UNIVERSITY OF CALIFORNIA, SAN DIEGO

Interconnect Synthesis

A Thesis submitted in partial satisfaction of the requirements
for the degree
Master of Science

in

Computer Science (Computer Engineering)

by

Sriram Venkatesh

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Ronald L. Graham

2018

The thesis of Sriram Venkatesh is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I thank my advisor, Professor Andrew B. Kahng for his invaluable support and guidance throughout the course of my degree. I thank him for giving me the opportunity to do research and for giving me the exposure to a myriad of interesting problems and ideas. His ideologies about research and life have influenced me strongly. I would also like to thank my thesis committee members, Professors Chung-Kuan Cheng and Ronald L. Graham, for their time to review my research and for their valuable comments. Finally, I would like to thank my former and current labmates and affiliates of the UCSD VLSI CAD Laboratory (Dr. Wei-Ting Chan, Dhanya Ganesh, Kwangsoo Han, Soowan Hong, Minsoo Kim, Hyein Lee, Dr. Jiajia Li, Uday Mallappa, Christopher Moyes, Tushar Shah, Lutong Wang, Bangqi Xu, Ahmed Youssef) for their support and valuable discussions.

The material in this thesis is based on the following publications.

Chapter 2 is in part a reprint of C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and S. Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 10-17.

Chapter 3 is in part a reprint of A. B. Kahng, C. Moyes, S. Venkatesh and L. Wang, "Wot the L: Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 2-9.

Chapter 4 is in part a reprint of the submitted draft of S. Hong, A. B. Kahng, S. Venkatesh and L. Wang, "Layer-Balancing ECO Clock Optimization for Improved Model-Hardware Correlation and Design Closure", *Proc. DAC*, 2018, submitted draft.

My co-authors (Dr. Charles J. Alpert, Dr. Wing-Kai Chow, Kwangsoo Han, Soowan Hong, Professor Andrew B. Kahng, Dr. Zhuo Li, Derong Liu, Christopher Moyes and Lutong Wang, listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

2013	B.E.(Hons), Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani, India
2013-2016	Design Engineer, ARM Holdings, Bangalore, India
2016-2018	Graduate Student Researcher, University of California, San Diego
2018	M. S. in Computer Engineering, University of California, San Diego

PUBLICATIONS

C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and **S. Venkatesh**, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 10-17.

A. B. Kahng, C. Moyes, **S. Venkatesh** and L. Wang, "Wot the L: Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 2-9.

ABSTRACT OF THE THESIS

Interconnect Synthesis

by

Sriram Venkatesh

Master of Science in Computer Science (Computer Engineering)

University of California, San Diego, 2018

Professor Andrew B. Kahng, Chair

In recent times, even small improvements in performance and power are seen as huge wins in digital integrated circuit (IC) design. In advanced technology nodes, design of energy-efficient chips with high yields faces many challenges. Notably, aspects of interconnect design are now among the most significant challenges to obtaining ICs with low power, high performance and high yield. This thesis presents new techniques to (i) improve the construction of interconnects, (ii) improve the estimation of wirelengths of interconnects given a placement, and (iii) improve manufacturing yield by eliminating imbalance of metal layer usage in interconnects used for clock distribution.

This thesis has three main contributions, presented in the three main chapters. First, this

thesis presents two tree construction algorithms for simultaneous improvement of wirelengths and source-to-sink pathlengths of routing trees. Second, this thesis defines a new property of placed signal nets and the corresponding pin locations, and proposes an improved lookup table to accurately estimate wirelengths of these nets. Finally, this thesis presents a technique to improve yield of ICs by layer-balancing the clock paths of each launch-capture register-pair in the design.

Chapter 1

Introduction

In recent years, it has become increasingly difficult for IC designers to achieve significant improvements in performance, power and area for multiple generations of products. Synthesis of interconnects (signal and clock nets) in advanced VLSI technology nodes has been a particularly challenging factor. In addition, the latest nodes and the corresponding foundry libraries have shown multiple model-hardware miscorrelation (MHM) issues, which result in low yields. Thus, a great deal of recent research has focused on improved interconnect construction and improved correlation between models and silicon. This thesis describes several new techniques that tackle the above-mentioned challenges.

Chapter 2 of this thesis describes two algorithms, PD-II and Detour-aware Steinerization, for iterative improvement of spanning and Steiner trees. These algorithms build on the widely-used Prim-Dijkstra construction [1] to simultaneously improve the wirelengths (WLs) and source-to-sink pathlengths (PLs) of routing trees. This translates to lower-power interconnects with smaller delays.

Chapter 3 of this thesis defines a new characteristic of pointsets (i.e., the collection of placed pins of a net) called *L-ness*, and provides an accurate lookup table for WL estimation of nets. The L-ness characteristic and the improved lookup-based estimation enable better, more

relevant evaluation of tree-construction heuristics; they also enable placers to estimate WL faster and more accurately, leading to better routability.

Chapter 4 of this thesis aims to alleviate the model-hardware miscorrelation (MHM) issue seen in the back-end-of-line stack. Different metal layers are affected by varying amounts of variation, leading to hold violations and reduced yields. This work describes a method to layer-balance the routes of clock nets to each launch-capture pair in a design, thereby making the design more robust to variation. This work also explores alternative strategies which feed back MHM-induced hold violation information to earlier stages of the physical design flow to guardband against MHM issues.

Chapter 2

Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees

This chapter presents two new algorithms which iteratively improve the widely-used Prim-Dijkstra algorithm and obtain trees with lower wirelengths and pathlengths.

2.1 Introduction

In recent technology nodes, wire capacitance has become a key challenge to design closure, and this problem worsens with each successive technology node [2]. Today, a digital implementation flow cannot simply use minimum wirelength (WL) trees for routing estimates in placement and optimization, nor can they be used for timing-driven routing of critical nets. Routing an advanced-node design with minimum WL trees leads to untenable source-to-sink distances, yielding high delays for many nets. One cannot afford to use a shortest path tree also, which achieves optimal source-to-sink pathlength (PL) for each sink, due to increased WL which worsens dynamic power and routing congestion. For these reasons, timing-driven tree construction that trades off WL and PL becomes a critical technology for modern designs.

The Prim-Dijkstra (*PD*) [1] construction is generally regarded as the best available spanning tree algorithm for achieving this tradeoff and has the additional advantage of simplicity [4].[1] This algorithm has been used for over 20 years to construct high-performance routing trees in leading semiconductor design methodologies and electronic design automation (EDA) tools, as can be seen by related patents assigned to IBM, Synopsys, Cadence and other entities ([25] [26] [27] [28] [29] [30] [31] [32]). Further, the authors of [9] performed an evaluation that compared *PD* to other spanning tree constructions such as BRBC [10], KRY [11], etc. in 2006; they concluded that *PD* obtained the best tradeoff between WL and PL. That paper [9] argues that the *PD* wirelength cost is minimal enough to be practically free. However, this claim is now suspect because today's designs are significantly more power-sensitive than a decade ago: now, a 1% reduction in power is viewed as a big win for today's design teams performing physical implementation. Consequently, even a small WL savings with similar timing can have a high impact on value. A deeper discussion of prior art is given in Section 2.2.

The *PD* construction balances between WL and source-to-sink PL by blending the Prim and Dijkstra spanning tree constructions [5][6] via a weighting factor, α. When $\alpha = 0$, *PD* is identical to Prim's algorithm [5] and constructs a minimum spanning tree (MST). As α increases, *PD* constructs a tree with higher WL but better PL; when $\alpha = 1$, *PD* is identical to Dijkstra's algorithm [6] and constructs a shortest-path tree (SPT). *PD* begins with a tree consisting just the source node, then iteratively adds the edge e_{ij} that minimizes $d_{ij} + \alpha \cdot l_i$, where node v_i is in the current tree and v_j is not in the current tree, d_{ij} is the distance between nodes v_i and v_j, and l_i is the PL from the source to v_i in the current tree.

