37	Average Accuracy	98.90	Average Speedup	3.06	Average Accuracy	98.86

Table 5.7 Results of Clustering on the Initial Training Set Generated by Shoniker's Initial Design, with an Extra Randomised Set for Every Cluster

2 Clusters				3 Clusters			
Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy	Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bitcell	35.70	3.36	100.00	bitcell	42.17	2.85	100.00
charge_pump1	44.32	4.96	100.00	charge_pump1	51.98	4.21	100.00
charge_pump2	71.15	4.68	98.00	charge_pump2	79.63	4.19	95.60
sense_amp1	48.43	2.92	100.00	sense_amp1	44.28	2.91	100.00
bias_gen	39.78	3.03	100.00	bias_gen	44.13	2.72	100.00
opamp1	52.90	2.27	100.00	opamp1	51.79	2.32	100.00
shift_reg	706.30	1.85	97.33	shift_reg	665.70	1.98	96.33
buffer_chain	200.90	11.25	95.83	buffer_chain	201.13	10.51	94.33
Average Speedup	4.29	Average Accuracy	98.90	Average Speedup	3.96	Average Accuracy	98.28
4 Clusters				5 Clusters			
Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy	Dataset	Average Corners to Convergence	Average Speedup	Average Accuracy
bitcell	47.76	2.51	100.00	bitcell	51.65	2.32	100.00
charge_pump1	60.20	3.62	100.00	charge_pump1	68.29	3.17	100.00
charge_pump2	90.23	3.69	97.00	charge_pump2	99.92	3.31	96.80
sense_amp1	45.96	2.70	99.29	sense_amp1	53.03	2.36	99.86
bias_gen	48.79	2.46	100.00	bias_gen	53.58	2.24	100.00
opamp1	53.42	2.25	100.00	opamp1	51.82	2.32	100.00
shift_reg	617.15	2.08	93.00	shift_reg	616.81	2.12	92.00
buffer_chain	193.40	10.50	96.00	buffer_chain	189.46	10.62	93.33
Average Speedup	3.73	Average Accuracy	98.16	Average Speedup	3.56	Average Accuracy	97.75

Table 5.8 Results of Clustering on the Initial Training Set Generated by the Original Heuristic, with an Extra Plackett-Burman Design Added into Every Cluster

The observation that some datasets do benefit from this heuristic suggests that it could be useful, especially if the selected initial training set provides enough information about the structure of the dataset, as seems to be the case with custom_sa. Another problem is that this heuristic at this stage acts on every dataset regardless of its size or performance using Shoniker's original algorithm. A simple way to compensate for both of the above problems is to introduce clustering later in the simulation. This idea will be investigated in Section 5.2.3.

5.2.3 Clustering Based on the Performance of the Dataset Under Analysis

After applying the simple *k*-means clustering preprocessing technique on the input space of datasets in Section 5.2.1, the main conclusion was that while clustering seems to provide significant benefits to big and difficult datasets like shift_reg, it can be counter-productive for small and straightforward datasets. This problem could possibly be addressed by monitoring the performance of the dataset under analysis. The problem, however, is deciding what is a

straightforward dataset. At runtime, every function is unknown, so we can only guess whether a function is straightforward by monitoring the performance of the algorithm on the function.

For this chapter, the solution is based on the previous observations of the effectiveness of the algorithm for our benchmark circuits. This requires the same assumption that has been made about the provided datasets, and that assumption is that these functions are reasonably representative of the entire class of functions that would be produced from a corner analysis of a circuit. From Table 3.2, it can be seen that the majority of the output functions take no more than a third of the corners simulated to converge to a maximum with 4-sigma confidence. Taking into account that these results are averages of 100 runs starting with randomised initial training sets, and that some datasets have almost exactly 3.0x speedup, it makes sense to make the boundary between an "easy" and a "difficult" dataset a little stricter. It was decided then to move that boundary to 2.5x speedup. In other words, a dataset is considered difficult at runtime, if after more than 40% of the corners have been selected for simulation, the algorithm still has not converged to 4-sigma confidence. This also addresses the conclusion made in Section 5.2.2, as now the algorithm has more information about the output function and can make more intelligent choices when partitioning the input space.

2 Clusters			
bitcell	25.19	4.76	100.00
charge_pump1	34.67	6.33	99.80
charge_pump2	55.12	6.01	98.80
sense_amp1	57.26	2.65	100.00
bias_gen	36.86	3.26	100.00
opamp1	48.87	2.46	100.00
shift_reg	588.01	1.96	97.00
buffer_chain	159.13	13.91	99.83
Average Speedup	5.17	Average Accuracy	99.43

3 Clusters			
bitcell	24.21	4.96	100.00
charge_pump1	34.27	6.39	100.00
charge_pump2	55.61	5.98	99.40
sense_amp1	56.61	2.69	100.00
bias_gen	36.64	3.28	100.00
opamp1	49.82	2.41	100.00
shift_reg	636.62	1.75	92.33
buffer_chain	162.42	14.32	99.83
Average Speedup	5.22	Average Accuracy	98.95

4 Clusters			
bitcell	23.90	5.02	100.00
charge_pump1	34.54	6.37	100.00
charge_pump2	56.13	5.94	98.80
sense_amp1	57.91	2.68	99.71
bias_gen	36.69	3.28	100.00
opamp1	48.81	2.46	100.00
shift_reg	538.09	2.05	90.33
buffer_chain	133.78	16.12	99.83
Average Speedup	5.49	Average Accuracy	98.58

5 Clusters			
bitcell	23.94	5.01	100.00
charge_pump1	34.84	6.32	100.00
charge_pump2	57.12	5.86	99.60
sense_amp1	59.53	2.67	99.86
bias_gen	36.74	3.27	100.00
opamp1	51.65	2.32	100.00
shift_reg	555.56	1.99	88.67
buffer_chain	138.51	15.87	100.00
Average Speedup	5.41	Average Accuracy	98.52

6 Clusters			
bitcell	23.81	5.04	100.00
charge_pump1	34.45	6.39	99.80
charge_pump2	56.17	5.91	98.40
sense_amp1	60.90	2.63	99.57
bias_gen	36.66	3.28	100.00
opamp1	50.60	2.37	100.00
shift_reg	556.68	1.99	86.33
buffer_chain	129.68	16.30	99.83
Average Speedup	5.48	Average Accuracy	98.00

7 Clusters				8 Clusters			
bitcell	23.58	5.09	100.00	bitcell	24.19	4.96	100.00
charge_pump1	34.02	6.46	99.80	charge_pump1	34.27	6.42	99.60
charge_pump2	55.87	5.97	99.40	charge_pump2	55.45	6.00	98.40
sense_amp1	60.26	2.63	99.86	sense_amp1	59.54	2.66	100.00
bias_gen	36.76	3.27	100.00	bias_gen	36.59	3.28	100.00
opamp1	51.55	2.33	100.00	opamp1	55.76	2.15	100.00
shift_reg	569.37	1.93	90.67	shift_reg	579.26	1.91	91.67
buffer_chain	133.69	16.15	99.83	buffer_chain	133.60	16.31	100.00
Average Speedup	5.48	Average Accuracy	98.70	Average Speedup	5.46	Average Accuracy	98.71

Table 5.9 Results of Splitting the Datasets into Two and Three Clusters if they have not Converged after 40% of the Corners have been Simulated

From this starting point, a new heuristic was developed. The algorithm proceeds like Shoniker's algorithm as long as fewer than 40% of the total number of corners have been selected for simulation. When more than 40% of corners have been selected for simulation, the remaining points are clustered based on the output values of simulated corners and the proximity of the unsimulated corners to the simulated corners. So, for single output functions, all of the simulated points are separated into several tiers based on how big their output value is, the number of tiers being equal to the number of clusters. Then, for each tier, the distance between the points that have not been assigned to a cluster yet and the respective closest point from the tier is calculated. The reason why points are assigned to clusters based on the proximity to the closest point in a tier, rather than to the central point in a tier, is because an unsimulated point is more likely to have the output value close to a neighbouring point, rather than the central point. Finally, all the unassigned points are separated into their own tiers based on their proximity to the closest point in the current cluster, with the number of these tiers being set equal to the number of clusters which have not been assigned any unsimulated points yet. This cluster is then regarded as complete, and the process moves on to the next tier of simulated corners. The process continues until every point has been assigned to a cluster.

After the clusters have been determined, each cluster gets additional training points from the Plackett-Burman design to ensure proper representation of each cluster's subspace. The Plackett-Burman design was selected since it is compact and the results of Section 5.1.2 have shown that this design is well suited for the algorithm. The calculated boosting factors would carry over to the new clusters. Additionally, pointwise termination heuristic from Section 4.1.3.3 with

σ_{eff} was used, as it allows slightly increased efficiency without sacrificing accuracy. This heuristic was, again, tested for up to 8 clusters. The results of these runs are provided in Table 5.9.

Reviewing the results summarised in Table 5.9, it is clear that there is in fact some benefit to using more clusters for the more difficult circuits, which causes the average speedup to rise, and the small datasets (sense_amp1) surprisingly do not lose in the speedups or, mostly, in the termination accuracies. However, the termination accuracy is still too low. In order to increase the accuracy on several problematic datasets, a series of heuristic improvements was introduced. First, separate points would not be pronounced safe under any condition, to eliminate the risk of accidentally removing the true maximum point from consideration. Second, from preliminary tests, it was found that introducing clustering later in a run positively influences the eventual accuracy, and so that threshold was set at 50% of the dataset. Third, no point from the extra Plackett-Burman design for each cluster would be aliased with any other point, meaning the number of sampled points would be increased by the theoretical size of the Plackett-Burman design for this dataset. Fourth, a central point would be added to the extra Plackett-Burman design to keep it more consistent with the Shoniker's problem design for selecting the initial training set. Finally, the clusters have a separate termination threshold to compensate for the lower accuracy of such heuristic. While more informed clustering results in higher termination accuracies than straightforward clustering, it is still mostly not good enough. This is not such an extreme problem as for the straightforward clustering cases, so simply setting a higher termination threshold for the resulting clusters should be enough. Setting a higher termination threshold, additionally, allows the computational priority to be shifted into exploration to give GPMs more accuracy within each partition. For Table 5.10, such threshold would be set to 6-sigma.

