Timing-Driven Routing in VLSI Physical Design under Uncertainty

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE FACULTY OF ENGINEERING

by

Radhamanjari Samanta

Supercomputer Education and Research Centre and
Centre for Nano Science and Engineering
Indian Institute of Science
BANGALORE – 560 012

NOVEMBER 2013
TO

*My Parents*
Acknowledgements

First and foremost I would like to express my deep and sincere gratitude to my advisor Prof. Soumyendu Raha for his constant encouragement, suggestions and support throughout my research work. He helped me understand the rigorous mathematical and algorithm development part of VLSI physical design. Without his assistance, I would have never enjoyed this blend of mathematics and engineering simultaneously. His friendly and approachable nature helped me a lot in understanding the problems and implementing. He always extended his support, be it technical or a personal problem. I am also very thankful to my co-advisor Prof. Santanu Mahapatra for his valuable and instant help, whenever asked. I am deeply indebted to Prof. Adil I. Erzin of Sobolev Institute of Mathematics for his guidance and help during my PhD. Though he was not my official guide, I always felt like being his own student. He would always respond to my queries at the earliest. I feel very fortunate to visit Sobolev Institute of Mathematics for a short research visit during 2009 and for the meaningful interaction I had there with Prof. Erzin and Dr. Zalyubovskiy. I also would like to thank Prof. Virendra Singh of IIT Mumbai (then prof at SERC, IISc) for several fruitful discussions and interactions at various point of time.

I would like to thank all the staff of SERC and CeNSE for their help. I would like to thank the CeNSE chairperson Prof. Rudra Pratap for providing me funds to present at a conference.

I must acknowledge all my labmates Saswati Di, Nigam, Samar, Raghu, Vishal and Milind for helping me during my research work and for their pleasant company. Discussions with Nigam and Samar often helped me understand the mathematical part of
my work. Saswati Di was a wonderful senior and friend. Spending time with her always
made me feel better.

Last but not the least, it is my family without whom I could not have completed my
thesis. My late grandfather was the constant source of inspiration to me. He embedded
the idea of “Plain Living and High Thinking” in my heart. My parents always encouraged
me to aspire for a research career and I don’t know how to put into words my eternal
thanks to them. And my warmest gratitude goes to my brother Ramanuj, who has been
my friend, philosopher and guide.
Abstract

The multi-net Global Routing Problem (GRP) in VLSI physical design is a problem of routing a set of nets subject to limited resources and delay constraints. Various state-of-the-art routers are available but their main focus is to optimize the wire length and minimize the overflow. However optimizing wire length do not necessarily meet timing constraints at the sink nodes. Also, in modern nano-meter scale VLSI process the consideration of process variations is a necessity for ensuring reasonable yield at the fab. In this work, we try to find a fundamental strategy to address the timing-driven Steiner tree construction (i.e., the routing) problem subject to congestion constraints and process variation.

For congestion mitigation, a gradient based concurrent approach (over all nets) of Erzin et. al., rather than the traditional (sequential) rip-and-reroute is adopted in order to propagate the timing/delay-driven property of the Steiner tree candidates. The existing sequential rip-up and reroute methods meet the overflow constraint locally but cannot propagate the timing constraint which is non-local in nature. We build on this approach to accommodate the variation-aware statistical delay/timing requirements.

To further reduce the congestion, the cost function of the tree generation method is updated by adding history based congestion penalty to the base cost (delay). Iterative use of the timing-driven Steiner tree construction method and history based tree construction procedure generate a diverse pool of candidate Steiner trees for each net. The gradient algorithm picks one tree for each net from the pool of trees such that congestion is efficiently controlled.
As the technology scales down, process variation makes process dependent parameters like resistance, capacitance etc non-deterministic. As a result, Statistical Static Timing Analysis or SSTA has replaced the traditional static timing in nano-meter scale VLSI processes. However, this poses a challenge regarding the max/min-plus algebra of Dijkstra like approximation algorithm that builds the Steiner trees. A new approach based on distance between distributions for finding maximum/minimum at the nodes is presented in this thesis. Under this metric, the approximation algorithm for variation aware timing driven congestion constrained routing is shown to be provably tight and one order of magnitude faster than existing approaches (which are not tight) such as the MVERT.

The results (mean value) of our variation aware router are quite close to the mean of the several thousand Monte Carlo simulations of the deterministic router, i.e the results converge in mean. Therefore, instead of running so many deterministic Monte Carlo simulations, we can generate an average design with a probability distribution reasonably close to that of the actual behaviour of the design by running the proposed statistical router only once and at a small fraction of the computational effort involved in physical design in the nano regime VLSI.

The above approximation algorithm is extended to local routing, especially non-Manhattan lambda routing which is increasingly being allowed by the recent VLSI technology nodes. Here also, we can meet delay driven constraints better and keep related wire lengths reasonable.
Keywords

Global Routing; Steiner Tree; Elmore Delay; Gaussian Distribution; Probability Density Function; Process Variation; Statistical Static Timing Analysis; Gradient Method.
Publications based on this Thesis


Contents

Acknowledgements ii
Abstract iv
Keywords vi
Publications based on this Thesis vii

1 Introduction 1
  1.1 Organization ............................................................. 8

2 Timing-driven Routing 11
  2.1 Introduction .............................................................. 11
  2.2 Global Routing Model .................................................. 13
  2.3 Problem Formulation ................................................... 14
  2.4 Algorithm MAD .......................................................... 15
     2.4.1 Illustration of the steps of MAD .................................. 16
  2.5 Congestion Aware Tree Selection ..................................... 18
  2.6 History Based MAD ..................................................... 21
  2.7 Diversity of the Generated Steiner Trees ............................ 21
  2.8 Experimental Results .................................................. 22
  2.9 Conclusion ............................................................... 27
5.6.1 Experimental Results .............................................. 55
5.6.2 Conclusion ...................................................... 56

6 Conclusion .................................................................. 57
  6.1 Summary ............................................................. 57
  6.2 Future Work ........................................................ 58

Appendix: A .................................................................. 59

Appendix: B .................................................................. 63
  6.2.1 Grid graphs with $n \geq 2$ terminals ......................... 69

Bibliography ............................................................... 75
## List of Figures

1.1 Steps of Routing: Global Routing and Detailed Routing [77] 1  
1.2 VLSI Circuit: A Hypergraph [45] 2  
1.3 Spanning tree(a), Steiner tree(b) and Rectilinear Steiner tree(c) 2  
1.4 Minimum wirelength does not necessarily mean minimum delay and vice-versa 4  

2.1 Global Graph Construction 13  
2.2 Vias connecting the layers 13  
2.3 Candidate Steiner points 17  
2.4 Intermediate tree 17  
2.5 Final Steiner tree generated by MAD 18  
2.6 Histogram of delay difference 24  
2.7 Delay of FLUTE(red) and our router(green) generated trees on ibm01 benchmark 24  

3.1 Min distribution of two given normal distributions 29  
3.2 Approximating min distribution to normal distribution when the given distributions almost completely overlap 30  
3.3 Approximating min distribution to normal distribution when the variances of the given distributions differ by large value 31  
3.4 3σ pruning 32  
3.5 Hellinger distance between min and given distributions 34
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Max Overflow with and without Gradient</td>
<td>42</td>
</tr>
<tr>
<td>4.2</td>
<td>Total Overflow with and without Gradient</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Generating Steiner points on different $\lambda$-Geometry</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Reduction of hex Steiner points for #40(1) at max_dist = 0.05</td>
<td>51</td>
</tr>
<tr>
<td>5.3</td>
<td>Steiner pts for hex &amp; rec trees of #70(1) at different adjacency dist</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Wirelength and Run time for #70(1) at different adjacency dist</td>
<td>52</td>
</tr>
<tr>
<td>5.5</td>
<td>Rec tree generated on OR Library instance 10(1) by MAD</td>
<td>54</td>
</tr>
<tr>
<td>5.6</td>
<td>Rec tree generated on OR Library instance 10(2) by MAD</td>
<td>54</td>
</tr>
<tr>
<td>6.1</td>
<td>Lemma 1</td>
<td>64</td>
</tr>
<tr>
<td>6.2</td>
<td>Lemma 2</td>
<td>66</td>
</tr>
<tr>
<td>6.3</td>
<td>Lemma 3</td>
<td>68</td>
</tr>
<tr>
<td>6.4</td>
<td>Claim</td>
<td>70</td>
</tr>
</tbody>
</table>
List of Tables

2.1 IBM/ISPD 98 Benchmark Suite ............................................. 23
2.2 Results of MAD with and without Gradient on Examples Derived from
IBM/ISPD98 benchmarks .................................................... 25
2.3 % of FLUTE generated trees selected by Gradient .................... 26
3.1 Comparison between Visweswariah et al [83], our approach and Monte
Carlo simulations on ISCAS’85 benchmarks .............................. 36
4.1 Comparison of Wirelength and Delay between Deterministic (Monte Carlo)
and Statistical(Variation Aware) MAD on Examples Derived from ISPD98
Benchmarks ..................................................................... 45
5.1 Comparison between properties of different $\lambda$-Geometry Routing ... 48
5.2 Comparison of Wirelength and Delay for different $\lambda$-Geometry Routing
on OR-Library Instances ...................................................... 53
5.3 Comparison between Deterministic (Monte Carlo) and Statistical(Variation
Aware) MAD for different $\lambda$-Geometry on OR-Library Benchmarks ... 56
List of Abbreviations

AT     Arrival Time
CDF    Cumulative Distribution Function
CMP    Chemical Mechanical Polishing
FLUTE  Fast Look Up Table based wirelength Estimation technique
GradAl Gradient Algorithm
GRP    Global Routing Problem
IC     Integrated Circuit
IMAD   Iterative MAD
ISCAS  International Symposium on Circuits and Systems
ISPD   International Symposium on Physical Design
MAD    Modified Algorithm Dijkstra
MCF    Multi Commodity Flow
MVERT  Minimum Violation Elmore Routing Tree
NCR    Negotiated Congestion Routing
NP     Non deterministic Polynomial time
OR-Library  Operations Research Library

PDF  Probability Density Function

R & R  Rip-up and Reroute

RSMT  Rectilinear Steiner Minimal Tree

RT  Required Time

RV  Random Variable

SSTA  Statistical Static Timing Analysis

STA  Static Timing Analysis

VLSI  Very Large Scale Integration
Chapter 1

Introduction

As the fabrication technology scales down in the nano regime, the complexity of the integrated circuits increase exponentially. Therefore the efficiency of Computer-Aided-Design (CAD) tools are becoming more and more important for VLSI physical design. VLSI physical design is broadly divided in the following phases: partitioning, floorplanning, placement, routing, compaction and verification. In the layout design process, partitioning is the first step. Partitioning is done to divide the chip in smaller subsystems which are easily manageable. Floorplanning phase aims to find the relative position of hard (known geometric features) and soft (unknown geometric features) blocks on the chip so that total area is minimized. In the placement phase, exact location of all the circuit components are assigned based on a number of objectives so that the performance of the IC is maximized. Routing is typically divided in two phases, global and detailed.
routing. In the global routing step, an approximate connection of nets are done subject to resource and delay constraints. In the detailed routing step actual geometric layout of the nets are found out subject to various geometric constraints. The Fig. 1.1 [77] shows the two steps of routing. The overall area of the chip is reduced in the compaction step, without making design rule violations. Finally in the verification step, all the design rules and functionalities are checked and the layout is verified by circuit extraction. When the design verification step meets all the functionalities, fabrication can be started.

With the scaling of VLSI technology, higher clock frequency and better operating speed is becoming essential. Interconnect delay is thus becoming a major contributing factor in circuit performance. In this work, we aimed at studying algorithms for global routing making them aware of the process variation in the nanometer scale of VLSI.
design in terms of interconnect delay variability and in terms of resource availability on the chip.

A VLSI circuit is a collection of modules and a set of nets, which can be represented as a hypergraph $H(V, E)$ as shown in Fig. 1.2. Vertex set $V = \{v_i|i = 1, 2, .., n\}$ denotes modules. Hyperedge set $E = \{e_j|j = 1, 2, .., m\}$ denotes nets. Each net $e_j$ is a subset of $V$ with cardinality $|e_j| \geq 2$. The modules in $e_j$ are called the pins of $e_j$ and they have same electric potential. As spanning trees(Fig. 1.3.a) have larger wirelengths, Steiner trees(Fig. 1.3.b) are generally used for connecting the terminals of a multi-terminal net. Rectilinear Steiner tree(Fig. 1.3.c) allows edges in horizontal and vertical directions only. Finding Minimum(delay/wirelength) Steiner Tree is a NP Complete problem.

Most of the recent global routing techniques focus on minimizing wirelength and congestion but do not take into account the timing constraint [13,14,18,22,23,35,38,50,51, 57,61–63,67]. Mere minimization of wirelength and overlap without considering timing and process variation will not address the real-time feasibility of electrical connections. Optimization of wirelength and then minimizing congestion is a completely different objective from optimizing delay and congestion. It is not sufficient to minimize the wirelength only as the routing might not meet timing at the sinks, which is absolutely essential for the large scale VLSI designs. As stated by Moffitt et. al. [58] in ISPD 2008, timing-driven routing is becoming more and more important.

With the scaling down of technology, interconnect delay is playing significant role in circuit performance. There are several methods for computing wire delays. The lumped C method ignores the interconnect delay and the total wire capacitance is applied at the gate output. Elmore delay metric is used in this work. Although this metric sometimes give inaccurate results under certain cases, it is a widely accepted method for computing delay as it has very high fidelity [9] and can be calculated very efficiently. The delay of any segment of interconnect is found out by multiplying the resistance of the segment with the downstream capacitance. For a trees structured network, sum of all the delays from the root are done to get the final delay value. Explicit moment matching techniques are usually more accurate as Elmore delay matches the first moment but these metrics are
based on matching multiple moments. AWE [65], PRIMA [59] are methods with explicit moment matching. But the computation time of these methods are higher compared to Elmore delay. SPICE or other circuit simulators are the most accurate method for computing interconnect delay but they are the slowest method.

A simple example is shown in Fig. 1.4 to demonstrate that optimum wirelength does not necessarily mean optimum delay and vice-versa.