One problem with the *PD* algorithm is that it greedily adds edges, which becomes problematic with higher fanout trees. Once an edge is added, it is never removed from the final solution, making it impossible for *PD* to recover from a potentially poor choice. This can lead

[1] For global routing, spanning trees are often preferred to Steiner trees since global routing commonly decomposes multi-fanout nets into two-pin nets. A spanning tree provides the router with an obvious decomposition. However, Steiner trees are not well-suited for this because the Steiner points become unnecessary constraints that restrict the freedom of the router to resolve congestion.

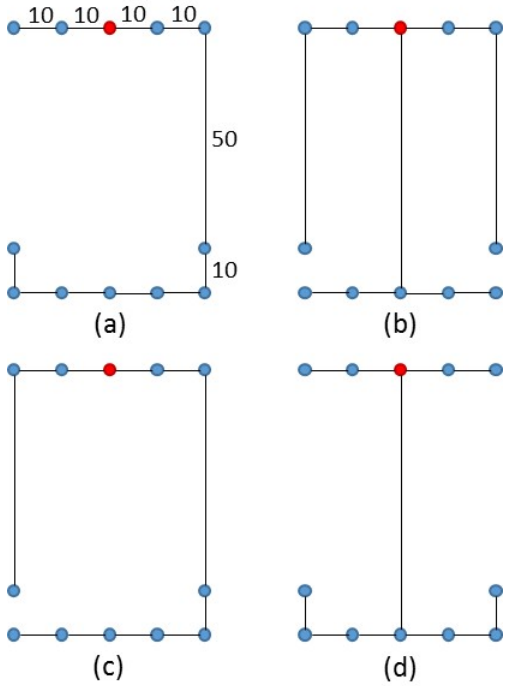


Figure 2.1: An example instance showing suboptimality of *PD*. The red node is the source. (a) shows the MST obtained when $\alpha = 0.2$, (b) shows the SPT obtained when $\alpha = 0.8$, and (c) shows the solution when $\alpha = 0.4$. The tradeoff in (c) is clearly suboptimal in both WL and PL, as compared to (d).

to trees that are suboptimal in both WL and maximum PL. Figure 2.1 shows such an example. When α is small (0.2), *PD* obtains the MST solution (a) with $WL = 150$ and $PL = 130$. When α is large (0.8), *PD* obtains the SPT solution (b) with $WL = 240$ and $PL = 80$. However, when $\alpha = 0.4$, *PD* obtains the solution (c) with suboptimal values of both WL and PL ($WL = 190$ and $PL = 120$). This solution (c) is inferior for both objectives than the solution (d) with $WL = 160$ and $PL = 90$. Thus, $\alpha = 0.4$ generates a poor solution for both WL and PL.

This chapter makes the following contributions:

- To fix the shortcomings in *PD*, one needs to directly optimize PL in the tree construction, which requires a new problem formulation. We propose incorporating total *detour cost*, the amount of suboptimal PL for each node, into the tradeoff. The correct formulation of the objective is paramount since it drives any optimization which follows. This work seeks

to optimize the detour cost to all sinks instead of just the worst one, as proposed in prior works [36].

- Next, a new algorithm, which we call *PD-II*, is proposed. The idea is to recover the tree, that has any edges poorly chosen by *PD*, using an iterative improvement method according to the proposed objective function.

- Since Steiner trees are most commonly useful for timing prediction and physical synthesis, an algorithm for converting balanced spanning trees into balanced Steiner trees is proposed. The resulting *Detour-Aware Steinerization* (DAS) algorithm optimizes both WL and detour cost to achieve a tree with similar properties to those obtained by the *PD-II* spanning tree algorithm.

- Finally, three sets of experiments are presented. The first shows that *PD-II* is able to meaningfully shift the Pareto curve obtained by the *PD* algorithm, obtaining up to 18% improvement in PL for the same WL. The second experiment demonstrates the value of the *DAS* algorithm versus more standard Steinerization methods. The third experiment shows that the proposed Steiner construction outperforms those of SALT [36] for medium- and high-fanout nets, a recent state-of-the-art academic tool, achieving up to 36.48% PL improvement for similar WL.

The following sections of this chapter are organized as follows. Section 2.2 briefly reviews related works in the areas of spanning and Steiner tree constructions. Section 2.3 presents the proposed problem formulation that incorporates both WL and detour cost. Section 2.4 presents the *PD-II* heuristic for spanning tree optimization, and Section 2.5 presents the *DAS* heuristic for Steiner tree optimization. Section 2.6 reports our experimental results, and Section 2.7 concludes the chapter.

2.2 Previous Work

There is a rich history on spanning tree and Steiner tree constructions. Many focus on minimizing WL or minimizing longest PL. (Our present work studies constructions that consider both metrics.)

Spanning Tree Constructions. As discussed previously, the Prim and Dijkstra constructions achieve optimal WL and PL, respectively. Spanning tree algorithms that optimize both are called *shallow-light* constructions [10] [12]; they seek to optimize WL and PL simultaneously to within constant factors of optimal. Shallow-light constructions have in many ways been a "holy grail" in VLSI CAD literature for over 25 years. The *PD* algorithm is "shallow-light in practice", but no such formal property has ever been established [1]. Cong et al. [13] give the Bounded Prim (BPRIM) extension of Prim's MST algorithm [5], which produces trees with low average WL and bounded PLs, but possibly unbounded WL. The BRBC algorithm of Cong et al. [10] produces a tree that has WL no greater than $1 + 2/\varepsilon$ times that of an MST, and radius no greater than $1 + \varepsilon$ times that of an SPT. Khuller et al. [12] contemporaneously develop a method similar to BRBC.

Minimum WL Heuristic Steiner Tree Constructions. Several works describe heuristic algorithms for Steiner tree constructions with minimized WL. Kahng and Robins [15] give the iterated 1-Steiner (I1-S) heuristic which greedily constructs a Steiner tree through iterative Steiner point insertion, resulting in trees with close to optimal WL. Ho et al. [7] propose an algorithm (HVW) to optimally edge-overlap *separable* MSTs to obtain Steiner trees, while Borah et al. [16] present a greedy heuristic (BOI) to convert spanning trees to RSMTs with performance similar to the I1-S heuristic. Chu and Wong [33] propose FLUTE which uses pre-computed look-up tables for Steiner construction to find solutions more efficiently than the prior art.

Rectilinear Steiner Arborescence (RSA) Constructions. The NP-complete [17] *rectilinear Steiner arborescence* (RSA) problem seeks to find a minimum-WL tree in the Manhattan plane that achieves optimal PL for every sink. Rao et al. [3] present the first heuristic for the RSA

problem. Cong et al. [18] address the construction of RSAs with the A-tree algorithm, while Kahng and Robins [19] give a simple adaptation of their Iterated 1-Steiner algorithm to the RSA problem.

Steiner Constructions that Optimize WL and PL. Recently, Scheifele [35] has proposed a method to construct Steiner trees for which Elmore delays are bounded. Given an RSMT solution (i.e., FLUTE), [35] iteratively finds the vertex that breaks its ε-based metric, and reroutes the vertex to the source via a shortest path, which indirectly balances between RSMT and RSA. On the other hand, Elkin and Solomon [34] propose a more direct shallow-light Steiner tree construction method (ES). The main idea is to identify breakpoints and reconnect those breakpoints to the root directly by a Steiner SPT so that there is no detour from the root to the breakpoints. The authors of [34] build a Hamiltonian path and check the accumulated distance along the Hamiltonian path to find proper breakpoints, such that the final Steiner tree meets the given shallowness and lightness criteria. Recently, Chen et al. [36] present SALT, which further improves the ES method [34]. The key contributions are (i) tighter criteria to identify breakpoints, and (ii) using an MST instead of a Hamiltonian path. With some post-processing such as L-shape flipping, the method shows superior tradeoffs between pathlength and wirelength compared to any state-of-the-art spanning/Steiner tree construction methods. Comparisons to the method of [36] are included in Section 2.6 below.