3 Clusters				4 Clusters			
bitcell	26.07	4.60	100.00	bitcell	25.58	4.69	100.00
charge_pump1	35.21	6.23	99.80	charge_pump1	35.43	6.21	100.00
charge_pump2	58.06	5.74	100.00	charge_pump2	58.33	5.72	100.00
sense_amp1	62.73	2.54	100.00	sense_amp1	62.46	2.55	100.00
bias_gen	37.25	3.23	100.00	bias_gen	37.30	3.22	100.00
opamp1	46.91	2.56	100.00	opamp1	46.74	2.57	100.00
shift_reg	654.59	1.74	95.00	shift_reg	640.29	1.74	96.33
buffer_chain	135.20	15.97	99.83	buffer_chain	140.63	15.86	100.00
Average Speedup	5.33	Average Accuracy	99.33	Average Speedup	5.32	Average Accuracy	99.54

Table 5.10 Results of Not Discarding Safe Points, Clustering at Half, Expanded Extra Training Set and Increasing Termination Threshold for Clusters to 6-sigma

A further improvement was to increase the termination threshold for eventual subsets to 8 sigma. This produces 100% accuracy for all the functions that had to be clustered. Thus, we conclude that, to achieve the target accuracy, we need to double the termination sigma level for the clusters, compared to the unclustered case. So, for example, if the unclustered case is told to terminate with the 3-Sigma rule, the clustered termination threshold should be 6-Sigma. From preliminary tests on the most challenging output (30 runs on the rise_time output from circuit shift_reg), it was found that partitioning into 4, 5, and 6 clusters had the highest accuracy of prediction for the output. Table 5.11 summarises the results of the 100 tests using this combination of heuristics.

4 Clusters				5 Clusters				6 Clusters			
bitcell	25.66	4.68	100.00	bitcell	25.47	4.71	100.00	bitcell	25.63	4.68	100.00
charge_pump1	35.58	6.19	100.00	charge_pump1	35.41	6.23	100.00	charge_pump1	35.52	6.21	100.00
charge_pump2	58.59	5.71	100.00	charge_pump2	57.37	5.80	100.00	charge_pump2	57.80	5.77	100.00
sense_amp1	65.15	2.51	100.00	sense_amp1	65.42	2.49	100.00	sense_amp1	65.89	2.49	100.00
bias_gen	37.24	3.23	100.00	bias_gen	37.33	3.22	100.00	bias_gen	37.33	3.22	100.00
opamp1	46.81	2.56	100.00	opamp1	46.67	2.57	100.00	opamp1	47.15	2.55	100.00
shift_reg	697.33	1.61	95.67	shift_reg	720.83	1.60	99.33	shift_reg	716.69	1.65	97.33
buffer_chain	137.24	15.32	100.00	buffer_chain	131.30	16.38	100.00	buffer_chain	136.32	16.11	99.67
Average Speedup	5.23	Average Accuracy	99.46	Average Speedup	5.38	Average Accuracy	99.92	Average Speedup	5.34	Average Accuracy	99.63

Table 5.11 Results of 8-sigma Termination Rule for Subclusters

The final improvement would be to not add the extra training set for small datasets when clustering difficult functions of such datasets. Table 5.12 summarises by how much the training set increases for each dataset, should the dataset be found to be difficult and then consequently split up into multiple clusters.

Dataset	Clustering Threshold	Extra Training Points	Approximate Relative Increase for 4 Clusters	Approximate Relative Increase for 5 Clusters	Approximate Relative Increase for 6 Clusters	Has "Difficult" Functions
bitcell	60.00	9.00	1.60	1.75	1.90	No
charge_pump1	108.00	9.00	1.33	1.42	1.50	No
charge_pump2	162.00	9.00	1.22	1.28	1.33	No
sense_amp1	60.00	9.00	1.60	1.75	1.90	Yes
bias_gen	60.00	5.00	1.33	1.42	1.50	No
opamp1	60.00	5.00	1.33	1.42	1.50	No
shift_reg	540.00	9.00	1.07	1.08	1.10	Yes
buffer_chain	900.00	9.00	1.04	1.05	1.06	Yes

Table 5.12 Relative Increases of Training Set Sizes in Case of Clustering

From Table 5.12, it is seen that the small datasets significantly increase the size of their training sets. However, only one dataset (sense_amp1) selects too many extra corners at the moment of partitioning, thus causing worse speedups. Then, the extra training set will not be added

if the total number of added points is 20% or more of the size of the training set at the moment of split. The updated Table 5.11 is provided as Table 5.13.

4 Clusters				5 Clusters				6 Clusters			
bitcell	25.66	4.68	100.00	bitcell	25.47	4.71	100.00	bitcell	25.63	4.68	100.00
charge_pump1	35.58	6.19	100.00	charge_pump1	35.41	6.23	100.00	charge_pump1	35.52	6.21	100.00
charge_pump2	58.59	5.71	100.00	charge_pump2	57.37	5.80	100.00	charge_pump2	57.80	5.77	100.00
sense_amp1	61.33	2.56	100.00	sense_amp1	62.04	2.55	100.00	sense_amp1	61.54	2.55	100.00
bias_gen	37.24	3.23	100.00	bias_gen	37.33	3.22	100.00	bias_gen	37.33	3.22	100.00
opamp1	46.81	2.56	100.00	opamp1	46.67	2.57	100.00	opamp1	47.15	2.55	100.00
shift_reg	697.33	1.61	95.67	shift_reg	720.83	1.60	99.33	shift_reg	716.69	1.65	97.33
buffer_chain	137.24	15.32	100.00	buffer_chain	131.30	16.38	100.00	buffer_chain	136.32	16.11	99.67
Average Speedup	5.23	Average Accuracy	99.46	Average Speedup	5.38	Average Accuracy	99.92	Average Speedup	5.34	Average Accuracy	99.63

Table 5.13 Report for not Adding Extra to the Training Set, if it Increases by More Than 20 Percent

The final best accuracy and speedup are then 99.92%, and 5.38x, respectively, compared to 100% and 4.74x, as found originally.

5.3 Discussion

In this chapter, the potential benefits of partitioning the input space into several sub regions was explored. Following the results of Chapter 4, it was suggested that there could be a way to expand the number of points that can be safely pruned away because they were not likely to have the global maximum. This led to the idea that separating the input set into several disjoint subsets (that is, clustering) could be an efficient way to go about this. The preliminary finding proved there could be great potential in this, and several heuristics were developed and refined.

Throughout testing the various developed heuristics, it was found that the more clusters that were used, the faster the algorithm would converge to what it assumed to be the global maximum. However, the termination accuracies were unacceptably poor, further corroborating one of the conclusions of the previous chapter that with more points under observation of a GPM, the individual errors for each separate corner are balanced out, and the average error of a collection of corners leads to the expected termination accuracies.

Bringing more structure into the selection of the initial training set provided interesting results. On the one hand, the termination accuracy improved to much more acceptable levels. On

the other hand, there would be very little chance of arriving at the correct value should such an "improved" initial training set turn out to be a poor representation of the total search space.

It was found that by taking a more informed approach to clustering greatly increases the overall efficiency of the general algorithm, significantly increasing the speedup, while preserving the termination accuracy at expected levels. By getting an idea of where to locate some of the higher valued outputs of the objective function, reasonable guesses can be made as to where the true global maximum is likely to be located. Such a technique seems to work particularly well on the functions for which the original algorithm would have difficulty producing quick results.

The next step would be to test the limits of the best available algorithm. Applying some insight into generating a difficult dataset, it should be possible to expand the scope of the project and eliminate the possibility of only having good performance for the datasets provided. Additionally, having total control over new datasets might provide additional insights into what makes a circuit produce functions that are difficult to analyse using existing variation design techniques. The next chapter will explore some of these questions.

Chapter 6: Building Hard Datasets

6.1 Sources of Difficulty in the Datasets

6.1.1 Observations from the Provided Datasets

The datasets provided to us by Solido Design Automation did not indicate the details of the circuits (e.g., the schematics) whose simulated behaviour produced the datasets. We could only guess about what functions were being implemented by looking at the names of the datasets. However, to achieve an important goal of the project, which is to understand what design structures can make circuits difficult to verify, we really do need full netlists for the circuits. One way to overcome the limitations of Solido's datasets was to create difficult datasets of our own, either from simulating difficult circuits or by creating synthetic datasets (i.e., without netlists).

It is clear that our algorithm would poorly handle datasets with sudden narrow spikes in the magnitudes of the output values. Indeed, we attempted to improve the performance for those cases using clustering. If such spike could be localised within a smaller region of focus of the input domain, then there would be a greater chance for the algorithm to accidentally sample it. However, as proved by several outputs of the provided datasets, simple clustering was not found to be effective. Another observation was that the algorithm poorly handles flat functions, like sen_dip_pctg of sense_amp1. Perhaps that is because the GPM has difficulty modelling what looks like uncorrelated noise. Finally, assuming that the benchmark circuit shift_reg indeed models the delay, rise time and fall time of the signals of a shift register, a plausible explanation for the difficulty of this dataset would be that internal race conditions can cause sudden non-linear behaviour in the positive feedback-stabilized outputs and are thus not well modelled by a GPM, which is best suited for modelling smooth well-correlated functions. All of these insights were applied in attempts to generate our own difficult datasets.

6.1.2 Creating Difficult Datasets

Several datasets were generated following the observations outlined in Section 6.1.1. The first of those is one that models uniform noise. It was generated from MATLAB, with different noise powers in the otherwise constant output functions, with the input values having no relation

to the outputs. Therefore, one would expect a GPM to have difficulty accurately modelling such a noisy output. Looking at the results provided in the relevant tables in this section, that is indeed the case. In fact, the best average speedup for this dataset (not shown in the tables) is 1.01x for the riskier algorithm that relies on single-point termination.

Next, datasets generated from the circuits were studied. The circuits were synthesised assuming 28-nm CMOS technology, and simulated in HSPICE, the commercial version of SPICE[5]. The first such new dataset was generated from a phase detector circuit, which is itself a part of a delay locked loop (DLL) design [68]. The phase detector detects the relative phase difference between two input clock signals and generates two pulsed outputs, UP and DOWN, that allow later stages of the DLL to phase-lock one adjustable (with variable delay) clock to a fixed reference clock. This circuit should have an increased probability of race conditions and nonlinear behaviours. Figure 6.1 demonstrates the basic functionality of the circuit

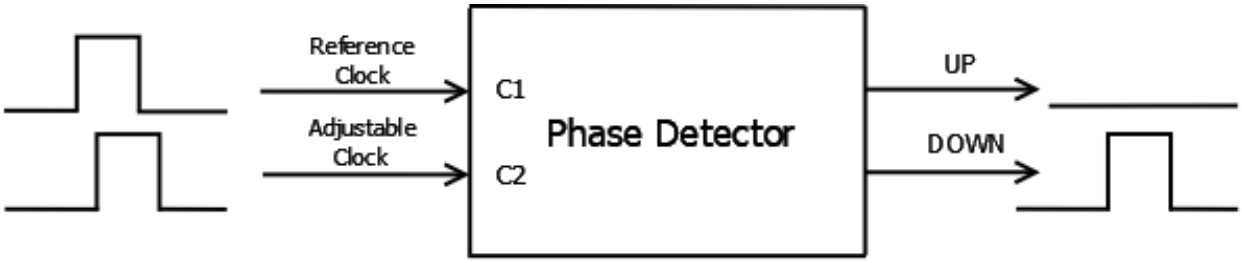


Figure 6.1 Basic Functionality of a Phase Detector Circuit

The detailed functionality is demonstrated in Figure 6.2. In the top half of Figure 6.2, C1 (the reference clock signal) leads C2 (the clock to be phase-locked to the reference) by 250 ps, which causes the circuit to generate a pulse at the UP output, while only producing a constant low (with some noise) at the DOWN output. In the bottom half of Figure 6.2, the situation is reversed. C1 lags C2 by 250 ps, and as a result a DOWN pulse is generated.

