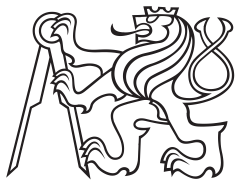


Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Novel Geometric-Programming Formulations in Computer-Aided Design of Integrated Circuits

Adam Bosák

**Supervisor: Mgr. Jakub Mareček, Ph.D.
Supervisor–specialist: Dmytro Mishagli, Ph.D.
May 2022**

I. Personal and study details

Student's name: **Bosák Adam** Personal ID number: **492310**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Cybernetics**
Study program: **Open Informatics**
Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Novel Geometric-Programming Formulations in Computer-Aided Design of Integrated Circuits

Bachelor's thesis title in Czech:

Nové modely geometrického programování pro návrh integrovaných obvodů

Guidelines:

The computer-aided design of integrated circuits has long benefitted from geometric-programming (GP) formulations, wherein GP is a broad generalization of linear programming that can be solved to any fixed precision in polynomial time. So far, however, GP formulations utilized only rudimentary models of delay. Independently, there is a long tradition of work on statistical static timing analysis (SSTA). Interesting recent contributions include those of dr. Mishagli and prof. Blokhina from the University College Dublin. This line of research approximates the delay as a random variable using Gaussian mixture models (or rather, radial basis functions, RBF) and introduces effective methods for implementing certain operations therewith (e.g., convolutions). This can be seen as an extension of histogram approximations of random variables. This dissertation aims to explore the use of histogram approximations and RBF approximations within the geometric-programming formulations of computer-aided design of integrated circuits. In particular, the student shall:

- Get acquainted with the recommended literature.
- Get acquainted with CVXPY, the Python library.
- Implement certain operations for histogram approximations and compare those against Monte Carlo sampling.
- Extend known GP formulations of Boyd et al. towards the use histogram approximations and RBF approximations.
- Implement these extensions using CVXPY in Python and test them on at least one sample circuit.

Bibliography / sources:

- [1] S. Boyd, S.-J. Kim, L. Vandenberghe, A. Hassibi: A Tutorial on Geometric Programming. Optimization and Engineering, 8(1):67-127, 2007.
- [2] S. Boyd, S.-J. Kim, S. S. Mohan, M. Horowitz, D. Patil: Circuit Design via Geometric Programming. ICCAD Tutorial, 2004.
- [3] D. Mishagli, E. Koskin, E. Blokhina: Path-Based Statistical Static Timing Analysis for Large Integrated Circuits in a Weak Correlation Approximation. IEEE International Symposium on Circuits and Systems (ISCAS) 2019.
- [4] J. Freeley, D. Mishagli, T. Brazil, E. Blokhina: Statistical simulations of delay propagation in large scale circuits using graph traversal and kernel function decomposition. 15th International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2018.

Name and workplace of bachelor's thesis supervisor:

Mgr. Jakub Mareček, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Dmytro Mishagli, Ph.D. University College Dublin

Date of bachelor's thesis assignment: **14.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Mgr. Jakub Mareček, Ph.D.
Supervisor's signature

prof. Ing. Tomáš Svoboda, Ph.D.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to express my considerable appreciation to Dr. Jakub Mareček and Dr. Dmytro Mishagli for their valuable and constructive suggestions during the planning and development of this research work. Their willingness to give their time so generously has been very much appreciated. The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 “Research Center for Informatics” is also gratefully acknowledged.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of the university thesis.

20. May 2022

.....

Abstract

Geometric programming is often used in the layout and timing problems of circuit design. The gate sizing problem is perhaps the best known special case. To the best of our knowledge, the literature has used either deterministic delay models or approximating statistical models, so far. A significant part of the thesis includes the solution to finding a multiplication of two numbers using only optimization variables that can then be used in the SSTA algorithm. This thesis also presents two new solutions of the statistical approach to the gate sizing problem using mixed-integer programming and geometric programming. In particular, a histogram approximation of the statistical approach is employed. We apply our proposed algorithms to the ISCAS benchmark circuit and compare the results with the deterministic approach.

Keywords: Mixed-Integer Programming, Geometric Programming, Gate Sizing, Statistical Static Timing Analysis

Supervisor: Mgr. Jakub Mareček, Ph.D.

Geometrické programování se často používá při problémech s uspořádáním a časováním při návrhu obvodů. Problém velikosti hradel je možná nejznámějším speciálním případem. Literatura dosud buď používala deterministické modely nebo aproximující statistické modely zpoždění. Významnou částí práce je řešení hledání násobení dvou čísel pouze pomocí optimalizačních proměnných, což lze následně použít v algoritmu SSTA. Tato práce také představuje dvě nové řešení statistického přístupu k problému velikosti hradel pomocí smíšeného celočíselného programování a geometrického programování. Zejména se používá aproximace histogramem statistického přístupu. Aplikujeme námi navržené algoritmy na obvod rodiny ISCAS a porovnááme výsledky s deterministickým přístupem.

Klíčová slova: Smíšené Celočíselné Programování, Geometrické Programování, Optimalizace Hradel, SSTA

Contents

Acronyms	1
List of Notations	3
Vector Spaces and Sets	3
Linear Algebra	3
Functions and Other	3
1 Introduction	5
2 Related Work: Geometric Programming for Circuit Design	7
2.1 Monomials and posynomials	7
2.2 Geometric Program	8
2.3 Solving a GP	8
2.3.1 How MOSEK solves GP	8
2.4 GP extensions and generalized posynomials	9
2.4.1 Simple extensions	9
2.4.2 Fractional powers	10
2.4.3 Maximum of posynomials	10
2.4.4 Other non-trivial extensions	10
2.5 Generalized Geometric program	11
3 Statistical Static Timing Analysis	13
3.1 Introduction	13
3.2 Exact computation	14
3.2.1 Maximum	14
3.2.2 Convolution	15
3.2.3 New edges	16
3.2.4 Shifting a histogram	17

3.3 SSTA as Mixed-integer problem	18
3.3.1 Unary encoding	18
3.3.2 Maximum and Convolution	19
3.3.3 Normalization	21
3.3.4 Problem Tightening	23
3.3.5 Scalability	24
3.3.6 SSTA as an optimization problem	25
3.4 SSTA via Geometric Programming	26
3.4.1 Maximum, Convolution and SSTA	26
3.4.2 Problem relaxation	26
3.5 Discussion and Conclusions	29
4 Gate Sizing	31
4.1 Background review: Deterministic approach	32
4.1.1 Formulating the delay	33
4.1.2 Formulating the upper bounds	34
4.1.3 Formulating the GGP	34
4.2 Statistical approach	35
4.2.1 Regression model	36
4.2.2 Distribution bounds	38
4.2.3 Objective function	39
4.2.4 Final Mixed-Integer Program	40
4.2.5 Final geometric program	41
4.3 Discussion and Conclusions	43
5 Conclusions and Critical Overview of the Thesis	45
Bibliography	47

A Source Code

Figures

3.1 Sketch of new edges	17
3.2 Histogram with 5 bins and width of the bin is 1	19
3.3 Maximum of 2 histograms with 200 bins and 200 unary variables with overflow	22
3.4 Maximum of 2 histograms with 200 bins and 200 unary variables with normalization.	23
3.5 Comparison of 3 methods of the SSTA realization.	24
3.6 Scalability of a three-term multiplication model.	25
3.7 Scalability of a GP model with relaxation constraints and fixed number of bins	28
3.8 Scalability of a GP model with relaxation constraints and fixed number of gates	28
4.1 A high-level diagram of c17 circuit from ISCAS-85	31
4.2 DAG of the c17 circuit from ISCAS-85	32
4.3 Delay simulation of the inverter	37
4.4 Change of distributions	38
4.5 Scalability of the final geometric program	42
4.6 Comparison of the delay distributions	43

Tables

4.1 Input gate parameters of the ISCAS-85 c17 circuit	35
4.2 Optimal sizing parameters of the ISCAS-85 c17 circuit using the deterministic model	35
4.3 Optimal sizing parameters of the ISCAS-85 c17 circuit using the statistical GP model	41



Acronyms

CAD Computer-Aided Design

GP Geometric Program

SP Signomial Program

GGP Generalized Geometric Program

SSTA Statistical Static Timing Analysis

STA Deterministic Static Timing Analysis

RV Random Variable

PDF Probability Density Function

CDF Cumulative Distribution Function

VLSI Very Large Scale Integration

MAPE Mean Absolute Percentage Error

ISCAS-85 1985 International Symposium on Circuits And Systems

DAG Directed Acyclic Graph

VaR Value-at-Risk

CVaR Conditional Value-at-Risk

ISPD International Symposium on Physical Design

IC Integrated Circuit

RBF Radial Basis Function

List of Notations

Vector Spaces and Sets

\mathbb{R}	the set of real numbers
\mathbb{N}	the set of natural numbers without 0
\mathbb{R}^n	the vector space of n -dimensional vectors
\mathbb{R}_+^n	the cone of non-negative n -dimensional vectors
\mathbb{R}_{++}^n	the cone of strictly positive n -dimensional vectors
(a, b)	the open interval of real numbers between a and b
$[a, b]$	the closed interval of real numbers between a and b

Linear Algebra

$\mathbf{A} \in \mathbb{R}^{m \times n}$	a matrix with m rows and n columns
$\mathbf{a} \in \mathbb{R}^m$	a vector with m rows and 1 column
$A_{i,j}$	component in i -th row and j -th column of the matrix \mathbf{A}
a_i	component in i -th row of the vector \mathbf{a}
$\mathbf{A}_{i,:}$	i -th row of the matrix \mathbf{A}
$\mathbf{A}_{:,i}$	i -th column of the matrix \mathbf{A}
$\mathbf{A}_{i,:n}$	first n components in the i -th row of the matrix \mathbf{A}
$\mathbf{A}_{i,n:}$	i -th row of the matrix \mathbf{A} without first $n - 1$ components
$\mathbf{1}$	1 -th vector with appropriate number of rows



Chapter 1

Introduction

The Integrated Circuits (ICs) must work at expected frequencies with respect to the timing constraints specified in their designs, which is checked by Computer–Aided Design (CAD) tools. At the same time, designers want to minimize the area taken by the designs on a chip and the power consumption. This leads to a particular class of optimization problems. The gate sizing is one of such problems.

There have been many approaches introduced since 1980s to tackle the gate sizing. This was summarized in a seminal paper by Boyd *et al.* in 2005 [8], where a critical assessment of the field was done as well as the general statement of the gate sizing problem as a Geometric Program (GP) was made. However, in that paper as well as in subsequent ones up to the most recent work [28], a deterministic models of a gate delay are used. Such models while give advantage it terms of low algorithm complexity and, thus, high overall speed of computations, give too pessimistic (i.e. overestimated) delays of Very Large Scale Integration (VLSI) designs. An attempt to mitigate this brought to a rise of a new field, Statistical Static Timing Analysis (SSTA). A novel approach within the SSTA was recently developed at University College Dublin (Ireland) [20, 25, 26]. This approach allows one to consider non-Gaussian distributions of the gates' delays without any loss of information while keeping complexity low enough to use it for the circuit optimization problems.

In this thesis, a histogram–based version of the approach [20, 26] is presented. We use histogram approximations of the distribution functions and show their potential in the evaluation of the SSTA and further VLSI optimization. We present two formulations of the problem. We use mixed-integer programming for the first and geometric programming for the second. We introduce more effective relaxations and solutions to their drawbacks. We then show how to formulate and solve the gate sizing problem using the histogram–based approach to SSTA.

The thesis is organized as follows.

Chapter 2 gives an introduction to Geometric Programming and Generalized Geometric Programming. The Chapter is a useful handout for the basics of the Geometric Programming.

Chapter 3 introduces a formulation of the SSTA in terms of histograms. In this Chapter, it is shown how the SSTA can be formulated as an optimization problem. Two formulations are given, using (i) a mixed–integer programming and (ii) geometric programming.

Chapter 4 is dedicated to the gate sizing problem. The problem is stated and the background review is given. The problem is then solved using two approaches to the SSTA developed in the previous Chapter. The results are compared with the deterministic approach.

Chapter 5 presents conclusions and critical overview of the work.

Chapter 2

Related Work: Geometric Programming for Circuit Design

This chapter introduces geometric programming. In the first section, monomial and posynomial functions are presented. Geometric program is defined in the second subsection with a quick note on how geometric program is solved. In the following subsections, some extensions to geometric programming and generalized geometric programs are discussed.

It has been observed ([7]) that GP or Generalized Geometric Program (GGP) either approximate very well or is equivalent to circuit design optimization problems.

GP and GGP can be converted into a nonlinear convex program using logarithmic transformation of the variables, objective and constraint functions. Fortunately, convex programs can nowadays be very efficiently solved. Furthermore, GP always finds a global optimum solution for a feasible problem, requires no parameter tuning and no starting point. In addition, the strong duality holds for any GP in convex form and so the duality gap is zero¹.