2.3 Problem Formulation

A *signal net* $V = \{v_0, v_1, ..., v_{n-1}\}$ is a set of n terminals, with v_0 as the *source* and the remaining terminals as *sinks*. We define the underlying routing graph to be a connected weighted graph $G = (V, E)$, where each edge $e_{ij} \in E$ has a cost d_{ij}. We are concerned with the case where G is a complete graph with each e_{ij} having cost equal to the Manhattan distance d_{ij}. A *routing*

Table 2.1: Notation

Notation	Meaning
V	signal net, $V = \{v_0, v_1, ... v_{n-1}\}$ having $n-1$ sinks
G	routing graph in the spanning tree context
T	routing tree, which is a spanning subgraph of G
v_0	source node of the signal net V, which is the root of T
e_{ij}	edge from node v_i to v_j
$par(v_i)$	parent node of v_i
l_j	cost of the unique v_0 to v_j path in a tree, $v_0, v_j \in T$
d_{ij}	cost of the edge e_{ij}
m_{ij}	Manhattan distance from node v_i to v_j
W_T	total wirelength of a tree
Q_i	detour cost of node v_i, $Q_i = l_i - m_{i0}$
Q_T	detour cost of a tree, $= \sum_{n-1}(Q_i)$
C	weighted cost of a tree, $= \alpha \cdot Q_T + (1-\alpha) \cdot W_T$
$\Delta C_{e,e'}$	the change in the weighted cost that results from removing edge e and adding e', used in *PD-II*
α	weighting factor used in *PD* and *PD-II*
D	flipping distance used in *PD-II*
P_T	sum of pathlengths of a tree, $= \sum_{n-1}(l_j)$

tree $T = (V, E')$ is a spanning subgraph of G with $|E'| = n-1$.[2] Given a routing tree T, the cost of the unique $v_0 - v_i$ path in T is l_i, the *radius* of T is $r(T) = max_{1 \le i \le n-1} l_i$, and the *wirelength* (WL) of T is $W_T = \sum_{e_{ij} \in T} d_{ij}$. All notations used in our work are listed in Table 2.1.

Initially, the tree consists only of v_0. The *PD* algorithm iteratively adds edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

$$(\alpha \cdot l_j) + d_{ij} \; s.t. \; v_j \in T, \; v_i \in V - T \tag{2.1}$$

The *PD* algorithm can result in trees with either large WL or PL, as shown in Figure 2.1. To alleviate this issue, conventional *shallow-light* tree constructions [10] [13] [36] focus on

[2] Our use of G and T pertains to the spanning tree context. In the rectilinear Steiner tree context, the underlying routing graph would be the Hanan grid [20], and a Steiner routing tree would be a spanning tree over $\{V \cup S\}$, where S is a set of Steiner points taken from the Hanan grid. For simplicity, as long as meanings are obvious, we will use terms from the spanning tree context in the Steiner tree context as well.

bounding the *shallowness* and *lightness* to optimize the tree cost. Lightness η means that the WL of a tree is at most η times of the MST WL. A tree has shallowness ζ if PL to each sink in the tree is at most ζ times the source-to-sink Manhattan distance (MD). However, shallowness alone does not adequately represent the quality of a routing tree. Figure 2.2 shows two examples that have the same shallowness and lightness. It is clear that Figure 2.2(b) is preferable to Figure 2.2(a) since the left sinks have shorter PLs, but shallowness does not capture the difference.

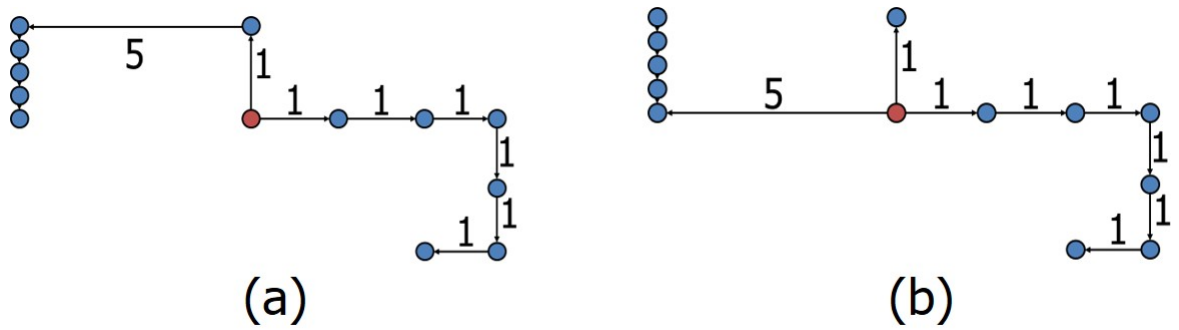


Figure 2.2: Two routing trees that have the same lightness and shallowness.

With the above in mind, we define a new *detour cost* metric as follows. Detour cost Q_i of a sink v_i is the difference between PL from v_0 to v_i in T and the Manhattan distance from v_0 to v_i. The detour cost of the tree T, denoted by Q_T, is the sum of the detour cost values of all the sinks in the tree, i.e., $Q_T = \sum_{1 \leq i \leq n-1} Q_i$. Since *PD* iteratively adds edges and nodes to the growing tree, if a sink v_j close to the source incurs high detour, then all downstream sinks (descendants of v_j) will also have high detour and hence long PL. We therefore propose the following formulation to capture the problem of simultaneously reducing WL and detour cost of a spanning tree:

Simultaneous WL and Detour Cost Reduction (SWDCR) Problem. Given a spanning tree $T = (V, E)$, minimize the weighted sum of WL and detour cost of the tree.

$$\text{Minimize } \alpha \cdot \sum Q_i + (1 - \alpha) \cdot W_T \tag{2.2}$$

where $0 \leq \alpha \leq 1$. We present a heuristic algorithm *PD-II* in Section 2.4 for tackling the SWDCR problem.

Once the spanning tree construction is converted into a Steiner tree, there is a change in the tree topology. We propose and address the following formulation to further optimize the detour cost of a Steiner tree:

Detour Cost Reduction in Steiner Trees (DCRST) Problem. Given a Steiner tree, minimize the tree detour cost.

$$\text{Minimize } Q_T \tag{2.3}$$

$$\text{s.t. } W_{T,new} \leq W_{T,init} \tag{2.4}$$

$$Q_{T,init} \geq Q_{T,new} \tag{2.5}$$

To address the DCRST problem, we present our algorithm *DAS* below in Section 2.5.

2.4 The *PD-II* Spanning Tree Construction

This section presents the *PD-II* algorithm that performs iterative edge-swapping which simultaneously improves the detour cost and WL. The key idea of the *PD-II* algorithm is to start with a spanning tree and swap edges to improve the tradeoff between detour cost and WL. The algorithm can take any spanning tree as input, but it makes sense to start with the *PD* solution since it should already be relatively strong for both objectives. We note that while *PD* can be quite slow for higher-fanout nets, it can be sped up significantly by using a sparsified nearest-neighbor graph instead of the complete graph.