The functions generated from the phase detector circuit were the rise time and fall time of the two output signals, and the inputs were the phase difference between the clocks, the rise time and fall time characteristics for the adjustable clock generated by the circuit and the input clock signal to be synchronised, and the die temperature. Due to the complexity of the circuit, and its intended purpose, which implies dealing with race conditions, the output functions can be expected to be very sensitive to small PVT changes at certain input phases. The tests done on the circuit

with the algorithms presented in this work supported that hunch. The results of some of these tests are summarised in the relevant tables at the end of this section.

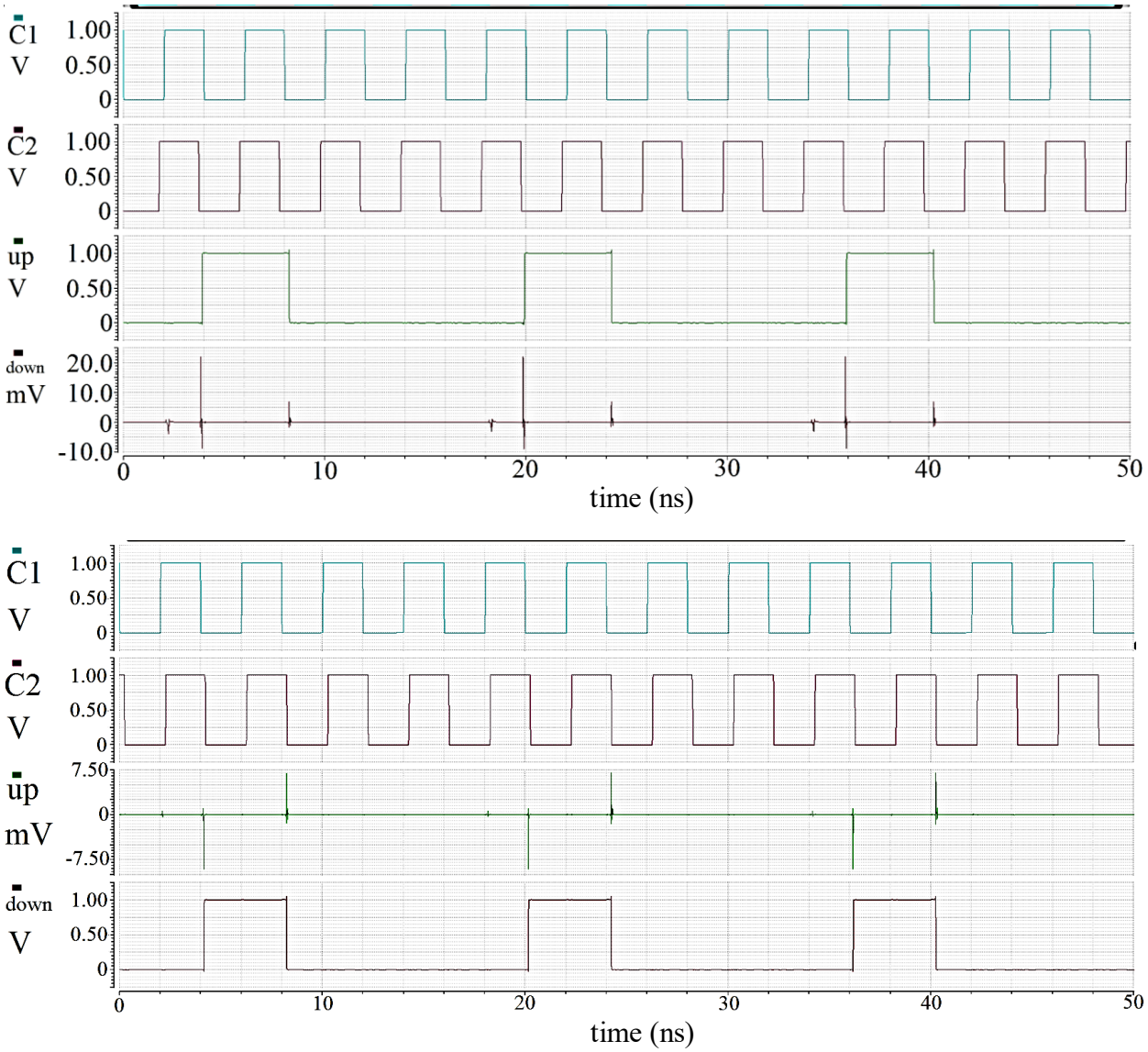


Figure 6.2 Functionality of the Phase Detector Circuit

The second circuit was the differential sense amplifier used in SRAM arrays [16]. The tested output function was the fall time of the complementary output in reaction to reading a positive value from a mock SRAM cell. The function was chosen for its simplicity and importance in IC design. Also, the positive feedback of the sense amplifier will produce a highly nonlinear output response. The inputs supplied were the PVT characteristics (modelsets ss, tt, and ff, temperature, and power voltage), and the voltage difference between the two complementary bit lines received from the mock cell. Interestingly, this dataset turned out to be easy for Shoniker's

algorithm. Despite the fact that several output points had significantly higher values than the others, the algorithms very rarely failed to find the global maximum, and the speedups were usually around 3.5x.

The third and last custom circuit design was the "glitch eraser" circuit, shown in Figure 6.3. Glitch eraser is designed to deglitch the two complementary input signals, X1 and X2, in case those inputs are corrupted by noise spikes, such as spikes caused by cosmic radiation or other sources of soft errors. The circuit fails if two glitches occur too close together in time, and that presents an interesting set of functions. The glitches were modeled as triangular waves with varying widths (that is, length of effect), and phase differences between the two glitches. In addition to those inputs, the process corners were also applied with the three standard values for the transistor modelset, as well as the die temperature and the power voltage level V_{DD}. The functions to be explored were the maximum voltage for the output signal that would be expected to be HIGH (Y1), and the minimum voltage for the output signal that would be expected to be LOW (Y2). So, for correct operation of the glitch eraser, output Y1 should be close to V_{DD} and Y2 should be close to GND (the ground voltage), whereas for incorrect operation this would be reversed. The sudden separation between the values of the modelled functions at the points of correct operation versus the points where the circuit fails would make this a difficult circuit. Table 6.1 gives the truth table for the circuit.

X1	X2	Y1	Y2
Glitched LOW	LOW	X	X
LOW	Glitched LOW	X	X
Glitched HIGH	HIGH	X	X
HIGH	Glitched HIGH	X	X
Glitched LOW	HIGH	LOW	HIGH
LOW	Glitched HIGH	LOW	HIGH
Glitched HIGH	LOW	HIGH	LOW
HIGH	Glitched LOW	HIGH	LOW

Table 6.1 Truth Table for the Glitch Eraser Circuit

The circuitry and the functionality are provided in Figures 6.3 and 6.4, respectively. In Figure 6.4, two triangular pulses with the duration of 1 ns affect the complimentary inputs X1 and X2, 1 ns apart. The 1 ns width for the pulses is perhaps too wide for a glitch, and could model something more like an extremely brief power surge, but this width is the biggest values for the duration of the pulses (the other are 250 ps, 500 ps and 750 ps), and these values still produce an interesting behaviour for the output functions. The circuit is able to catch those irregularities in the

voltage levels, and produce the corresponding complementary outputs Y1 and Y2 at the required levels, with the noise spikes removed.

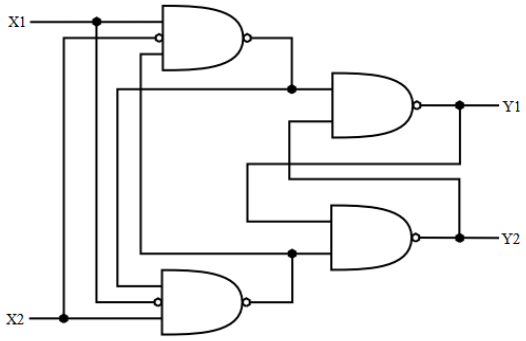


Figure 6.3 Glitch Eraser Circuit Diagram

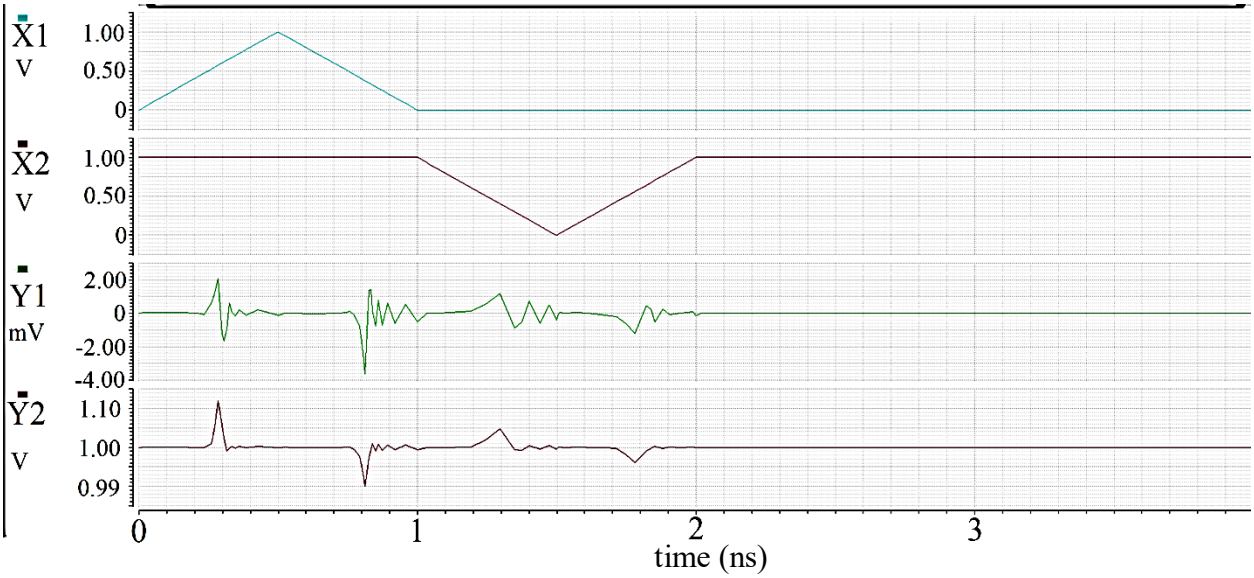


Figure 6.4 Functionality of the Glitch Eraser Circuit

The distributions of the generated output functions are provided next.