These facts implicate that GP modelling gives us a very powerful tool for solving various optimization problems.

2.1 Monomials and posynomials

Let us first define monomial and posynomial functions as these are used in the GP. According to [7], monomial is a function: $\mathbb{R}_{++}^n \rightarrow \mathbb{R}_{++}$ ²

$$f(x_1, x_2, \dots, x_n) = cx_1^{a_1} \cdots x_n^{a_n}, \quad (2.1)$$

where $c > 0$ and $a_i \in \mathbb{R}$. We call c coefficient of the monomial and a_i exponents of the monomial. We refer to a sum of monomials as a posynomial ([7]), i.e., function in the form

$$f(x_1, x_2, \dots, x_n) = \sum_{k=1}^K c_k x_1^{a_{1k}} \cdots x_n^{a_{nk}}. \quad (2.2)$$

It is clear from the definitions that for the set of all monomials A and for the set of all posynomials B , it holds that $A \subseteq B$. One should also note that posynomials are closed under addition, multiplication, positive scaling and results in posynomial when divided by a monomial.

¹Please see [14] for proof.

²The domain of the monomials is the non-negative quadrant of \mathbb{R}^n . We assume that the optimum values cannot be zero and so the domain is in the form \mathbb{R}_{++}^n .

2.2 Geometric Program

Based on [7], geometric program is an optimization problem in the form

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 1, \quad i = 1, \dots, n \\ & && g_i(x) = 1, \quad i = 1, \dots, p, \end{aligned} \quad (2.3)$$

where x_i are optimization variables, f_i are posynomial functions, and g_i are monomials. We call (2.3) a geometric program in a standard form. One should distinguish geometric optimization from the geometric programming defined in (2.3). Geometric optimization is referred to the optimization of parameters driven by the geometry of a system.

It is sometimes easy to transform a program that does not look like a GP into one using some elementary operations, such as division or inverse. Time showed that more complex extensions are needed for the optimization of real-world problems. For example, for the digital circuit gate sizing problem on which we concentrated. This led to the idea of generalized posynomials and GGP which is further discussed in (2.4).

2.3 Solving a GP

As is widely known [14], minimization of a general polynomial optimization problem is NP-Hard. This is due to the fact that the objective function is non-convex. Even deciding the local optimality of a non-convex quadratic program is NP-Hard. However, in geometric programming, we consider a very special, posynomial objective. With a logarithmic change of all variables, GP in standard form can be turned into a convex program which can be solved with polynomial-time algorithms. The change of variables is $y_i = \log x_i$, $b_{ik} = \log c_{ik}$ and $b_l = \log c_l$. Program is then in the form ³

$$\begin{aligned} & \text{minimize} && \log \sum_{k=1}^{K_0} \exp(a_{0k}^T y + b_{0k}) \\ & \text{subject to} && \log \sum_{k=1}^{K_i} \exp(a_{ik}^T y + b_{ik}) \leq 0, \quad i = 1, \dots, n \\ & && a_l^T y + b_l = 0, \quad l = 1, \dots, p, \end{aligned} \quad (2.4)$$

where n is a number of inequalities, p is a number of equalities, $a_{ik} = [a_{ik}^{(1)}, a_{ik}^{(2)}, \dots, a_{ik}^{(m)}]^T$, i.e., exponents of the k -th monomial of the i -th posynomial in a vector. Notice that this can be an arbitrarily close approximation of the original posynomial problem.

2.3.1 How MOSEK solves GP

Solver MOSEK has been used for gate sizing optimization. Having a deeper understanding of how the GPs are solved in MOSEK will be important in the section about relaxation of the problem 3.4.2.

³Proof for 2.4 being a convex program is located in [14].

According to the [4, 13] the exponential cone is defined as a convex subset of \mathbb{R}^3 as

$$K_{\text{exp}} = \{(x, y, z) : x \geq ye^{z/y}, y > 0\} \cup \{(x, 0, z) : x \leq 0, z \geq 0\},$$

as shown in [13], such set is the closure of the following points

$$\{(x, y, z) : x \geq ye^{z/y}, x > 0, y > 0\}.$$

The proof of the convexity of the cone is the positive semidefinite Hessian of $f(y, z) = ye^{z/y}$ for $y > 0$. We can now clearly see that the inequality from (2.4)

$$\log \sum_{k=1}^{K_i} \exp(a_{ik}^T y + b_{ik}) \leq 0,$$

can be written using

$$u_k \geq \exp(a_{ik}^T y + b_{ik}), \text{ (equiv. } (u_k, 1, a_{ik}^T y + b_{ik}) \in K_{\text{exp}}), \quad (2.5)$$

$$\sum_k u_k = 1. \quad (2.6)$$

Minimization of the posynomial leads to a minimization of the $\sum_k u_k$. Note that for each monomial, we need to introduce 1 exponential cone and 2 new variables: auxiliary variable u_k and a slack variable s such that $-a_{ik}^T y + s = b_{ik}$ and 2 new constraints: (2.5) and constraint for the slack.

2.4 GP extensions and generalized posynomials

According to the definition in [7], generalized posynomials are functions formed from posynomials using the following operations: addition, multiplication, positive fractional power, and maximum. By the definition, one can see that generalized posynomials are closed under all operations from the previous sentence.

In order to be able to transform generalized posynomial into a posynomial or a program into a standard-GP-compatible form, some basic GP extensions need to be addressed.

Unless specified otherwise, f denotes posynomials and g monomials in the next subsections.

2.4.1 Simple extensions

The following two trivial extensions are possible due to the already mentioned fact, observed in [7], that posynomials are closed under division. The inequality constraint

$$f_i(x) \leq g_i(x),$$

can be changed into

$$f_i(x)/g_i(x) \leq 1, \quad (2.7)$$

using the fact that $f_i(x)/g_i(x)$ is a posynomial. We can also similarly transform the equality constraint

$$g_i(x) = g_j(x)$$

into

$$g_i(x)/g_j(x) = 1. \quad (2.8)$$

Finally, the maximization of the objective monomial function is equivalent to the minimization of the inverse.

2.4.2 Fractional powers

By the definition of standard GP (2.3), we already know that decimal powers of posynomials are standard-GP-compatible inequalities. However, it would not be valid for some fractional powers. In [7], a broadly used trick is introduced that is typically used in linear programming. For every posynomial f_i powered by fraction z , we can introduce new auxiliary variable s_i . These should act as upper bounds and substitutions for the functions f_i . From

$$f_1(x) + \dots + f_i(x)^z + \dots + f_n(x) \leq 1,$$

we get

$$\begin{aligned} f_i(x) &\leq s_i \\ f_1(x) + \dots + s_i^z + \dots + f_n(x) &\leq 1, \end{aligned} \quad (2.9)$$

which are standard-GP-compatible inequalities.

2.4.3 Maximum of posynomials

Very similar trick in [7] is used when minimizing the maximum of a finite number of posynomials. From

$$\text{minimize } \max\{f_1(x), \dots, f_n(x)\},$$

we get

$$\begin{aligned} &\text{minimize } s \\ &\text{subject to } f_i(x) \leq s, \forall i \\ &\text{variables } s, x, \end{aligned} \quad (2.10)$$

which is again in a valid standard form.

2.4.4 Other non-trivial extensions

In [7], some more extensions are introduced. Since they are not needed for the digital circuit gate sizing problem that we concentrated on, we will just mention them and not dive deeper.

First extension is an approximation and fitting of the non-posynomial objective function using the first-order Taylor expansion.

Next is the so-called Signomial Program (SP). SP subject / objective function $s : \mathbb{R}_{++}^n \rightarrow \mathbb{R}$, is in the form

$$s(x_1, x_2, \dots, x_n) = \sum_{i=1}^N c_i g_i(x_1, x_2, \dots, x_n). \quad (2.11)$$

Note that $\mathbf{c} \in \mathbb{R}^n$. Signomial subject function allows to express not only upper bound inequalities but also lower bound inequalities or equalities. Equality constraints are used in network modelling⁴. It is important to note that compared to GP, a signomial program cannot be transformed into a convex one and the dual gap is non-zero.

■ 2.5 Generalized Geometric program

Based on the informations and definitions in [7], the GGP is in the form

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 1, \quad i = 1, \dots, n \\ & && g_i(x) = 1, \quad i = 1, \dots, p, \end{aligned} \tag{2.12}$$

where g_1, \dots, g_p are monomials, f_0, \dots, f_n are generalized posynomials. Using the GP extensions in 2.4, GGP can be transformed into an equivalent GP which has already all the good qualities described in 2. This transformation can be easily done during the parsing of the program. Using the previous fact, solvers do not need to have any transformations from GGP into GP implemented.

⁴Please see [14].

Chapter 3

Statistical Static Timing Analysis

In this chapter, the delay of a signal in digital circuits is calculated using the SSTA. Finding a delay of the circuit is critical for the gate sizing problem. First, general principles of SSTA are outlined. Then, a *histogram approximation* to SSTA with exact numbers is discussed. Finally, two SSTA algorithm formulations as an optimization problem are presented. This constitutes one of the original contributions by the Author.


3.1 Introduction

Deterministic Static Timing Analysis (STA) is a standard way to take into account systematic process variations [29]. At the same time, the delay values computed in such a way are too pessimistic [6]. This results in increased chips' cost, when these delays are attempted to mitigate [31]. In modern ultra-VLSI circuits (5nm and below), the impact of random correlated processes and fluctuations is significant and cannot be neglected. That is why the SSTA is been developed.

SSTA addresses randomness in a natural way, treating the delays in a system as Random Variables (RVs). The analysis then allows one to determine the mean value of the delay across selected paths. The maximum delay corresponds to the critical path. Current industrial realizations of SSTA allow one to determine moments of delay distributions and/or their quantiles [10, 11, 12]. In principle, SSTA can give a slack¹ distribution of the whole circuit, i.e., Probability Density Function (PDF) and/or Cumulative Distribution Function (CDF). This makes SSTA comparable to Monte Carlo simulations in terms of accuracy. At the same time, SSTA algorithms are much less resourceful.

In this work, we use a simplified version of the approach to SSTA proposed in [20, 26]: a histogram approximation. Only in the estimation of the maximum, we assume that the inputs are uncorrelated. The same code of the general SSTA algorithm is used for all the exact computation 3.2 and both optimization formulations (3.3, 3.4). Crucial parts of the algorithm are the computations of maximums and convolutions. Their solutions are presented in the next sections. One can see in the Algorithm 1 the general version of the algorithm² based on [20].

¹A *slack* is defined as a difference between the required arrival time of a signal and its actual arrival time.

²Full algorithm is at  or can be found in the repository <https://github.com/bosakad/GP-Optimization>.

Algorithm 1: General SSTA algorithm**Data:** Histogram approximation of N gates, number of gates N **Result:** A histogram of the delay of a circuit**for** $i \leftarrow 1, \dots, N$ **do** $M \leftarrow$ max of input gates; $C \leftarrow$ convolution of gate(i) and M ; propagate C further as input PDF; $D \leftarrow$ max of output PDFs;

3.2 Exact computation

In this section, we will present the exact computation of the maximum and the convolution. We will not dive too deeply into its problematic, as its purpose is mainly for comparison and explanation of the optimization problem. In this section, PDFs are represented by the histogram approximation. Each histogram is represented by a pair of NumPy array of individual bin values and a NumPy array of edges. General assumption for the histograms is to have the same array of edges. This assumption also holds for both optimization problems.

Unless specified otherwise, N denotes the number of bins for each histogram in the following subsections.

3.2.1 Maximum

We will try to find the histogram approximation of the maximum ξ of two independent random variables η, ζ . We assume set of bins $M = \{0, 1, \dots, N-1\}$, the histogram samples η_i and ζ_i take the value in the interval $[A, B] \forall i$. Given the edges interval $[A, B]$, we partition \mathbb{R} into N -intervals n_1, n_2, \dots, n_N with points e_1, e_2, \dots, e_{N+1} such that e_i is the start of the interval n_i and e_{i+1} is the end of n_i , $e_1 = A$, $e_{N+1} = B$ and $|n_i| = |B - A|/N, \forall i$. The midpoints of the intervals m_1, m_2, \dots, m_N are also given. The maximum can be formulated as

$$\xi = \begin{cases} \eta, & \text{if } \eta \geq \zeta, \\ \zeta, & \text{if } \eta < \zeta. \end{cases} \quad (3.1)$$

Using the law of total probability, the probability at realization $z \in [A, B]$ can be written as

$$p(\xi = z) = p(\xi = z, \eta \geq \zeta) + p(\xi = z, \eta < \zeta). \quad (3.2)$$

Using the definition in (3.1), we can change (3.2) to

$$p(\eta = z, \eta \geq \zeta) + p(\zeta = z, \eta < \zeta). \quad (3.3)$$

In our case of independent RVs η and ζ , we can make the final changes. The probability of a maximum is

$$p(\xi = z) = p(\eta = z) \cdot p(\zeta \leq z) + p(\zeta = z) \cdot p(\eta < z). \quad (3.4)$$

The discrete random variable formulation and histogram estimations h_η, h_ζ and h_ξ are now easily expressed as follows at $i \in M$

$$h_{\xi}[i] = h_{\eta}[i] \cdot \sum_{k=1}^i h_{\zeta}[k] + h_{\zeta}[i] \cdot \sum_{k=1}^{i-1} h_{\eta}[k]. \quad (3.5)$$

Note that the upper bound of the second sum is $i - 1$. This is due to the fact that in (3.3) in the second term there is a strict inequality as one part of the joint probability.