We initially populate the *neighbors* of each node using the following method. We say that v_i is a *neighbor* of v_j if the smallest bounding box containing v_i and v_j contains no other nodes. The worst-case number of neighbor nodes for each node is $\Theta(n)$. For example, every red point in

Figure 2.3 is a neighbor of every green point, and vice versa. However, Naamad et al. [23] show that the expected number of maximal empty boxes amidst n random points in a plane is bounded above by $O(n\ log\ n)$, so it is reasonable to expect the average number of neighbors per node to be $O(log\ n)$.

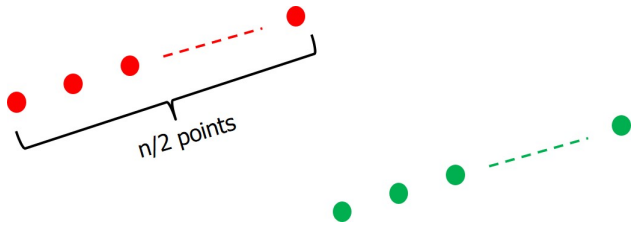


Figure 2.3: Example showing $\Theta(n^2)$ asymptotic worst-case complexity of the number of *neighbor* relationships. Each green node is a neighbor to each red node.

Analysis of random placements of net sinks show this to be true. The number of neighbors for 100K random point sets of size 16, 32, and 64 yields an average number of neighbors per node of 6.3, 8.7 and 11.3, respectively. Real placements should generally have even fewer neighbors, since cells tend to align horizontally or vertically. For the testcases described in Section 2.6.1, the average number of neighbors is 2.58, 4.27, 6.15 and 8.24 for small, medium, large and huge nets, respectively. Hence, in practice, runtime complexity of iterating through the neighbors of a node has logarithmic complexity.

An $O(n\ log\ n)$ runtime complexity can be obtained for *PD* using a binary heap implementation and an adaptation of Scheffer's MST code [21][22]. Since *PD* solutions are generally good, though sometimes suboptimal, it makes sense to post-process the *PD* solution to obtain a better one. The key technique for *PD-II* is *edge flipping*, whereby one edge is removed from the original tree and replaced with a new edge. Figure 2.4(a) shows an example tree, represented as a DAG, representing a topological ordering starting at the source. Figure 2.4(b) shows an example transform in which one edge is removed and replaced with a new red edge, thereby obtaining a different tree. Note that one of the directed edges in the new (rooted) tree is reversed from its previous orientation in order to maintain a well-formed rooted tree. This approach recalls

the iterative improvement operation used in BOI [16], but the application of *flipping* is more restricted to focus on WL vs. PL improvements.

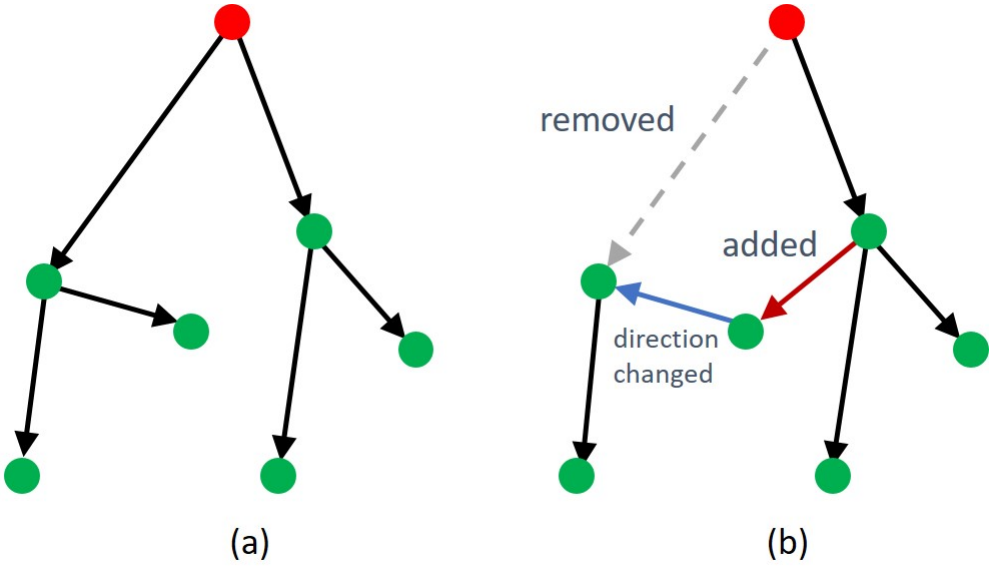


Figure 2.4: Illustration of *PD-II* edge *flipping*.

For each edge pair, we define the *flip cost* as the cost associated with edge flipping, i.e., the cost of removing edge e_{ij} and adding edge $e_{i'j'}$. *Flip cost* $\Delta C_{e,e'} = \alpha \cdot (Q_{T_{i'j'}} - Q_{T_{ij}}) + (1-\alpha) \cdot (d_{i'j'} - d_{ij})$, where α is a weighting factor;[3] $Q_{T_{ij}}$ and $Q_{T_{i'j'}}$ are the detour costs of the trees before and after edge flipping, respectively; and d_{ij} and $d_{i'j'}$ are the lengths of edges being removed and inserted, respectively.

Pseudocode for *PD-II*, Algorithm 1 is given below. Essentially, *PD-II* takes an input tree and searches for edge flips that improve flip cost.[4] If the flip cost improves, the swap is taken. Considering all pairs of possible swaps could be expensive, so we define the *flipping distance* D to be equal to the number of edges in the DAG that require a change in direction to preserve topological ordering, i.e., rooted orientation. For the swap in Figure 2.4, $D = 1$. In practice,

[3]The parameter α can be determined by the timing constraints. If a net is critical, a higher value of α can be used to achieve lower delays, but if arcs through the net have positive slacks, α can be small to save wirelength. Hence, α allows topology optimization and can be chosen to best satisfy the design specifications on a per-net basis.

[4]Flipping cannot be added into the original PD tradeoff objective since the flip cost objective cannot be correctly computed until an entire tree is constructed. Hence, we propose PD-II as a post-processing algorithm which improves a given spanning tree.

using $D > 1$ has little benefit (but more runtime) compared to $D = 1$, so we use $D = 1$ for all experiments.

Algorithm 1 Algorithm *PD-II*

Input: Spanning tree $T_{in} = (V, E_{in})$, with $E_{in} \subseteq E$
Output: Spanning tree $T_{out} = (V, E_{out})$, with $E_{out} \subseteq E$

```
1:  Initialize T_out ← T_in
2:  repeat
3:     Initialize largest detour cost reduction, ΔC_best ← 0
4:     for all e ∈ E_out do
5:        E_e ← candidateEdges(e, D)
6:        for all e' ∈ E_e do
7:           ΔC_{e,e'} ← flipCost(e, e')
8:           if ΔC_{e,e'} < ΔC_best then
9:              ΔC_best ← C_{e,e'}
10:             e_best ← e ; e'_best ← e'
11:          end if
12:       end for
13:    end for
14:    if ΔC_best < 0 then
15:       Remove e_best, insert e'_best and change direction of associate edges
16:    end if
17: until ΔC_best == 0
```

Line 3 of Algorithm 1 initializes the best flip cost to zero. Line 5 computes the set of candidate edges E_e that can be flipped with edge e, as restricted by the flipping distance D. For each candidate edge $e' \in E_e$, we calculate the flip cost for the edge pair (e, e') and find the edge pair (e_{best},e'_{best}) with lowest flip cost in Lines 6-12. These edges are swapped if the lowest flip cost is less than zero (Lines 14-16). The algorithm continues until no more flip-cost improvement is obtained (Line 17).