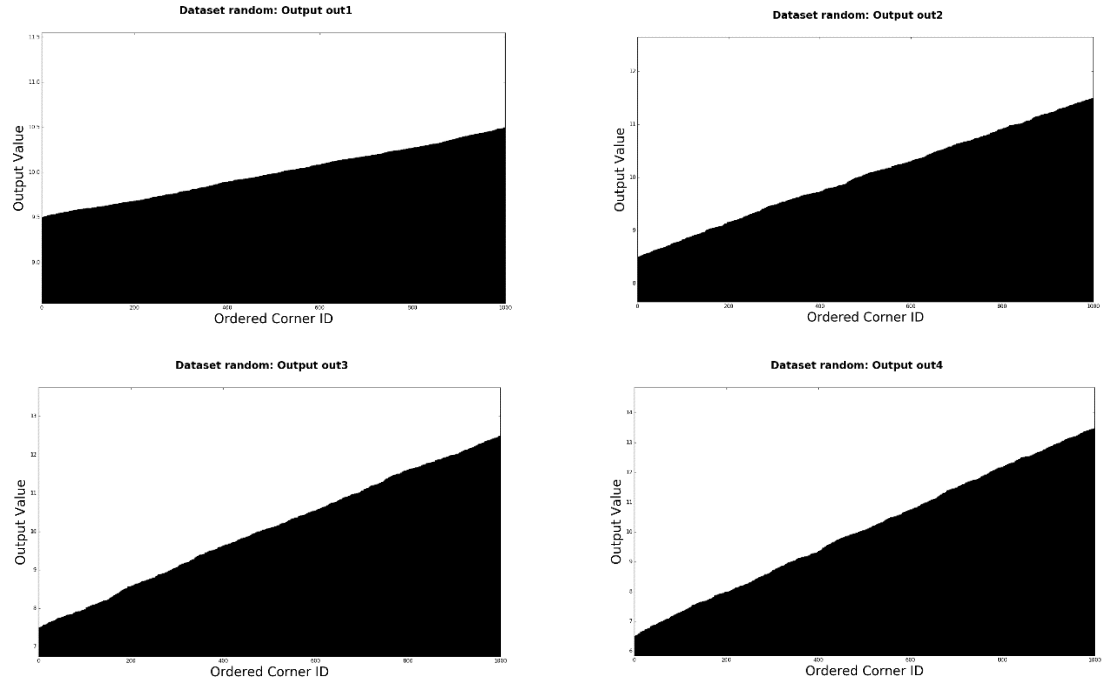


Figure 6.5 Distributions of the Output Functions of a Uniformly Random Dataset

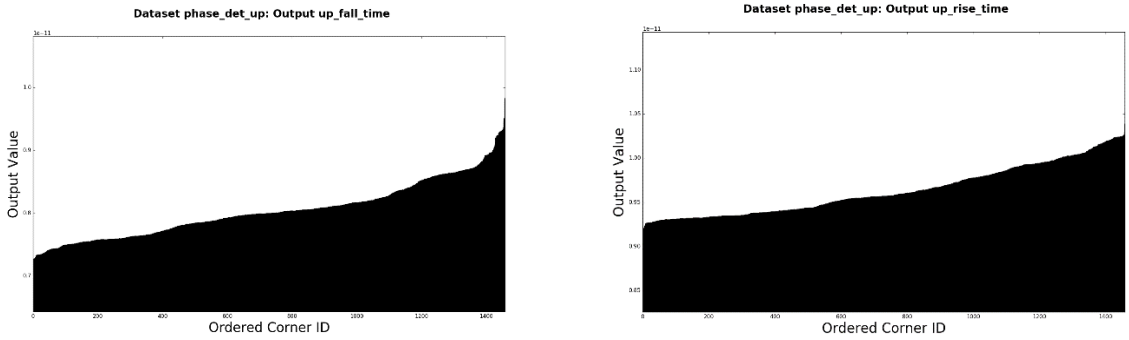


Figure 6.6 Distributions of the Output Functions of the phase_det_up Dataset

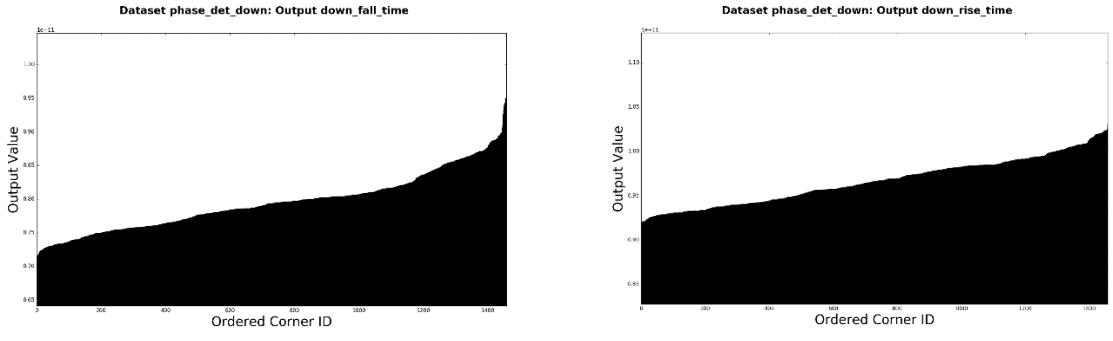


Figure 6.7 Distributions of the Output Functions of the phase_det_down Dataset

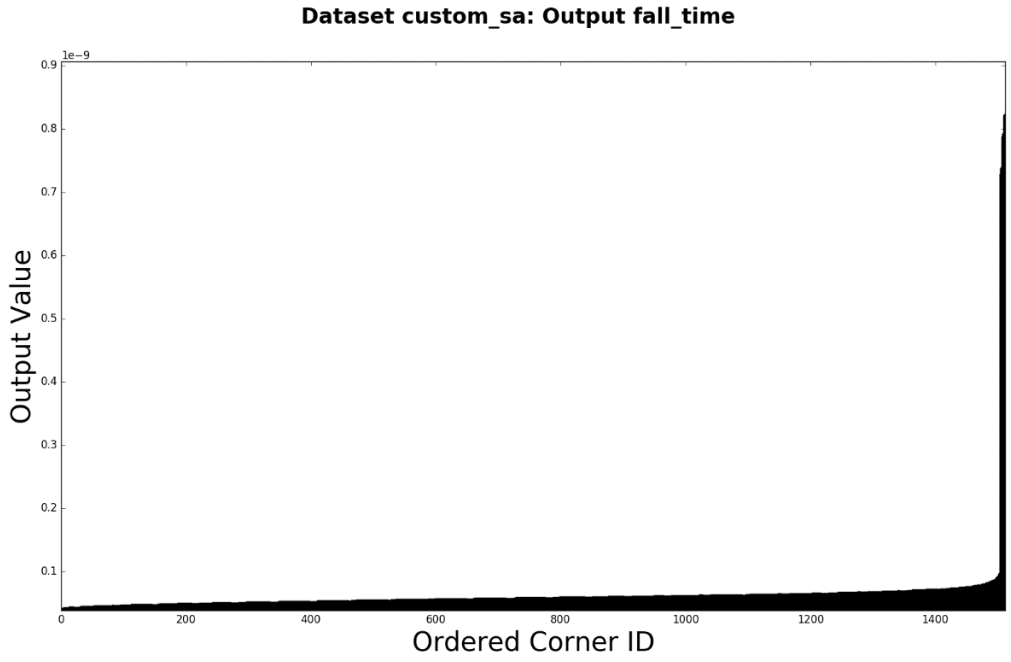


Figure 6.8 Distribution of the Output Function of the custom_sa Dataset

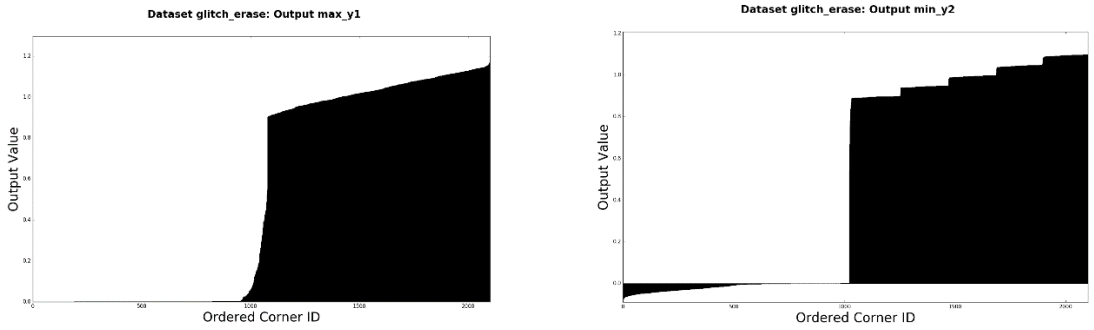


Figure 6.9 Distributions of the Output Functions of the glitch_erase Dataset

The histograms provide interesting insights into the structures of the output functions of these datasets. The histograms of the output functions of the random dataset (Figure 6.4) are of a trapezoidal shape, as one would expect from samples of a uniform random variable. As the noise power becomes larger, the slope of the trapezoid also becomes steeper. Of course, such a simple shape does not mean that the dataset is easy to analyse as the output values (by design) have no simple relation to the inputs for this dataset.

The shapes of the phase detector functions are rather flat, but still a few points are noticeably bigger than most others. This distribution looks similar in shape to the three functions in the shift_reg dataset. This means we could expect these datasets to be relatively challenging.

The custom sense amplifier output function has nine output values that are between 7.45 and 8.43 times bigger than the tenth highest value. The total size of the dataset is 1512 corners, meaning that 99.4% of the corner outputs lie within just over seven percent of the total range of values in the dataset. The fifteen highest values and the lowest value, with corresponding input values, are provided in Table 6.2.

Rank	Model_set	delta_v	Temperature	vvdd	fall_time
1	tt	0.25	125	0.85	8.24E-10
2	ss	0.25	125	0.85	8.24E-10
3	ff	0.25	125	0.85	8.23E-10
4	ss	0.25	100	0.85	7.94E-10
5	tt	0.25	100	0.85	7.92E-10
6	ff	0.25	100	0.85	7.88E-10
7	ss	0.25	125	0.9	7.40E-10
8	tt	0.25	125	0.9	7.38E-10
9	ff	0.25	125	0.9	7.28E-10
10	ss	0.05	125	0.85	9.77E-11
11	ss	0.05	100	0.85	9.66E-11
12	ss	0.05	50	0.85	9.34E-11
13	ss	0.05	75	0.85	9.32E-11
14	ss	0.05	27	0.85	9.19E-11
15	ss	0.05	125	0.9	9.07E-11
1512	ff	0.25	-50	1.05	4.29E-11

Table 6.2 The Fifteen Highest Values and the Lowest Value of the custom_sa Dataset

From Table 6.2 it can be seen that the six highest values are located in a two-dimensional neighbourhood, or, alternatively, on two adjacent ridges. The next three highest points are also located on a ridge, defined by the modelset of the CMOS technology. Looking at only the nine highest points, it would seem that the modelset parameter defines a more or less flat ridge for the points, and the conclusion could be that ridges are defined by that parameter. However, looking at the points ranked 10 to 14, the ridge parameter then seems to be the temperature value, and not even all the temperature values along the dimension.