A big advantage of this formulation is its speed-up potential. The first part of the optimization could be the precomputation of the cumulative sum for both histograms. The cumulative sum represents the cumulative distribution function, so 3.4 could be used. The cumulative sum can be easily computed in linear time. The second part of the speed-up could be a vectorization of the algorithm. Note that in (3.4) only 1 index is used for all functions and all values of the functions are independent of each other. The calculation of (3.4) can then also be done in linear time with vectorized instructions. The total time complexity of the optimized maximum algorithm is $\theta(2 \cdot N) = \theta(N)$.

The unoptimized version of the algorithm of this method is shown in the Algorithm 2 in pseudocode³ and will be important when formulating the maximum as an optimization problem in 3.3.

Algorithm 2: Maximum

Data: Number of bins N , two vectors x, y of histogram values

Result: Maximum of two histograms in m

$m \leftarrow \mathbf{0}$;

for $i \leftarrow 1, \dots, N$ **do**

for $j \leftarrow 1, \dots, i$ **do**

$m[i] \leftarrow m[i] + x[i] \cdot y[j]$;

if $i \neq j$ **then**



$m[i] \leftarrow m[i] + x[j] \cdot y[i]$;


3.2.2 Convolution

The convolution of two histograms can be performed using the `convolve` function from the NumPy library, which uses Fast Fourier transform. However, this solution cannot be used in the optimization formulation of the problem, so a general convolution has been used instead:

$$(f * g)(z) = \sum_{-\infty}^{\infty} f(k) \cdot g(z - k), \quad (3.6)$$

where f, g are some functions. The time complexity of the convolution is $\theta(N^2)$. The pseudocode⁴ can be seen in Algorithm 3. The formula (3.6) implies that the values of the edges must be changed. The value of the first edge has to be added to all other edges. This can be done in many ways and is discussed in the following subsections.

³Full code of maximum is at , vectorized version at  or can be found in the repository <https://github.com/bosakad/GP-Optimization>.

⁴Full code of the convolution can be seen at , or can be found in the repository <https://github.com/bosakad/GP-Optimization>.

Algorithm 3: Convolution**Data:** Number of bins N , two vectors x, y of histogram values**Result:** Convolution of two histograms in c $c \leftarrow \mathbf{0}$;**for** $z \leftarrow 1, \dots, N$ **do**

for $k \leftarrow 1, \dots, z$ do	$c[z] \leftarrow c[z] + x[k] \cdot y[z - k]$;
---	--

3.2.3 New edges

As already discussed, we need to add the first value of the interval to all edges after each convolution. The first option is to add the first value to all edges and unite them during the SSTA algorithm when the edges of the second histogram differ. The receipt is simple: find the new array of edges \mathbf{e} and modify the PDFs of histogram approximations f_α, f_β with the new changed edges. The array of edges of f_α is given as \mathbf{e}^α , similarly \mathbf{e}^β denote the array of edges of f_β .

To find the \mathbf{e} , we partition \mathbb{R} into N -intervals n_1, n_2, \dots, n_N with points e_1, e_2, \dots, e_{N+1} such that e_i is a start of the interval n_i and e_{i+1} is the end of the n_i , $e_1 = \min\{e_0^\alpha, e_0^\beta\}$, $e_{N+1} = \max\{e_{N+1}^\alpha, e_{N+1}^\beta\}$ and $\forall i: |n_i| = |e_{N+1} - e_1|/N$.

The easiest way to do this modification is to use the function `rv_histogram` from `scipy.stats` library. This function finds a distribution function that fits the given histogram. Let us say that F is the fitted cumulative distribution function of the f_α . Then PDF at the realization $z \in n_i$ of the new histogram $f_{\alpha'}$ with the desired edges \mathbf{e} is

$$f_{\alpha'}(z) = F(e_{i+1}) - F(e_i). \quad (3.7)$$

Another way to do so is by exact integration over the bins of the histogram:

$$f_{\alpha'}(z) = \int_{e_i}^{e_{i+1}} f_\alpha(x) dx. \quad (3.8)$$

The same would be done for $f_{\beta'}$. Solution (3.8) gives more precise results and bypasses the problem of fitting functions, which is relatively computationally demanding, all at the cost of a slightly longer code. Exact integration can be performed in $O(N)$ for the whole histogram⁵.

⁵Full code of (3.8) can be seen at [\[9\]](#), and (3.7) at [\[9\]](#), or can be found in the repository <https://github.com/bosakad/GP-Optimization/>.

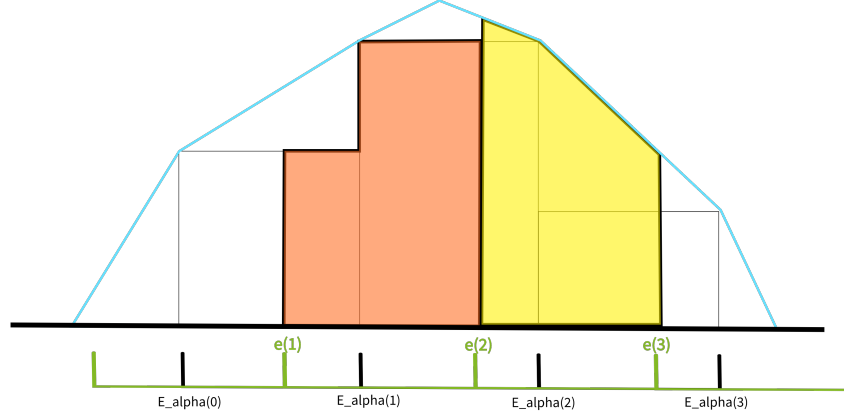


Figure 3.1: Sketch demonstrating the integration over the fitted function (3.7) (yellow surface) and the exact integration (3.8) (red surface). Blue function represents the fitted PDF.

When one looks at the SSTA algorithm in Alg. 1, a problem with such a union of edges is evident. After convolutions of the input gates, every time a maximum and then convolution should be computed, the edges will differ and have to be united. Taking into account the two inputs for each gate, the function (3.7) or (3.8) is called twice per gate. Moreover, more problems are to come when trying to solve a convolution optimization problem.

A different and more straightforward solution is presented in the next subsection.

3.2.4 Shifting a histogram

A second solution to the problem of adding a value to the edges is a simple shifting. We can shift the whole histogram to the left or right by the number of bins determined by a value that is to be added to all edges divided by the length of the bins and floored. Shifting the value of a bin by such a number simulates the addition of the first value to all edges. We assume n bins, set of bins $B = \{0, 1, \dots, n - 1\}$ the identical edges array of the histograms is given as $\mathbf{e} \in \mathbb{R}^{(n+1) \times 1}$, histograms as h_α, h_β , s denotes the shift. The shift can be computed as ⁶

$$s = \lfloor \frac{|e_0|}{e_1 - e_0} \rfloor. \quad (3.9)$$


In the case of $e_0 > 0$, the new changed histogram $h_{\alpha'}$ (equivalently for $h_{\beta'}$) at point $x \in B$ will look like

$$h_{\alpha'}[x + s] = h_\alpha[x] \quad (3.10)$$

In the case of $e_0 < 0$, the shift is very similar:

$$h_{\alpha'}[x] = h_\alpha[x + s] \quad (3.11)$$

When shifting to the right, there are s unoccupied positions on the left. These are nullified. Similarly, done when shifting to the left. Having the starting interval set correctly, this does not

⁶Please see the code at , or can be found in the repository <https://github.com/bosakad/GP-Optimization/>.

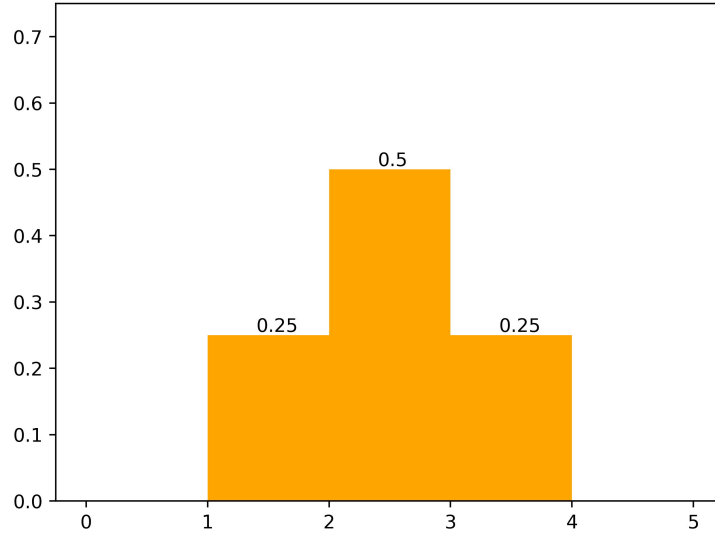


Figure 3.2: Histogram with 5 bins and width of the bin is 1

The histogram in Figure 3.2 can be represented by a 5x3 matrix:

$$h = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

It is evident that not all real values can be encoded by a finite number of binary values. The more columns we have, the more precise the encoding will be.

In the next subsections, N denotes the number of rows (bins) and M denotes the number of columns (unary variables).

■ 3.3.2 Maximum and Convolution

In this section, a procedure is presented on how to find the multiplication of two real numbers encoded in the unary notation and how it can be used in the convolution and the multiplication.

The best way to explain this is to look at an example. Let us have two natural numbers α , β encoded in unary notation in vectors \mathbf{a} , \mathbf{b} . By definition, the multiplication of two numbers is equal to repeated addition: $\alpha \cdot \beta = (\beta^{(1)} + \beta^{(2)} + \dots + \beta^{(\alpha)})$. This can be used in vector parallel. Each 1 in the multiplier \mathbf{a} can be interpreted as one addition of all ones in the multiplicands vector \mathbf{b} and can be easily written as matrix multiplication:

$$\mathbf{1}_{1 \times \alpha}^T \begin{bmatrix} \mathbf{a} & \mathbf{a} & \dots & \mathbf{a} \end{bmatrix}_{\alpha \times \beta} \mathbf{b}_{\beta \times 1} \quad (3.12)$$

Having this explained, we can move on and show how we can enforce the multiplication of two unary variables. We are given two unary variables x and y , our goal is to find their product. To do

$$\begin{aligned}
 \mathbf{Z}_{1,:}\mathbf{1} &\leq \mathbf{1}^T \mathbf{v}_1 \\
 &\dots \\
 &\dots \\
 \mathbf{Z}_{N,:}\mathbf{1} &\leq \mathbf{1}^T \mathbf{v}_N,
 \end{aligned} \tag{3.14}$$

we then propagate \mathbf{Z} further in the SSTA. It is important to bear in mind that the constraints (3.14) will work only for a maximization problem. Inequality signs should be flipped for the minimization problem. These particular constraints bring us a new problem and will be discussed in section 3.3.3.

Let me end this subsection with a quick discussion about edge modifications. We can add a new argument on why to use histogram shift (3.2.4) over the computation of the histogram with new edges (3.2.3). In order to simulate the integration over the bins to find the new histogram (3.8), we would need to introduce new constraints. In contrast, the shift of the bins can be done very quickly and without any new constraints.

■ 3.3.3 Normalization

Using the same notation as in the previous subsection, we can find a large problem in (3.14). Knowing the elements of the matrix $Z_{i,j}$ take only the value 0 or 1. Thus, the maximum possible value reached on the left side of the inequality is M . However, the other side of the inequality can reach even larger values than M . This would lead, in the case of a minimization problem, to infeasibility in certain situations as the inequality sign would be flipped in (3.14). This is why a maximization problem is more suitable.

So what happens if the right side of the constraint (3.14) for the i -th bin takes the value $M + a$, where $a \in \mathbb{N}$? This becomes infeasible for the minimization, for the maximization, the maximum possible value M is set for $Z_{i,1} + \dots + Z_{i,M}$ (if other constraints allow it), and the information about a ones is cut. This information-cut leads to many problems and will be called an overflow. The obvious first problem is an unnatural cutting of high peaks at some height. Secondly, we already know from the section 3.3.1 that the probability of each bin is calculated as a ratio of the number of non-zero elements in the row and the number of overall non-zero elements times the width of the bin. Therefore, not only will the mean value change but also the standard deviation will increase drastically.