The number of candidates for edge flipping can be very large when D is unbounded. The worst-case number of edges is $(n/2)^2$, giving Algorithm *PD-II* a worst-case time complexity of $O(n^3)$, where n is the number of sinks. However, with the distance restriction, the complexity reduces to $O(D \cdot n^2)$, and in practice it converges rapidly. To show this, we take two large blocks from an industrial design and run a production Steiner package on an Intel Xeon 2.7GHz machine

(CPU E5-2680), using RHEL5. The first design has 1.9 million datapath nets, and the total runtime for the Steiner package which uses *PD* for its spanning tree construction requires 59.3 seconds. Adding *PD-II* to the Steiner package increases the runtime to 62.7 seconds, for a net penalty of 3.4 seconds. The second design with 4.0M datapath nets requires 124.0 seconds for running the default Steiner package. Adding *PD-II* to the Steiner package increases the runtime from 124.0 seconds to 125.8 seconds, for a net penalty of 1.8 seconds. Consequently, the runtime cost of using *PD-II* is negligible, averaging less than one additional second of runtime per million nets.

2.5 The Detour-Aware Steinerization Algorithm (*DAS*)

For global routing, spanning tree constructions such as *PD-II* are sometimes preferred to Steiner trees since global routing commonly decomposes multi-pin nets into two-pin nets. However, for timing estimation, congestion prediction, or general physical synthesis optimization, a Steiner tree is required since spanning trees will have too much WL. The previous spanning tree formulation can easily be extended to Steiner trees; the definitions of WL and PL do not change. However, since finding the minimum wirelength Steiner is NP-complete, FLUTE WL is used as a proxy for minimum Steiner tree cost.

To transform a spanning tree into a Steiner tree, the linear-time algorithm of [7] is invoked. It maximizes edge-overlaps in the spanning tree by creating a Steiner node. We call the algorithm *HVW* after the algorithm's creators: Ho, Vijayan, and Wong. *HVW* traverses the tree from the leafs and iteratively maximize overlaps with the currently visited edge and its immediate children edges. However, this basic construction can be inefficient both in terms of WL and PL. Hence a new Steinerization algorithm, called *DAS* for Detour-Aware Steinerization is proposed below.

DAS has two phases of optimization. The first phase seeks to reduce WL while minimizing the detour cost penalty (Lines 1-14). This phase does a bottom-up tree traversal and makes edge

Algorithm 2 The Detour-Aware Steinerization Algorithm (*DAS*)

Input: Steiner tree $T_{St,in}$
Output: Improved Steiner tree $T_{St,out}$

1: //First phase: wire recovery at the cost of small additional PL
2: $p_T^{max} \leftarrow$ maximum PL of the Steiner tree
3: Do Breadth-First Search (BFS) from the leaf node
4: **for all** v_i **do**
5: $v_j \leftarrow par(v_i)$; $d_{ji} \leftarrow$ edge length to v_i;
6: $o_{ji} \leftarrow$ overlap length with other edges to v_i
7: $\Delta d_{ji} \leftarrow d_{ji} - o_{ji}$
8: **for all** v_k in {all *neighbors* of v_i} **do**
9: $\Delta d_{ki} \leftarrow d_{ki} - o_{ji}$; $p_i \leftarrow$ PL to node v_i
10: **if** ($\Delta d_{ki} < \Delta d_{ji}$ && ($p_i \leq 0.5 \cdot p_T^{max}$) **then**
11: Disconnect v_i to v_j and reconnect v_i to v_k
12: **end if**
13: **end for**
14: **end for**
15: //Second phase: detour cost reduction with bounded WL
16: $W_{T,init} \leftarrow$ Init. Steiner tree WL; $Q_{T,init} \leftarrow$ Init. Steiner tree detour cost
17: Do Breadth-First Search (BFS) from the source node
18: **for all** v_i **do**
19: $v_j \leftarrow par(v_i)$; $d_{ji} \leftarrow$ Initial edge length to v_i
20: **for all** v_k in {all *neighbors* of v_i} **do**
21: $e_{ki} \leftarrow$ Edge from v_k to v_i; $d_{ki} \leftarrow$ Edge length from v_k to v_i
22: $W_{T,new} \leftarrow W_{T,init} + d_{ki} - d_{ji}$
23: $Q_{T,new} \leftarrow$ detour cost tree with edge e_{ki}
24: **if** ($W_{T,new} \leq W_{T,init}$) && ($Q_{T,new} < Q_{T,init}$) **then**
25: Disconnect v_i to v_j and reconnect v_i to v_k
26: $W_{T,init} \leftarrow W_{T,new}$; $Q_{T,init} \leftarrow Q_{T,new}$
27: **end if**
28: **end for**
29: **end for**

swaps which reduce WL. For each edge e_{ji} in the Steiner tree, the edge e_{ji} is removed from the tree and replaced with e_{ki} where v_k is a nearest neighbor of v_i if the WL improves and the PL is not overly degraded. (i.e., $p_i \leq 0.5 \cdot p_T^{max}$).

After the first phase, since PL (or detour cost) is not targeted, there still may be room to improve for that dimension. Hence, a second phase (Lines 15-29) seeks to optimize detour cost Q_T without degrading WL. This second phase performs a *top-down* tree traversal to minimize Q_T. This is because the detour cost Q_i to a node v_i affects not only the PL to the node, but also the PL to the downstream nodes of v_i. Thus, more opportunity for large Q_T reductions exists in the edges near the source v_0. For each edge e_{ji} in the Steiner tree, the edge e_{ji} is removed and replaced

with e_{ki}, where v_k is the possible parent among the nearest neighbors of v_i, to reduce Q_T without degrading WL. This process is repeated for all the nodes in the tree with non-zero detour cost.

Algorithm *DAS* has a worst-case time complexity of $O(n^2)$. However, with the sparsified nearest neighbor graph implementation described in Section 2.4, *DAS* runs much faster than $O(n^2)$ and is closer to $O(n\ log\ n)$ in practice. For 100K nets, *DAS* runs in 0.86 seconds for 16-terminal nets, 1.71 seconds for 32-terminal nets and 4.83 seconds for 64-terminal nets.

2.6 Experimental Setup and Results

2.6.1 Experimental Setup

The algorithms described above are implemented in C++. The following experiments are performed on a 2.7 GHz Intel Xeon server with 8 threads. Testcases are generated from the DAC-2012 contest benchmarks [37], with pin locations for each net are extracted from ePlace placement solutions [38]. Since finding a solution with optimal WL and PL is trivial for two- and three-pin nets, our experiments focus on nets with fanout larger than two. The 749K total nets are divided into four groups (small, medium, large, huge) by their terminal count, as shown in Table 2.2.

Table 2.2: Net statistics for Superblue benchmark designs.

	small	medium	large	huge
$\|V\|$	$4-7$	$8-15$	$16-31$	$32+$
#nets	533029	128463	46486	20853

While our algorithms optimize Q_T, Q_T itself does not adequately capture the quality of the tree. Instead, results are reported based on two normalized metrics, W_{Tnorm} (normalized WL) and P_{Tnorm} (normalized PL). W_{Tnorm} is defined as the ratio of the tree WL to the MST WL for spanning trees. P_{Tnorm} is defined as the ratio of sum of PLs of each node in the tree to the sum of Manhattan distances from source to each node. The optimal value any tree could have for either metric is one, which makes the corresponding Pareto curve more intuitive.

2.6.2 Experiment I - Spanning Tree Results

In the following results, *PD* and *PD-II* refer respectively to the *spanning trees* constructed using the *PD* and *PD-II* algorithms. Figure 2.5 shows normalized WL and PL tradeoff curves for *PD* and *PD-II*, for the 46486 large nets. Each point in the curves represents the average (W_{Tnorm}, P_{Tnorm}) over all the nets for a particular value of α. We sweep α from 0.05 to 0.95, in steps of 0.05, to obtain both the *PD* and *PD-II* curves. We observe that the blue *PD-II* Pareto curve is clearly better than the red *PD* curve.