Figure 1.4: Minimum wirelength does not necessarily mean minimum delay and vice-versa

The computation of delay is heavily dependent on pendant subtrees. Therefore optimizing delay and congestion is a multi-objective constraint, which is the focus of this work. In the example, Fig. 1.4(a) shows that, given 3 pins, 1(source), 2(sink), and 3(sink), we first draw the Hanan grid by drawing horizontal and vertical lines through them. The intersecting points are the generated Steiner points. Fig. 1.4(b) shows the minimum wire-length(WL = 3 units) tree configuration and Fig. 1.4(c) shows another tree configuration for the net, whose wirelength, WL = 4 units. Now, we compute the
CHAPTER 1. INTRODUCTION

delay of the sink nodes for both the tree. In the figure, all \( R_i = R \) and all \( C_i = C \) 

Though the tree in Fig. 1.4(b) has minimum WL(3 units), it has \( 3RC \) delay whereas 

tree in Figure 1.4(c) has 4 units of WL but \( 2RC \) delays at the sinks as can be seen from 

the following equations. For, Fig. 1.4(b),

\[
\text{Delay at node2, } d_{12} = R_1 \cdot C_3 + (R_1 + R_2) \cdot C_2 \simeq 3RC \\
\text{Delay at node3, } d_{13} = R_1 \cdot C_2 + (R_1 + R_3) \cdot C_3 \simeq 3RC
\]

and for, Fig. 1.4(c),

\[
\text{Delay at node2, } d_{12} = (R_4 + R_2) \cdot C_2 \simeq 2RC \\
\text{Delay at node3, } d_{13} = (R_4 + R_3) \cdot C_3 \simeq 2RC
\]

With the shrinking down of process technology, circuit performance has become highly sensitive to the variability in digital ICs. The fluctuations during the manufacturing process like chemical mechanical polishing(CMP) or optical proximity effects etc. can influence the variability. As explained in [7], the main factors causing physical parameter variation are critical dimension, oxide thickness, channel doping, wire width and wire thickness. The variability in physical parameter variation in turn cause electrical parameter variations like fluctuations in saturation current, gate capacitance, threshold voltage, wire resistance and wire capacitance and these in turn cause variability in gate delay, slew rate and wire delay. The traditional deterministic Static Timing Analysis(STA) computes the expected time of a digital circuit for a particular condition where all the process parameters like oxide thickness, gate length, temperature, voltage etc. are considered to be uniform and equal across the chip. STA is done multiple times for various corners by varying these parameter sets within a range of possible values. The delay of the circuit with process variation can be predicted when a significant number of STA is run for various process conditions. Program Evaluation and Review Technique(PERT) based STA [47] and other methods for STA appears in [39, 54]. But the limitations of STA is that it cannot predict intra die variations which ultimately results
in a highly optimistic or pessimistic analysis. Also with the drastic increase in number of corners files [55] and exponential rise in number of runs [82] make the deterministic STA ineffective and highly time consuming.

SSTA replaced the deterministic STA to effectively model the process and environmental variation. Several approaches for SSTA exist in the literatures. The simplest and most general method which can address process variation irrespective of the nature of the distribution and correlation is numerical integration method [44]. But it is a computationally extensive procedure. Another general approach which basically runs a sample of deterministic STAs is Monte Carlo simulation but this also requires significant amount of time for predicting variation accurately. The other statistical approaches consider the timing quantities as Gaussian random variables instead of deterministic values. These methods are based on computation of statistical max/min and addition of delay probability density functions(PDF)s. Based on how the max function is used, this idea leads to two different approaches. One is path-based [2,60] and another is block-based approach [1,6,16,30,52,83]. The first approach is based on identifying candidate paths which are likely to be critical and then computing statistical max operation on these paths. This approach is simple and can handle correlation efficiently but the performance of the analysis is highly dependent on identifying the path which is to be done rigorously beforehand. The block-based approach does not require identifying the paths as a topological traversal of the circuit is followed here. Two basic operations statistical plus and max are done on the fan-in nodes although the statistical max/min operation considering correlation is a non-trivial problem. The runtime complexity of the block-based approach is not very high and therefore most of the recent SSTA methods are developed using this approach.

In the block-based approach, usually all the timing quantities are expressed in the standard first-order canonical form, which is a linear function of independent normal RVs. The canonical form is expressed as

\[ d = a_0 + \sum_{i=1}^{n} a_i \Delta X_i + a_{n+1} \Delta R_a, \]

where \( a_0 \) is the mean delay, \( \Delta X_i \) represents the variation of \( n \) independent RVs from
the mean value, $i = 1, 2, ..., n$, and $\Delta R_a$ is the residual independent random variation. All $a_i$s, $i = 1, 2, ..., n$, represent the sensitivity coefficient associated with each RV. The most important operations involved in block-based analysis are statistical sum and max. The sum of two Gaussian delay distributions is easily expressed in the canonical form, where the resultant mean or nominal value will be the sum of the mean values of the given distributions and similarly the sensitivity coefficients will also be added. Except the coefficient for independent variation is the root sum square of the individual independent contributions. But to re express the max/min of two Gaussian distributions in canonical form is a complex problem as the exact max/min distribution of two given distributions may not remain Gaussian particularly if the individual distributions are significantly overlapping. For multiple Gaussian input distributions, iterative pair-wise max/min is done to get the final max/min. Then the actual max/min distribution is needed to be approximated to Gaussian distribution. The order in which the pair-wise max/min is found, can vary the error due to approximation. Also the approximation error depends on the methods used to approximate the actual distribution to normal distribution. The most adopted method of SSTA is developed by Visweswariah et. al. [83]. The approach of [83] and [16] is summarised in [8] as follows. First the tightness probability (i.e. the probability of the arrival time of one distribution is higher than the other distribution) is computed.

If $A(\mu_a, \sigma^2_a)$, $B(\mu_b, \sigma^2_b)$ are two given distributions, which can be expressed in the canonical form as $A = a_0 + \sum_{i=1}^{n} a_i \Delta X_i + a_{n+1} \Delta R_a$ and $B = b_0 + \sum_{i=1}^{n} b_i \Delta X_i + b_{n+1} \Delta R_b$, where $A$ and $B$ have means and variances $\mu_a(a_0), \sigma^2_a$ and $\mu_b(b_0), \sigma^2_b$ respectively and $r$ represents the covariance between $A$ and $B$.

Then, $\sigma^2_a = \sum_{i=1}^{n+1} a_i^2$ and $\sigma^2_b = \sum_{i=1}^{n+1} b_i^2$ and $r = \sum_{i=1}^{n} a_i b_i$

Tightness probability is defined as $T_A = P(A > B) = \Phi \left( \frac{\mu_a - \mu_b}{\theta} \right)$, where $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$ and $\Phi(x) = \frac{1}{2} \left[ 1 + erf \left( \frac{x}{\sqrt{2}} \right) \right]$ and $\theta = \sqrt{\sigma^2_a + \sigma^2_b - 2r}$.

Following Clark’s formula [27] for computing max of two Gaussian RVs, mean and
CHAPTER 1. INTRODUCTION

variance of $C = \max(A, B)$ is derived as follows.

$$
\mu_c = \mu_a T_A + \mu_b (1 - T_A) + \theta \phi \left( \frac{\mu_a - \mu_b}{\theta} \right)
$$

$$
\sigma^2_c = \left( \sigma^2_a + \mu^2_a \right) T_A + \left( \sigma^2_b + \mu^2_b \right) (1 - T_A) + (\mu_a + \mu_b) \theta \phi \left( \frac{\mu_a - \mu_b}{\theta} \right) - \mu^2_c
$$

But the distribution of $C = \max(A, B)$ may not be Gaussian, therefore the approximated Gaussian distribution $C'$ is determined. The sensitivity coefficient of $C'$ is given as,

$$
c_i = a_i T_A + b_i (1 - T_A), \text{ where } i = 1, 2, ..., n.
$$

The sensitivity coefficient of the random component of $C'$ is calculated as $(\sigma^2_c - \sum_{i=1}^{n} c_i^2)$

As stated in [83], since the max is approximated in the canonical Gaussian distribution, accuracy is compromised and when the individual input signals have considerable overlapping, accurate approximation to the canonical Gaussian form will be very hard. Also the delay distributions are assumed to be Gaussian, but in reality they might be non-normal distributions. There exists some approaches for nonnormal STA which will predict the variation quite accurately but they are tremendously computationally expensive. Some of the works [44, 90] model device parameters as quadratically dependent. Also nonlinear dependence on device parameter variation is explored in [17, 46].

1.1 Organization

In this thesis, we will illustrate an $O(n^2 \log n)$ timing-driven Steiner tree generation method MAD(Modified Algorithm Dijkstra) developed by Erzin et al [31, 33] and extend this method to address process variation and congestion. The existing timing-driven routing methods [42, 86, 87] are based on Minimum Violation Elmore Routing Tree(MVERT) [41] and MVERT has a complexity of $O(n^4)$ ($n$ is the number of sinks). Some algorithms [10–12] are there based on Elmore delay model, but they are of complexity at least $O(n^3 \log n)$. A set of minimum delay candidate Steiner trees for each net is generated by iterative MAD or IMAD by slightly varying the cost function calculating delays at the nodes. To make the delay-driven trees congestion-aware, a penalty based on overflow occurred in the previous iterations (history-based congestion cost) is added.
to the exiting base cost (delay). Thus we have a diverse and dissimilar pool of candidate set of trees. Now to efficiently address congestion, a gradient based approach of Erzin et al [31,33] is applied concurrently on all the nets. It will choose a tree for each net so that the congestion is minimized and the delay-driven property of the Steiner trees are preserved. Chapter 2 describes the timing-driven algorithm, history-based congestion cost function and how overflow is reduced using the gradient algorithm.

Process variation becomes very dominant in the recent VLSI technology and the variation-dependent parameters like resistance and capacitances become uncertain in nature thereby making the delay of the nodes non-deterministic. To model the process variations efficiently, SSTA is required to be done. To find the maximum/minimum at the nodes, a new and efficient approach is proposed in this thesis for SSTA, based on distance between distributions. In the traditional approach, the approximated normal distribution of the actual max/min is propagated and this incurs significant error when the given distributions have considerable amount of overlapping. The proposed idea does not need to approximate the max/min distribution and thus incurs no error due to approximation. Chapter 3 presents the proposed approach of computing max/min of a set of Gaussian random variables.

Based on the proposed distance metric, we have derived the variation-aware (statistical) version of the timing-driven Steiner tree construction method by taking its variation dependent parameters as Gaussian distributions instead of deterministic quantities. While selecting the minimum delay edges, distance between the candidate edge distributions and the actual min distribution are considered in the variation-aware routing method. It is observed that the results of the variation aware router and several thousand Monte Carlo simulations of the deterministic router converge in mean. Therefore, we can run the statistical router only once instead of running so many deterministic Monte Carlo simulations which will take huge time as the proposed router can predict the variation quite accurately. In chapter 4, the variation-aware router is presented and a comparative study between the deterministic and the statistical router on derived ISPD98 benchmarks are given.
Also it has been observed that non-manhattan routing reduces the wirelength and delay more than the traditional Manhattan routing. Since the recent fabrication technology does not any more restrict the routing in only rectangular direction, the timing-driven variation-aware Steiner tree generation method is also extended on different $\lambda$-geometries, i.e. on Manhattan as well as Non-Manhattan grids. The use of Non-Manhattan geometry helps generating Steiner trees with reduced delay and wire-length. Timing-driven and variation-aware non-Manhattan local routing is illustrated in chapter 5.

Finally, summary of the thesis and several future directions based on the presented work are suggested in chapter 6.
Chapter 2

Timing-driven Routing

2.1 Introduction

In this chapter, a brief description of the timing-driven congestion-aware global routing algorithm developed by Erzin et al [31] is given. Next my contribution has been included to update the cost function of the tree construction method by adding history-based congestion penalty terms to the base delay cost to further reduce the congestion. There have been numerous works in the area of routing in the recent past but none of the state of the art routers consider timing. Almost all the recent routers [22,35,43,57,61,62,64,68,85] are based on constructing an initial routing tree with optimized wirelength using Fast Look Up Table based wirelength Estimation technique (FLUTE) [25, 26]. FLUTE is a look up table based method which produces optimal wirelength trees for nets of size 9 (containing 9 pins) and for larger nets, a divide and conquer approach is used. After constructing the initial tree, rip-up and reroute (R & R) in various ordering of nets are done in the existing literatures. But only optimizing the wirelength will not ensure the electrical connectivity because they will not necessarily meet timing at the sinks. As suggested by Moffitt et al [58], there is increasing demand of timing-driven routing algorithms and there are not many works focused in this area. There are a few global routing algorithms [41,42,86,87] which consider timing and their basic tree construction method is a non-Hanan routing algorithm, MVERT [41] which is of complexity $O(n^4)$ ($n$...
is the number of sinks). In these algorithms, edge flexibility and movable Steiner points are used for congestion mitigation. Flexibility of a Steiner point is presented in [41] to meet the timing-driven target and retain the routability. For delay-driven global routing, Hu et. al. [42] explores the flexibility in the Steiner tree. In [86, 87], Yan et.al exploits the movability of the Steiner points to reduce the congestion locally. But these timing-driven algorithms are implemented on very small number of nets, whereas the algorithm which we have used and extended to consider congestion and variation constraints is of complexity $O(n^2 \log n)$ and is provably tight.

Also GRP was formulated as a multi-commodity flow Problem [40] as well. With each net a certain flow of unit size is associated. Each edge has a flow capacity. With respect to the objective function we may get a min-cost multi-commodity flow problem or concurrent multi-commodity flow problem [3, 15, 36, 76]. Meta-heuristics to solve GRP can be found in Timber-Wolf [29](simulated annealing), [21] (evolution algorithm), [34] (genetic algorithm) and [89] (tabu search).

The timing-driven algorithm [31] Modified Algorithm Dijkstra(MAD) constructs timing-driven Steiner trees for each net. The Gradient based approach which is used for minimizing the overflow considers all the nets concurrently and provides an $(1 + \epsilon)$ - approximation solution. This is a concurrent approach considering all the nets simultaneously contrary to the existing approaches of sequential R & R. The algorithms are implemented on derived ISPD98 benchmarks and the drastic reduction of overflow is observed.

Global Routing model is described in brief in section 2.2. In section 2.3, the problem is formulated and section 2.4 describes the algorithm MAD(Modified Algorithm of Dijkstra) and its iterative modifications(IMAD). IMAD is applied to create minimum critical delay Steiner trees for each net. History based IMAD described in section 2.6 is used to create congestion-aware trees for each net. Also we have used FLUTE [26] to generate another set of Steiner candidate trees for each net. Finally a gradient algorithm is used to pick a tree for each net from its candidate set of trees, such that the congestion/overflow is minimal. The Gradient method is described in section 2.5. We run MAD without Gradient on IBM/ISPD98 benchmarks and this gives us the initial congestion of the
chip. Then we run Gradient on the trees generated by IMAD, history-based MAD and FLUTE. The results show how effectively the gradient algorithm reduces the congestion. The benchmarks are modified by assigning resistance, capacitance values to the wires. We show that 66.4% trees picked by Gradient are generated by MAD and the rest are from FLUTE. The experimental results are shown in section 2.8 and section 2.9 gives the conclusion.

2.2 Global Routing Model

The global graph is constructed by partitioning the layout area into several rectangular tiles called global bins. Each global bin represents a vertex and each edge corresponds to a boundary between two adjacent global bins as shown in Fig. 2.1. A set of Steiner points (constructed by the grid graph) are added to the set of terminal points, i.e the given pin locations for the net and then the Steiner tree for the net is created on that graph. Therefore Steiner tree \( t_i = (v_i, e_i) \), where \( v_i = n_i \cup v_i,\text{Steiner} \) and \( v_i,\text{Steiner} \subseteq V, e_i \subseteq E \). 

\[
G(V,E) \text{ is the global graph and } n_i \text{ represents the terminal points. The horizontal and vertical connections can be on different layers with vias connecting the layers as given in}
\]
Fig. 2.2. Supply \( s(e) \) of an edge is defined as the number of available routing tracks it contains and the demand, \( d(e) \) is the number of wires that utilize an edge \( e \). Overflow of an edge \( e \) is defined as,

\[
\text{Overflow}(e) = \begin{cases} 
    d(e) - s(e), & \text{if } d(e) > s(e) \\
    0, & \text{otherwise}
\end{cases}
\]

\[2.3\] Problem Formulation

Given an undirected graph \( G = (V, E) \), \(|V| = m\). To each edge \((i, j) \in E\), two non-negative parameters \( r_{ij} \) (resistance) and \( c_{ij} \) (capacitance) are assigned. Consider the subset of vertices \( S = \{0, 1, \ldots, n\} \subseteq V \), where vertex 0 is the source of the graph (since the graph is undirected, it has no source and we call vertex 0 the source of signal) and nodes \( S \setminus \{0\} \) are the sinks or terminals. We call nodes from \( V \setminus S \) intermediate nodes.