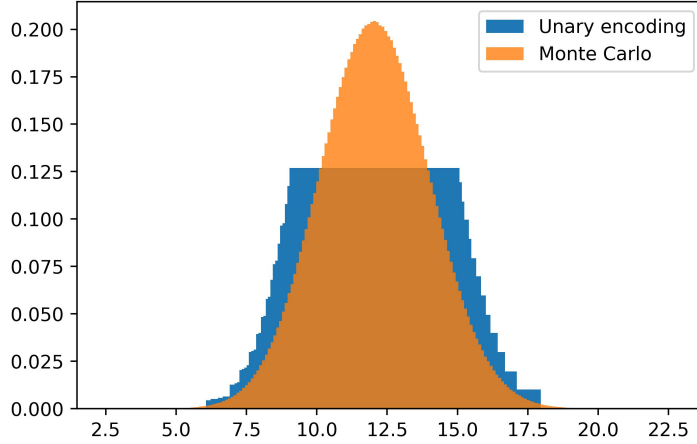


Figure 3.3: Maximum of 2 histograms with 200 bins and 200 unary variables with overflow

The solution to this is a normalization of the histogram. Using the notation of (3.14), we can divide each sum of the row \mathbf{v}_q for the bin q by a parameter $x \in \mathbb{R}_{++}$, constraints (3.14) will change in the following

$$\begin{aligned}
 \mathbf{z}_{1,:} \mathbf{1} &\leq \mathbf{1}^T \mathbf{v}_1 \cdot \frac{1}{x} + 0.5 \\
 &\dots \\
 &\dots \\
 \mathbf{z}_{N,:} \mathbf{1} &\leq \mathbf{1}^T \mathbf{v}_N \cdot \frac{1}{x} + 0.5,
 \end{aligned} \tag{3.15}$$

where the scalar 0.5 rounds the right side as it becomes a positive real number after division by $x \in \mathbb{R}_{++}$. We denote $w_i = \mathbf{1}^T \mathbf{v}_i$, then the parameter x should have properties such that $\frac{\max\{w_1, \dots, w_N\}}{x} \leq M$. Too large x prevents overflow, but also loses some information by rounding the divided rows with small sums. Too small x does not cut any information at the cost of a possible overflow. Taking the ideal (minimal possible) x we get

$$\frac{\max\{w_1, \dots, w_N\}}{M} = x. \tag{3.16}$$

So, theoretically, such a perfect x can be found. Can it also be done for the optimization scenario? Assuming that we are solving a minimization problem, finding a maximum of integer variables w_q should not be a problem. Dividing it by a scalar constant M is also not a problem. Thus, the answer is yes, we can find such x , but we cannot use it for the division, as $\frac{w_q \cdot M}{\max\{w_1, \dots, w_N\}}$ is yet again not a convex function.

We now have two options on how to solve this. First, we can take the maximal possible value of w_q and not let any overflow happen even in the worst case. This would be $N \cdot M^2 + (N - 1) \cdot M^2$ for a maximum and $N \cdot M^2$ for a convolution (derived from Algorithms 4 and 2). Such values are unfortunately too large, and we would lose too much information by division.

The second option is to set x as a parameter. For demonstration purposes, $x = \frac{1}{30} \cdot N \cdot M$ and $x = \frac{1}{22} \cdot N \cdot M$ were set for the maximum and the convolution consecutively.

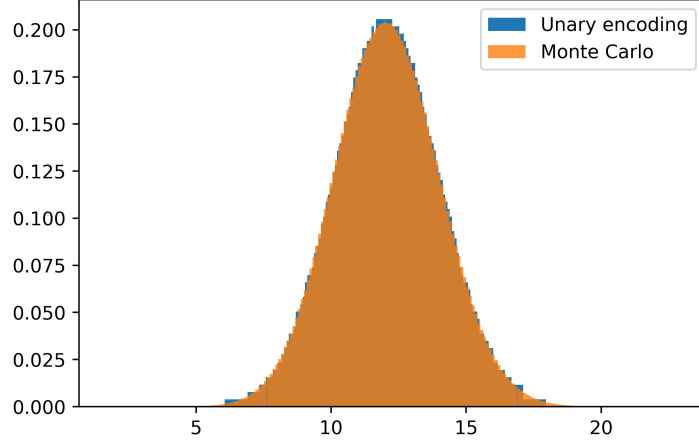


Figure 3.4: Maximum of 2 histograms with 200 bins and 200 unary variables with normalization.

3.3.4 Problem Tightening

In order to speed up the convergence rate of the optimization algorithm, we need to introduce constraints that tighten the problem as much as possible. An ideal tightening should be performed with a minimal number of new variables and constraints.

Good tightening constraints in our case of convolution and maximum are constraints that enforce a separation of zeros and ones in the matrix row. We denote the new matrix representing the convolution / maximum by $\mathbf{Z} \in \{0, 1\}^{N \times M}$ with $N \cdot M$ unary variables, we introduce (3.15), and for each row i we introduce

$$\begin{aligned}
 Z_{i,1} &\geq Z_{i,2} \\
 Z_{i,2} &\geq Z_{i,3} \\
 &\dots \\
 &\dots \\
 Z_{i,N-1} &\geq Z_{i,N}.
 \end{aligned} \tag{3.17}$$

Such constraints give a solver a good starting point, thus having this done for all rows, the convergence is drastically faster for more complex problems.

We do not have to concentrate on the separate maximums and convolutions and instead use the steps in SSTA algorithm to tighten our problem. Rather than finding a $\{0, 1\}^{M \times N}$ matrix, as presented in (3.15), after each convolution and maximum, we can keep c_b pairs of variable indices for each bin b after performing the maximum without introducing any constraints. We further propagate the list of the indices as one of the inputs for the convolution and use these pairs to introduce three-term multiplication constraints after the convolution. We denote ζ , η and

ξ the random variables, their histogram approximations by $\hat{\zeta}$, $\hat{\eta}$ and $\hat{\xi}$. Let $X = \{(x, y, z) | (x, y)$ are variables saved after the maximum of $\hat{\zeta}$ and $\hat{\eta}$, z is a variable saved after convolution of the maximum with $\hat{\xi}\}$. For each triplet (x, y, z) we can find s , new constraints similar to (3.13) are:

$$\begin{aligned}
 s &\leq x, \\
 s &\leq y, \\
 s &\leq z, \\
 s &\geq x + y + z - 2.
 \end{aligned} \tag{3.18}$$

Again, the last constraint is not needed for the maximization problem in all cases. Next, we can create a new matrix $\mathbf{Z} \in \{0, 1\}^{N \times M}$ with $N \cdot M$ unary variables, introduce constraints (3.15) using the new variables s , also constraints (3.17). We then propagate \mathbf{Z} further. One can see in Figure 3.5 the comparison of the relaxed problems with the original problem⁸.

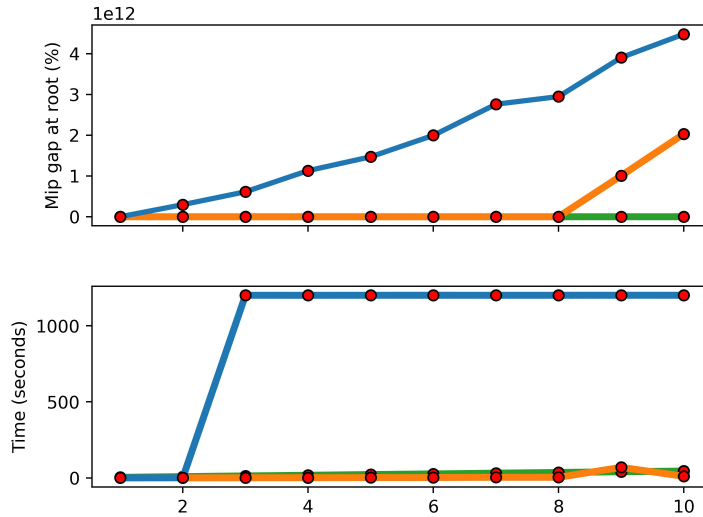


Figure 3.5: Comparison of 3 methods of the SSTA realization. The methods are tested on a “ladder” of maximums and convolutions with 10 bins, 10 unary variables, and a time limit of 20 minutes (1200 seconds). The blue line indicates the original method, the orange line indicates a method with separation constraints (3.17) and green line a method with separation constraints and a three-term multiplication model (3.18).

3.3.5 Scalability

If we look at the algorithms and their solutions, we will find that the number of variables and constraints introduced for each convolution/maximum is very large. Even more so in the case of a tightened problem with separation constraints and a three-term multiplication model. However, if we look at the Figure 3.6, we can see that the number of constraints and variables scale linearly with the number of gates.

⁸Full code of three-term multiplication model with separation constraints in MOSEK API is at [📄](#), its vectorized version at [📄](#) or all can be found in the repository <https://github.com/bosakad/GP-Optimization>.

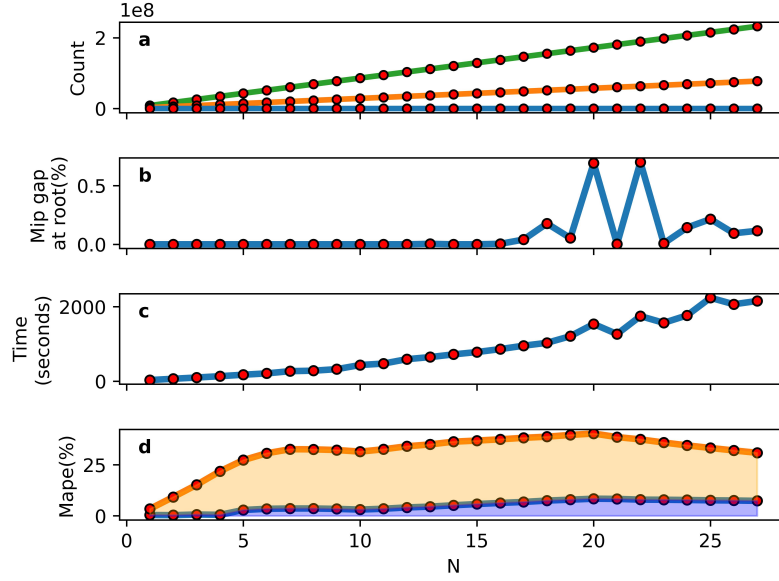


Figure 3.6: Scalability of a three-term multiplication model (3.18 and 3.17) tested on a ladder of maximums and convolutions with 20 bins, 10 unary variables, and no time limit. The subplots show: (a), the increase in the number of non-zeros (blue line), variables (orange line), and constraints (green line). (b), MIP gap at a root node in percentage, MIP gap tolerance is set to 1%; (c), time in seconds; (d), Mean Absolute Percentage Error (MAPE) of the standard deviation (orange line) and mean (blue line) compared to Monte Carlo.

Please pay attention to the number of non-zeros indicating that the problem was presolved. Also, having only 10 unary variables causes a higher error in standard deviation. The sign of a good relaxation is a nearly zero relative (MIP) gap at the root node in a relaxation tree, enabling us to solve even a problem with 232475400 constraints and 77506400 variables.

3.3.6 SSTA as an optimization problem

Having the convolution and the maximum functional, we can dive into the SSTA as an optimization problem. We assume N gates, n bins, and m unary variables. First, we need to enforce the variables to fit the Gaussian distribution. To do so, we can generate numbers with the probability of normal distribution with a given mean value and standard deviation, create a histogram using these numbers, and represent it in unary encoding using the receipt 3.3.1. For each input gate g , we have a matrix $\mathbf{E}^g \in \{0, 1\}^{n \times m}$ created from generated numbers with Gaussian probability and a matrix $\mathbf{Z}^g \in \{0, 1\}^{n \times m}$ of binary variables. For each gate, for each pair $i, j \in \mathbb{N}, i \leq N, j \leq M$, we introduce

$$\mathbf{E}_{i,j}^g \leq \mathbf{Z}_{i,j}^g. \quad (3.19)$$

We can introduce all needed constraints and matrix variables mentioned in (3.15) and (3.17) as we traverse the circuit. We can then specify the objective function. The objective is the sum of all the variables of the last (sink) gate. This function is minimized. Flip the inequality of (3.19) for the maximization task.

the N -th gate we will have m^{N-1} monomials in the last bin, each with N variables. This clearly leads to an exponential growth in monomials after each convolution and maximum for a constant number of bins. As shown in 2.3.1, for each monomial in the posynomial we need to introduce an exponential cone, two continuous variables, and two constraints. Thus, for a constant number of bins, the growth of variables, cones, and constraints is exponential with the number of gates. For constant number of gates $N + 1$, the growth in variables, cones, and constraints with number of bins can be computed as the sum of all bins $\sum_{i=1}^m i^N$, we know thanks to Faulhaber's formula¹⁰ that this sum can be expressed as a polynomial of degree $N + 1$, and so is the growth.