The Pareto curve makes the improvement trend clear, but makes it difficult to measure the degree of improvement of *PD-II*. To compare the two algorithms more robustly, we analyze the results in the following way; (1) select different percentages of permissible WL degradation with respect to MST WL (i.e., WL thresholds $= 1\%$, 2%, 4%, 7%, 10% and 15%), and (2) for each net, find the minimum P_{Tnorm} solution that meets the WL threshold across all solutions with different α. The results are averaged across all the nets and summarized in Table 2.3. Each entry in the table corresponds to the normalized PL P_{Tnorm}. To find the percentage improvement, one is subtracted from each value, since 1.0 is a lower bound. For example, a reduction from 1.15 to 1.12 results in an improvement of 20%, i.e., $(1-(1.12-1.0)/(1.15-1.0)) \cdot 100\%$.
We observe the following.

- *PD-II* gives better results than *PD* for all classes of nets. This makes sense since it strictly improves upon an existing *PD* solution.

- Small nets obtain relatively small improvement, ranging from 0.26% to 1.63%; however, huge nets show significant improvements, ranging from 4.91% to 18.87%. Trends for medium and large nets lie in between. This is because the detour cost is close to optimal for smaller nets, but is much larger for bigger nets. For example, with a 1% WL threshold, the average normalized PL for *PD-II* is 1.097 for small nets but 1.376 for large nets.

- When the WL threshold is tight (such as 1% or 2%), the improvement of *PD-II* is much

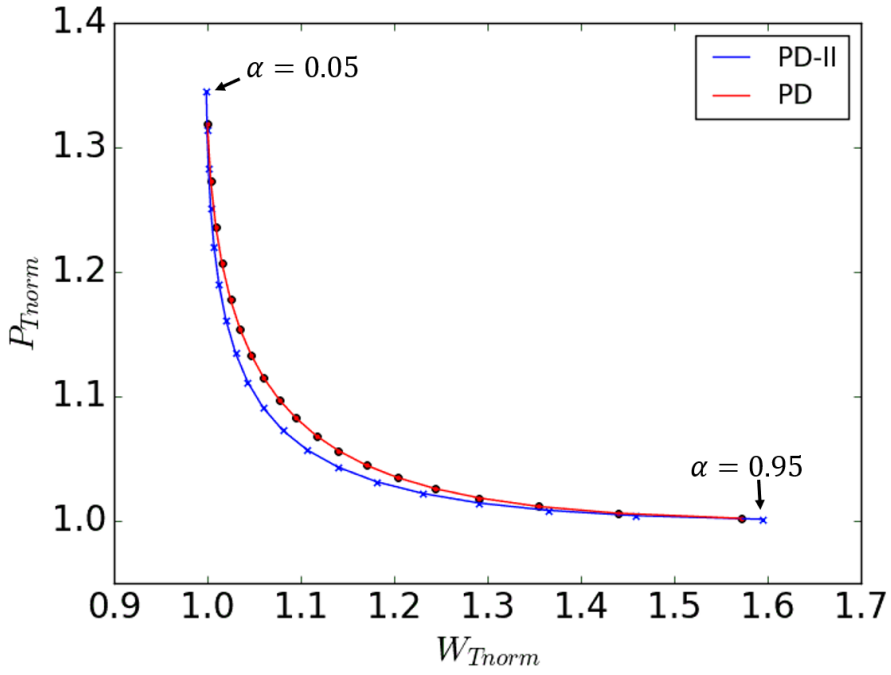


Figure 2.5: WL and PL tradeoff for various α.

smaller as compared to looser constraints of 10% or 15%. This makes sense because a looser constraint gives the algorithms more freedom to reduce PL. A threshold of 1% means the topology cannot deviate much from the minimum-length spanning tree.

2.6.3 Experiment II - Steiner Tree Results

Our next experiments compare (*PD* + *HVW* + *DAS*) and a baseline flow (PD + *HVW*) to show the value of *DAS*. *HVW* refers to the Steiner tree obtained after performing edge-overlapping as described by Ho et al. [7], and *DAS* refers to the Steiner tree after applying *DAS* algorithm to the *HVW* tree. Figure 2.6 shows the normalized WL and PL tradeoff comparison for the two flows for the set of large nets. Steiner tree W_{Tnorm} is defined as the ratio of total WL of the tree to the FLUTE WL[5] [33] and P_{Tnorm} is defined as the ratio of sum of PLs of all sinks in the tree to

[5]FLUTE constructs optimal RSMTs for nets with terminal sizes up to 9, and near-optimal RSMTs for nets with higher terminal counts.

Table 2.3: Comparisons of the best P_{Tnorm} for *PD* and *PD-II* across different WL thresholds.

$\|V\|$	Method	WL threshold					
		1%	2%	4%	7%	10%	15%
small	*PD*	1.0972	1.0927	1.0819	1.0680	1.0569	1.0427
	PD-II	1.0970	1.0923	1.0812	1.0672	1.0561	1.0420
	Imp. (%)	**0.26**	**0.42**	**0.78**	**1.15**	**1.36**	**1.63**
med.	*PD*	1.1888	1.1746	1.1483	1.1189	1.0974	1.0723
	PD-II	1.1870	1.1706	1.1423	1.1122	1.0909	1.0668
	Imp. (%)	**0.93**	**2.33**	**4.07**	**5.66**	**6.62**	**7.68**
large	*PD*	1.2981	1.2698	1.2216	1.1723	1.1390	1.1006
	PD-II	1.2895	1.2545	1.2025	1.1533	1.1219	1.0870
	Imp. (%)	**2.89**	**5.66**	**8.64**	**11.00**	**12.32**	**13.52**
huge	*PD*	1.3952	1.3550	1.2873	1.2210	1.1777	1.1302
	PD-II	1.3758	1.3238	1.2526	1.1876	1.1488	1.1056
	Imp. (%)	**4.91**	**8.79**	**12.06**	**15.14**	**16.27**	**18.87**

the sum of source-to-sink Manhattan distances. Each point in the curve represents the average (W_{Tnorm}, P_{Tnorm}) over all nets, for a particular value of α. It is clear that *DAS* adds significant value to the Steiner construction, pushing its Pareto curve further left and down compared to the one from the baseline.

Similarly to Table 2.3, Table 2.4 shows normalized PL across a range of permissible WL degradations for *HVW* versus *HVW+DAS*. We observe the following:

- DAS always obtains better results than HVW. Again, this makes sense since DAS starts with an HVW solution and further refines it to improve both WL and PL.
- Improvements for DAS can be quite significant, ranging from 8.36% to 83.67%.
- DAS improves results more significantly for smaller fanout nets than for larger ones. This may suggest there is still further room for improvement in Steinerization.
- Larger WL thresholds correspond to larger normalized PL improvements, which again is likely due to more freedom for the algorithm to find a solution that reduces detour cost.