Each \( i \in S \) has the capacitance \( c_i \) and vertex 0 has also the resistance \( r_0 \).

Let us consider a Steiner tree \( T(V, E) \) spanning \( S \) and rooted in 0. We are using the Elmore delay metric. Let \( k \) be an arbitrary terminal in \( T \) and define Elmore delay as \( t_k(T) \). Let us denote:

- \( P_k(T) \) - path from 0 to \( k \) in \( T \). Also \( P_{uv}(T) \) denotes path from \( u \) to \( v \) in \( T \). Argument \( T \) is omitted where it is obvious from context;

- \( T_j \) \( (j \in V(T)) \) is downstream subtree of \( T \) with root \( j \); \( T_e \) \( (e \in E(T)) \) is downstream subtree of \( T \) rooted in the head node of arc \( e \);

- \( C(H) \) \( (R(H)) \) - total capacitance (resistance) of subgraph \( H \), \( C(H) = \sum_{e \in E(H)} c_e + \sum_{i \in V(H)} c_i \). Also we use notations: \( C_j = C(T_j) \), \( C_e = C(T_e) \) and \( R_{uv} = R(P_{uv}) \).

The Elmore delay \([66,69]\) along the arc \((i, j) \) in \( T \) is defined as follows.

\[
d_{ij} = d_{ij}(T) = r_{ij} \left( \frac{c_{ij}}{2} + C_j \right).
\]

\[2.1\]
is given by
\[ t_k = t_k(T) = r_0C_0 + \sum_{(i,j) \in P_k(T)} d_{ij}. \quad (2.2) \]

The maximum among all the delays to terminals in \( T \) is called the \textit{critical delay} and is denoted by \( t^*(T) \). Terminal \( k \): \( t_k(T) = t^*(T) \) is a \textit{critical terminal} in tree \( T \). Here the problem of finding such a tree spanning \( S \) in \( G \) that provides minimum critical delay is defined:
\[ \max_{i \in S} t_i(T) \rightarrow \min_T. \quad (2.3) \]

This problem is known to be NP-hard [49]. The following section describes the proposed approximate algorithm, Modified Algorithm of Dijkstra (MAD) for solving the problem (2.3).

\section{Algorithm MAD}

\textbf{Steps of the Algorithm:}

\begin{enumerate}
  \item \textbf{Step 0.} Set tree \( T = (0, \emptyset) \) and delay \( t_0 = 0 \);
  \item \textbf{Step 1.} Find \( (i,j) = \arg \min_{(u,v) \in E: \ u \in T, v \notin T} \left\{ t_u(T \cup \{u,v\}) + d_{uv} \right\} \),
\end{enumerate}

where
\[ t_u(T \cup \{u,v\}) = t_u(T) + r_0(c_{uv} + c_v) + \sum_{e \in P_u(T)} r_e(c_{uv} + c_v). \]

Set \( T = T \cup \{i,j\} \) and recalculate the delays \( t_k \) (\( k \in T \)):

- If \( j \) is an intermediate vertex, then set \( t_j = t_i + d_{ij} \), and the delays \( t_k \) in all the other vertices do not change;
- if \( j \) is a terminal, then cut off all the pendent subtrees not containing terminal vertices and recalculate all the delays in \( T \) by formula (2.2).

If not all terminals are included in \( T \) then go to Step 1.

\textbf{Time Complexity of Algorithm MAD:} The generated set of Steiner points is reduced to \( O(cn) \), where \( n \) is the number of sinks/terminal points and \( c \) is a small factor. The reduction is based on a clustering technique adopted from [24].
time complexity of MAD is $O(n^2 \log n)$. The number of iteration of Step 1 is upper bounded by $n$. The complexity of each Step 1 is $O(r \log r + g)$ based on Fibonacci-heap implementation, where $r$, the total number of nodes in the graph is $O(cn)$ and $g$, the number of edges is a linear function of $n$. Therefore the worst-case complexity of MAD is $O(n^2 \log n)$.

The algorithm admits the following iterative modifications.

Algorithm IMAD-1 at iteration $k + 1$ constructs a Steiner tree with MAD using the values of delays from the tree built at the previous iteration.

Let $T^k$ be the tree constructed by the algorithm at the $k$-th iteration and $d^k_{ij}$ be delay along the arc $(i, j) \in T^k$. The algorithm uses MAD to construct $T^{k+1}$ and at each step of MAD the edge $(u, v)$ that minimizes the value: $t_u(T^{k+1} \cup \{u, v\}) + d^k_{uv}$ ($(u, v) \in E$, $u \in T^{k+1}$, $v \notin T^{k+1}$) is attached to the tree.

Algorithm IMAD-2 at iteration $k + 1$ constructs a Steiner tree with MAD using the values of delays from all the trees built at all the previous iterations.

Let $d^k_{ij}$ be the arithmetic mean of delays $d^l_{ij}$ along the arc $(i, j)$ in trees $T^l$ ($l \leq k$). The algorithm uses MAD to construct $T^{k+1}$ and at each step of MAD the edge $(u, v)$ that minimizes the value: $t_u(T^{k+1} \cup \{u, v\}) + d^k_{uv}$ ($(u, v) \in E$, $u \in T^{k+1}$, $v \notin T^{k+1}$) is attached to the tree.

The algorithms stop, when $T^{k+1} = T^l$ ($l \leq k$), or maximum number of iterations is performed.

2.4.1 Illustration of the steps of MAD

Here we will show how MAD creates a rectilinear Steiner Minimal tree (RSMT) from a given set of points. The hexagonal version can be easily replicated just by changing the Steiner point generation step. In Fig. 2.3, the small circles 0, 1 and 2 are the given terminal points of a net, which are also called sinks. To generate Steiner points for rectilinear geometry, vertical and horizontal lines are drawn through nodes 0, 1 and 2
resulting the grid structure in fig 2.3. The crosses at the intersection of the lines are the candidate Steiner points. Now, we start from root node 0 with an initial trivial (zero edges) tree. We assume delay at node 0, \( t_0 = 0 \). To make the example brief and simple, we will not get into the details of the delay calculation using the formulas. Instead we will assume the delay values at the nodes and generate the tree. Two edges \((0, 3)\) and \((0, 5)\) are going out from root 0. Let us assume, \( \{t_0(T \cup \{0, 3\}) + d_{03}\} < \{t_0(T \cup \{0, 5\}) + d_{05}\} \).

Therefore edge \((0, 3)\) is selected by MAD. Since 3 is an intermediate (Steiner) vertex, just the delay at node 3 is updated as \( t_3 = t_0 + d_{03} \). In the next iteration, \((0, 5), (3, 1)\) and \((3, 4)\) are the candidate edges and edge \((0, 5)\) is added to the tree. Delay of node 5, \( t_5 \) is calculated. Similarly in the next iteration, edge \((5, 6)\) is added to the tree and \( t_6 \) is calculated. Next the minimum delay edge picked up by MAD is \((5, 1)\). But the node to be added is 1, which is a sink. Therefore all the pendent subtrees not containing terminal vertices are to be removed from the tree. The resulting tree in this iteration is shown in Fig. 2.4. Now the delays of the nodes in the tree have changed because subtrees of some of the nodes have changed. Therefore \( t_0, t_5 \) and \( t_1 \) are calculated using
eqn (2.2). Similarly edges (1, 7), (5, 6) and (7, 2) are added in the next three consecutive iterations and the delays of the added nodes $t_7$ and $t_6$ are calculated. Now the last node added to the tree is sink 2. Therefore after cutting-off the redundant subtrees, the tree is as in Fig. 2.5. Since all the sinks are added to the tree, the iteration stops here.

2.5 Congestion Aware Tree Selection

Algorithm IMAD constructs a set of timing-driven Steiner trees for each net in the global graph.

The algorithm is applied when the following inputs are provided. Logical network given as a set of nets and primary inputs with AT(Arrival Time)s and primary outputs with RT(Required Time)s, Number of layers, Specific resistance and capacitance and maximum number of channels $Q_{ij}$ (capacity of corresponding global edge) in each layer, and Resistances and capacitances of vias.

The best trees generated by IMAD are stored as candidate tree set for that net. To increase the cardinality of the candidate set, we use FLUTE [26] along with IMAD. FLUTE is a fast and accurate method for construction of rectilinear Steiner minimal
tree (RSMT). We create Steiner trees for each net applying L-routing on two pin nets decomposed from multi-pin nets (decomposition done by FLUTE). We check the delay of the FLUTE generated trees. If the tree generated from FLUTE is new (i.e. if it was not already generated by IMAD) and has comparable delay with IMAD generated trees, then it is added to the candidate tree set of that net. Therefore, we have several feasible Steiner trees of various types in the candidate set for each net. A gradient based method [31] is described below which is used to pick one tree for each net from its candidate set $Q^s$, so that the total overflow of the edges will be minimum. Consider the problem of optimal use of routing resources, i.e. the available routing tracks. $S$ is the total number of nets. Index $e \in E$ is used for edge of global graph $G$, and index $t'$ is used for trees, where

$$t' \in J = \bigcup_{s=1}^{S} Q^s$$

Set $a_{e,t'} = \begin{cases} 1, & \text{if edge } e \in \text{ the tree } t' \\ 0, & \text{otherwise} \end{cases}$ and $x_{t'} = \begin{cases} 1, & \text{if tree } t' \text{ is selected} \\ 0, & \text{otherwise}. \end{cases}$

Then the problem is defined as follows:

$$f(x) = \sum_{e \in E} \left( \max \left\{ 0, \sum_{t' \in J} a_{e,t'} x_{t'} - q_e \right\} \right)^2 \rightarrow \min_{x_{t'} \in [0,1]} ; \quad (2.4)$$

$$\sum_{t' \in Q^s} x_{t'} = 1, \quad s = 1, \ldots, S \quad (2.5)$$

Objective (2.4) is the sum of penalties for capacity overflows of routing resources $q_e, e \in E$. If there is no overflow, then the penalty (2.4) is zero. Linear relaxation of (2.4) - (2.5) is considered. The function $f(x)$ is convex and smooth, therefore the gradient algorithm, described below can be applied.

For the reason of simplicity, the objective function is rearranged:

$$\tilde{f}(y) = \sum_{e \in E} \left( \max \left\{ 0, y_e - q_e \right\} \right)^2 ;$$
where \( y_e = \sum_{t' \in J} a_{et'} x_{t'} \), \( e \in E \).

We use the following notations below: let \( \varepsilon > 0 \) be desired precision of solution and \( L \) be a lower bound of objective function of problem (2.4) - (2.5).

**Algorithm GradAl.**

**Step 0.** Initialization.

- \( \hat{x}_{t'} := \frac{1}{|Q^s|} \) (\( t' \in Q^s \), \( s = 1, \ldots, S \));
- \( \hat{y}_e := \sum_{t' \in J} a_{et'} \hat{x}_{t'} \) (\( e \in E \));
- \( L := 0 \).

**Step 1.** Iteration. Calculate the values

- \( g_e = \max\{0, \hat{y}_e - q_e\} \) (\( e \in E \));
- \( t'_s = \arg \min_{t' \in Q^s} \sum_{e \in E} g_e a_{et'} \)

and set

- \( \delta_{t'} = \begin{cases} 1 - \hat{x}_{t'}, & t' = t'_s; \\ \hat{x}_{t'}, & t' \neq t'_s; \end{cases} \) for each \( t' \in Q^s \), \( s = 1, \ldots, S \);
- \( z = (z_e), \ z_e = \sum_{t' \in J} a_{et'} \delta_{t'}, \ e \in E \);
- \( GZ = \sum_{e \in E} g_e z_e \).

Recount the lower bound: \( L := \max\{L, \tilde{f}(\tilde{y}) + 2GZ\} \).

If \( \tilde{f}(\tilde{y}) - L \leq \varepsilon \max\{1, L\} \), then STOP. Else GOTO Step 2.

**Step 2.** Move to a new point.

Calculate \( \tilde{p} = \min \left\{ 1, -\frac{GZ}{ZZ} \right\} \), \( GZ \) is defined above, \( ZZ = \sum_{e \in E} (z_e)^2 \), and set

- \( \hat{x}_{t'} := \hat{x}_{t'} + \delta_{t'} \tilde{p}, \ (t' \in J) \);
- \( \hat{y}_e := \hat{y}_e + z_e \tilde{p}, \ (e \in E) \).

Then GOTO Step 1.
Claim. The algorithm provides \((1 + \varepsilon)\)-approximation solution of the problem (2.4) - (2.5). Its time complexity is \(O(|E||J|\varepsilon^{-1}\ln \varepsilon^{-1})\).

Proof. Proof given in Appendix A.

\[\square\]

2.6 History Based MAD

To reduce the overflow further a penalty based on congestion history is added to the original cost function. A trade-off between the delay and the overflow is done so that the delay is not increased beyond a certain range to mitigate the congestion.

The history based cost at \((i + 1)^{th}\) iteration of an edge \(e \) is given as
\[
h_{e}^{i+1} = \begin{cases} h_{e}^{i} + c, & \text{if edge } e \text{ has overflow} \\ h_{e}^{i}, & \text{otherwise} \end{cases}
\]
where \(h_{e}^{i}\) is the history based cost of edge \(e \) at \(i^{th}\) iteration and \(c\) is a constant which can be increased to give more weightage to congestion in the cost function. The actual cost of the edge \(e \) is now calculated as
\[
c_{e} = (b_{e} + h_{e}).p_{e},
\]
where \(b_{e}\) represents the base cost of Step 1 of algorithm MAD of section 2.4. \(p_{e}\) represents the current congestion penalty. This approach is called Negotiated Congestion Routing (NCR) \[53\]. \(p_{e}\) can be obtained using the following equations. Let the density of an edge,
\[
D(e) = \frac{\text{demand(current number of wires passing through the edge)}}{\text{supply(capacity)}}
\]
Then the congestion penalty term \(p_{e}\) \[67\] for edge \(e \) is defined as,
\[
p_{e} = \begin{cases} \exp(\beta(D(e) - 1)), & \text{if } D(e) > 1 \\ D(e), & \text{otherwise} \end{cases},
\]
where \(\beta\) is a positive constant.

2.7 Diversity of the Generated Steiner Trees

The performance of the Gradient algorithm depends on the diversity of the candidate Steiner tree set. The dissimilarity between the trees will give a number of routing possibilities for each net. Using a combination of trees, the congestion can be effectively reduced. To measure the diversity of the Steiner tree pool, we use the metric described in \[37\]. Diversity of the pool,
\[ D = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} D_{ij}}{n(n-1)} = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 - S_{ij}}{n(n-1)} = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 - \frac{\text{comm}(E_i, E_j)}{\text{max}(E_i, E_j)}}{n(n-1)}, \]

where \( n \) = number of candidate Steiner trees in the pool,
\( S_{ij} \) = similarity between two trees \( i \) and \( j \),
\( D_{ij} = (1 - S_{ij}) \) = dissimilarity between two trees \( i \) and \( j \),
\( \text{comm}(E_i, E_j) \) = percentage of common edges shared by tree \( i \) and \( j \), and
\( \text{max}(E_i, E_j) \) = maximum number of edges in either tree.