Finally, the glitch_erase circuit provides an interesting structure in that there are two clearly defined, separated to extremes and abruptly changing levels in the output functions. This structure could easily confuse GPMs which would be expecting significantly higher or lower responses from a function. The output min_y2, which models the lowest response of the Y2 signal, the signal that would be pulled to V_{DD}, over the 4 ns illustrated on Figure 6.4, provides additional dimensions to the problem in that some of the output values are negative, and in that the circuit setup makes the

second higher level have five "stepped" levels of its own. This is because one of the inputs for glitch_erase is five different V_{dd} levels (0.9 V, 0.95 V, 1.00 V, 1.05 V, and 1.1 V), so min_y2 cannot go much higher than the corresponding value of V_{DD}. A GPM should easily model that the output has near perfect correlation between the corner values and the value of the V_{DD} input. However, this might also introduce the problem of the GPM being too reliant of that input value and thus having problems at the low level of the function, as the corner values in that level are much less correlated to V_{DD}, and so that GPM would be a worse fit for the function.

As intended, Shoniker's algorithm struggles greatly on the functions of the custom datasets, and the introduced improvements are able to safely prune away anywhere from 11% to 77% more of the corner simulations, compared to Shoniker's final algorithm, excluding the random dataset. Some of the algorithms described in the previous sections were tested on the new datasets. The results of those tests are provided in the following Tables 6.3-6.13.

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1451.59	1.00	100.00
phase_det_down	1453.35	1.00	100.00
custom_sa	458.16	3.30	100.00
random	999.64	1.00	100.00
glitch_erase	2042.24	1.03	100.00

Table 6.3 Results of Applying Standardised Preprocessing (Section 4.1.3.3) on the Custom Circuits

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
custom_sa	436.45	3.46	100.00
phase_det_down	1454.20	1.00	100.00
phase_det_up	1451.89	1.00	100.00
random	999.25	1.00	100.00
glitch_erase	1064.89	1.97	98.50

Table 6.4 Results of Applying the Final Pointwise Neighbourhood Termination Heuristic (Section 4.1.3.3) on the Custom Circuits

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1436.68	1.01	100.00	phase_det_up	1412.96	1.03	99.58
phase_det_down	1442.34	1.01	100.00	phase_det_down	1419.45	1.03	100.00
custom_sa	106.36	14.22	100.00	custom_sa	119.47	12.66	100.00
random	997.29	1.00	99.75	random	998.08	1.00	99.75
glitch_erase	1959.81	1.07	100.00	glitch_erase	1895.82	1.11	99.5

Table 6.5 Results of Applying Clustering on the Randomised Initial Training Set (Section 5.2.2) with Randomised Initial Training Set for Each Cluster; Results for 2 and 3 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1406.98	1.04	99.50	phase_det_up	1373.87	1.06	100.00
phase_det_down	1395.19	1.05	99.00	phase_det_down	1386.36	1.05	99.50
custom_sa	129.81	11.65	100.00	custom_sa	134.80	11.22	100.00
random	994.89	1.01	99.50	random	997.63	1.00	100.00
glitch_erase	1791.33	1.17	100.0	glitch_erase	1776.06	1.18	100.00

Table 6.6 Results of Applying Clustering on the Randomised Initial Training Set (Section 5.2.2) with Randomised Initial Training Set for Each Cluster; Results for 4 and 5 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1445.60	1.01	100.00	phase_det_up	1434.65	1.02	100.00
phase_det_down	1435.62	1.02	99.50	phase_det_down	1425.12	1.02	99.00
custom_sa	84.87	17.82	100.00	custom_sa	120.83	12.51	100.00
random	999.24	1.00	100.00	random	995.39	1.00	99.50
glitch_erase	1918.75	1.09	96.50	glitch_erase	1890.99	1.11	100.00

Table 6.7 Results of Applying Clustering on the Randomised Initial Training Set (Section 5.2.2) with a Plackett-Burman Design for the Initial Training Set for Each Cluster; Results for 2 and 3 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1396.33	1.04	100.00	phase_det_up	1376.87	1.06	99.00
phase_det_down	1411.95	1.03	98.00	phase_det_down	1359.04	1.07	98.50
custom_sa	125.54	12.04	100.00	custom_sa	137.23	11.02	100.00
random	997.01	1.00	99.75	random	997.72	1.00	100.00
glitch_erase	1608.85	1.32	81.00	glitch_erase	1609.46	1.32	87.50

Table 6.8 Results of Applying Clustering on the Randomised Initial Training Set (Section 5.2.2) with a Plackett-Burman Design for the Initial Training Set for Each Cluster; Results for 4 and 5 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1350.42	1.08	100.00	phase_det_up	1318.87	1.11	100.00	phase_det_up	1320.70	1.10	100.00
phase_det_down	1414.19	1.03	100.00	phase_det_down	1396.60	1.05	100.00	phase_det_down	1401.95	1.04	97.50
custom_sa	465.85	3.25	99.00	custom_sa	469.23	3.22	98.00	custom_sa	473.72	3.19	100.00
random	1000.00	1.00	100.00	random	999.41	1.00	100.00	random	998.03	1.00	99.00
glitch_erase	1635.65	1.28	100.00	glitch_erase	1670.86	1.26	100.00	glitch_erase	1627.00	1.29	100.00

Table 6.9 Results of Applying Clustering Based on the Performance of Datasets, Final Version (Section 5.2.3), for 4, 5 and 6 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1431.36	1.02	100.00	phase_det_up	1378.70	1.06	100.00	phase_det_up	1317.02	1.11	100.00
phase_det_down	1437.85	1.01	100.00	phase_det_down	1399.94	1.04	100.00	phase_det_down	1295.55	1.13	100.00
custom_sa	161.63	9.35	82.00	custom_sa	146.63	10.31	99.00	custom_sa	155.66	9.71	100.00
random	999.47	1.00	100.00	random	999.13	1.00	100.00	random	998.80	1.00	100.00
glitch_erase	1890.57	1.11	100.00	glitch_erase	1796.52	1.17	100.00	glitch_erase	1742.43	1.21	100.00

Table 6.10 Results of Applying Straightforward Clustering (Section 5.1.1) with a Randomised Initial Training Set for Each Cluster, for 2, 3 and 4 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1259.09	1.16	100.00	phase_det_up	1245.12	1.17	100.00
phase_det_down	1306.87	1.12	100.00	phase_det_down	1292.65	1.13	100.00
custom_sa	173.80	8.70	99.00	custom_sa	175.27	8.63	100.00
random	998.54	1.00	99.50	random	998.50	1.00	100.00
glitch_erase	1727.27	1.22	100.00	glitch_erase	1740.11	1.21	100.00

Table 6.11 Results of Applying Straightforward Clustering (Section 5.1.1) with a Randomised Initial Training Set for Each Cluster, for 5 and 6 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1448.66	1.01	100.00	phase_det_up	1448.66	1.01	100.00	phase_det_up	1402.04	1.04	100.00
phase_det_down	1452.08	1.00	100.00	phase_det_down	1450.08	1.01	100.00	phase_det_down	1441.07	1.01	100.00
my_sense_amp	292.63	5.17	100.00	my_sense_amp	157.72	9.59	100.00	my_sense_amp	179.19	8.44	100.00
random	999.67	1.00	100.00	random	999.67	1.00	100.00	random	999.58	1.00	100.00
glitch_erase	1959.91	1.07	100.00	glitch_erase	1738.05	1.21	100.00	glitch_erase	1689.70	1.24	100.00

Table 6.12 Results of Applying Straightforward Clustering (Section 5.1.2) with a Plackett-Burman Design for the Initial Training Set for Each Cluster, for 2, 3 and 4 Clusters

Dataset	Average Corners to Termination	Average Speedup	Average Accuracy	Dataset	Average Corners to Termination	Average Speedup	Average Accuracy
phase_det_up	1393.65	1.05	100.00	phase_det_up	1303.12	1.13	100.00
phase_det_down	1428.62	1.02	100.00	phase_det_down	1426.24	1.02	100.00
my_sense_amp	227.00	6.66	100.00	my_sense_amp	234.65	6.44	100.00
random	999.24	1.00	100.00	random	998.92	1.00	100.00
glitch_erase	1705.08	1.23	100.00	glitch_erase	1736.60	1.21	100.00

Table 6.13 Results of Applying Straightforward Clustering (Section 5.1.2) with a Plackett-Burman Design for the Initial Training Set for Each Cluster, for 5 and 6 Clusters

Tables 6.3-6.13 demonstrate improvements in speedups without loss of accuracy, compared to Shoniker's final algorithm, for almost any developed algorithm. Table 6.3 shows results that are close to those of Shoniker's final algorithm, and demonstrate that it struggles a lot with every dataset, except custom_sa, with the average speedup being 1.47x. The best results of our algorithms are for each dataset: 1.17x for phase_det_up (compared to 1.00x of Shoniker's algorithm), 1.13x for the phase_det_down dataset (compared to 1.00x), 17.82x for the custom_sa dataset (compared to 3.30x), 1.00x for the random dataset (compared to 1.00x), and 1.29x for the glitch_erase dataset (compared to 1.03x), among heuristics that terminated with 100% accuracy for the respective datasets.

6.1.3 Comparison with an Industrial Tool

In order to get a better sense of how the algorithms compare to the industrial standards, the commercial tool Solido Variation Designer version 3.5.7 (latest available version to us, current release is version 4.1.9) was used on the three custom circuits. The results of this round of

experiment are provided in Table 6.14. The tool does not provide a lot of flexibility in how the minimal corner selection will be carried out. So, for example, we could not set the desired sigma-confidence level. We could, however, set up the corners according to the ones in our datasets.

Dataset	Output Name	Corners to Convergence	Speedup	Average Speedup	Max Found
phase_det_down	down_rise_time	987.00	1.48	1.44	TRUE
	down_fall_time	1034.00	1.41		TRUE
phase_det_up	up_rise_time	636.00	2.29	1.82	TRUE
	up_fall_time	1088.00	1.34		TRUE
custom_sa	fall_time	59.00	25.63	25.63	FALSE
glitch_erase	max_y1	938.00	2.24	2.66	TRUE
	min_y2	681.00	3.08		TRUE

Table 6.14 Results of Analysis by Industrial Tool

Dataset	Output Name	Corners to Convergence	Speedup	Average Speedup	Max Found
phase_det_down	down_rise_time	1029.00	1.42	1.53	TRUE
	down_fall_time	873.00	1.67		TRUE
phase_det_up	up_rise_time	421.00	3.46	2.23	TRUE
	up_fall_time	885.00	1.65		TRUE
glitch_erase	max_y1	1328.00	1.58	1.66	TRUE
	min_y2	1195.00	1.76		TRUE

Table 6.15 Best Results of the Best Algorithms as Applied on the Custom Datasets

From Table 6.14, it can be seen that Variation Designer 3.5.7 significantly outperforms any algorithm described in the work. However, what is interesting to note, Variation Designer returned an incorrect worst-case corner when analysing the easy dataset custom_sa, the tenth-worst-case corner, in fact. Perhaps, this suggests that Variation Designer 3.5.7 is not well suited to handling datasets with spikes as it selects the highest "regular" value in the custom_sa fall_time function (see Table 6.2). The other results are far better than the respective average results by any of our algorithms. Even setting a lower termination threshold for our algorithms does not come close to the speedups of Variation Designer (however, the up_fall_time of the phase_det_up dataset gets very close to Variation Designer for the 3-sigma termination rule, losing by about 60 corners, on average). At the same time, some performances of our informed clustering algorithms do terminate to the correct global maximum faster than Variation Designer for three out of four outputs in the phase_det_up and phase_det_down datasets. The best results of our algorithms are summarised in Table 6.15.