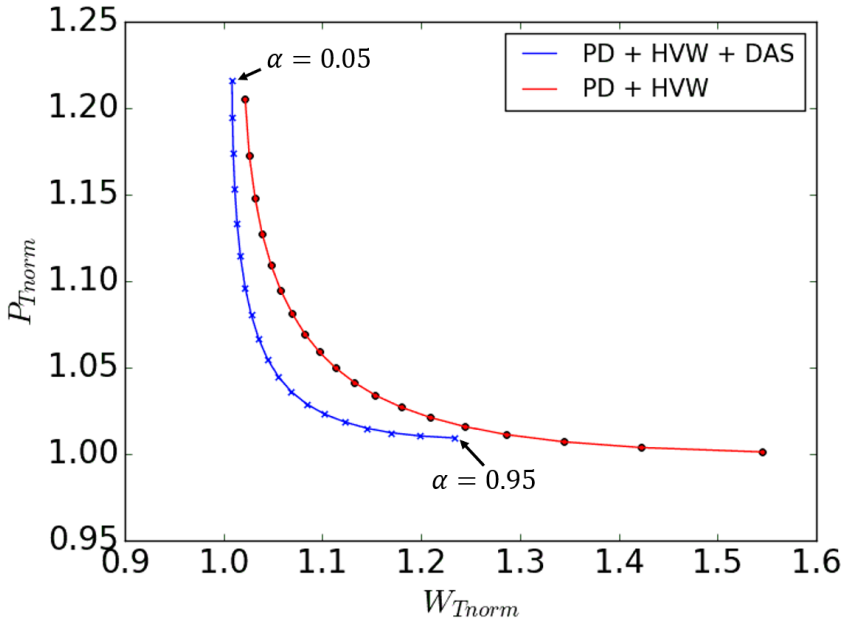


Figure 2.6: WL and PL tradeoff for Steiner tree constructions.

2.6.4 Experiment III - Comparison with SALT [36]

Our final set of experiments compares the best combined flow (*PD-II* + *HVW* + *DAS*) with the results from the state-of-the-art academic Steiner tree construction, SALT [36]. SALT uses FLUTE [33] to generate its initial input and improves the initial construction to reduce PL. For nets with less than 10 terminals, FLUTE produces the optimal WL and may also produce excellent or even optimal PL, in which case running SALT is not even necessary. Hence, the cases for which FLUTE produces excellent PL are in some sense uninteresting. If FLUTE produces a good tradeoff curve, then SALT simply returns the FLUTE solution. Our approach can do something similar using the following simple metaheuristic: (1) run both FLUTE and (*PD-II* + *HVW* + *DAS*) in parallel; (2) if FLUTE is better than (*PD-II* + *HVW* + *DAS*) for both WL and PL, return the FLUTE solution, else return the (*PD-II* + *HVW* + *DAS*) solution. Essentially, the metaheuristic returns a solution identical to SALT's when the FLUTE solution is dominant. Note that for large and huge nets, the FLUTE solution almost never is dominant.

Figure 2.7 shows normalized WL and PL tradeoff curves for the metaheuristic flow and

Table 2.4: Comparisons of the best P_{Tnorm} for (1) *PD* + *HVW* and (2) *PD* + *HVW* + *DAS* across different WL thresholds.

$\|V\|$	Method	WL threshold					
		1%	2%	4%	7%	10%	15%
small	(1)	1.0233	1.0241	1.0250	1.0249	1.0236	1.0202
	(2)	1.0126	1.0115	1.0097	1.0073	1.0054	1.0033
	Imp. (%)	**46.14**	**52.31**	**61.15**	**70.85**	**77.30**	**83.67**
med.	(1)	1.0786	1.0821	1.0828	1.0757	1.0649	1.0489
	(2)	1.0665	1.0629	1.0532	1.0385	1.0277	1.0168
	Imp. (%)	**15.43**	**23.30**	**35.78**	**49.07**	**57.24**	**65.58**
large	(1)	1.1637	1.1644	1.1547	1.1275	1.1026	1.0728
	(2)	1.1440	1.1347	1.1087	1.0760	1.0553	1.0357
	Imp. (%)	**12.01**	**18.07**	**29.73**	**40.36**	**46.08**	**50.93**
huge	(1)	1.2278	1.2091	1.1606	1.1107	1.0812	1.0538
	(2)	1.228	1.209	1.161	1.111	1.081	1.054
	Imp. (%)	**8.36**	**15.14**	**27.82**	**36.36**	**39.74**	**41.69**

SALT for (a) small, (b) medium, (c) large and (d) huge nets. For small nets, SALT actually achieves better solutions than the metaheuristic until the normalized WL is about 2.3% higher than optimal.[6] However, for medium, large and huge nets, the Pareto curve for the metaheuristic outperforms the one from SALT, especially as nets increase in size. For huge nets, SALT achieves $W_{Tnorm} = 1.0370$, $P_{Tnorm} = 1.141$ for $\varepsilon = 1.281$, which is its knee point in the tradeoff curve. The knee point in the metaheuristic's tradeoff curve corresponds to $W_{Tnorm} = 1.024$ and $P_{Tnorm} = 1.121$ at $\alpha = 0.35$, which achieves 35.13% WL and 14.18% PL improvements compared to SALT at its $\varepsilon = 1.281$.

Since SALT optimizes shallowness and not detour cost, Figure 2.8 presents the same set of data but using SALT's proposed metrics. SALT dominates our method according to the shallowness metric. Thus, SALT is superior with respect to its proposed metric, while *PD-II* + *HVW* + *DAS* is superior with respect to its metric.

Finally, Table 2.5 compares our best recipe to SALT using the same methodology as

[6]For {small, medium, large, huge} nets, FLUTE results for {55.6, 7.9, 0.03, 0}% of nets have smaller WL and PL than our results. As expected, FLUTE results are dominant for small nets, but our algorithm gives better PL for large and huge nets.

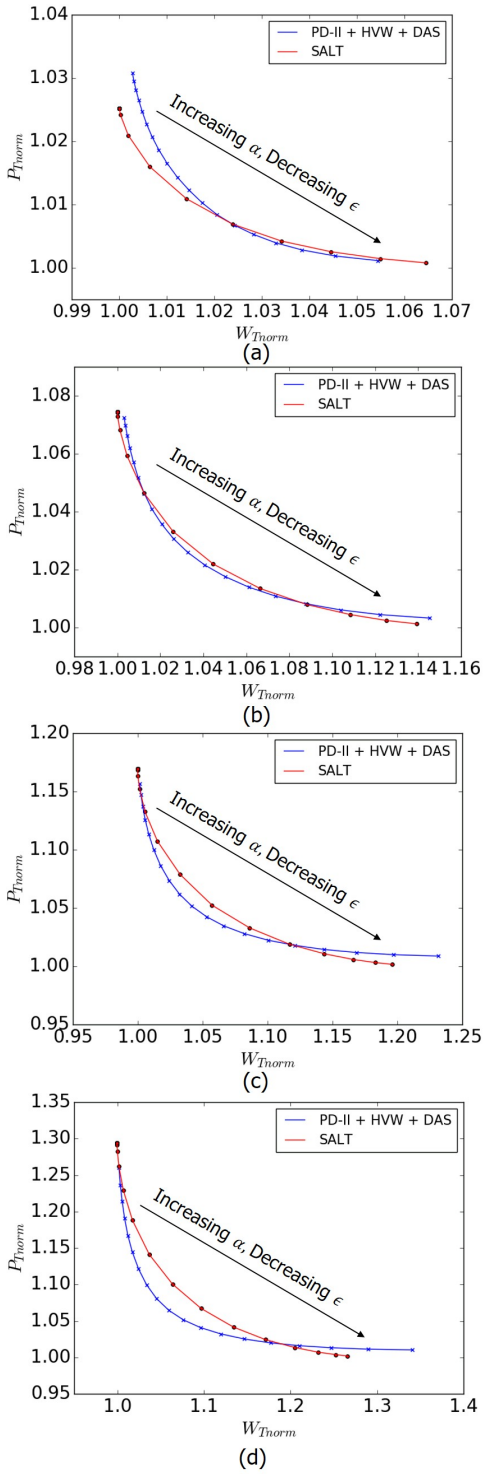


Figure 2.7: Normalized WL and PL for our metaheuristic and SALT on nets with $|V|$ = (a) 4 to 7, (b) 8 to 15, (c) 16 to 31 and (d) 32+.