### 2.8 Experimental Results

Algorithm IMAD, history based MAD and FLUTE are used to generate a set of perspective trees for each net. For the experiment, the maximum number of candidate trees used in the pool are 7. The average diversity of the trees are 32%. A tree is chosen for each net using the gradient algorithm such that minimal residual capacity of global edges is maximal. All algorithms are implemented in C on a quad-core AMD Opteron machine on Linux. We note that no benchmark suites are available for delay driven routing. Existing methods [42, 86, 87] consider examples too small (maximum number of nets 1294) to be meaningful for modern VLSI physical design practices. Also, being a multi-objective NP-hard problem, our delay driven routing with practical running times reduces overflow similar to the existing algorithms and unlike congestion aware only routing algorithms. We run MAD without Gradient and IMAD with Gradient on a set of examples derived from modifying the IBM/ISPD 98 benchmark suite by assigning resistance(0.016 KΩ) and capacitance(0.47 Ff) values to the wires. The modified examples are renamed with suffix “dd” to the benchmark designs. The description of the IBM benchmark suite is given in table 2.1. Maximum number of iterative MAD used to generate the timing-driven Steiner tree is 7.

The results are shown in table 2.2. From table 2.2, it can be seen how history based MAD and Gradient can reduce the overflow efficiently. It is observed that the trees generated by MAD algorithm are almost of the same quality of FLUTE. We compare
our results with FLUTE since it is being used by most of the recent efficient routers.

Fig. 2.7 shows the Elmore delay of the trees generated by FLUTE and our router on some nets of ibm01 benchmark (assuming same resistance and capacitance to calculate Elmore delay for both cases). It is obvious from the figure that the delay of the smaller and medium sized nets are almost same and for the larger nets our router have slightly higher delay. Since very large nets are scarce in these benchmarks, the overall performance of MAD remains almost same as FLUTE. In table 2.3, we show the percentage of FLUTE generated trees selected by the gradient algorithm. Average 33.6% selected trees are generated by FLUTE, i.e. 66.4% selected trees are from MAD, which establishes the importance of MAD as a timing-driven router.
CHAPTER 2. TIMING-DRIVEN ROUTING

Figure 2.6: Histogram of delay difference

Figure 2.7: Delay of FLUTE (red) and our router (green) generated trees on ibm01 benchmark
Table 2.2: Results of MAD with and without Gradient on Examples Derived from IBM/ISPD98 benchmarks

<table>
<thead>
<tr>
<th>Bench Name</th>
<th>Grids</th>
<th># of Nets</th>
<th>MAD w/o Gradient</th>
<th>History based Mad with Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max ovfl</td>
<td>Total ovfl</td>
</tr>
<tr>
<td>ibm01dd</td>
<td>64 x 64</td>
<td>11507</td>
<td>16</td>
<td>2494</td>
</tr>
<tr>
<td>ibm02dd</td>
<td>80 x 64</td>
<td>18429</td>
<td>17</td>
<td>2692</td>
</tr>
<tr>
<td>ibm03dd</td>
<td>80 x 64</td>
<td>21621</td>
<td>12</td>
<td>660</td>
</tr>
<tr>
<td>ibm04dd</td>
<td>96 x 64</td>
<td>26163</td>
<td>12</td>
<td>1574</td>
</tr>
<tr>
<td>ibm06dd</td>
<td>128 x 64</td>
<td>33354</td>
<td>16</td>
<td>2975</td>
</tr>
<tr>
<td>ibm07dd</td>
<td>192 x 64</td>
<td>44394</td>
<td>9</td>
<td>270</td>
</tr>
<tr>
<td>ibm08dd</td>
<td>192 x 64</td>
<td>47944</td>
<td>22</td>
<td>3531</td>
</tr>
<tr>
<td>ibm09dd</td>
<td>256 x 64</td>
<td>50393</td>
<td>10</td>
<td>1920</td>
</tr>
<tr>
<td>ibm10dd</td>
<td>256 x 64</td>
<td>64227</td>
<td>16</td>
<td>2483</td>
</tr>
</tbody>
</table>
Table 2.3: % of FLUTE generated trees selected by Gradient

<table>
<thead>
<tr>
<th>Bench Name</th>
<th>% of FLUTE tree selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01dd</td>
<td>34.1</td>
</tr>
<tr>
<td>ibm02dd</td>
<td>37.8</td>
</tr>
<tr>
<td>ibm03dd</td>
<td>33.8</td>
</tr>
<tr>
<td>ibm04dd</td>
<td>38.3</td>
</tr>
<tr>
<td>ibm06dd</td>
<td>34.9</td>
</tr>
<tr>
<td>ibm07dd</td>
<td>27.2</td>
</tr>
<tr>
<td>ibm08dd</td>
<td>34.1</td>
</tr>
<tr>
<td>ibm09dd</td>
<td>31.7</td>
</tr>
<tr>
<td>ibm10dd</td>
<td>30.1</td>
</tr>
<tr>
<td>Average</td>
<td>33.6</td>
</tr>
</tbody>
</table>
2.9 Conclusion

In this chapter, the timing-driven Steiner tree construction algorithm, MAD \cite{31,72} has been described and to improve the congestion, a history based penalty is added to the existing base cost while constructing delay-driven Steiner trees. Also the complexity of the algorithm is one order less than the existing timing-constrained Steiner tree construction algorithms such as MVERT \cite{41}. A gradient based method \cite{32}, which preserves the timing-driven property of the trees is described for minimizing the overflow. The algorithms have been implemented on modified benchmarks derived from large industry-standard benchmarks called ibm/ISPD 98 benchmarks. None of the available timing-driven global routing algorithms work on such large number of nets.
Chapter 3

Computing Max/Min of a Set of Gaussians: A New Distance Based Approach

3.1 Introduction

The deterministic Static Timing Analysis (STA) can not take care of the process variations which become very predominant with scaling down of technologies. To model the process variations efficiently, SSTA is essential to use. There are mainly two kinds of Statistical Static Timing Analysis (SSTA) methods described in the literatures. One is block based [1, 6, 16, 30, 52, 83], and another is path based [2, 60]. Computing maximum or minimum of two distributions is a very important step in SSTA. Given two distributions, finding the maximum or minimum of them is not a trivial problem. The generated max or min distribution (whichever the case) could be a completely new distribution, if the given distributions are overlapping. For example, Fig. 3.1 shows two given Gaussian distributions shown in red and green line and their generated min distribution (blue line). The generated min distribution is not Gaussian. The formulas used to generate the min distribution is described in detail in the preliminaries section 3.2.1 of this chapter.
Figure 3.1: Min distribution of two given normal distributions

Usually an approximation of the max/min distribution is done to get a normal distribution and this approximated normal distribution is propagated along the paths for timing analysis instead of using the exact max or min of the given distributions. Computing Monte Carlo simulations will produce accurate timing analysis but it would take huge time and thus can not be used for large designs. A detailed study of the statistical timing analysis is shown in [8]. The approximation to normal distribution is usually based on matching the mean and variance [16, 83] of the actual max/min distribution and the re expressed canonical Gaussian distribution. However it could incur error if the two given distributions are overlapping and the generated max/min is a completely asymmetric distribution. In that case, approximation based on mean and variance only could be erroneous and the cumulative inaccuracy could lead to a wrong critical path determination. In Fig. 3.2, two given distributions are shown in red and green line. Their resultant min distribution is shown using blue line and the pink line shows the Gaussian approximation of the actual min distribution. This approximated distribution is propagated in [83] and [16], which are the two most popular methods of SSTA. It can be observed that the approximated min distribution is quite far from the given distributions, though the given distributions almost completely overlap with each other. Intuitively the min or max of them should be very close to them. In Fig. 3.3, we observe that the generated min distribution is quite skewed because of the difference in variances
of the given distributions. Therefore approximating the actual min distribution to a normal distribution and propagating it will not represent the true minimum. Thus only approximation based on the variance will result in significant error for overlapping distributions. Also in [70], there are several examples of approximating the max between two distributions which demonstrate that these approximations become poorer and poorer with the increase in difference of variances. We propose two methods to compute the approximated normal distribution based on considering the distance from the actual distribution to the distribution to be propagated. Two approaches will be described in the following sections. One, where we generate a new normal max/min distribution which will be closest to the actual max/min. In the second approach, we pick one edge from the given set of edges, which is closest to the actual max/min. For both the cases, proximity of the distributions is measured in terms of Hellinger distance. As we consider the whole of the distribution the error could be minimized. Also Hellinger distance computation will take linear time only. Therefore this approach will be comparable in computational perspective too. The first approach could be computationally expensive and therefore
in this chapter we have implemented the second one. The method described in [83] is compared with our work and several thousand Monte Carlo simulations are done on ISCAS'85 benchmark circuits. Our method takes comparatively higher time but the solutions are very close to Monte Carlo simulations compared to the results of [83]. Therefore the error due to approximation can be greatly reduced using this approach. The next section 3.2.1 describes briefly about how the exact max/min distributions are found out and section 3.2.2 gives the definition of Hellinger distance. Section 3.3 proposes the two ideas of finding and propagating the max/min distribution. The experimental results are described in section 3.4. Section 3.5 concludes the chapter.
3.2 Preliminaries

3.2.1 Exact Distribution of Maximum/Minimum of two Gaussian Random Variables

Let $X_1, X_2$ denote two Gaussian random variables with means $(\mu_1, \mu_2)$, variances $(\sigma_1^2, \sigma_2^2)$, and correlation coefficient $\rho$. The minimum of two Gaussian random variables is not necessarily one of them, particularly when the distributions of the given two random variables are overlapping. First we set the $3\sigma$ pruning condition for finding the minimum distribution as shown in the Fig. 3.4. If the distribution of $X_1$ and $X_2$ are non-overlapping, then the simple expression will determine the minimum. If

$$\mu_1 + 3\sigma_1 < \mu_2 - 3\sigma_2$$

$$\Rightarrow \mu_1 - \mu_2 < -3(\sigma_1 + \sigma_2)$$

$$\Rightarrow |\mu_1 - \mu_2| > 3(\sigma_1 + \sigma_2) \quad (3.1)$$

then, $X_1$ will be the minimum. Otherwise, when the distribution of $X_1$ and $X_2$ are overlapping, the distribution of $X=\min(X_1, X_2)$ is obtained from [70]. The mean and
CHAPTER 3. COMPUTING MAX/MIN: A NEW DISTANCE BASED APPROACH

variance of $X = \min(X_1, X_2)$ is shown in the following.

$$E(X) = \mu_1 \Phi \left( \frac{\mu_2 - \mu_1}{\theta} \right) + \mu_2 \Phi \left( \frac{\mu_1 - \mu_2}{\theta} \right) - \theta \phi \left( \frac{\mu_2 - \mu_1}{\theta} \right)$$ (3.2)

$$E(X^2) = (\sigma_1^2 + \mu_1^2) \Phi \left( \frac{\mu_2 - \mu_1}{\theta} \right) + (\sigma_2^2 + \mu_2^2) \Phi \left( \frac{\mu_1 - \mu_2}{\theta} \right) - (\mu_1 + \mu_2) \theta \phi \left( \frac{\mu_2 - \mu_1}{\theta} \right)$$ (3.3)

$$V(X) = E(X^2) - [E(X)]^2$$ (3.4)

where $\phi(.)$ and $\Phi(.)$ are the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) of the standard normal distribution, i.e.,

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$ and $\Phi(x) = \frac{1}{2} \left[ 1 + erf \left( \frac{x}{\sqrt{2}} \right) \right]$ and

$$\theta = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\rho \sigma_1 \sigma_2}$$

The error function $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is a defined function in C. Using this function, the calculation of mean and variance become much easier. In a similar way, the maximum between two distributions can also be found out.

3.2.2 Hellinger Distance

Hellinger distance [88] can be used to measure distance between two distributions. The Hellinger distance between two discrete probability distributions $P = (p_1 \ldots p_k)$ and $Q = (q_1 \ldots q_k)$ is, $H(P, Q) = \frac{1}{\sqrt{\pi}} \sqrt{\sum_{i=1}^k (\sqrt{p_i} - \sqrt{q_i})^2}$. By definition, the Hellinger distance satisfies triangle inequality. In Fig. 3.5, the two given distributions $D_1$ and $D_2$ are shown in red and green lines respectively. The generated minimum distribution, $D_{\min} = \text{Min}(D_1, D_2)$ is shown in blue line. The Hellinger distance $H(D_{\min}, D_1) = 0.014$ and $H(D_{\min}, D_2) = 0.209$. Therefore based on Hellinger distance, the distribution $D_1$ is closer to the actual min distribution $D_{\min}$. 
CHAPTER 3. COMPUTING MAX/MIN: A NEW DISTANCE BASED APPROACH

3.3 Approximating normal distribution to the actual max/min

3.3.1 Picking a virtual wire with approximated Gaussian distribution

Given two timing quantities,

\[ A = a_0 + \sum_{i=1}^{n} a_i \Delta X_i + a_{n+1} \Delta R_a \]  
\[ B = b_0 + \sum_{i=1}^{n} b_i \Delta X_i + b_{n+1} \Delta R_b, \]

the mean and variance of the distribution of \( C = \min(A, B) \) is found out using the formulas in (3.1) or (3.2), (3.3) and (3.4), according to whether the distributions of \( A \) and \( B \) are overlapping or not. Now, we have to find out a Gaussian distribution representing timing quantity, \( C' = c_0 + \sum_{i=1}^{n} c_i \Delta X_i + c_{n+1} \Delta R_c \), so that Hellinger distance \( H(C, C') \) is minimum among all \( C' \) expressed in the above form. To obtain the distribution \( C' \) and the coefficients \( c_i \)'s, solving \( \frac{\partial H(C, C')}{\partial c_i} = 0 \) will give closed form expressions of \( c_i \)'s. This generated \( C' \) is a better approximated normal distribution compared to the existing [83] approach, where only mean and variances are matched to generate the approximated normal distribution because in the proposed approach the whole distribution is matched and thereby it incurs less error.


### 3.3.2 Picking a wire from existing edges

For practical purposes the above approach may take longer time to compute the coefficients of the max/min each time. Therefore we propose here another approach which will help in choosing the critical path quite quickly and accurately. Here to propagate the delay a wire is chosen from the existing edges which is closest in terms of distance from the actual max/min distribution. The max/min distribution is computed similarly using the formulas in (3.1) or (3.2), (3.3) and (3.4), according to whether the given distributions are overlapping or not. Then the Hellinger distance from the actual min/max to the existing edges are computed by iterative pair-wise comparison. This way the min/max for any given set of edge distributions is decided. Thus there will be no need to find the critical path separately as the min/max is one of the given distributions. Also the propagated min/max will be the closest to the actual min/max and there is no need to approximate the actual min/max distribution. The Hellinger distance is computed from the actual min/max distribution, which might not at all be a Gaussian distribution. Thus no error is incurred for approximation.