It can be argued then that Variation Designer is simply more consistent in its analysis and predictions. This can be explained by the fact that Variation Designer has been in continuous

development for around a decade, for commercial use, meaning the standards and knowledge in Solido Design Automation are far higher than those for a graduate project. Figures 6.10, 6.11, and 6.12 summarise the best performances of our algorithms.

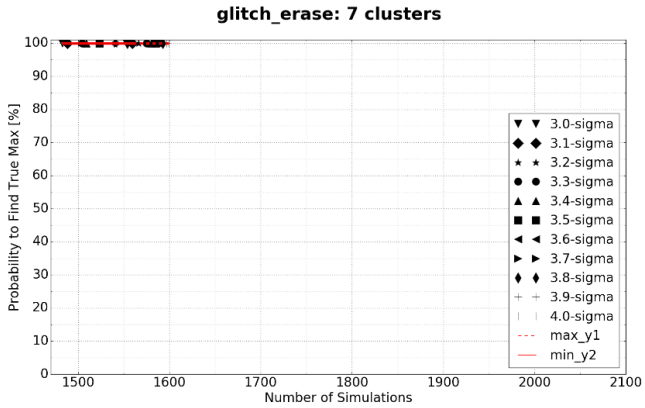


Figure 6.10 Results of Informed Clustering, 7 Clusters, on the glitch_erase Dataset

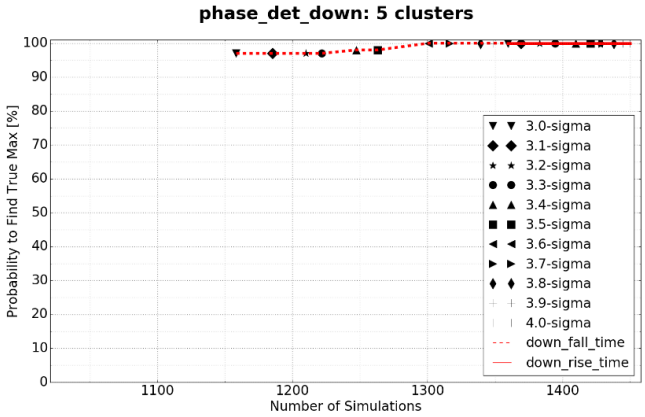


Figure 6.11 Results of Informed Clustering, 5 Clusters, on the phase_det_down Dataset

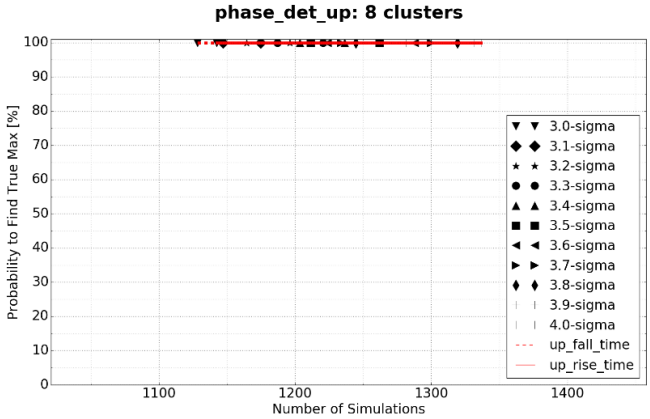


Figure 6.12 Results of Informed Clustering, 8 Clusters, on the phase_det_up Dataset

Another interesting observation is the convergence diagram for the glitch_erase outputs, provided in Figure 6.13 and Figure 6.14. In those figures, the x axis corresponds to how many corners have been simulated so far, and the y axis shows the predicted values and uncertainties of the worst predicted corner, with the current known maximum as the baseline.

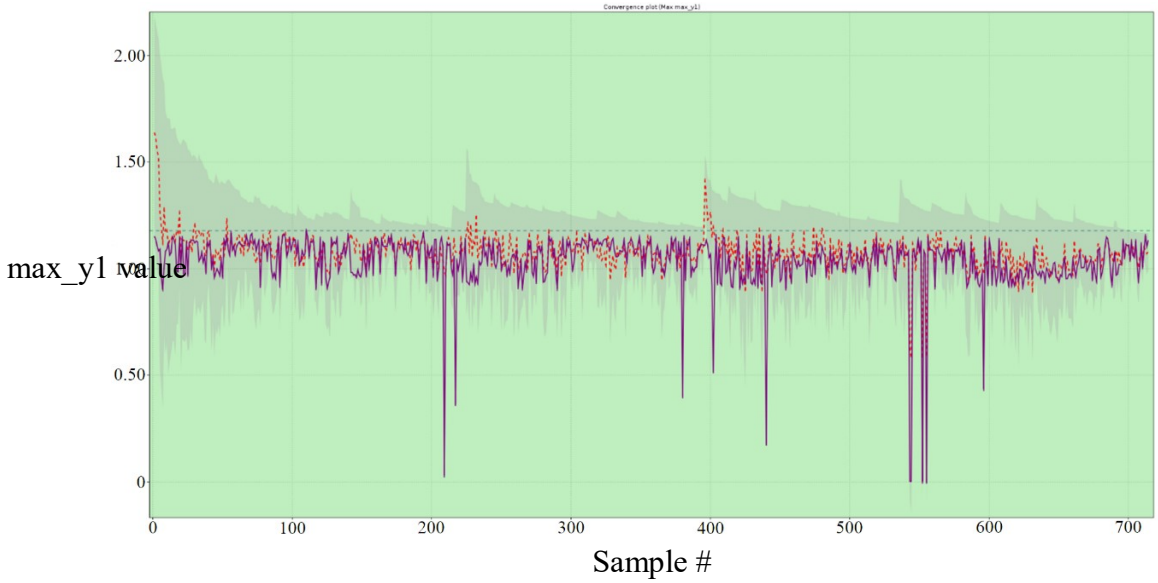


Figure 6.13 Convergence Plot of the max_y1 Output of the glitch_erase Dataset

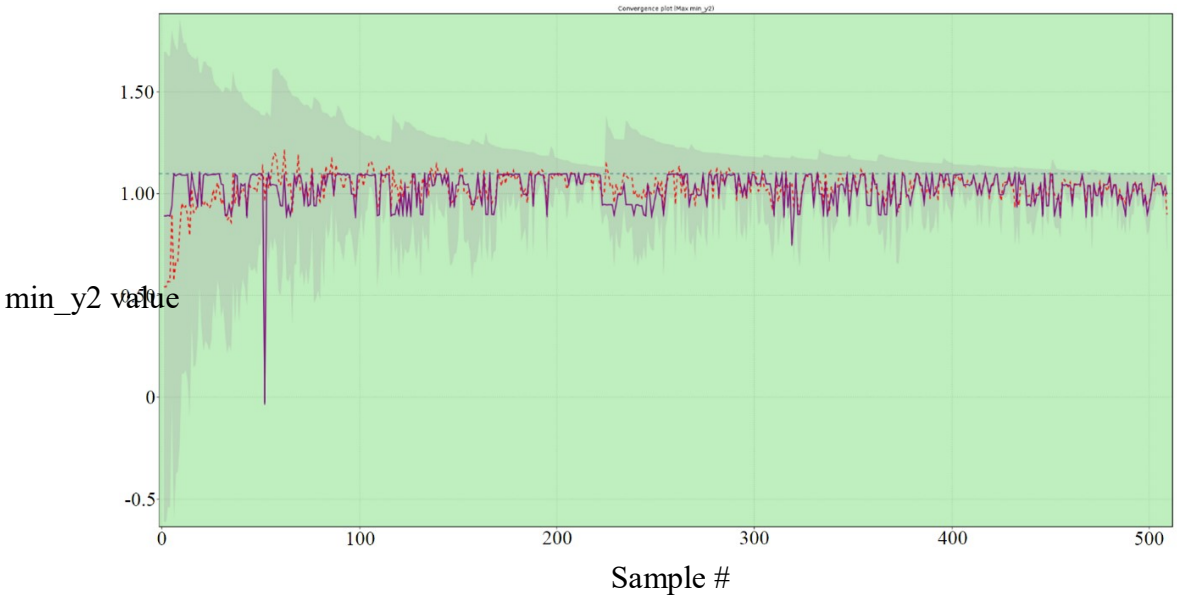


Figure 6.14 Convergence Plot of the min_y2 Output of the glitch_erase Dataset

Figure 6.13 and Figure 6.14 show the evolution of the uncertainty (grey shaded region) of Variation Designer in its eventual selected guess for the global maximum (the constant blue dashed line). The red dashed line is the prediction $\hat{y}$ of the corner that has the highest $\hat{y} + k * \sigma$, and the

solid purple line is the value of the latest simulated corner. It is interesting to see in Figure 6.9 that the algorithm came extremely close to convergence at least three times, before selecting a corner with significantly smaller output than was expected and causing the uncertainty to rise and thus continue looking for the maximum. This could be explained by the nature of the dataset, which has two very distinct levels: near the GND voltage level and near VDD. A smart clustering procedure would definitely help in this case as it should be able to separate these two levels, and accelerate convergence to the selected corner.

6.2 Discussion

Looking at the constructed worst-case circuits proved to be very helpful for improving our understanding of the various algorithms for minimal worst-case corner selection. Having a custom dataset allows more flexibility in exploring performances of algorithms and this flexibility allowed us to highlight strengths or weaknesses of each particular heuristic.

This chapter provided a very preliminary analysis of what could make a dataset difficult to analyse with Shoniker's algorithm. From our experience with these datasets, these guesses turned out to be mostly correct. The performance of the heuristics developed in Chapters 4 and 5 showed clear improvement on these difficult datasets, which suggests that they could become more valuable as the feature size of circuits grows smaller and the number of corners for confident design verification grows larger.

Having access to an industrial tool also provided significant benefit. Quite unsurprisingly, the tool outperformed every algorithm developed here by a large margin. Interestingly, one of the custom datasets highlighted a significant weakness in the design of the tool. The visualisation techniques of the tool also suggested insights into how their algorithm works and what conditions could confuse the tool. Overall, the performance of algorithms developed in this work could be judged to be reasonably good, although there is a long way to go to reliably achieve the speed-ups of the commercial tool. It could partially be explained by the apparent less strict requirements for termination from Variation Designer 3.5.7.