We can reduce this by a very simple trick: we can introduce m new positive variables (monomials) and set appropriate constraints. At the beginning of the traverse, we initiate two empty sets of vectors N_{succ} and N_{pred} of the successor and predecessor vectors of variables. We denote the function $f_{pred} : G \rightarrow N_{pred}$ that maps the predecessor posynomials and $f_{succ} : G \rightarrow N_{succ}$ that maps the new created monomials. After the convolution of the gate i , we save the result vector of posynomials \mathbf{x}_i in N_{pred} , we also create a new vector of one-variable monomials $\mathbf{n}_i \in \mathbb{R}_{++}^{m \times 1}$ and store it in N_{succ} . We then use this vector \mathbf{n}_i as a new vector representing the gate histogram and further propagate it in the SSTA. We also update the functions $f_{pred}(i) = \mathbf{x}_i$ and $f_{succ}(i) = \mathbf{n}_i$. For the last gate l , we save the vector \mathbf{n}_i and denote it by \mathbf{s} . Relaxed SSTA is in the form¹¹

$$\begin{aligned}
 & \text{minimize} && \mathbf{1}^T \mathbf{s} \\
 & \text{subject to} && f_{pred}(g) \leq f_{succ}(g), \quad \forall g \in G \\
 & && \mathbf{e}_g \leq \mathbf{z}_g \leq \mathbf{1}, \quad g = 1, \dots, n.
 \end{aligned} \tag{3.21}$$

Such relaxation gives the exact same solution as the non-relaxed version. For each convolution, we introduce $(m/2)(1 + m)$ new exponential cones, thus $(2m/2)(1 + m)$ help variables, and m new upper bound variables. We just decreased the exponential growth of variables, cones and constraints to a linear one with the number of gates, and a high-degree polynomial growth with the number of bins to always quadratic. The numbers are a little different for the maximum; however, the growths before and after relaxations are identical. The scalability of the GP model with the relaxation constraints is shown in Figures 3.7 and 3.8.

¹⁰See [24] for more details.

¹¹Full code of the optimized convolution can be seen at <https://github.com/bosakad/GP-Optimization/>, optimized maximum at <https://github.com/bosakad/GP-Optimization/>, or can be found in the repository <https://github.com/bosakad/GP-Optimization/>.

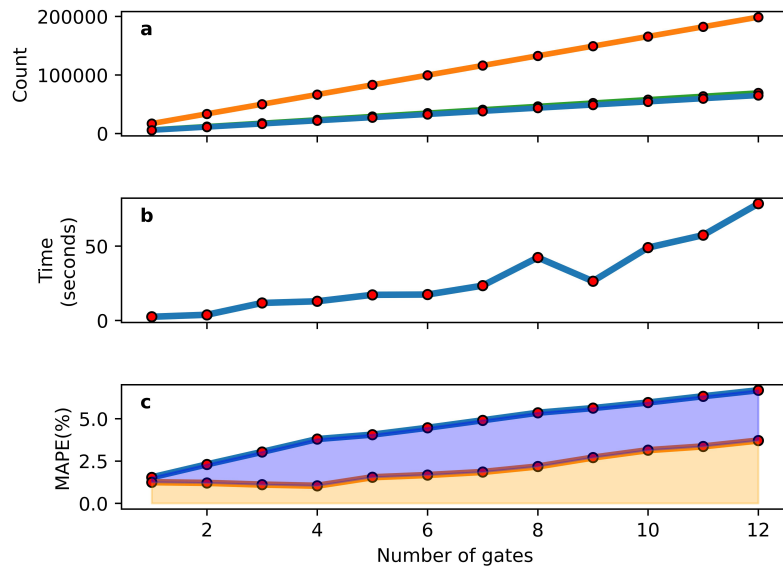


Figure 3.7: Scalability of a GP model with relaxation constraints (3.21) tested on a ladder of maximums and convolutions with fixed 60 bins and no time limit. The subplots show: **a**, the increase in the number of cones (blue line), variables (orange line), and constraints (green line - overlapped with blue line); **b**, time in seconds; **c**, MAPE of the standard deviation (orange line) and mean (blue line) compared to Monte Carlo.

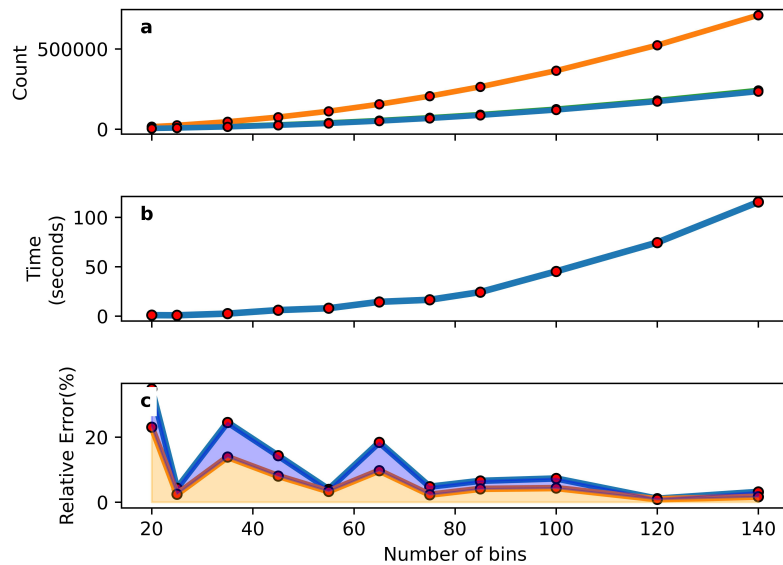


Figure 3.8: Scalability of a GP model with relaxation constraints (3.21) tested on a ladder of maximums and convolutions with fixed 8 gates and no time limit. The subplots show: **a**, the increase in the number of cones (blue line), variables (orange line), and constraints (green line - overlapped with blue line); **b**, time in seconds; **c**, absolute relative error of the standard deviation (orange line) and mean (blue line) compared to Monte Carlo.

One should pay attention to the fact that (3.21) can be changed into a standard-GP-compatible inequality using the simple extension (2.7) as $\forall i : y_i$ is a monomial. Thus, relaxed (3.20) is a

generalized geometric program.

■ 3.5 Discussion and Conclusions

In this chapter, we presented a solution to finding a delay distribution using the SSTA algorithm in terms of histogram approximations.

- (i) Firstly, we introduced and derived the general algorithms for the maximum and the convolution. We also showed its speed-up potentials and their implementation in Python language.
- (ii) We expanded these to be solvable as an optimization problem. First formulation uses mixed-integer programming. We presented how scalability issues of such formulations can be addressed. We introduced two relaxations to this problem, (3.17) and (3.18), and the error-resolving constraints (3.15). Despite NP-hardness of the computation of the general mixed-integer program, we computed the delay distribution even for 28 gates using the relaxation constraints.
- (iii) Second formulation uses the geometric programming. We derived the scalability problem, its solution (3.21), and showed its potential (3.7–3.8),

We can now compute the delay distribution of the whole circuit for at least 400 gates under 7 minutes using the GP model, and 28 gates in 34 minutes using the mixed-integer model. Delay is a vital part of the gate sizing problem, which is discussed in the next chapter.

Chapter 4

Gate Sizing

In this chapter, we will introduce an example of a real-world geometric optimization problem. In particular, we focus on the digital circuit gate sizing problem. In the first subsection, we show various deterministic formulations of the problem. In the second subsection, we present some other reviews and two of our formulations with a histogram approximation of SSTA using the mixed-integer programming and geometric programming.

Our goal is to minimize the worst-case circuit delay, also called a critical path, under the maximum area and circuit power consumption constraints. The main optimization variable is a vector \mathbf{x} of sizing parameters. Sizing parameter $x_i \geq 1$ is a scale factor of the i -th gate, i.e. a scale factor of transistors the i -th gate is made from. In the case of $x_i = 1$, the gate is at its minimal possible size. As the number of transistors increases, so does the speed, size, and consumption of the circuit.

For the demonstration purposes, let me use the circuit c17 shown in Fig. 4.1 from the 1985 International Symposium on Circuits And Systems (ISCAS-85) benchmark family [9, 21].

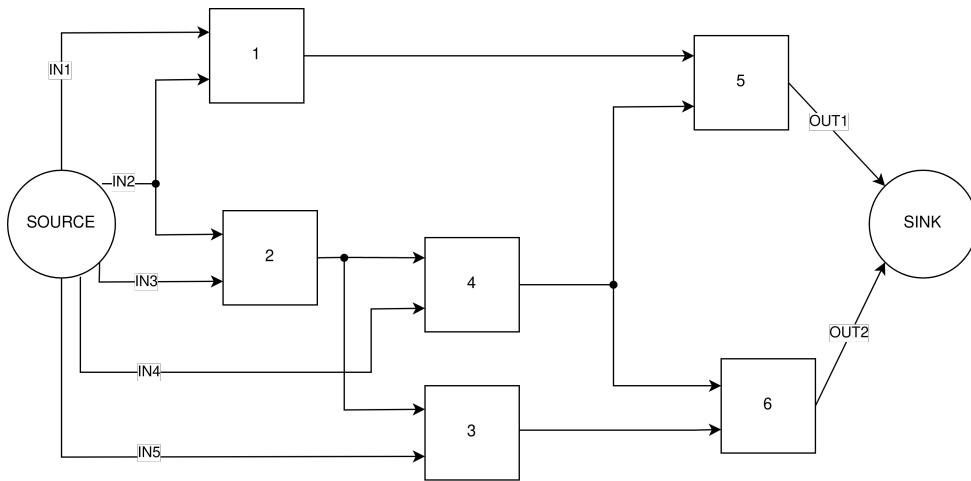


Figure 4.1: A high-level diagram of c17 circuit from ISCAS-85. A circuit has 6 gates, 5 inputs and 2 output gates.

General convention is that digital circuits are represented in the following way. Each circuit has a single source and a sink. Fan-out gates of the source are called primary input gates. Fan-in gates of the sink are called primary output gates. From the mathematical point of view, digital circuits can be described by Directed Acyclic Graphs (DAGs). Such graphs have the following properties:

- DAG is formed by vertices and edges.

- Each vertex (logic gate) is connected by edges to another vertex.
- Each edge has an orientation.
- Graph has no cycles.

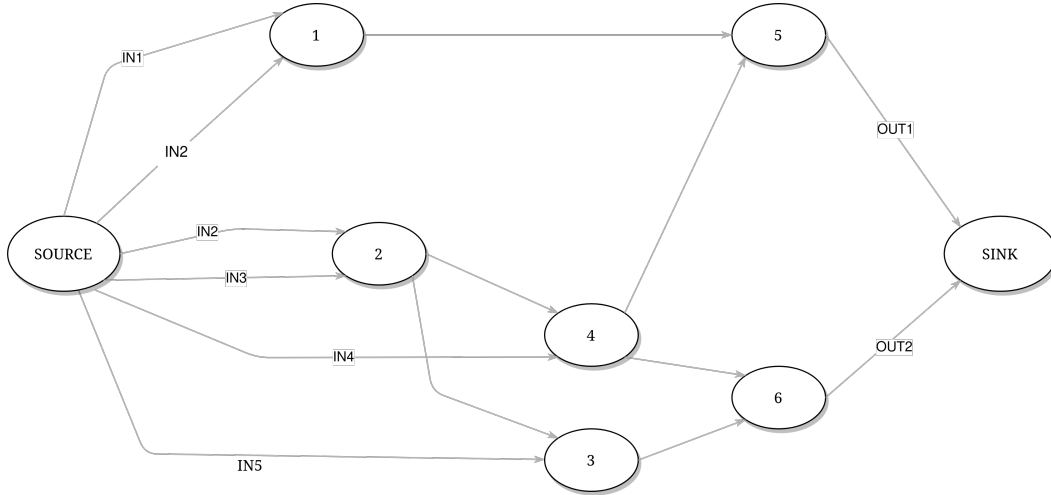


Figure 4.2: The digital circuit c17 expressed by a Directed Acyclic Graph.

These requirements are needed so the logic works as expected.

A standard-cell-based methodology is used: each gate (square) in Fig. 4.1 represents a generalization of the logical elements, so-called logic cells. Some particular examples are NAND, AND, and OR gates.

4.1 Background review: Deterministic approach

To our knowledge, the best-known formulation is Boyd’s deterministic gate sizing model [7] that uses the so-called RC -model of a delay. This is our main inspiration and will be discussed in more detail in Subsections about delay (4.1.1), area and power consumption models (4.1.2) and final program (4.1.3). It is appropriate to say that Joshi and Boyd [23] introduced a more efficient formulation of Boyd’s model: a reformulation of the [7] using soft-max, soft-min and timing constraints, suitable for pseudo-Newton methods. This seems too complicated to extend to the SSTA, and so our main inspiration remains Boyd’s original model.

A classic, simple model [15] used signal arrival times (AT) and required time (RT) for each gate of the circuit to analyze the delay. A slack is defined as $S(n) = AT(n) - RT(n)$ for each gate n . The critical path has the minimal sum of slacks. This leads to maximization of the slacks¹. The hindrance of such a method is that each time a sizing parameter is changed for some gate, time expensive re-evaluation of the whole circuit has to be done in some cases. The same area model as in Boyd’s approach [7] was used. The power in a gate is $P = P_{\text{load}} + P_{\text{internal}} + P_{\text{leakage}}$, where P_{load} is power lost in charging and discharging of the gate, P_{internal} depends on the transition time and on the internal loads and P_{leakage} is due to a leakage of a current.