### 3.4 Experimental Results

In this section, we compare the experimental results of our approach described in section 3.3.2 with the approximation used by Visweswariah et al [83] on ISCAS’85 benchmarks. Both the methods are implemented in C in a 4-core AMD Opteron machine. The process variation coefficients are randomly generated and same values are considered for both the methods. We obtain the coefficients of different arrival times of the output nodes. For comparison purpose, we compute the sum of the mean of all the output timing distributions and the sum of the variances for both the methods. We also do several thousand Monte Carlo simulations by varying the delays at the input nodes and take the average of the simulations. The experimental results are shown in table 3.1. Though our method takes higher time compared to [83], but it performs better as our results are much more closer to the Monte Carlo computations.
### Table 3.1: Comparison between Visweswariah et al [83], our approach and Monte Carlo simulations on ISCAS’85 benchmarks

<table>
<thead>
<tr>
<th>Bench Name</th>
<th>Visweswariah et al [83]</th>
<th>Our method</th>
<th>Monte Carlo Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Variance</td>
<td>Run time(s)</td>
</tr>
<tr>
<td>c6</td>
<td>5.86</td>
<td>2.11</td>
<td>0.005</td>
</tr>
<tr>
<td>c17</td>
<td>27.69</td>
<td>10.65</td>
<td>0.01</td>
</tr>
<tr>
<td>c432</td>
<td>4949.83</td>
<td>2086.27</td>
<td>0.14</td>
</tr>
<tr>
<td>c880</td>
<td>393.22</td>
<td>160.51</td>
<td>0.26</td>
</tr>
<tr>
<td>c1908</td>
<td>14031.1</td>
<td>6435.58</td>
<td>0.68</td>
</tr>
<tr>
<td>c2670</td>
<td>817.992</td>
<td>339.665</td>
<td>0.95</td>
</tr>
<tr>
<td>c3540</td>
<td>21205.5</td>
<td>9110.73</td>
<td>1.63</td>
</tr>
<tr>
<td>c5315</td>
<td>4746.65</td>
<td>1956.26</td>
<td>5.16</td>
</tr>
<tr>
<td>c6288</td>
<td>26288</td>
<td>11805.3</td>
<td>6.70</td>
</tr>
<tr>
<td>c7552</td>
<td>9298.53</td>
<td>3863.69</td>
<td>13.4</td>
</tr>
</tbody>
</table>

### 3.5 Conclusion

In this chapter, we have proposed two ideas for propagating max/min based on Hellinger distance. The advantage of this approach over the existing approaches is that, since it is based on distance between the actual min/max distribution and the candidate distribution, it incurs less error. Whereas approximating the distribution by matching mean and variance could be erroneous when the distributions have significant overlapping. Comparing the results with Monte Carlo simulations prove the efficiency of our approach.
Chapter 4

Variation-Aware MAD

4.1 Introduction

As the device size is scaling down very fast, the parameter variation affects circuit performances significantly. Since the interconnect delay increases rapidly compared to the gate delay with the shrinking of technology, process variation has large impact on the performance of interconnects and therefore it is becoming very difficult to predict the interconnect delay accurately. There are some methods or computer-aided design (CAD) tools such as HSPICE available, which can give accurate results but they are based on Monte Carlo method. Varying the input variables randomly for a large number of times, the mean of all the output samples are computed in this case. Monte Carlo based approaches provide very close solutions but at the cost of a very long time. With statistical timing analysis, modeling the parameter (capacitance, resistance etc.) variations statistically gives an effective variation aware solution for digital circuits in the nanometer era.

SSTA is needed in [1, 5, 71, 81, 83] due to uncertainty in different parameters and various approaches are used to take care of this variations from process and environment.

Berkelaar [5] described how to calculate statistical delay and a method is shown to find the max of the delays. Following that approach, we try to identify the wire/set of wires which are likely to carry the minimum delay as described in chapter 3.

Sivaswamy and Bazargan [78] studied a variation aware modeling and compared the
delay between the deterministic and statistical timing router.

We introduce process variation while constructing a timing-driven global router. In this chapter we will derive the statistical version [73] of the algorithm MAD described in section 2.4. Variation-aware IMAD(Iterative MAD) finds a set of timing driven perspective Steiner trees for each net in a global graph. We generate the delay of the trees considering process variation. In place of deterministic quantities, all capacitances and resistances of the trees are assumed to be correlated Gaussian random variables with the means same as the respective quantities in deterministic algorithm and standard deviation as 7% of their respective mean. We use the Gradient based concurrent approach of section 2.5 to pick up a tree from each set (generated by IMAD) while trying to reduce the overall congestion.

The total wire-length and equivalent delay distributions of all the nets are computed and finally the results between the deterministic and the statistical router are compared.

The main motivation of this chapter is to incorporate variation while constructing the delay driven trees. The rest of the chapter is organized as follows. In section 4.2, some of the definitions and backgrounds such as formulas and metrics used in the statistical version are mentioned. section 4.3 describes the proposed variation aware router. The Gradient algorithm for choosing congestion-aware trees was already described in section 2.5. Section 4.5 shows the comparative experimental results of the deterministic and the statistical routing algorithm on benchmarks derived from IBM/ISPD98 benchmark suite by assigning resistances, capacitances to the wires. Section 4.6 gives the conclusion and future directions.

4.2 Preliminaries

4.2.1 A few properties of Gaussian Random Variables

Considering x, y, z and t as Gaussian random variables and a, b, c and d as constants, the following equations can be used to compute the means and variances of the given
expressions [56]. The mean and variance of each variable is denoted with the usual notation $E[\text{variable}]$ and $V[\text{variable}]$ respectively and $Cov[x, y]$ represents covariance between $x$ and $y$.

\begin{equation}
E[x + y] = E[x] + E[y]
\end{equation}

\begin{equation}
V[x + y] = V[x] + V[y] + 2Cov[x, y]
\end{equation}

\begin{equation}
Cov[x + y, t] = Cov[x, t] + Cov[y, t]
\end{equation}

\begin{equation}
E[ax + y] = aE[x] + E[y]
\end{equation}

\begin{equation}
V[ax + y] = a^2V[x] + V[y] + 2aCov[x, y]
\end{equation}

\begin{equation}
Cov[ax + y, t] = aCov[x, t] + Cov[y, t]
\end{equation}

\begin{equation}
E[xy] = E[x]E[y] + Cov[x, y]
\end{equation}

\begin{equation}
E[ax] = aE[x]
\end{equation}

\begin{equation}
+ V[x]V[y] + Cov[x, y]^2
\end{equation}

\begin{equation}
Cov[xy, t] = E[x]Cov[y, t] + E[y]Cov[x, t]
\end{equation}

\begin{equation}
Cov[x, a] = 0
\end{equation}

\begin{equation}
Cov[ax + by, cz + dt] = acCov[x, z] + adCov[x, t]
+ bcCov[y, z] + bdCov[y, t]
\end{equation}

### 4.3 Statistical MAD - A variation aware version of MAD

The objective is to construct routing trees for each net using MAD algorithm considering process variation. In place of deterministic quantities, all the variation dependent input values, i.e., all capacitances and resistances of the trees are taken as stochastic variables with Gaussian distributions having the same mean as their respective quantities in nominal design and standard deviation as 7% of their respective mean. Therefore, we get the final delay of each branch or edge as a Gaussian PDF. Now to pick the edge with minimum delay, we cannot choose the generated minimum distribution, because the generated minimum distribution might be a completely different distribution which does not physically represent any edge. Therefore, we need to find out which of the delay distribution of the candidate edges is closest to the actual minimum distribution. That way we pick the minimum delay edge in each iteration in the statistical version.

In this section, we derive a statistical expression to choose a minimum delay edge to be added to the tree. Let the root resistance \( r_o \) be normally distributed with mean and variance \( \mu_{r0} \) and \( \sigma_{r0}^2 \) respectively. Similarly, let us assume \( c_{uv} \) (capacitance of edge \( uv \)), \( c_v \) (capacitance of node \( v \)), \( r_e \) (resistance of the edge denoted as \( e \)), \( r_{uv} \) (resistance of edge \( uv \)) to be Gaussian random variables with mean and variances \( \mu_{cuv}, \sigma_{cuv}^2, \mu_{cv}, \sigma_{cv}^2; \mu_{re}, \sigma_{re}^2; \) and \( \mu_{r_{uv}}, \sigma_{r_{uv}}^2 \) respectively. Also covariance among all these random variables are known.

Let \( t_u(T) \) be 0 for initial tree. Let us suppose, there are two edges \( (u_1, v_1) \) and \( (u_2, v_2) \) among which the minimum delay edge has to be chosen. If we consider \( \{t_{u1}(T \cup \{u1, v1\}) + d_{u1v1}\} \) as Gaussian random variable \( X_1(\mu_1, \sigma_1^2) \) and \( \{t_{u2}(T \cup \{u2, v2\}) + d_{u2v2}\} \) as Gaussian random variable \( X_2(\mu_2, \sigma_2^2) \), then we have to first find out \( Y = min(X_1, X_2) \).
To find the expectation of $X_1$,

$$
\mu_1 = E(X_1) = E[t_{u1}(T \cup \{u1, v1\}) + d_{u1v1}]
= E[t_{u1}(T) + r_0(c_{u1v1} + c_{v1}) + \sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1}) + d_{u1v1}]
= E[t_{u1}(T)] + V[r_0(c_{u1v1} + c_{v1})] + V[\sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1})] + E[d_{u1v1}]
$$

and the variance of $X_1$,

$$
\sigma_1^2 = V(X_1) = V[t_{u1}(T \cup \{u1, v1\}) + d_{u1v1}]
= V[t_{u1}(T) + r_0(c_{u1v1} + c_{v1}) + \sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1}) + d_{u1v1}]
= V[t_{u1}(T)] + V[r_0(c_{u1v1} + c_{v1})] + V[\sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1})] + V[d_{u1v1}]
+ 2Cov[t_{u1}(T), r_0(c_{u1v1} + c_{v1})]
+ 2Cov[r_0(c_{u1v1} + c_{v1}), \sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1})]
+ 2Cov[t_{u1}(T), \sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1})]
+ 2Cov[t_{u1}(T), d_{u1v1}] + 2Cov[r_0(c_{u1v1} + c_{v1}), d_{u1v1}]
+ 2Cov[\sum_{e \in P_{u1}(T)} r_e(c_{u1v1} + c_{v1}), d_{u1v1}]
$$

We can expand the terms $t_{u1}$ and $d_{u1v1}$ using equations (2.2), and (2.1). The expectation of all the terms can be found out using equations (4.1), (4.4) and (4.7). The covariance of the terms can be determined using equations (4.3), (4.6), (4.10), (4.12), (4.13). The variance of the expanded terms are obtained using (4.2), (4.5), (4.9). In a similar way, we can find the mean $\mu_2$ and variance $\sigma_2^2$ of $X_2$. The mean and variance of the distribution of $Y = \min(X_1, X_2)$ is found out using the formulas in (3.1) or (3.2), (3.3) and (3.4), according to whether the distributions of $X_1$ and $X_2$ are overlapping or not. Then to choose the minimum delay edge, we find the Hellinger distance of $Y$ from $X_1$ and $Y$ from $X_2$ as described in section (3.2.2) and the distribution which has less
divergence from $Y$ is selected. Accordingly the edge is also selected, i.e. if $X_1$ has been chosen, then edge $(u_1, v_1)$ is to be added to the tree. otherwise $(u_2, v_2)$ will be added to the tree. In this way, from a set of candidate edges $u, v$ s.t. $(u, v) \in E; u \in T, v \notin T$, the minimum delay edge can be found out. The iterations continue until all the sinks are added to the tree.

4.3.1 Properties of Statistical MAD

The theoretical analysis of statistical MAD and its approximation ratio is derived in Appendix B.

4.4 Congestion-aware tree selection for each net

A set of timing-driven variation-aware Steiner trees for each net are generated using variation-aware IMAD. Now to pick one tree for each net from its candidate set, the Gradient algorithm described in section 2.5 is used such that the total overflow of the edges is minimum.

![Max Overflow with and without Gradient](ISPD98-derived-benchmarkinstances.png)

Figure 4.1: Max Overflow with and without Gradient

The Fig. 4.1, 4.2 show how the max and total overflows can be reduced using the proposed Gradient algorithm. The blue lines show the overflows when Gradient is not applied and the red lines show the overflows when Gradient is applied on the ISPD98
Figure 4.2: Total Overflow with and without Gradient
derived benchmark instances.
4.5 Experimental results

The algorithm MAD and its statistical version are implemented in C in a 4-core AMD Opteron machine. The algorithm is tested on a set of examples derived from modifying the IBM/ISPD 98 benchmark suite by introducing resistance and capacitance values to wires along with their probability distributions. Horizontal layers have capacitance 0.47 $Ff$ and resistance 0.016 $K\Omega$ and the vertical layers have capacitance 0.49 $Ff$ and resistance 0.018 $K\Omega$. The modified examples are renamed by suffixing “vdd” to the original benchmark design. The deterministic algorithm inputs (resistance and capacitance) are varied randomly between $-3\sigma$ to $+3\sigma$ ($\sigma$ is the standard deviation of the inputs of statistical version) i.e inputs are in the range of $\mu \pm 3\sigma$. The distribution of the inputs are Gaussian. 99.99% of the samples can be generated within the range of $-3\sigma$ to $+3\sigma$. The reason for truncating the limit beyond $3\sigma$ is because of the practicality of computation. Around 1000 Monte Carlo simulations are run for each sample and the mean of all these results are calculated. The statistical version is run with the mean of the inputs same as the deterministic input and standard deviation as 7% of the mean.