Chapter 7: Conclusions and Future Work

7.1 Main Contributions

This thesis described attempts to improve upon Michael Shoniker's algorithm for minimal worst-case corner selection when finding the maximum of an output function. The improvements studied include:

- A more in-depth study of the distributions of the output values of the datasets with the first attempts to improve performance for the datasets that suffer from overfitting and sudden changes in output values (spikes and ridges).
- More detailed experimentation on the termination rules and more attempts to prune away some "safe" corners before terminating the entire algorithm, thus improving speedups of the analyses of output functions. Introduction of neighbourhood termination in order to improve the termination accuracy.
- Study of the effects of partitioning the input space into several clusters for the purposes of pruning away groups of corners according to the *k*-means clustering procedure, for between 2 and 8 clusters. Exploration of selecting the initial training set for the algorithms according to Plackett-Burman designs. Partitioning of the input space based on the observed output values after a fraction of the corners have been simulated, thus allowing the algorithm to take advantage of the benefits for big and difficult circuits and avoid the disadvantages for small and easy circuits introduced by running the straightforward *k*-means clustering of the input space.
- Design of custom datasets for the purposes of further exploration of what design decisions make circuits difficult to analyse. Comparison of the performance heuristics developed in Chapters 4 and 5 to that of the commercial tool, Solido Variation Designer 3.5.7, performing the same function as the heuristics introduced in this thesis.

First, it was noted that the termination criterion used by Shoniker had some inconsistencies, and these inconsistencies were studied. The effect of multiple near-worst-case corners, in particular, meant that the algorithm would, theoretically, tend to be more optimistic than expected.

Thus, the performance of the algorithm with a stricter pruning and termination thresholds was studied. As a result, it was found that increasing the termination σ rule by one would take care of most of such inconsistencies, and thus the σ=4 termination rule was adopted exclusively in all further work.

Making the GPR assumption that every test point is an independent variable drawn from the same joint Gaussian distribution provided interesting insights into the behaviour of the GPM for some particularly difficult datasets. Specifically, it was found that proclaiming test points as unlikely to be equal to the global maximum based only on their predicted distributions proved to be a risky strategy. Consequently, mechanisms for considering several neighbouring points for pointwise termination were developed. These changes provided minor (approximately 10%) improvements to the speedups of our datasets. This also gave the insight that taking into account the predicted values of the neighbours of the target corner, and not just those of the target corner on its own, could be very beneficial.

The early studies of neighbourhood-based group termination suggested that it was an even riskier strategy than pointwise termination. The naïve *k*-means clustering approach was tested and inverse correlations were observed between the number of clusters and the number of corners simulated to convergence, as well as between the number of clusters and the accuracy of finding the true global maximum. As the number of clusters grew bigger, the termination accuracies and the number of corners simulated tends to decrease. This was improved by applying a deterministic initial design for the initial selection of corners. This helped when the algorithm was optimising big and difficult datasets, however, the performance on the small and easy datasets became worse. The clustering heuristic was further improved to base the clustering on the observed responses, which helped to increase the termination accuracy to acceptable levels and avoided the need for more complicated termination heuristics.

Looking at the custom circuits and comparing the performance of the industrial tool with the algorithms provided some interesting insights. It was found that the tool is able to reliably (but not always) achieve significantly better speedups on the custom circuits compared to the developed algorithms. Surprisingly, the tool failed to find the maximum of one of the datasets. This could point to the tool not being able to handle spikes and high plateaus in the structure of the output functions. It is also possible that the tool has more relaxed requirements for termination.

7.2 Future Work

There are a number of directions in the project that can be explored in the future, some of which we have attempted to implement but were not successful. The biggest one of those that we tried was the multiobjective optimization. That functionality was preliminarily explored in Shoniker's thesis. We attempted to extend his ideas to the informed clustering approach (Section 5.2.3). In our implementation, if 50% of the corners have been simulated, we would perform *k*-means clustering on the multiple output dimensions, and assign the unsimulated corners to the clusters as described in Section 5.2.3 otherwise. This led to reduced accuracies and reduced speedups, however, so other ways should be considered.

Another attempted direction of research was trying to generate the training set of the smallest size that would terminate the function to a certain sigma confidence level while finding the true global maximum. For that purposes we used the adapted versions of two popular feature selection techniques: sequential forward selection (SFS)[69] and sequential backward selection (SBS)[70].

For SBS, we would run our algorithms on a dataset, and then select the smallest training set that terminated the function with 3-Sigma confidence. Then, we would remove one corner from the smallest training set, and look at how much closer the convex hull on the scatter plots gets to the 3-Sigma termination line. The smallest such change would mean that termination was the least affected by removing that corner, and so the new training set is taken as the smallest and the procedure is repeated until we cannot remove any corner without causing any of the points on the scatter plot to be above the termination line.

SFS is the opposite approach. We would start with no corners in the training set and would add one corner at a time to see which added corner brings the convex hull closest to the 3-Sigma termination line. This procedure runs until a training set is found that puts all the points on the scatter plot below the termination threshold.

Obviously, neither SBS nor SFS are ideal methods for the task, however, the ideal method would take enormous amount of time to run. Even these methods were only viable for small datasets, no bigger than charge_pump1 (216 corners). Tests for the charge_pump1 dataset found

that the best speedup achieved could be 9.56x for the 3-Sigma termination rule, compared to 5.16x for Shoniker's algorithm. However, just finding the smallest training sets is not enough. It is important to understand what makes that selection the smallest, and we were not successful in determining that.

A significant next step would be to find the most theoretically sound and efficient way to partition the input space into subregions, as early as possible in the execution of the algorithm. In Section 5.2.2, for example, it was shown that certain circuits (e.g., custom_sa) get significant improvements for the earliest stages after such a partition. Becoming more confident in the generated GPMs is critical to this. Thus, it would be very helpful to develop mechanisms to quickly locate spikes in the dataset. Another potentially useful work would be additive kernels, for example as explored in [71], which is essentially automatic construction of kernels from several basis kernels based on the observed outputs. If we know that a kernel models the function well, we can be more confident in its predictions, and thus partition the input space, or even terminate, earlier. As an intermediate step, we could measure some regression metrics, such as the ones implemented in the scikit-learn package[72] , for example coupling them with the cross-validation step, as described in Section 2.3.3.

The algorithms described in this work are very general. There is no reason to apply them only for the task of analysing the responses of integrated circuits. Any problem that requires optimisation through observing responses is well suited for this algorithm. For example, optimising a production process in chemical engineering tasks could require selecting process parameters that maximize the output. Such processes could easily be too large to analyse directly, requiring expensive simulations to model system performance for each choice of parameter settings accurately.

Conversely, the focus might be put into integrating knowledge of integrated circuit functions into the algorithm for a more tailored approach. For example, if a circuit needs to be optimised for a timing constraint, and one of the inputs is the process parameter, it would make sense to spend more computational effort on exploring regions represented by the slow-slow process, as this region would be most likely to contain the worst timing performance of the circuit.

Moving on to optimising a function over a nongridded input space could make the algorithm applicable for looking for rare-event failures in circuits. The algorithm could spend a

comparable amount of time finding high-sigma corners, like finding a 6-sigma (one-in-a-billion) worst case corner, in a fraction of the time that is required for a full-factorial simulation. Such an application would be useful for designing extremely variation-aware circuits or circuits that are required to be extremely reliable.

Finally, should the computational cost of GPR be optimised enough and the problem of locating spikes be solved adequately enough, these algorithms could be used for cost function optimisations used in other machine learning techniques, such as neural networks. This would ensure the lowest value of the cost function meaning that the performance of the learned structure (e.g., the trained neural network) would be the best possible using simpler designs of the networks, compare to the extremely complex structures like Deep Neural Networks. An existing framework, such as Robust Bayesian Optimization [73][74], can be a basis for these developments.

References

[1] Moore, G.E., 1965. Cramming more components onto integrated circuits. *Electronics, 38*(8), pp.80-91.

[2] Asenov, A., 1998. Random dopant induced threshold voltage lowering and fluctuations in sub-0.1/spl mu/m MOSFET's: A 3-D" atomistic" simulation study. *IEEE Transactions on Electron Devices, 45*(12), pp.2505-2513.

[3] Kuhn, K.J., Giles, M.D., Becher, D., Kolar, P., Kornfeld, A., Kotlyar, R., Ma, S.T., Maheshwari, A. and Mudanai, S., 2011. Process technology variation. *IEEE Transactions on Electron Devices, 58*(8), pp.2197-2208.

[4] McConaghy, T., Breen, K., Dyck, J. and Gupta, A., 2012. *Variation-aware design of custom integrated circuits: a hands-on field guide*. Springer Science & Business Media.

[5] Nagel, L.W. and Pederson, D.O., 1973. *SPICE: Simulation program with integrated circuit emphasis. Electronics Research Laboratory*, College of Engineering, University of California.

[6] March, J.G., 1991. Exploration and exploitation in organizational learning. *Organization science, 2*(1), pp.71-87.

[7] Shoniker, M, 2015. *Accelerated Verification of Integrated Circuits Against the Effects of Process, Voltage and Temperature Variations* (Master dissertation, University of Alberta Edmonton).

[8] Koh, M., Mizubayashi, W., Iwamoto, K., Murakami, H., Ono, T., Tsuno, M., Mihara, T., Shibahara, K., Miyazaki, S. and Hirose, M., 2001. Limit of gate oxide thickness scaling in MOSFETs due to apparent threshold voltage fluctuation induced by tunnel leakage current. *IEEE Transactions on Electron Devices, 48*(2), pp.259-264.

[9] Stolk, P.A., Widdershoven, F.P. and Klaassen, D.B.M., 1998. Modeling statistical dopant fluctuations in MOS transistors. *IEEE Transactions on Electron devices, 45*(9), pp.1960-1971.

[10] Asenov, A., Kaya, S. and Brown, A.R., 2003. Intrinsic parameter fluctuations in decananometer MOSFETs introduced by gate line edge roughness. *IEEE Transactions on Electron Devices, 50*(5), pp.1254-1260.

[11] Stapper, C.H., 1985. The effects of wafer to wafer defect density variations on integrated circuit defect and fault distributions. *IBM Journal of Research and Development, 29*(1), pp.87-97.

[12] Keane, J. and Kim, C.H., 2011. Transistor aging. *IEEE Spectrum, 48*(5), pp.28-33.

[13] Mehrotra, V., Nassif, S., Boning, D. and Chung, J., 1998, December. Modeling the effects of manufacturing variation on high-speed microprocessor interconnect performance. In *Electron Devices Meeting, 1998. IEDM'98. Technical Digest., International* (pp. 767-770). IEEE.

[14] Calhoun, B.H., Cao, Y., Li, X., Mai, K., Pileggi, L.T., Rutenbar, R.A. and Shepard, K.L., 2008. Digital circuit design challenges and opportunities in the era of nanoscale CMOS. *Proceedings of the IEEE, 96(2)*, pp.343-365.