¹Note that depending on definition, the problem can be either to minimize or to maximize the delay or slack. In this thesis, we speak of minimization of the maximum delay in a circuit.

Similarly old approach introduced in [5] formulated the gate sizing problem as a linear program. They linearized the delay model for this purpose. The main aim was to minimize power consumption and tune the delay by adapting the load drive capabilities. The introduced linear program has always a global optimum and can be solved very quickly using the Simplex method. Nevertheless, the delay model can lack accuracy.

Three years later, in a paper [30] the *RC* delay model mentioned above was introduced together with the first usage of posynomial functions. The gate sizing problem was formulated as a minimization of the area subject to a given maximal delay; however, other formulations are possible. A method to finding a global optima of the equivalent constraints in the form of a sum of exponentials is presented. We show in 2.3.1 better ways of solving this.

A more complex solution called NP-Separate has been recently introduced in [17] that sizes not only cells of the circuit but also PFET and NFET transistors of each cell. The drawbacks of such method are: unlike our approach, type of cells matter - ideally they should have regular, symmetric layout patterns (such inverters XOR, NAND, NOR, ...); Secondly, overlapping of NP cell instances can occur leading to unroutable layout. However, taking a 'closer' look at the problem might, as shown, yield better results. This is a completely different approach from standard cell-based methodology, but is worth reading for an interested reader.

Another approach has been published in [16], where is a min-max resource usage formulated as the resource sharing problem where each gate is a customer. A parallelization of the method has been presented in the paper as well. A typical delay *RC*-model was selected. Their results compared to those presented at International Symposium on Physical Design (ISPD) 2013 are significantly faster and reduce total power consumption better.

A similar approach to Boyd's model was presented in [27]. The area and delay model is again the same as in Boyd's approach [7]. Power consumption is not taken into account. Naidu adds pipelining constraints enforced by a 0 – 1 variables for each gate representing presence of the pipelining register. This complicates and changes the delay model in some ways and is not taken into account in our formulation. See also a promising new improvement of such a formulation with the convex first-order methods in [28] for more information.

■ 4.1.1 Formulating the delay

In case of *RC*-model, to formulate the delay objective function, we need to compute the input capacitance of each gate, the load capacitance of each gate, the driving resistance, the delay of each gate and lastly find the maximum expected delay of the circuit. Following [7, 8], we choose the input capacitance C_i and driving resistance R_i as follows

$$C_i = \alpha_i + \beta_i x_i, \quad (4.1)$$

$$R_i = \gamma_i / x_i, \quad (4.2)$$

where α is the internal capacitance, β is the wire load capacitance and γ is a gate resistance constant.

Load capacitance is computed as a sum of the fan-out input capacitances. If it is an output gate, the load capacitance C_i^{load} should be given. In our case (Figure 4.1) we have

$$\begin{aligned}
C_1^{\text{load}} &= C_5^{\text{input}} \\
C_2^{\text{load}} &= C_3^{\text{input}} + C_4^{\text{input}} \\
C_3^{\text{load}} &= C_6^{\text{input}} \\
C_4^{\text{load}} &= C_5^{\text{input}} + C_6^{\text{input}}
\end{aligned} \tag{4.3}$$

With C_5^{load} and C_6^{load} given, we have all load capacitances computed. Delay of a gate is a product of its driving resistance and its load capacitance:

$$D_i = R_i C_i^{\text{load}}. \tag{4.4}$$

As a last step, we express the maximum of all possible paths in the circuit.

$$\begin{aligned}
D = \max\{ & D_1 + D_5, \\
& D_2 + D_4 + D_5, \\
& D_2 + D_4 + D_6, \\
& D_4 + D_5, \\
& D_4 + D_6, \\
& D_2 + D_3 + D_6, \\
& D_3 + D_6\}.
\end{aligned} \tag{4.5}$$

Note that (4.1) is a posynomial, (4.2) is a monomial, (4.3) is a sum of posynomials, (4.4) is a posynomial, and hence the maximum delay is a generalized posynomial (because of the maximum). A good solution to finding a maximum delay (4.5) is using the dynamic programming.

■ 4.1.2 Formulating the upper bounds

In Boyd's formulation, the upper bounds are given as a scalar of maximum area and as a scalar of maximum power consumption. The total area can be computed as

$$A = \sum_{i=1}^n a_i x_i, \tag{4.6}$$

where a_i is the area of gate i with unit scaling. Total power can be expressed as

$$P = \sum_{i=1}^n f_i e_i x_i, \tag{4.7}$$

where f_i is a frequency of transition of the i -th gate and e_i is the energy loss of the gate transitions.

■ 4.1.3 Formulating the GGP

We already know that delay (4.5) is a generalized posynomial. Area (4.6) and power consumption (4.7) are posynomials. Let us formulate the Boyd's gate sizing GGP then

$$\begin{aligned}
& \text{minimize} && D \\
& \text{subject to} && A \leq A_{max} \\
& && P \leq P_{max} \\
& && x_i \geq 1, \quad i = 1, \dots, n.
\end{aligned} \tag{4.8}$$

Let me now show this procedure on a specific example. Consider the following scenario: maximum area is 35, maximum power consumption is 55, and the gate parameters can be seen in Table 4.1.

Paramater	Gate 1	Gate 2	Gate 3	Gate 4	Gate 5	Gate 6
Frequency	4	0.8	1	0.8	1.7	0.5
Energy Loss	1	2	1	1.5	1.5	1
Area scale	1	1	1	1	1	1
Output Capacitance	-	-	-	-	7	5

Table 4.1: Input gate parameters of the ISCAS-85 c17 circuit

The MOSEK solver has been used for the optimization. The optimal value was reached after 23 iterations and 8.5 milliseconds. The value of the primal problem (maximal delay) is 4.07 with a zero duality gap. The optimal sizing parameters for the gates are listed in Table 4.2.

	Gate 1	Gate 2	Gate 3	Gate 4	Gate 5	Gate 6
Sizing parameters	2.38	13.19	3.13	7.21	4.33	3.09

Table 4.2: Optimal sizing parameters of the ISCAS-85 c17 circuit using the deterministic model

4.2 Statistical approach

In the previous sections, the gate sizing problem was considered for the case of deterministic delays, i.e. each delay was given by a single scalar value. In this section, and this is the main goal of the thesis, we discuss the statistical formulation of the problem.

In the case where delays in an integrated circuit cannot be determined precisely, they are given by random variables. Thus, one should speak of statistical properties of circuits: delays can be described in terms of their moments (mean value, standard deviation) or by corresponding distributions.

We are not the first to solve the gate-sizing problem using a statistical approach. The first statistical attempts occurred even in the year 2000 in [22]. Jacobs and Berkelaar approximated the distribution of the maximum by a normal distribution and found an analytical expression of mean and standard deviation of the maximum of two normal distributions. After the traverse, they use these moments to minimize the delay of the circuit. The benefit of such a solution is that the gate sizing problem can be formulated as a nonlinear program and is very quick due to the formulas of the moments — they presented feasibility for up to a few thousand gates. On the other hand, they also lose a lot of information by matching moments of the normal distribution of the maximum, as the distribution of the of two RVs is not a normal one.

Five years later, a sensitivity based statistical solution was presented in [1]. Their theory is based on the perturbation bounds of the delay of the circuit. One can identify by using the bounds the

highest sensitive gate for sizing and prune out the less sensitive ones without explicitly propagating their effect. The optimization objective is defined on these upper bounds. The runtime of such a method is linear with the circuit size.

Very recently, the two-phase gate sizing method was introduced in [19]. In the first phase, the timing yield of the circuit is optimized by computing the criticality of each gate, ranking them by this criticality and gate size the group of gates with the highest level of criticality. In the second phase, the incremental method finds the group of most effective gates to size by the probabilities of aging-induced delay degradation and position on the critical path due to process variation and aging. Such a method should result in lower area overhead.

The hindrance behind all these previous statistical methods is that they either use the so-called canonical first-order (or second-order) delay model, or approximate the non-Gaussian delay distribution by a Gaussian one. This is what we do differently: we consider the whole distribution instead of some parameterized function, and the only information loss is caused by the assumption of the uncorrelated RVs and by the histogram approximation. This loss can be reduced by an increase in the number of bins and by setting the correct interval. The idea of our approach is identical to Boyd's RC -model (4.8). We want to minimize the critical path of the circuit, i.e. minimize the maximum delay of all possible paths. In statistical formulation, this maximum delay is represented by the sink node distribution. PDF of the sink node distribution is given to us as an output of the SSTA algorithm after the circuit traverse.

In this section, we will formulate the gate sizing problem as that for exact statistical optimization. The first formulation is the so-called mixed-integer formulation, the second one is the model using geometric programming.

4.2.1 Regression model

As a first step, we have to find a model that can change parameters of the distribution of the delay according to the change in area and power. The easiest way to do so is to use linear least squares regression. We assume the same model for each gate. Our methods are not explicitly restricted to these models, and so other more efficient models can be used. Using the simulations, we can find n distributions of 1 gate, each differing from others in area and power. Then, using the linear least squares for each bin b we can find for the mixed-integer model an affine function $\varphi_b : \mathbb{R}_{++}^2 \rightarrow \mathbb{R}$ such that

$$\varphi_b(a, p) = z + xa + yp. \quad (4.9)$$

By restricting the function to a set $X = \{(a, p) \mid a \in \mathbb{R}_{++}, p \in \mathbb{R}_{++}\} \subseteq \mathbb{R}_{++}^2$ of area and power around given data points, we can assume the function to be $\varphi_b : X \rightarrow [0, 1]$, also the histogram built using these functions with the same area and power should integrate to 1 and thus generate us a distribution.

Affine function works great for the mixed-integer model. However, we need to have all coefficients of the function positive for the GP. An affine function with positive coefficients cannot decrease. We would like the probability of last bins to decrease with increase in area and power. A better fit was used instead for the GP formulation, we have n distributions, we denote data points of area and power for distribution i by a_i and p_i , vector \mathbf{c} contains consecutive probabilities of bin b for each distribution, \mathbf{x} is an optimization variable:

$$\min \left\| \begin{bmatrix} a_1 & p_1 & 1/a_1 & 1/p_1 \\ \dots & \dots & \dots & \dots \\ a_n & p_n & 1/a_n & 1/p_n \end{bmatrix} \mathbf{x} - \mathbf{c} \right\|_2^2 \quad (4.10)$$

s.t. $\mathbf{x} > \mathbf{0}$.

Fig. 4.3 shows results of the simulation of an inverter gate synthesized using 28 nm technology with Cadence Virtuoso RF Solution Circuit Simulation tool. The results were kindly provided by Pierre Bisiaux (UCD, Dublin). One can see that the delay distribution of a gate can be approximated by a normal distribution. We shall take this as an evidence of a Gaussian distribution of delays of *individual* logic gates (at least for 28 nm technology and above).

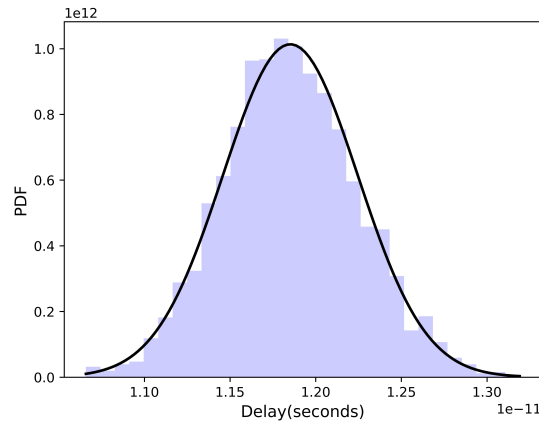


Figure 4.3: Delay simulation of the 28 nm inverter (purple histogram) and fitted PDF of a Gaussian distribution using `norm.fit` from `scipy.stats` library (black function). The estimated mean of the Gaussian distribution is 1.18×10^{-11} s, estimated standard deviation is 3.93×10^{-13} s.

To demonstrate the validity of the proposed methods, we use synthetic data. We generated 7 Gaussian distributions and 7 Lognormal distributions, then applied (4.10) on both data sets. In the following subsections, we will call the model gained from Gaussian distributions as the 'Gaussian model', similarly will be called the 'Lognormal model'. In Fig. 4.4, we can see how the distributions change according to the sizing parameter. We will use these two plotted models when demonstrating the final GP on the c17 circuit.