For both the algorithms, a tree is chosen for each net using the gradient algorithm described in section 2.5 such that the minimal residual (current) capacity of global edges is maximal. Then the delay and wire length of all the nets for both the algorithms are compared as given in table 4.1. It can be seen, that the results are close enough as the absolute percentage error is nominal. Also the max and total overflows for the deterministic and statistical case is presented in the table.
Table 4.1: Comparison of Wirelength and Delay between Deterministic (Monte Carlo) and Statistical (Variation Aware) MAD on Examples Derived from ISPD98 Benchmarks

<table>
<thead>
<tr>
<th>Bench Name</th>
<th>Grids</th>
<th># of Nets</th>
<th>Deterministic $\mu$</th>
<th>Statistical $\mu$</th>
<th>Max Ovfl</th>
<th>Tot Ovfl</th>
<th>Wirelength</th>
<th>Delay (ps)</th>
<th>Max Ovfl</th>
<th>Tot Ovfl</th>
<th>Wirelength</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01vdd</td>
<td>64 x 64</td>
<td>11507</td>
<td>65338.73</td>
<td>24002.76</td>
<td>5</td>
<td>290</td>
<td>24002.76</td>
<td>65338.73</td>
<td>5</td>
<td>290</td>
<td>64037</td>
<td>23592</td>
</tr>
<tr>
<td>ibm02vdd</td>
<td>80 x 64</td>
<td>18429</td>
<td>183867.21</td>
<td>31263.39</td>
<td>4</td>
<td>57</td>
<td>31263.39</td>
<td>183867.21</td>
<td>4</td>
<td>57</td>
<td>28641</td>
<td>23157</td>
</tr>
<tr>
<td>ibm03vdd</td>
<td>80 x 64</td>
<td>21621</td>
<td>157261.57</td>
<td>38482.46</td>
<td>2</td>
<td>3</td>
<td>38482.46</td>
<td>157261.57</td>
<td>2</td>
<td>3</td>
<td>28641</td>
<td>23157</td>
</tr>
<tr>
<td>ibm04vdd</td>
<td>96 x 64</td>
<td>26163</td>
<td>175184.04</td>
<td>54010.09</td>
<td>6</td>
<td>140</td>
<td>54010.09</td>
<td>175184.04</td>
<td>6</td>
<td>140</td>
<td>175264</td>
<td>54563</td>
</tr>
<tr>
<td>ibm06vdd</td>
<td>128 x 64</td>
<td>33354</td>
<td>295558.06</td>
<td>63776.47</td>
<td>6</td>
<td>92</td>
<td>63776.47</td>
<td>295558.06</td>
<td>6</td>
<td>92</td>
<td>305840</td>
<td>62803</td>
</tr>
<tr>
<td>ibm07vdd</td>
<td>192 x 64</td>
<td>44394</td>
<td>394126.43</td>
<td>83566.72</td>
<td>1</td>
<td>1</td>
<td>83566.72</td>
<td>394126.43</td>
<td>1</td>
<td>1</td>
<td>403819</td>
<td>83183</td>
</tr>
<tr>
<td>ibm08vdd</td>
<td>192 x 64</td>
<td>47944</td>
<td>441704.62</td>
<td>91227.01</td>
<td>5</td>
<td>47</td>
<td>91227.01</td>
<td>441704.62</td>
<td>5</td>
<td>47</td>
<td>443804</td>
<td>90673</td>
</tr>
<tr>
<td>ibm09vdd</td>
<td>256 x 64</td>
<td>50393</td>
<td>443716.46</td>
<td>91289.17</td>
<td>4</td>
<td>75</td>
<td>91289.17</td>
<td>443716.46</td>
<td>4</td>
<td>75</td>
<td>452941</td>
<td>91907</td>
</tr>
<tr>
<td>ibm10vdd</td>
<td>256 x 64</td>
<td>64227</td>
<td>627420.87</td>
<td>126671.32</td>
<td>4</td>
<td>38</td>
<td>126671.32</td>
<td>627420.87</td>
<td>4</td>
<td>38</td>
<td>634979</td>
<td>125251</td>
</tr>
</tbody>
</table>

% error

- ibm01vdd: 1.99, 1.70
- ibm02vdd: 1.24, 1.00
- ibm03vdd: 2.27, 1.20
- ibm04vdd: 2.04, 1.02
- ibm06vdd: 3.47, 1.52
- ibm07vdd: 2.45, 0.45
- ibm08vdd: 0.49, 0.61
- ibm09vdd: 2.08, 1.06
- ibm10vdd: 1.20, 1.12
4.6 Conclusion

Here a statistical variation aware version of a deterministic timing-driven congestion aware algorithm for construction of Steiner trees is derived. The delay and wire-length of the variation aware router are compared with Monte Carlo simulations of the deterministic algorithm. We see that the results of the statistical router differs by very small values from the mean of about 1000 simulations of the Deterministic Monte Carlos. Therefore, it will be more efficient to use the statistical router to consider the process variation instead of running so many Monte Carlo simulations, which take very long time.
Chapter 5

Variation-Aware MAD on Uniform $\lambda$-Geometry

5.1 Introduction

The multi-net rectilinear Steiner tree construction with the objective of wirelength minimization for routing in VLSI has received a lot of attention in the last few decades. As it has been observed now that non-Manhattan routing reduces the wirelength more than the traditional Manhattan routing, the routing in alternate directions are gaining increased interest. But there has not been much work on timing-driven non-Manhattan routing. In the traditional Manhattan architecture wires are allowed only in two directions along $0^0$ and $90^0$. The recent fabrication technology has made over-the-cell routing feasible. Therefore the mandatory rectangular route restriction is relaxed and other routing grids can be accommodated. Also since it has been observed that non-Manhattan routers provide solution with less wirelength and less delay, routing in other uniform orientations are becoming more desirable. The different routing architectures are formed according to their permitted orientations. Each $\lambda$ uniformly distributed architecture allows routing orientations forming consecutive angles of $\pi/\lambda$. $\lambda = 2$ denotes the M routing. The Y routing refers to $\lambda = 3$, i.e it permits wires in $0^0$, $60^0$ and $120^0$ and $\lambda = 4$, i.e wires oriented in $0^0$, $45^0$, $135^0$ and $180^0$ represents X routing. The properties
of the three routing architectures (M, X and Y) as observed from different literatures are shown in table 5.1. Geosteiner [84], a method based on linear programming has been proved most successful for constructing rectilinear and Euclidean Steiner trees. Coulston [28] extended the idea of GeoSteiner for creating exact octagonal Steiner minimum trees. A Multi Commodity Flow (MCF) based approach is explored for Y architecture in [20]. In [80], an exact algorithm with small scale optimal solution using branch and bound algorithm is proposed for computing hexagonal Steiner minimum trees. A fast heuristic method for constructing Y-routed trees has been reported in [75].

Almost all

Table 5.1: Comparison between properties of different $\lambda$-Geometry Routing

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Rectilinear $\lambda = 2$</th>
<th>Hexagonal $\lambda = 3$</th>
<th>Octagonal $\lambda = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wirelength [28]</td>
<td>Maximum</td>
<td>Intermediate</td>
<td>Minimum</td>
</tr>
<tr>
<td>Routing Direction</td>
<td>$0^0, 90^0$</td>
<td>$0^0, 60^0, 120^0$</td>
<td>$0^0, 45^0, 135^0, 180^0$</td>
</tr>
<tr>
<td>Via cost [48]</td>
<td>Minimum</td>
<td>Intermediate</td>
<td>Maximum</td>
</tr>
<tr>
<td>Candidate Steiner point [48]</td>
<td>$O(4)$</td>
<td>$O(9)$</td>
<td>$O(16)$</td>
</tr>
<tr>
<td>$O(\lambda^2)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing grid [19]</td>
<td>Regular</td>
<td>Regular</td>
<td>Not regular</td>
</tr>
</tbody>
</table>

the current works in non-Manhattan, even in Manhattan routing are based on wirelength optimization algorithms but as mentioned by Moffitt et al [58], there is increasing demand of timing-driven routing algorithms. Minimizing the wirelength merely does not ensure the physical electrical connectivity. Some of the very early literatures [10, 11, 41] address the timing constraint but they are of complexity $O(n^3)$ to $O(n^4)$ and the results are shown for very small sized nets and also do not consider $\lambda > 2$-geometry. A more recent work by Yan [87] constructs timing-driven octilinear tree. But since the experiments were done on random nets of maximum size 96 pins, we could not compare our work with them. In this chapter, the Elmore delay based method MAD of section 2.4 is used for constructing rectilinear and hexagonal Steiner trees. The Steiner points for different architectures are created according to their permitted directions and the proposed algorithm is then applied on the generated grid. The proposed method prunes
the pendant redundant (containing only Steiner points) subtrees and updates the delays as the delay of the nodes change with the configuration of the tree. Also it has been observed that the final Steiner minimum tree on any uniform architecture uses only a very small fraction of the generated Steiner points. Therefore we have adopted an effective and fast method from [24] to reduce the set of Steiner points using clustering of the points. The reduction of the Steiner points scales down the runtime by several times at the cost of increasing the wirelength by only a very small factor. We have implemented our method on the reduced Steiner (Hanan) point set and compared with [28] and [75] on OR-Library benchmarks [4]. For the sake of reading, we will refer [28] as method1 and [75] as method2. Method1, based on Geosteiner [84] can not provide solution beyond a certain pin no(20 pins) in considerable time whereas method2 provides solutions in small cpu time with larger wirelengths for upto 70 pin nets. Comparison between our method and the existing wire-length minimizing only methods is not relevant here since they do not consider the timing of the electrical pins. Though our objective is to optimize the delay of the sinks of $\lambda$-geometry tree, since there are no suitable benchmark openly available for this purpose, we compare the wirelengths with the mentioned methods which construct non-Manhattan tree. We observe that the hexagonal trees have improved delay and wirelength compared to rectilinear trees. Our method provides solutions much faster than method2 and also with smaller wirelengths for most of the instances of the benchmark. Our method generate solutions for upto 1000 pin nets in a trivial time. Since our proposed method does not depend on the routing geometry, it can be easily applied on any uniform orientation.

The rest of the chapter is organized as follows. In section 5.2, we describe the procedure for constructing different $\lambda$-geometry grids. The method for reduction of the Steiner point set is given in detail in section 5.3. A comparison between our proposed timing-driven $\lambda$-geometry Steiner tree construction method and two existing $\lambda$-geometry Steiner tree construction methods are demonstrated in section 5.4. Section 5.5 gives some future directions. The variation-aware version of MAD is also extended on different $\lambda$ orientations, which is described in section 5.6.
5.2 Problem Formulation

As we have already described, Steiner points are to be created for each net before constructing the tree. For rectilinear tree construction, the Steiner points are generated by passing two lines through each of the vertices along $0^\circ$ and $90^\circ$. Similarly for hexagonal routing, three lines along $0^\circ$, $60^\circ$ and $120^\circ$ are to be drawn through all the given vertices. In both the cases, the intersecting points generated by these lines are candidate Steiner points for the respective grids.

Fig. 5.1 describes how Steiner points are generated on different $\lambda$-Geometry.

![Figure 5.1: Generating Steiner points on different $\lambda$-Geometry](image)

After generating the Steiner points, we apply the timing-driven algorithm on the total set of the vertices. The algorithm basically adds up nodes one by one to the initially trivial tree such that the delay at the nodes in the tree remains minimum.

5.3 Reducing the cardinality of the Steiner point set

We have observed that the generated Steiner trees use less than 5% [24] of the total Steiner point set. But we have to explore all these points to create the optimal delay/wirelength tree which makes the cost of computation higher as the number of terminal points increase more and more. Therefore a strategy as has been suggested in [24], is applied here to reduce the cardinality of the Steiner point set. Reducing the Steiner points to a certain extent improves the run time of the trees very much with only slightly worse delays/wirelengths. Various methods were discussed in [24] among which we adopted the concept of clustering the Hanan points in our work. Basically a set of
adjacent (within a unit distance) Steiner points can be clustered together. All the adjacent points in a cluster have distance less than the given maximum adjacency distance and their slopes are same, i.e., clusters are always rectangular in nature. This way, the total set of points can be grouped as a set of clusters. Now from each cluster, one point is chosen to keep in the new set of Steiner points. This point is chosen on the basis of its overall distance from the set of terminal points. That is, the point in a cluster having the shortest total distance from all of the terminal points is the representative point for that cluster. Similarly each cluster contribute a point in the new reduced Steiner point set. As an example in Fig. 5.2, we show the Steiner points generated for constructing hexagonal tree of a 40 pin net instance of OR-Library. The original Steiner points are shown in blue and the pink points show the reduced set of points after clustering and selecting. All the pink points overlap with some blue points. The green circular points show the given terminal points. The excess blue points can be seen in the Fig. The maximum adjacency distance considered for clustering in this case is 0.05. All the terminal points are in the range \([0,1]\). Using the reduced Steiner point set over the original set, we observe that the wirelength is increased by a nominal 1.05 times though the runtime decreases by 6.40 times for this instance. Fig. 5.3 demonstrates the number of Steiner
CHAPTER 5. VARIATION-AWARE MAD ON UNIFORM $\lambda$-GEOMETRY

Figure 5.3: Steiner pts for hex & rec trees of #70(1) at different adjacency dist

Figure 5.4: Wirelength and Run time for #70(1) at different adjacency dist

points at different maximum adjacency distances for hex and rec grids for the instance 70(1). The reduction significantly improves the runtime for larger benchmarks. Fig. 5.4 shows the wirelength and runtime(sec) for the rectilinear trees generated for the instance 70(1) at different max adjacency distance. From an observation of delays/wirelengths and runtimes at different max adjacency distances over different instances of the OR-Library, we decided to take the max adjacency distance as 0.04 for constructing both hex and rec trees in general, as it provides a trade-off between the delays/wirelength and runtime. The clustering method is of $O(n^2)$ complexity.
Table 5.2: Comparison of Wirelength and Delay for different \( \lambda \)-Geometry Routing on OR-Library Instances

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rec  wl</td>
<td>Euc  wl</td>
<td>Hex  wl</td>
</tr>
<tr>
<td>10</td>
<td>1.91</td>
<td>1.60</td>
<td>1.79</td>
</tr>
<tr>
<td>20</td>
<td>3.26</td>
<td>2.85</td>
<td>3.05</td>
</tr>
<tr>
<td>30</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>40</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>70</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>80</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>90</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>100</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>250</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>500</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
<tr>
<td>1000</td>
<td>N.A</td>
<td>N.A</td>
<td>N.A</td>
</tr>
</tbody>
</table>

### 5.4 Experimental Results

The proposed method has been implemented in C in a Sun Microsystems 2.2 GHz Dual-Core Processor. The other methods were also run on the same machine.

The comparison between our method and method1 [28] and method2 [75] are shown in table 5.3. Method1 fails to generate solutions beyond 20 pin nets in considerable amount of time and method2 provides solutions up to only 70 pin nets in reasonable time. Wirelength and specifically runtime of our solution is much better than method2 for almost all the instances for both M and Y routing. Also our method generates the solution for up to 1000 pin nets in very small time for both the architectures. N.A in the table denotes “not available” for those instances method1 and method2 could not complete in reasonable time. It can be observed from table 5.3, that both the delay and the wirelength of hex trees are smaller than the rec trees. Thus it establishes the significance of non-Manhattan routing.
Figure 5.5: Rec tree generated on OR Library instance 10(1) by MAD

Figure 5.6: Rec tree generated on OR Library instance 10(2) by MAD
5.5 Summary

An Elmore delay based method for constructing Steiner trees on any uniform routing grid is proposed here. Since the Steiner points are created as a generalization of Hanan’s grid, this method is applicable to any uniform $\lambda$-geometry architecture. The effectiveness of reducing the cardinality of the Steiner point set to scale down the runtime has been utilized here. Our method produces very fast and quite good solution compared to the existing methods and can be applied on very large nets which other methods cannot complete in reasonable time.

5.6 Variation-Aware $\lambda$-MAD

In this section, we will show how we have extended algorithm MAD to consider process variation on uniform $\lambda$-Geometry [74]. The different grids for routing are generated as described in section 5.2. To reduce the search space, the clustering method in section 5.3 is applied for pruning the generated Steiner point set. Then the variation-aware MAD described in section 4.3 is applied on the reduced Steiner set.

5.6.1 Experimental Results

The algorithm MAD and its statistical version are implemented in C in a Sun Microsystems 2.2 GHz Dual-Core Processor. The algorithm is tested on a set of examples derived from modifying the OR-Library benchmark suite by introducing resistance and capacitance values to wires along with their probability distributions. The trees are generated on different $\lambda (= 2, 3)$-geometries. Trees generated by the $\lambda$ - MAD algorithm($\lambda = 2$) on two instances of OR-library with 10 sinks are shown in Fig. 5.5 and Fig. 5.6. The set of terminal points are given in the range $[0,1]$. Steiner points are generated using Manhattan grids as described in section 5.2. The clustering algorithm is applied after that to reduce the cardinality of the Steiner point set as shown in section 5.3. Finally the timing-driven routing is applied and the generated trees are shown. The deterministic
algorithm inputs (resistance and capacitance) are varied randomly between $-3\sigma$ to $+3\sigma$ ($\sigma$ is the standard deviation of the inputs of statistical version) i.e inputs are in the range of $\mu \pm 3\sigma$. Several thousand Monte Carlo simulations are run for each sample and the mean of all these results are calculated. The statistical version is run with the mean of the inputs same as the deterministic input and standard deviation as 7% of the mean.