[15] Li, X., Le, J., Celik, M. and Pileggi, L.T., 2005, November. Defining statistical sensitivity for timing optimization of logic circuits with large-scale process and environmental variations. In *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on* (pp. 844-851). IEEE.

[16] Pavlov, A. and Sachdev, M., 2008. *CMOS SRAM circuit design and parametric test in nano-scaled technologies: process-aware SRAM design and test* (Vol. 40). Springer Science & Business Media.

[17] Chang, L., Montoye, R.K., Nakamura, Y., Batson, K.A., Eickemeyer, R.J., Dennard, R.H., Haensch, W. and Jamsek, D., 2008. An 8T-SRAM for variability tolerance and low-voltage operation in high-performance caches. *IEEE Journal of Solid-State Circuits, 43*(4), pp.956-963.

[18] Sedra, A.S. and Smith, K.C., 1998. *Microelectronic circuits* (Vol. 1). New York: Oxford University Press.

[19] Cox, P., Yang, P., Mahant-Shetti, S.S. and Chatterjee, P.A.L.L.A.B., 1985. Statistical modeling for efficient parametric yield estimation of MOS VLSI circuits. *IEEE Journal of Solid-State Circuits, 20*(1), pp.391-398.

[20] Nassif, S.R., 2001. Modeling and analysis of manufacturing variations. In *Custom Integrated Circuits, 2001, IEEE Conference on*. (pp. 223-228). IEEE.

[21] Antoniadis, D.A. and Khakifirooz, A., 2008, December. MOSFET performance scaling: Limitations and future options. In *Electron Devices Meeting, 2008. IEDM 2008. IEEE International* (pp. 1-4). IEEE.

[22] Wang, X., Brown, A.R., Idris, N., Markov, S., Roy, G. and Asenov, A., 2011. Statistical threshold-voltage variability in scaled decananometer bulk HKMG MOSFETs: A full-scale 3-D simulation scaling study. *IEEE Transactions on Electron Devices, 58*(8), pp.2293-2301.

[23] Adamu-Lema, F., Wang, X., Amoroso, S.M., Riddet, C., Cheng, B., Shifren, L., Aitken, R., Sinha, S., Yeric, G. and Asenov, A., 2014. Performance and variability of doped multithreshold FinFETs for 10-nm CMOS. *IEEE Transactions on Electron Devices, 61*(10), pp.3372-3378.

[24] Wang, X., Cheng, B., Brown, A.R., Millar, C., Kuang, J.B., Nassif, S. and Asenov, A., 2013. Interplay between process-induced and statistical variability in 14-nm CMOS technology double-gate SOI FinFETs. *IEEE Transactions on Electron Devices, 60*(8), pp.2485-2492.

[25] Mutlu, A.A. and Rahman, M., 2005. Statistical methods for the estimation of process variation effects on circuit operation. *IEEE Transactions on Electronics Packaging Manufacturing, 28*(4), pp.364-375.

[26] Kroese, D.P., Brereton, T., Taimre, T. and Botev, Z.I., 2014. Why the Monte Carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics, 6*(6), pp.386-392.

[27] Montgomery, D.C., 2017. *Design and analysis of experiments*. John Wiley & Sons.

[28] Box, G.E. and Wilson, K.B., 1992. On the experimental attainment of optimum conditions. In *Breakthroughs in Statistics* (pp. 270-310). Springer New York.

[29] Plackett, R.L. and Burman, J.P., 1946. The design of optimum multifactorial experiments. *Biometrika, 33*(4), pp.305-325.

[30] Rasmussen, C.E. and Williams, C.K., 2006. *Gaussian processes for machine learning* (Vol. 1). Cambridge: MIT press.

[31] Krige, D.G., 1951. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy, 52*(6), pp.119-139.

[32] Gaussian Processes, 2017. Retrieved on August 25, 2017, from http://scikit-learn.org/stable/modules/gaussian_process.html

[33] Gaussian Processes, 2014. Retrieved on August 25, 2017, from http://scikit-learn.org/0.16/modules/gaussian_process.html

[34] Lophaven, S.N., Nielsen, H.B. and Søndergaard, J., 2002. *DACE-A Matlab Kriging toolbox, version 2.0*.

[35] Jones, D.R., Schonlau, M. and Welch, W.J., 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization, 13*(4), pp.455-492.

[36] Abramowitz, M. and Stegun, I.A., 1964. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables* (Vol. 55). Courier Corporation.

[37] Kohavi, R., 1995, August. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, No. 2, pp. 1137-1145).

[38] Zhang, H., Chen, T.H., Ting, M.Y. and Li, X., 2009, July. Efficient design-specific worst-case corner extraction for integrated circuits. *In Proceedings of the 46th Annual Design Automation Conference* (pp. 386-389). ACM.

[39] Fang, C., Huang, Q., Yang, F., Zeng, X., Zhou, D. and Li, X., 2016, June. Efficient performance modeling of analog integrated circuits via kernel density based sparse regression. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE* (pp. 1-6). IEEE.

[40] Li, X. and Cao, Y., 2008, March. Projection-based piecewise-linear response surface modeling for strongly nonlinear VLSI performance variations. In *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on* (pp. 108-113). IEEE.

[41] Blanton, R.D., Li, X., Mai, K., Marculescu, D., Marculescu, R., Paramesh, J., Schneider, J. and Thomas, D.E., 2015, November. Statistical learning in chip (SLIC). In *Computer-Aided Design (ICCAD), 2015 IEEE/ACM International Conference on* (pp. 664-669). IEEE.

[42] Chang, H.M., Cheng, K.T., Zhang, W., Li, X. and Butler, K.M., 2011, September. Test cost reduction through performance prediction using virtual probe. In *Test Conference (ITC), 2011 IEEE International* (pp. 1-9). IEEE.

[43] Qian, N., 1999. On the momentum term in gradient descent learning algorithms. *Neural networks, 12*(1), pp.145-151.

[44] Tsamardinos, I., Brown, L.E. and Aliferis, C.F., 2006. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine learning, 65*(1), pp.31-78.

[45] Khachaturyan, A., Semenovsovskaya, S. and Vainshtein, B., 1981. The thermodynamic approach to the structure analysis of crystals. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography, 37*(5), pp.742-754.

[46] Meadows, C.A., 1994, November. Formal verification of cryptographic protocols: A survey. In *International Conference on the Theory and Application of Cryptology* (pp. 133-150). Springer, Berlin, Heidelberg.

[47] Coudert, O. and Madre, J.C., 1990, November. A unified framework for the formal verification of sequential circuits. In *IEEE International Conference on Computer-Aided Design* (pp. 126-129).

[48] Gupta, A., 1992. Formal hardware verification methods: A survey. In *Computer-Aided Verification* (pp. 5-92). Springer US.

[49] Kern, C. and Greenstreet, M.R., 1999. Formal verification in hardware design: a survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES), 4*(2), pp.123-193.

[50] Camurati, P. and Prinetto, P., 1988. Formal verification of hardware correctness: Introduction and survey of current research. *Computer, 21*(7), pp.8-19.

[51] Althoff, M., Rajhans, A., Krogh, B.H., Yaldiz, S., Li, X. and Pileggi, L., 2013. Formal verification of phase-locked loops using reachability analysis and continuization. *Communications of the ACM, 56*(10), pp.97-104.

[52] Kanj, R., Joshi, R. and Nassif, S., 2006, July. Mixture importance sampling and its application to the analysis of SRAM designs in the presence of rare failure events. In *Design Automation Conference, 2006 43rd ACM/IEEE* (pp. 69-72). IEEE.

[53] Avizienis, A., Laprie, J.C. and Randell, B., 2001, May. Fundamental concepts of computer system dependability. In *Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments* (pp. 1-16).

[54] Hastings, W.K., 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika, 57*(1), pp.97-109.

[55] Au, S.K. and Beck, J.L., 2001. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic engineering mechanics, 16*(4), pp.263-277.

[56] Biegler, L.T., Grossmann, I.E. and Westerberg, A.W., 1985. A note on approximation techniques used for process optimization. *Computers & chemical engineering, 9*(2), pp.201-206.

[57] Downs, J.J. and Vogel, E.F., 1993. A plant-wide industrial process control problem. *Computers & chemical engineering, 17*(3), pp.245-255.

[58] Biegler, L.T., Cervantes, A.M. and Wächter, A., 2002. Advances in simultaneous strategies for dynamic process optimization. *Chemical engineering science, 57*(4), pp.575-593.

[59] Hanson, T.C., Bonaquist, D.P. and Jordan, M.D., Praxair Technology, Inc., 1994. *Chemical process optimization method*. U.S. Patent 5,315,521.

[60] Floudas, C.A., 2000. Global optimization in design and control of chemical process systems. *Journal of Process Control, 10*(2), pp.125-134.

[61] Babyak, M.A., 2004. What you see may not be what you get: a brief, nontechnical introduction to overfitting in regression-type models. *Psychosomatic medicine, 66*(3), pp.411-421.

[62] Rätsch, G., Onoda, T. and Müller, K.R., 2001. Soft margins for AdaBoost. *Machine learning, 42*(3), pp.287-320.

[63] McAndrew, C.C., Lim, I.S., Braswell, B. and Garrity, D., 2013, September. Corner models: Inaccurate at best, and it only gets worst…. In *Custom Integrated Circuits Conference (CICC), 2013 IEEE* (pp. 1-4). IEEE.

[64] Hartigan, J.A. and Wong, M.A., 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), pp.100-108.

[65] Garey, M.R. and Johnson, D.S., 2002. *Computers and intractability* (Vol. 29). New York: wh freeman.

[66] Jain, A.K., 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters, 31*(8), pp.651-666.

[67] Zhan, Y., Strojwas, A.J., Li, X., Pileggi, L.T., Newmark, D. and Sharma, M., 2005, June. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Proceedings of the 42nd annual Design Automation Conference* (pp. 77-82). ACM.

[68] Cockburn, B.F. and Boyle, K., 2006, May. Design and Characterization of a Digital Delay Locked Loop Synthesized from Black Box Standard Cells. In *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on* (pp. 1214-1217). IEEE.

[69] Marill, T. and Green, D., 1963. On the effectiveness of receptors in recognition systems. *IEEE transactions on Information Theory, 9*(1), pp.11-17.

[70] Whitney, A.W., 1971. A direct method of nonparametric measurement selection. *IEEE Transactions on Computers, 100*(9), pp.1100-1103.

[71] Duvenaud, D., 2014. *Automatic model construction with Gaussian processes* (Doctoral dissertation, University of Cambridge).

[72] Regression metrics, 2017. Retrieved on August 25, 2017, from http://scikit-learn.org/stable/modules/classes.html#regression-metrics

[73] Springenberg, J.T., Klein, A., Falkner, S. and Hutter, F., 2016. Bayesian optimization with robust Bayesian neural networks. In *Advances in Neural Information Processing Systems* (pp. 4134-4142).

[74] automl/RoBO, 2017. Retrieved on September 1, 2017, from https://github.com/automl/RoBO