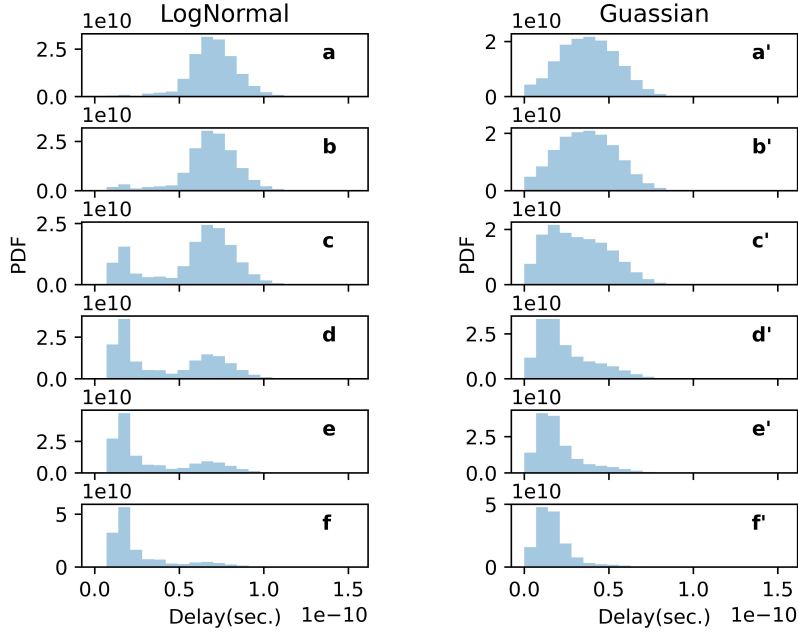


Figure 4.4: Change in the distributions of the gate delay according to the sizing parameters. The gate frequency is set to 0.5, energy loss to 1, and area coefficient to 1. The subplots show: **a-f**, consecutive PDFs of a Lognormal model according to the chosen sizing parameters (1, 2, 5, 10, 15, 25); **a'-f'**, consecutive PDFs of a Gaussian model according to the chosen sizing parameters (1, 2, 5, 10, 15, 25).

4.2.2 Distribution bounds

Before we dive into the final gate sizing program, at the beginning of the SSTA, we have to express the bounds of the PDFs using the regression model discussed above. In this subsection, a_i refers to the area of the gate i with unit scaling, f_i to the frequency of the transition of the gate i , e_i its transitions energy loss, and x_i gate sizing variable. Let us start with the bounds for the mixed-integer formulation. We restrict regression functions to the set X as mentioned in the previous subsection.

We denote the model (4.9) for each bin b by $\varphi_b : X \rightarrow [0, 1]$, number of unary variables M , number of bins N and we denote the matrix of unary variables representing the histogram approximation of the gate i as $\mathbf{A} \in \{0, 1\}^{N, M}$. For each gate i and its matrix \mathbf{A} , for each bin b , we introduce:

$$\hat{p}(b) = \varphi_b(a_i \cdot x_i, f_i \cdot e_i \cdot x_i), \quad (4.11)$$

$$\sum_{u=1}^M A_{b,u} \leq M \cdot \hat{p}(b) + 0.5, \quad (4.12)$$

$$\sum_{u=1}^M A_{b,u} \geq M \cdot \hat{p}(b) - 0.5. \quad (4.13)$$

Please note the similarity of (4.13) and (3.19). It is appropriate to point out that (4.13) is sufficient and (4.12) is just for the purposes of a faster convergence.

For the GP model, we assume the model (4.10) for each bin b to be $\varphi'_b : X \rightarrow [0, 1]$, number of bins N , vector of positive variables representing the histogram approximation of the gate i as $\mathbf{a} \in \mathbb{R}_{++}^{N \times 1}$. For each gate and its vector \mathbf{a} , for each bin b we introduce:

$$\hat{p}(b) = \varphi'_b(a_i \cdot x_i, f_i \cdot e_i \cdot x_i), \quad (4.14)$$

$$a_b \geq \hat{p}(b). \quad (4.15)$$

Note that a_b is a monomial and $\hat{p}(b)$ is a posynomial. Using once again trivial extension (2.7), we can say that this is a GGP-compatible inequality.

Also, the similarity of (4.15) and the inequalities in (3.20) is obvious.

4.2.3 Objective function

Ideally, we would like to minimize the mean value of the sink node. However, we cannot express the mean value as a convex function for both mixed-integer model and GP model. Let us first formulate the objective function for the mixed-integer model.

We assume n number of bins and m number of unary variables. We denote a set of unary variables by U and a set of bins by B , $\mathbf{d} \in \mathbb{R}^{n \times 1}$ midpoints of the bin intervals, and the variable matrix of the sink gate by $\mathbf{S} \in \{0, 1\}^{n \times m}$ given to us as an output of the SSTA. We can minimize a rough approximation of the mean

$$D = \sum_{b: b \in B} \left(\sum_{u: u \in U} S_{b,u} \right) \cdot d_b. \quad (4.16)$$

Another option is to minimize the tail of the distribution. This would minimize the expected value of the distributions tail, also known as the Conditional Value-at-Risk (CVaR). It is appropriate to note that we can minimize the CVaR only to some level, for example we cannot say we minimize the typical 99% CVaR as the position of the Value-at-Risk (VaR) depends on the yet not known number of ones in the last few bins. Let $B^{(k)}$ be the set of the last k bins of the histogram:

$$D = \sum_{b: b \in B^{(k)}} \left(\sum_{u: u \in U} S_{b,u} \right) \cdot d_b. \quad (4.17)$$

The objective for the GP model is very similar. Despite restricting the regression model, after some convolutions and maximums, the histogram does not integrate to one, and so we cannot speak of the mean minimization. For such problem, the minimization of the CVaR is ideal. Let $\mathbf{s} \in \mathbb{R}_{++}^{n \times 1}$ be the vector of positive variables representing the sink gate histogram given to us as an output of the SSTA, the objective is

$$D = \sum_{b: b \in B^{(k)}} s_b \cdot d_b. \quad (4.18)$$

This gives us a correct answer for the optimization variables; however, the distribution is in the first $n - k$ bins set to 1 (maximum possible). Let ϵ be a very a small number. We can overcome this problem by setting the objective to:

$$D = \sum_{b=1}^{n-k} s_b \cdot \epsilon + \sum_{b: b \in B^{(k)}} s_b \cdot d_b. \quad (4.19)$$

4.2.4 Final Mixed-Integer Program

We denote the number of gates n , set of gates $G = \{1, \dots, n\}$, number of bins m , set of bins $M = \{1, \dots, m\}$, number of unary variables u , correctly set interval $I = [h, k]$ partitioned into m equal subintervals each corresponding to 1 bin, variable matrix of the gate i by \mathbf{B}_i , set $B = \{\mathbf{B}_1, \dots, \mathbf{B}_n\}$, set of vectors $R = \{\mathbf{r}_1, \dots, \mathbf{r}_n\}$ of the corresponding right sides without the rounding scalar as mentioned in (4.13). We denote the i -th standard-basis vector by \mathbf{e}_i .

At first, we initiate two empty sets X and N_{succ} . We denote the functions $f_{trip} : G \times M \rightarrow X$ that maps the triplets and auxiliary variables for each gate and bin, $f_{sum} : G \rightarrow \mathbb{R}^{m \times 1}$ that maps the sum of all auxiliary variables for each gate in the vector, and $f_{succ} : G \rightarrow N_{succ}$ that maps the new created matrix variables. We initialize for all gates g and bins b : $f_{tri}(g, b) = \{\}$ and $f_{sum}(g) = \mathbf{0}$.


After performing the maximum and saving the pairs of unary variables for each bin, we perform the convolution with the gate g and store the corresponding l -th triplets (w, y, z) together with new auxiliary variable s for each bin b in the set X . Finally, we update the functions $f_{trip}(g, b) = f_{trip}(g, b) \cup \{(w, y, z, s)\}$ and $f_{sum}(g) = f_{sum}(g) + s \cdot \mathbf{e}_b$.

After each convolution at the gate g , we create a new matrix $\mathbf{N} \in \{0, 1\}^{m \times u}$ with unary variables, update the function $f_{succ}(g) = \mathbf{N}$ and propagate \mathbf{N} further. We also perform the histogram shift after each convolution according to 3.2.4 if h is nonzero. We use the matrix \mathbf{L} created after the maximum of the output gates to express the delay D from (4.16) or (4.17).

We use standard Boyd's RC model for area(4.6) denoted A and power consumption(4.7) denoted P . A_{max} denotes the maximum possible area, similarly P_{max} maximum possible power consumption, and x_i the sizing variable of gate i . The mixed-integer linear program is in the form²

$$\begin{aligned}
& \text{minimize} && D && && (4.20) \\
& \text{subject to} && \mathbf{B}_j \mathbf{1} \leq \mathbf{r}_j + 0.5 \cdot \mathbf{1}, && j = 1, \dots, n, \\
& && \mathbf{B}_k \mathbf{1} \geq \mathbf{r}_k - 0.5 \cdot \mathbf{1}, && k = 1, \dots, n, \\
& && s \leq w, && \forall g \in G : \forall b \in M : \forall (w, y, z, s) \in f_{trip}(g, b), \\
& && s \leq y, && \forall g \in G : \forall b \in M : \forall (w, y, z, s) \in f_{trip}(g, b), \\
& && s \leq z, && \forall g \in G : \forall b \in M : \forall (w, y, z, s) \in f_{trip}(g, b), \\
& && s \geq w + y + z - 2, && \forall g \in G : \forall b \in M : \forall (w, y, z, s) \in f_{trip}(g, b), \\
& && \mathbf{N}_{b, n-1} \geq \mathbf{N}_{b, 2}, && \forall g \in G : \forall b \in M : \mathbf{N} = f_{succ}(g), \\
& && \mathbf{N} \mathbf{1} \geq \mathbf{s} \cdot \frac{1}{d} - 0.5, && \forall g \in G : \mathbf{N} = f_{succ}(g), \mathbf{s} = f_{sum}(g), \\
& && A \leq A_{max}, \\
& && P \leq P_{max}, \\
& && x_i \geq 1, && i = 1, \dots, n.
\end{aligned}$$

The gate sizing mixed-integer program adds to the plain SSTA 3.3.6 new continuous variables. This complicates the convergence and could not finish with the c17 circuit. The largest circuit it could optimize was with 3 gates: 2 input gates and 1 output gate.

²Full mixed-integer program optimizing a toy circuit formulated in MOSEK API is at  or can be found in the repository <https://github.com/bosakad/GP-Optimization>.

4.2.5 Final geometric program

We denote the number of gates n , set of gates $G = \{1, \dots, n\}$, number of bins m , correctly set interval $I = [h, k]$ partitioned into m equal subintervals each corresponding to 1 bin, variable vector of the gate i by \mathbf{b}_i , set $B = \{\mathbf{b}_1, \dots, \mathbf{b}_n\}$, set of vectors $R = \{\mathbf{r}_1, \dots, \mathbf{r}_n\}$ of the corresponding right sides as mentioned in (4.15), at the beginning we initiate two empty sets of vectors N_{succ} and N_{pred} of the successor and predecessor vectors of variables. We denote the set of operations by $O = \{0, 1\}$, 0 represents the convolution and 1 represents the maximum. We denote the function $f_{pred} : G \times O \rightarrow N_{pred}$ that maps the predecessor posynomials and $f_{succ} : G \times O \rightarrow N_{succ}$ that maps the new created monomials.

After the convolution of the gate i , we save the result vector of posynomials $\mathbf{x}_i \in \mathbb{R}_{++}^{m \times 1}$ in N_{pred} , we also create a new vector of one-variable monomials $\mathbf{n}_i \in \mathbb{R}_{++}^{m \times 1}$ as in (3.21) and store it in N_{succ} . We update the functions $f_{pred}(i, 0) = \mathbf{x}_i$ and $f_{succ}(i, 0) = \mathbf{n}_i$. Similarly, for the maximum, we update $f_{pred}(i, 1) = \mathbf{x}_i$ and $f_{succ}(i, 1) = \mathbf{n}_i$. We also perform the histogram shift after each convolution according to 3.2.4 if h is nonzero. We then use this vector \mathbf{n}_i as a new vector representing the gate histogram and propagate it further in the SSTA. We use the vector \mathbf{n} created after the maximum of the output gates to express the delay D from (4.19).

We use the standard Boyd's *RC* model for area (4.6) denoted A and power consumption (4.7) denoted P . A_{max} denotes again the maximum possible area, similarly P_{max} maximum possible power consumption, and x_i the sizing variable of gate i . The final program is in the form³

$$\begin{aligned}
 & \text{minimize} && D && && (4.21) \\
 & \text{subject to} && \mathbf{b}_j \geq \mathbf{r}_j, && j = 1, \dots, n, \\
 & && f_{pred}(g, o) \leq f_{succ}(g, o), && \forall g \in G : \forall o \in O, \\
 & && A \leq A_{max}, \\
 & && P \leq P_{max}, \\
 & && x_i \geq 1, && i = 1, \dots, n.
 \end{aligned}$$


Note that D is a sum of monomials, thus a posynomial; GP-compatibility of constraints (3.21), (4.15) has already been discussed. Therefore, this is a generalized geometric program.