It can be observed from table 5.3 that the results of the statistical router differ by very small values from its deterministic counterpart. Therefore our proposed variation-aware router saves a lot of computational time while predicting the variation quite accurately. Also both the delay and the wirelength of hex trees are observed to be smaller than the rec trees. Thus it establishes the significance of non-Manhattan routing.

### 5.6.2 Conclusion

Here an Elmore-delay based delay-driven variation-aware statistical method for construction of Steiner trees on any uniform $\lambda$-Geometry is derived. To the best of our knowledge, there is no statistical $\lambda$-geometry router present in the literatures. From the experimental results, it can be seen that our statistical router is more efficient for considering the process variation instead of running so many Monte Carlo simulations, which takes very long time. Also our proposed method is not based on any particular geometry and therefore it can be extended to construct trees in any uniform orientations.
Chapter 6

Conclusion

This thesis focuses on timing-driven routing under process variation. The main contribution is a fast and accurate technique based on distance between distributions to pick max/min from a set of timing quantities which are usually represented as Gaussian random variables. Since distance is computed instead of matching only variances, the error due to approximation to normal distribution is completely avoided. The timing-driven variation-aware routing is also extended in other uniform \( \lambda \)-geometries.

6.1 Summary

Routing in VLSI physical design is a well studied problem. But most of the work is concentrated on optimizing the wirelength and congestion. Since timing is an important criteria to be addressed and optimizing wire length only will not necessarily meet timing, the objective of this work was to focus on timing-aware routing . Also the recent literatures [58] give emphasis to construct new delay-driven routing algorithms.

As the technology scales down, process variation has a significant impact on the circuit performance. Chapter 3 computes max/min based on Hellinger distance and therefore no error due to approximation to normal distribution is incurred. As the existing approaches compare second moment to match the distributions, approximation error could be very significant when the given distributions are significantly overlapping. Chapter 4 derives
the variation-aware version of the timing-driven algorithm MAD [31, 33] by taking the variation aware parameters like resistance, capacitance etc. as Gaussian random variables instead of deterministic quantities. During the propagation of the delays, the distance between distributions are considered for computing minimum delay edges. Since MAD is not dependent on any particular geometry or architecture, chapter 5 describes how it has been extended in uniform \(\lambda\)-orientations. Also routing on \(\lambda\)-geometry under variation is illustrated in the later part of that chapter. The next section 6.2 suggests some of the future directions related to this thesis.

6.2 Future Work

- Generating a set of trees with optimized wirelength and use the Gradient algorithm for concurrently mitigating congestion. This will be a similar approach to the existing state-of-the-art routers whose objective is to optimize the wirelength and congestion. But the novelty of the method will be concurrent reduction of congestion rather than using time-consuming sequential rip-up and reroutes.

- Using the gradient method hierarchically on timing-driven Steiner tree sets, i.e partitioning the layout area first and then using the gradient on a small cluster of nets based on their criticality may improve the congestion even more with a slight cost in run time.

- The timing-driven routing on \(\lambda\)-geometry will definitely have better performance if it is done after \(\lambda\)-geometry aware placement. This will further improve the quality of our routing trees.

- Generating timing-driven obstacle-aware Steiner tree construction on uniform \(\lambda\)-geometry. The existing algorithm on uniform geometry in chapter 5 can be extended to avoid regular shaped obstacles while routing.
Appendix A

Complexity of Gradient Algorithm [31]

Claim. The algorithm provides \((1 + \varepsilon)-\)approximation solution of the problem (2.4) - (2.5). Its time complexity is \(O(|E||J|\varepsilon^{-1} \ln \varepsilon^{-1})\).

Proof. It is first demonstrated that the objective function decreases at each iteration. Since function \(\tilde{f}(y)\) is convex, then

\[
\tilde{f}(y) \geq \tilde{f}(\tilde{y}) + \nabla \tilde{f}(\tilde{y})(y - \tilde{y}) = \tilde{f}(\tilde{y}) + \sum_{e \in E} 2g_e(y_e - \tilde{y}_e),
\]

where \(y\) is arbitrary feasible point. The last expression is estimated by solving the problem

\[
\sum_{e \in E} g_e y_e = \sum_{e \in E} g_e \sum_{t \in J} a_{et} x_t = \sum_{t \in J} \left( \sum_{e \in E} g_e a_{et} \right) x_t \to \min_x,
\]

The problem above decomposes into the independent subproblems for each net \(s\):

\[
\min_x \sum_{t \in J} \left( \sum_{e \in E} g_e a_{et} \right) x_t = \sum_{s=1}^{S} \min_{t \in Q^s} \left( \sum_{e \in E} g_e a_{et} \right) = \sum_{s=1}^{S} \sum_{e \in E} g_e a_{ets},
\]
the last equality follows from the definition of $t_s$. Thus,

$$
\sum_{e \in E} g_e (y_e - \tilde{y}_e) \geq \sum_{s=1}^{S} \sum_{e \in E} g_e a_{et_s} - \sum_{e \in E} g_e \tilde{y}_e
$$

$$
= \sum_{e \in E} g_e \left( \sum_{s=1}^{S} a_{et_s} - \sum_{t \in J} a_{et} x_t \right)
$$

$$
= \sum_{e \in E} g_e \left( \sum_{s=1}^{S} a_{et_s} (1 - x_{t_s}) - \sum_{t \in J, t \neq t_s} a_{et} x_t \right)
$$

$$
= \sum_{e \in E} g_e \sum_{t \in J} a_{et} \delta_t = GZ.
$$

Therefore,

$$
\tilde{f}(y) \geq \tilde{f}(\tilde{y}) + 2GZ,
$$

$L = \tilde{f}(\tilde{y}) + 2GZ$ is a lower bound for $\tilde{f}(y)$, and if $\tilde{f}(\tilde{y}) - L \leq \varepsilon \max\{1, L\}$, then $\tilde{y}$ defines a $(1 + \varepsilon)$-approximation. Otherwise the descent direction is defined by vectors $\delta$ and $z$.

The descent direction $\delta$ is from current point $\tilde{x}$ to the integer point $x = (x_t)$, where $x_t = 1$ if $t \in \{t_1, \ldots, t_s\}$, and $x_t = 0$, otherwise. In order to find a stride parameter $p \in [0, 1]$, the minimum of the following function can be found out

$$
h(p) = f(\tilde{y} + zp) = \sum_{e \in E} (\max\{0, \tilde{y}_e - q_e + z_e p\})^2,
$$

but a simpler function is used

$$
r(p) = \sum_{e \in E} (g_e + z_e p)^2.
$$

It is easy to check that $h(0) = r(0), h'(0) = r'(0), h(p) \leq r(p)$ for all $p \in [0, 1]$, and function $r$ reaches a minimum in $\hat{p} = \min \left\{ 1, -\frac{GZ}{Z} \right\}$.

If the current point $\tilde{y}$ is not optimal, then $h'(0) < 0, r'(0) < 0$ and $\hat{p} > 0$. Inequality $h(0) > h(\hat{p})$ follows from the expression $h(0) = r(0) > r(\hat{p}) \geq h(\hat{p})$.

Therefore, moving into point $\tilde{x}_t' = \tilde{x}_t + \delta_t \hat{p}$ ($t \in J$) and $\tilde{y}_e' = \tilde{y}_e + z_e \hat{p}$ ($e \in E$) does
Appendix A

not increase the objective function.

Now it is proved that each iteration has time complexity $O(|E||J|)$, and the total number of iterations is bounded by $O(\varepsilon^{-1} \ln \varepsilon^{-1})$.

A lower bound for $\rho = r(0) - r(\tilde{p})$ for one iteration is found out first. $\Delta = -2GZ$ is set. From definition of $r(p)$ and step (stride parameter) $\tilde{p}$ follows that if $\Delta \geq 2ZZ$, then $\rho \geq \Delta/2$, and if $\Delta \leq 2ZZ$, then $\rho \geq \frac{\Delta^2}{4ZZ}$. Finally

$$\rho \geq \min \left\{ \frac{\Delta}{2}, \frac{\Delta^2}{4ZZ} \right\} = \frac{\Delta}{2} \min \left\{ 1, \frac{\Delta}{2ZZ} \right\}$$

The same estimation takes place for argument $d = f(\tilde{y}) - f(\tilde{y} + z\tilde{p})$ of objective function, since $d \geq \rho$.

The upper bound for $D = f(\tilde{y}) - L$ is estimated in the following. $Q$ is defined as the total number of iterations. Parameters of one iteration is defined by upper index $l = 0, 1, \ldots, Q$. Step 0 corresponds iteration $l = 0$, then $\tilde{y}^0$ is initial point, $L^0 = 0$ and $D^0 = f(\tilde{y}^0) - L^0 = f(\tilde{y}^0)$. $c$ is defined as a number which is greater than each $f(\tilde{y}^l)$, $\Delta^l$ and $ZZ^l$, $l = 0, 1, \ldots$, even if the process is infinite. Such number exists since the feasible set of linear relaxation problem is compact.

From equalities $f(\tilde{y}^l) = f(\tilde{y}^{l-1}) - d^l$, $L^l = \max\{L^{l-1}, f(\tilde{y}^{l-1}) - \Delta^l\}$ and $D^l = f(\tilde{y}^l) - L^l$ follows that $D^l = \min\{D^{l-1}, \Delta^l\} - d^l$. Taking into account the above lower bound for $d^l$, the definition of number $c$ and inequality $\Delta^l \geq 0$, we get

$$D^l \leq \min\{D^{l-1}, \Delta^l\} - \frac{(\Delta^l)^2}{4c} \leq \max_{v \geq 0} \left\{ \min\{D^{l-1}, v\} - \frac{v^2}{4c} \right\}$$

Maximum of the last expression reaches when $v = D^{l-1}$, then

$$D^l \leq D^{l-1} - \frac{(D^{l-1})^2}{4c} = \left( 1 - \frac{D^{l-1}}{4c} \right) D^{l-1}$$

Continuing this inequality using inequality $D^{l-1} > \varepsilon$, which is true at each iteration
except the last one. As a result, we get

\[ D^l \leq \left(1 - \frac{\varepsilon}{4c}\right) D^{l-1} \leq \left(1 - \frac{\varepsilon}{4c}\right)^l D^0 \leq \left(1 - \frac{\varepsilon}{4c}\right)^l c \]

The number of iterations \( Q \) is bounded by \( T(\varepsilon) \), and

\[ \left(1 - \frac{\varepsilon}{4c}\right)^{T(\varepsilon)} c = \varepsilon \text{ and since } \lim_{\varepsilon \to 0} \frac{T(\varepsilon)}{\varepsilon^{-1} \ln \varepsilon^{-1}} = 4c \text{, the complexity do not exceed } O(\varepsilon^{-1} \ln \varepsilon^{-1}). \]
Appendix B

Properties of Statistical MAD

In this section, we investigate the performance of variation-aware MAD applied to grid graphs. The lemmas are derived from deterministic MAD properties [79]. We call $G$ a $M \times N$ grid graph if the set of vertices $V(G) = \{(m, n) | 0 \leq m \leq M, 0 \leq n \leq N\}$ for some $M$ and $N$ and edge set of graph, $E(G) = \{(v_1, v_2) | v_1 = (m, n), v_2 = (m, n + 1) \text{ or } v_2 = (m+1, n)\}$. Given the Graph $G(V, E)$, each edge $(i, j) \in E$ has two parameters, resistance and capacitance as Gaussian distributions, whose mean are $E[r_{ij}]$ and $E[c_{ij}]$ respectively.

Consider the subset of vertices $S = \{0, 1, \ldots, k\} \subseteq V$, where vertex 0 is the source of the signal and nodes $S \setminus \{0\}$ are recipients of the signal (terminal vertices or terminals). Each vertex $i \in S$ has $E[c_i]$ as the mean of its capacitance distribution. and $E[r_0]$ is the mean of the resistance distribution of the vertex 0. Consider a Steiner tree $T$ spanning $S$ and rooted in 0. Let $k$ be an arbitrary terminal in $T$ and the Elmore delay distribution at $k$ is $t_k(T)$. Let us denote:

- $P_k(T)$ - path from 0 to $k$ in $T$. Also we use denotation $P_{uv}(T)$ for path from $u$ to $v$ in $T$ below; we omit argument $T$ where it is obvious from context;

- $T_j (j \in V(T))$ is downstream subtree of $T$ with root $j$; $T_e (e \in E(T))$ is downstream subtree of $T$ rooted in the head node of arc $e$;

- $C(H)$ ($R(H)$) - total capacitance (resistance) distribution of subgraph $H$, $E[C(H)] = E \left[ \sum_{e \in E(H)} c_e \right] + E \left[ \sum_{i \in V(H)} c_i \right]$. Also we use notations: $C_j = C(T_j)$, $C_e = C(T_e)$ and $R_{uv} = R(P_{uv})$. 

63
Figure 6.1: Lemma 1

Considering Steiner tree \( T \), let \( \hat{t}_u \) be the delay distribution along the path \( P_u \) (disregarding subtrees) and \( \bar{t}_{uv} \) be the delay distribution in vertex \( v \) in subtree \( T_u \setminus T_v \).

Weighted grid graph \( G \) belongs to class \( \Gamma_1 \) if the expectation of resistances and capacitances of all the edges of one line (horizontal or vertical) are equal, i.e. given edges \( e_1 = ((m_1, n_1), (m_1, n_1 + 1)) \) and \( e_2 = ((m_1, n_2), (m_1, n_2 + 1)) \), then \( E[r_{e_1}] = E[r_{e_2}] \) and \( E[c_{e_1}] = E[c_{e_2}] \). The same property holds for horizontal edges.

Weighted grid graph \( G \) belongs to class \( \Gamma_2 \), if \( G \in \Gamma_1 \) and additionally for each \( e \in E(G) \), \( E[c_e] = k \cdot E[r_e] \) holds for some \( k \).

In this section, we will show that, if \( G \in \Gamma_2 \) and contains \( n \) terminal vertices, then Statistical MAD provides \( n \)-approximation solution of problem (2.3).

Lemma 1. Let \( T \) be a Steiner tree for \( S \) and \( u \in S \) be a leaf in \( T \) and \( P \) be the path connecting root 0 and \( u \). \( T \setminus P \) is a set of subtrees \( \{T_{(0)}, \ldots, T_{(k)}\} \). Denote by \( v_i \) the root of \( T_{(i)} \), \( v_i \in P \). Vertices \( \{v_1, \ldots, v_k\} \) split \( P \) into \( k + 1 \) parts: \( P = P_1 \cup P_2 \cup \cdots \cup P_k \), where \( P_1 = P_{0v_1} \), \( P_{k+1} = P_{v_ku} \) and \( P_i = P_{v_{i-1}v_i} \) for \( i = 1, \ldots, k \) (see Fig. 6.1). Then

\[
E[t_u(T)] = E[\hat{t}_u] + E\left[r_0 \sum_{i=0}^{k} C(T_{(i)})\right] + E\left[\sum_{i=1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)})\right]
\]

Proof. Let \( C_e^P = C(T_e \cap P_u) \) and \( C_e^T = C(T_e \setminus P_u) \). The lemma follows from the equalities below.
We know,

\[ t_u(T) = r_0 \left( \sum_{i=1}^{k+1} C(P_i) + \sum_{i=0}^k C(T(i)) + c_u \right) + \sum_{e \in P_u} r_e \left( \frac{c_e}{2} + (C_e^P + C_e^T) \right) \]

Therefore,

\[
E[t_u(T)] = E \left[ r_0 \left( \sum_{i=1}^{k+1} C(P_i) + \sum_{i=0}^k C(T(i)) + c_u \right) + \sum_{e \in P_u} r_e \left( \frac{c_e}{2} + (C_e^P + C_e^T) \right) \right]
\]

\[
= E \left[ r_0 \left( \sum_{i=1}^{k+1} C(P_i) + \sum_{i=0}^k C(T(i)) + c_u \right) \right] + E \left[ \sum_{e \in P_u} r_e \left( \frac{c_e}{2} + (C_e^P + C_e^T) \right) \right]
\]

\[
= E \left[ r_0 \left( \sum_{i=1}^{k+1} C(P_i) + c_u \right) \right] + E \left[ \sum_{i=0}^k C(T(i)) \right] + E \left[ \sum_{e \in P_u} r_e C_e^P \right] + E \left[ \sum_{e \in P_u} r_e C_e^T \right]
\]

\[
= E \left[ r_0 \left( \sum_{i=1}^{k+1} C(P_i) + c_u \right) \right] + E \left[ \sum_{e \in P_u} r_e C_e^P \right] + E \left[ \sum_{e \in P_u} r_e C_e^T \right]
\]

(6.1)

Now,

\[ \hat{t}_u = r_0 \left( \sum_{i=1}^{k+1} C(P_i) + c_u \right) + \sum_{e \in P_u} r_e C_e^P + \sum_{e \in P_u} r_e C_e^T \]

Therefore,

\[
E[\hat{t}_u] = E \left[ r_0 \left( \sum_{i=1}^{k+1} C(P_i) + c_u \right) + \sum_{e \in P_u} r_e C_e^P + \sum_{e \in P_u} r_e C_e^T \right]
\]

(6.2)
Substituting the value of $E[\hat{t}_u]$ from (6.2) into (6.1) we obtain,

$$E[t_u(T)] = E[\hat{t}_u] + E \left[ r_0 \sum_{i=0}^{k} C(T_{(i)}) \right] + E \left[ \sum_{e \in P} r_e C_e^T \right]$$

$$= E[\hat{t}_u] + E \left[ r_0 \sum_{i=0}^{k} C(T_{(i)}) \right] + E \left[ \sum_{i=1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)}) \right]$$

\[\square\]

**Lemma 2.** Given the path $P$ connecting the source vertex 0 with a terminal $u$ and $v \in P$ (see Fig. 6.2). Then

$$E[t_u(P)] = E[t_{0v}] + E[\hat{t}_{vu}] + E[R_{0v}(C_{vu} + c_u)]$$

**Proof.** Here,

$$C_{e}^{0v} = C(T_{e} \cap P_{0v})$$
Now we know,
\[
t_u(P) = r_0(C_{ov} + C_{vu} + c_u) + \sum_{e \in P_{0v}} \frac{r_e c_e}{2} + \sum_{e \in P_{vu}} \frac{r_e c_e}{2}
+ \sum_{e \in P_{0v}} r_e(C_{e0v} + C_{vu} + c_u) + \sum_{e \in P_{vu}} r_e(C_e + c_u)
\]

Therefore,
\[
E[t_u(P)] = E[r_0(C_{ov} + C_{vu} + c_u)] + E \left[ \sum_{e \in P_{0v}} \frac{r_e c_e}{2} \right] + E \left[ \sum_{e \in P_{vu}} \frac{r_e c_e}{2} \right]
+ E \left[ \sum_{e \in P_{0v}} r_e(C_{e0v} + C_{vu} + c_u) \right] + E \left[ \sum_{e \in P_{vu}} r_e(C_e + c_u) \right]
= \left( E[r_0C_{0v}] + E \left[ \sum_{e \in P_{0v}} \frac{r_e c_e}{2} \right] \right)
+ \left( E[r_0(C_{vu} + c_u)] + E \left[ \sum_{e \in P_{vu}} \frac{r_e c_e}{2} \right] \right)
+ \left( E[r_0(C_{e0v} + c_u)] + E \left[ \sum_{e \in P_{vu}} \frac{r_e c_e}{2} \right] \right)
+ E \left[ \sum_{e \in P_{vu}} r_e(C_{e0v} + c_u) \right]
\]

(6.3)

Now,
\[
\hat{t}_{0v} = r_0C_{0v} + \sum_{e \in P_{0v}} \frac{r_e c_e}{2} + \sum_{e \in P_{0v}} r_e C_{e0v}
\]

Therefore
\[
E[\hat{t}_{0v}] = E[r_0C_{0v}] + E \left[ \sum_{e \in P_{0v}} \frac{r_e c_e}{2} \right] + E \left[ \sum_{e \in P_{0v}} r_e C_{e0v} \right]
\]

(6.4)
And,

\[ \hat{t}_{vu} = r_0(C_{vu} + c_u) + \sum_{e \in P_{vu}} \frac{r_e c_e}{2} + \sum_{e \in P_{vu}} r_e (C_e + c_u) \]

Therefore,

\[ E[\hat{t}_{vu}] = E[r_0(C_{vu} + c_u)] + E\left[ \sum_{e \in P_{vu}} \frac{r_e c_e}{2} \right] + E\left[ \sum_{e \in P_{vu}} r_e (C_e + c_u) \right] \quad (6.5) \]

Substituting the value of \( E[\hat{t}_{0v}] \) from (6.4) and \( E[\hat{t}_{vu}] \) from (6.5) into (6.3), we get,

\[ E[t_u(P)] = E[\hat{t}_{0v}] + E[\hat{t}_{vu}] + E\left[ \sum_{e \in P_{0v}} r_e (C_{vu} + c_u) \right] = E[\hat{t}_{0v}] + E[\hat{t}_{vu}] + E[R_{0v}(C_{vu} + c_u)] \]

\[ \square \]

**Lemma 3.** Let \( T \) be a Steiner tree for \( S \), \( u \in S \) be a leaf in \( T \) and \( v \in P_u \). Then

\[ E[t_u(T)] = E[\hat{t}_{0v}] + E[\hat{t}_{vu} + R_{0v}C(T_v)] \]

**Proof.** Without loss of generality assume \( v = v_l \) for some \( l \) (see Fig. 6.3). According to
lemmas 1 and 2 the following equalities hold.

\[
E[t_u(T)] = E[\hat{t}_u + r_0 \sum_{i=0}^{k} C(T_{(i)}) + \sum_{i=1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)})] \\
= E[\hat{t}_u] + E\left[ r_0 \sum_{i=0}^{k} C(T_{(i)}) \right] + E\left[ \sum_{i=1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)}) \right] \\
= E[\hat{t}_0 + \hat{t}_{vu} + R_{0v}(C_{vu} + c_u)] \\
+ E\left[ r_0 \sum_{i=0}^{l-1} C(T_i) + r_0 \sum_{i=l}^{k} C(T_{(i)}) \right] \\
+ E\left[ \sum_{i=1}^{l} \left[ R(P_i) \left( \sum_{j=1}^{l-1} C(T_{(j)}) + \sum_{j \geq l} C(T_{(j)}) \right) \right] \right] \\
+ E\left[ \sum_{i=l+1}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)}) \right] \\
= E[\hat{t}_{ov} + r_0 \sum_{i=0}^{l-1} C(T_{(i)}) + \sum_{i=1}^{l} R(P_i) \sum_{j=1}^{l-1} C(T_{(j)})] \\
+ E[\hat{t}_{vu} + r_0 \sum_{i=l}^{k} C(T_{(i)}) + \sum_{i=l}^{k} R(P_i) \sum_{j \geq i} C(T_{(j)})] \\
+ E\left[ R_{0v}(C_{vu} + c_u) + \sum_{i=1}^{l} R(P_i) \sum_{j \geq l} C(T_{(j)}) \right] \\
= E[\tilde{t}_0] + E[\tilde{t}_{vu}] + E[R_{0v}C(T_v)]
\]

6.2.1 Grid graphs with \( n \geq 2 \) terminals

Here we consider operation of statistical MAD on grid graphs with \( n \) terminal vertices and estimate its approximation ratio in a special case.

Claim. Given graph \( G \) containing \( n \) terminal vertices. Assume MAD connects terminals in the following order:

\{u_1, u_2, \ldots, u_k, \ldots, u_n\} and denote partially constructed tree spanning set \{u_1, \ldots, u_k\} by
Proof. Tree $T^k$ differs from $T^{k-1}$ by a simple path to vertex $u_k$. Denote this path by $P_{vu_k}$ (see Fig. 6.4). As $v$ was attached at previous steps, it lies on a path to a terminal vertex and thus is root of a subtree containing terminal vertices. Without loss of generality we assume this tree to be non-empty and $v$ to be incident to single edge $(v, w)$ in tree $T^{k-1}$.

Lemma 3 implies the following equalities.

\begin{align}
E[t^*(T^k)] &= E[\hat{t}_{u_k}] + E[t^*(T^{k-1})].
\end{align}

From (6.7),

\begin{align}
E[\hat{t}_{0v}] = E[t_w(T^{k-1})] - E[\hat{t}_{vw}] - E[R_{0v}(C(T') + c_{vw})] - E[(r_0 + r_{vw})C(T')]
\end{align}
substituting $E[\hat{t}_{0v}]$ into (6.6):

$$E[t_{u_k}(T^k)] = E[\hat{t}_{vu_k}] + E[R_{0v}(C_{vu_k} + c_{u_k})] + E[R_{0v}(C(T') + c_{vu_k})] + E[t_w(T^{k-1})] - E[\hat{t}_{vw}] - E[R_{0v}(C(T') + c_{vu_k})] - E[(r_0 + r_{vw})C(T')]$$

$$= E[t_w(T^{k-1})] + E[\hat{t}_{vu_k}] + E[R_{0v}(C_{vu_k} + c_{u_k})] + E[\hat{t}_{0v}] - E[\hat{t}_{0v}]$$

$$- E[\hat{t}_{vw}] - E[(r_0 + r_{vw})C(T')]$$

$$= E[t_w(T^{k-1})] + E[\hat{t}_{vu_k} + R_{0v}(C_{vu_k} + c_{u_k})] + \hat{t}_{0v}$$

$$- E[\hat{t}_{0v} + \hat{t}_{vw} + (r_0 + r_{vw})C(T')]$$

According to Lemma 2 the expression in bold text equals $E[\hat{t}_{u}]$ and hence,

$$E[t_{u_k}(T^k)] \leq E[t_w(T^{k-1})] + E[\hat{t}_{u_k}] \leq E[t^*(T^{k-1})] + E[\hat{t}_{u_k}].$$

So if $u_k$ is a critical terminal in $T^k$ the Claim is proved. Otherwise let $u'$ be a critical terminal in $T^k$ and $v'$ be the last common vertex of paths $P_{0u_k}$ and $P_{0u'}$ in $T^k$.

$$E[t_w(T^k)] = E[t_w(T^{k-1})] + E[(r_0 + R_{0v'})(C_{vu_k} + c_{u_k})]$$

$$< E[t^*(T^{k-1})] + E[\hat{t}_{vu_k} + R_{0v}(C_{vu_k} + c_{u_k})]$$

$$\leq E[t^*(T^{k-1})] + E[\hat{t}_{u}].$$

\[\square\]

**Remark 1.** Let $S$ be the set of terminal vertices and $T$ be a Steiner tree for $S$. The previous claim just described yields the following estimation for critical delay in $T$:

$$E[t^*(T)] \leq E\left[\sum_{u \in S} \hat{t}_u\right].$$

**Theorem 1.** Given $G \in \Gamma_2$ and $S = \{0, 1, \ldots, n\}$. Let $T_{MAD}$ be the tree constructed by
MAD and $T_{opt}$ be an optimal tree spanning $S$. Then

$$\frac{E[t^*(T_{MAD})]}{E[t^*(T_{opt})]} \leq n.$$  

**Proof.** Use Remark 1. Let $u_d$ be the $d$-th terminal attached to $T_{MAD}$ and $T_{opt}(u_1, u_2, \ldots, u_d)$ be an optimal tree spanning set $\{u_1, u_2, \ldots, u_d\}$.

It is obvious that $E[\hat{t}_{u_d}] \leq E[t^*(T_{opt}(u_1, u_2, \ldots, u_d))]$. Hence,

$$E[t^*(T_{MAD})] \leq E \left[ \sum_{d=1}^{n} t^*(T_{opt}(u_1, u_2, \ldots, u_d)) \right] \leq E \left[ \sum_{d=1}^{n} t^*(T_{opt}(u_1, u_2, \ldots, u_n)) \right] \quad (6.9)$$

Now from equation 2.1 and 2.2,

$$t^*_k = r_0 C_0 + \sum_{(i,j) \in P_k(T)} d_{ij}$$

Therefore,

$$E[t^*_k] = E[r_0 C_0] + E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \left( \frac{c_{ij}}{2} + C_j \right) \right]$$

$$= E[r_0] \cdot E[C_0] + Cov[r_0, C_0] + E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right]$$

$$+ Cov \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] + E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} C_j \right]$$

$$+ Cov \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} C_j \right]$$
Therefore,

\[
E \left[ \sum_{d=1}^{n} t^* \left( T_{opt}(u_1, u_2, \ldots, u_n) \right) \right] = n \cdot E[r_0] \cdot E[C_0] + n \cdot Cov[r_0, C_0] 
+ n \cdot E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] 
+ \sum_{d=1}^{n} Cov \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] 
+ n \cdot E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} C_j \right] 
+ \sum_{d=1}^{n} Cov \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} C_j \right] 
\leq n \cdot E[r_0] \cdot E[C_0] + n \cdot Cov[r_0, C_0] 
+ n \cdot E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] 
+ n \cdot Cov_{\max} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] 
+ n \cdot E \left[ \sum_{(i,j) \in P_k(T)} r_{ij} \right] \cdot E \left[ \sum_{(i,j) \in P_k(T)} C_j \right] 
+ n \cdot Cov_{\max} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} C_j \right] 
\leq n \cdot E[t^* \left( T_{opt}(u_1, u_2, \ldots, u_n) \right)] 
\tag{6.10}
\]
where,

$$\text{Cov}_{\text{max}} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right] = \arg \max_{d=1, \ldots, n} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} \frac{c_{ij}}{2} \right]$$

and

$$\text{Cov}_{\text{max}} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} C_j \right] = \arg \max_{d=1, \ldots, n} \text{Cov} \left[ \sum_{(i,j) \in P_k(T)} r_{ij}, \sum_{(i,j) \in P_k(T)} C_j \right]$$

Therefore from 6.9 and 6.10,

$$E[t^*(T_{MAD})] \leq n \cdot E[t^*(T_{opt}(u_1, u_2, \ldots, u_n))]$$

which gives,

$$\frac{E[t^*(T_{MAD})]}{E[t^*(T_{opt})]} \leq n.$$
Bibliography


Applications of Information Technology (EAIT), Kolkata, India, pages 375–378, 2011.


[42] Jiang Hu and Sachin S. Sapatnekar. A timing-constrained simultaneous global route-