Let us show the GP model on ISCAS-85 c17 circuit. Consider the same input values as for the deterministic model (Table 4.1). We used both the Gaussian model and Lognormal model for the optimization. Using 35 bins, the minimum has been reached after 34 iterations and 1.18 seconds for the Gaussian model (similarly for the Lognormal model), the sizing parameters can be seen in Table 4.3. Note the difference of the results for different models. However, the significance of each

Sizing parameters	Gate 1	Gate 2	Gate 3	Gate 4	Gate 5	Gate 6
Gaussian	2.89	5.30	6.39	6.04	4.14	10.23
LogNormal	2.69	6.10	5.99	6.40	4.18	9.64

Table 4.3: Optimal sizing parameters of the ISCAS-85 c17 circuit using the statistical GP model and two regression models

gate remains almost the same for both of them.

³Full geometric program optimizing the c17 (4.1) formulated in CVXPY is at  or can be found in the repository <https://github.com/bosakad/GP-Optimization>.

It is clear, in the case of the final program (4.21), that we just add a few new constraints compared to the plain relaxed SSTA (3.21). As expected, the scaling will be again quadratic with the number of bins and linear with the number of gates. The scalability of (4.21) compared to the relaxed SSTA problem (3.21) can be seen in Figure 4.5.

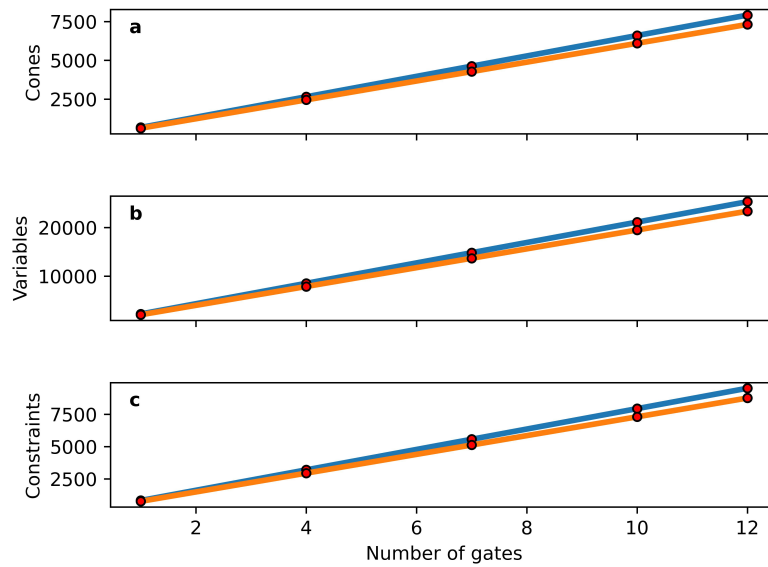


Figure 4.5: Scalability of the final geometric program (4.21) (blue line) compared to the plain SSTA using the GP model with relaxation constraints (3.21) (orange line) tested on the "ladder" of maximums and convolutions. The number of bins is set to 20 for both algorithms. The subplots show: **a**, the increase in the number of cones; **b**, the increase in the number of variables; **c**, the increase in the number of constraints.

In Figure 4.6, we can see the optimal delay distribution compared to the delay computed using Monte Carlo. Inputs for Monte Carlo were generated according to the optimal distributions precalculated by (4.21).

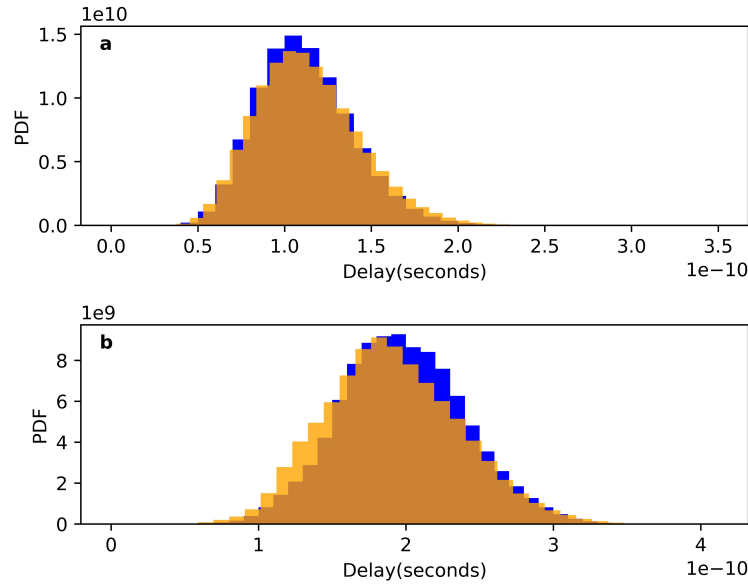


Figure 4.6: Comparison of the optimal delay distribution computed by (4.21) (blue distribution) with the delay distribution computed by Monte Carlo (orange distribution). The number of bins was set to 35. The subplots show: **a**, comparison of the optimal delay distributions computed using the Gaussian model; **b**, comparison of the optimal delay distributions computed using the Lognormal model.

4.3 Discussion and Conclusions

In this Chapter, we presented two statistical approaches to the gate sizing problem, namely:

- (i) The first approach, the so-called mixed-integer model, exhibits global convergence in theory and relatively good results for the SSTA computation in practice. Test circuits have been optimized using this formulation with correct results. It seems, however, that introducing constraints with continuous sizing variables adds too much complexity and is not usable for practical examples.
- (ii) The second approach uses geometric programming to compute the SSTA and further minimize the conditional value-at-risk (CVaR) of the circuit delay. This approach shows very promising results. The model scales better than the mixed-integer formulations. Additionally, the further constraints do not complicate the problem substantially, compared to the mixed-integer model.

One can see from the comparison of the statistical result (Table 4.3) with the deterministic one (Table 4.2) that the difference is significant. Our results are expected to be more accurate.

Chapter 5

Conclusions and Critical Overview of the Thesis

In this work, the statistical gate sizing problem for the Very Large Scale Integration (VLSI) circuits has been studied. Since the calculation of a delay is a crucial step for the sizing problem, the problem of Statistical Static Timing Analysis (SSTA) has also been addressed. For the latter, two approaches were used, mixed-integer and geometric programming, which resulted in two different formulations of the gate sizing problem.

Chapter 3 studies the problem of the maximum delay calculation in a digital circuit taking into account the statistical nature of delays, the so-called SSTA problem. Using a histogram representation of delays' probability density functions, we proposed two formulations of the SSTA algorithm as an optimization problem. The first formulation uses mixed-integer programming and the second one uses geometric programming. The so-called GP model scales significantly better after the relaxations (3.21) than the relaxed mixed-integer model (3.17) —(3.18), and gives more precise results because there is no loss of information caused by the unary encoding used in mixed-integer formulation.

The main achievement of Chapter 3 is the SSTA formulated as the Geometric Program. For the relaxed problem, we have demonstrated linear scaling with the number of gates and the quadratic scaling with the number of bins. The SSTA has been successfully computed using 30 bins for a circuit with 400 gates in 440 seconds on the RCI cluster. It should be noted that correlations between the delays were not taken into account.

Chapter 4 continues its predecessor and makes use of the two proposed formulations of the SSTA for the gate sizing problem. Thus, we propose two formulations of the statistical sizing problem via: (i) mixed-integer programming, and (ii) geometric programming. The general statement of the gate sizing problem is motivated by Boyd *et al.* [8] but extends that for the case of delays given by distributions (hence the name *statistical*).

The mixed-integer model works in theory; however, it cannot scale and is therefore not suitable for real-world circuits. The second of the proposed approaches, the GP model (4.21), gives promising results, and we believe that this might serve as a new direction for the community on how to solve the gate sizing problem in a statistical domain.

One should note that the histogram approximation, which is used in this work, has clear disadvantages: (i) as we increase the number of gates, we have to increase the size of the interval, and with that the number of bins; (ii) we also have to know the interval before the computation (trial and error methods worked well to prove the concept, however, this is a clear limitation that needs to be addressed). On the other hand, such an approach allowed us to (i) perform the robust optimization of delays' distributions (unlike other statistical approaches, where only the statistical moments are taken into account), and (ii) perform computations in polynomial-time using GP. Last but not



Bibliography

- [1] Aseem Agarwal, Kaviraj Chopra, and David Blaauw. Statistical timing based optimization using gate sizing. In *Design, Automation and Test in Europe*, pages 400–405. IEEE, 2005.
- [2] Akshay Agrawal, Steven Diamond, and Stephen Boyd. Disciplined geometric programming. *Optimization Letters*, 13(5):961–976, 2019.
- [3] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.
- [4] MOSEK ApS. MOSEK optimizer API for Python. Version 9.3.20. <https://docs.mosek.com/latest/pythonapi/index.html>. Accessed: 2022-05-02.
- [5] MRCM Berkelaar and JAG Jess. Transistor sizing in mos digital circuits with linear programming. In *Proc. of the European Design Automation Conference, (Mierlo, The Netherlands)*, pages 217–221, 1990.
- [6] David Blaauw, Kaviraj Chopra, Ashish Srivastava, and Lou Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE transactions on computer-aided design of integrated circuits and systems*, 27(4):589–607, 2008.
- [7] S Boyd, SJ Kim, and L Vandenberghe. Hassibi., a.(2007). a tutorial on geometric programming. *Optimization and Engineering*, 8(1):67–127.
- [8] Stephen P Boyd, Seung-Jean Kim, Dinesh D Patil, and Mark A Horowitz. Digital circuit optimization via geometric programming. *Operations research*, 53(6):899–932, 2005.
- [9] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark designs and a special translator in fortran. In *Proc. ISCAS*, pages 695–698, 1985.
- [10] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay functions. In *Proc. DAC*, pages 71–76, Jun 2005.
- [11] Hongliang Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *Proc. ICCAD*, pages 621–625, Nov 2003.
- [12] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis under spatial correlations. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 24(9), 2005.

- [28] Srinath R Naidu. A convex programming solution for gate-sizing with pipelining constraints. *Optimization and Engineering*, pages 1–36, 2021.
- [29] S. Sapatnekar. *Timing*. Springer-Verlag, 2004.
- [30] Sachin S Sapatnekar, Vasant B Rao, Pravin M Vaidya, and Sung-Mo Kang. An exact solution to the transistor sizing problem for cmos circuits using convex optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(11):1621–1634, 1993.
- [31] C. Visweswariah. Death, taxes and failing chips. In *Proc. DAC*, pages 343–347. IEEE, June 2003.

Appendix A

Source Code

- Project.zip
- Github: <https://github.com/bosakad/GP-Optimization>

```
[ 4096] GP-Optimization
|-- [  991] README.md
|-- [  0] __init__.py
|-- [ 4096] docs
|-- [  634] Makefile
|-- [   xxx] ...
|-- [  284] requirements.txt
|-- [ 4096] src
`-- [ 4096] gateSizing
    |-- [ 4096] Inputs.outputs
    |-- [  360] c17.v
    |-- [ 4096] data
    |-- [ 771119] Inverter_chain_global_var.csv
    |-- [ 3114947] MonteCarlo_10inv_in2VDD_mismatch.csv
    |-- [  236] ex1.bench
    |-- [ 4832] generatedDistros.npz
    |-- [ 1382] model.npz
    |-- [  502] model_MIXED_INT.npz
    |-- [ 1382] model_Normal.npz
    |-- [  516] test.v
    |-- [  744] testParsing.txt
    |-- [ 6444] verbose.stdout
    |-- [ 3809] Parser_Matrix.py
    |-- [ 32513] Plotter.py
    |-- [ 14182] SSTA.py
    |-- [  0] __init__.py
    |-- [ 40239] cvxpyVariable.py
    |-- [ 5003] deterministicDelay.py
    |-- [ 10429] distrFitter.py
    |-- [ 4096] examples_monteCarlo
    |-- [ 4344] infinite_ladder_montecarlo.py
    |-- [ 6369] montecarlo.py
    |-- [ 6731] histogramGenerator.py
    |-- [ 72799] mosekVariable.py
    |-- [  655] node.py
    |-- [ 18029] optimizeGates.py
    |-- [ 6334] optimizeGatesSimple_Boyd.py
    |-- [ 9305] optimizeGatesVectorized_Boyd.py
    |-- [ 1995] plot_cadence_mc.py
    |-- [ 25472] randomVariableHist_Numpy.py
    |-- [ 30981] test_Operations_Cvxpy.py
    |-- [ 21934] test_RV_Numpy.py
    |-- [ 7969] test_SSTA_Numpy.py
    |-- [ 16748] test_SSTA_cvxpy.py
    |-- [ 14681] test_algorithms.py
    |-- [ 61154] test_infiniteLadder.py
    |-- [ 33908] test_mosekOperations.py
    |-- [ 152985] unaryTesting.ipynb
    |-- [ 11195] unaryTesting.py

22 directories, 145 files
```