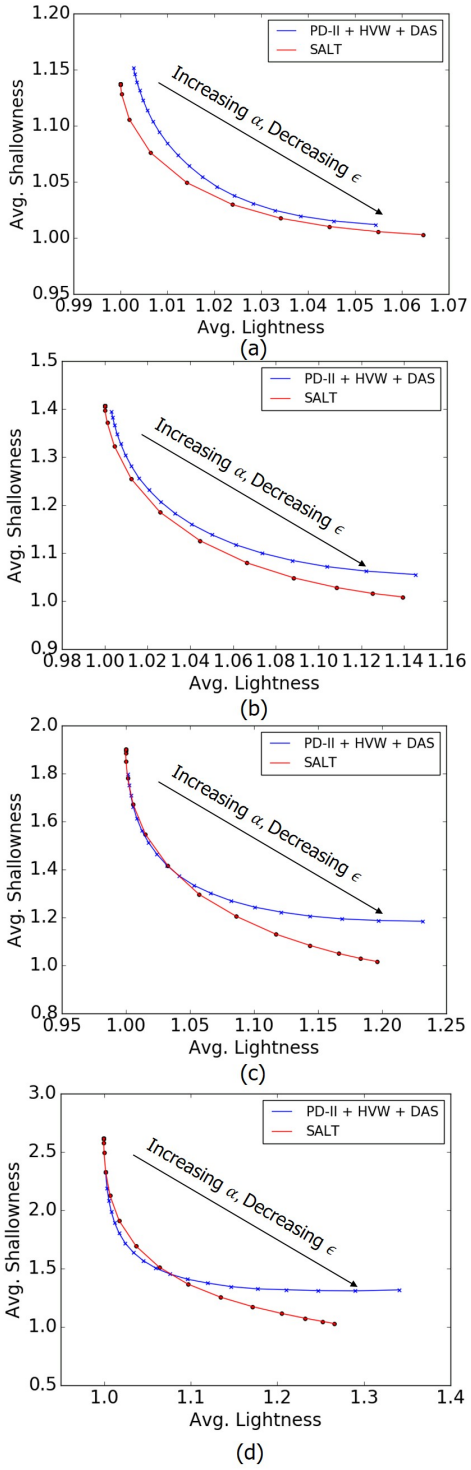


Figure 2.8: Average shallowness and lightness for our metaheuristic and SALT on nets with $|V| =$ (a) 4 to 7, (b) 8 to 15, (c) 16 to 31 and (d) 32+.

Tables 2.3 and 2.4. Note that we use FLUTE WL as a lower bound. We observe the following:

- For small nets, and WL thresholds below 10%, SALT outperforms the proposed approach. SALT is also better on medium nets with WL thresholds below 2%. This makes sense since trees in this space will closely resemble FLUTE constructions. SALT starts with a FLUTE construction and iteratively improves it, so in the space where FLUTE obtains good trees for WL and PL, such an approach outperforms the algorithm proposed in this work. Note that the magnitude of the improvement is still small. For example, for small nets and a 1% threshold, SALT is 0.99% away from the optimal normalized path length, while our approach is 1.26% away.

- For large and huge nets, and for medium nets with thresholds larger than 2%, the proposed approach performs better, reaching a peak of 36.46% improvement for huge nets with a 10% threshold. This is the domain for which the optimal tradeoff can be considerably different from FLUTE. These arguably form the class of more interesting instances where the tradeoff between WL and PL becomes increasingly important.

- As WL threshold increases, the improvement of our approach vs. SALT improves too, especially around the 7% and 10% WL threshold ranges. However, for large and huge nets the improvement is somewhat less at the 15% threshold.

Runtime. For the benchmarks studied, SALT's total runtime is 2762 seconds. By contrast, the *PD-II + HVW + DAS* algorithms, as implemented and optimized within a commercial EDA tool's code base, take 361 seconds in total. Thus, PD-II today runs more than 7 times faster than SALT.

Delay. Below, we show the impact of WL and PL improvement on delay. We estimate delays of nets produced by our algorithms and by SALT, based on the Elmore delay model with resistance of 37.318Ω per micron of wire, capacitance of 0.228fF per micron of wire, and 0.67fF pin capacitance per sink. For the solutions produced by our approach and SALT with WL

Table 2.5: Comparisons of the best P_{Tnorm} for (1) SALT and (2) *PD-II* + *HVW* + *DAS* across different WL thresholds.

$\|V\|$	Method	WL threshold					
		1%	2%	4%	7%	10%	15%
small	(1)	1.0099	1.0093	1.0082	1.0067	1.0053	1.0036
	(2)	1.0126	1.0115	1.0097	1.0073	1.0054	1.0033
	Imp. (%)	**-27.29**	**-23.85**	**-17.98**	**-8.80**	**-0.86**	**7.90**
med.	(1)	1.0652	1.0619	1.0547	1.0435	1.0337	1.0213
	(2)	1.0665	1.0629	1.0532	1.0385	1.0277	1.0168
	Imp. (%)	**-1.95**	**-1.66**	**2.76**	**11.32**	**17.63**	**21.15**
large	(1)	1.1564	1.1475	1.1261	1.0961	1.0720	1.0432
	(2)	1.1440	1.1347	1.1087	1.0760	1.0553	1.0357
	Imp. (%)	**7.91**	**8.66**	**13.77**	**20.92**	**23.09**	**17.31**
huge	(1)	1.2744	1.2574	1.2205	1.1688	1.1277	1.0763
	(2)	1.2278	1.2090	1.1606	1.1107	1.0811	1.0536
	Imp. (%)	**17.01**	**18.79**	**27.18**	**34.44**	**36.46**	**29.71**

threshold 2%, we calculate the sum of all sink delays for each net, and the average of this sum across all nets. For {small, medium, large, huge} nets, the average sum of sink delays from PD-II is lower than the average sum of sink delays from SALT by {-0.0005, 0.24, 1.54, 5.62}%. As seen with the WL and PL comparison, our algorithm has slightly larger delays for small nets and smaller delays for higher-fanout nets.

In summary, while our approach does not uniformly outperform SALT, it does provide a superior tradeoff for the most interesting class of nets that are far from optimal in terms of PL and WL.[7]

2.7 Conclusion

This work shows that the classic *PD* spanning tree algorithm that balances between Prim's and Dijkstra's algorithm can have a bad tradeoff that ends up with both WL and PL being highly suboptimal. A new spanning tree heuristic *PD-II* is demonstrated to significantly improve both

[7]The PD-II algorithm has been released as part of a leading commercial tool, with demonstrated improvements of timing and wirelength.

WL and total detour cost compared to *PD*. Further, this work extends the construction to Steiner tree with the *DAS* algorithm that directly improves trees according to both objectives. The algorithms are shown to be fast and practical. They are also suitable for integration into existing commercial routers, and can be applied in conjunction with any existing spanning and Steiner tree constructions for simultaneous WL and PL improvements. Compared to the recent SALT algorithm, our construction generates clear improvements according to the proposed metrics, especially for medium-size and larger nets. Future research includes (i) revisiting the still-open question of worst-case detour from a *PD* construction; (ii) learning-based estimation of the best α for any given instance (i.e., set of pin locations of a signal net); and (iii) extending the detour cost objective to encompass sink criticality, "global" radius, and other additional criteria.

2.8 Acknowledgments

Chapter 2 is in part a reprint of C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and S. Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 10-17.

I would like to thank my co-authors Dr. Charles J. Alpert, Dr. Wing-Kai Chow, Kwangsoo Han, Professor Andrew B. Kahng, Dr. Zhuo Li and Derong Liu. I also thank Mr. Gengjie Chen and the other authors of SALT [36] for their kindness in performing studies with their code so that we could report comparisons with SALT. I also thank the authors of [38] for making their ePlace executable available for us to gather placement data.

Chapter 3

Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics

This chapter describes a property of nets and their pin locations, and provides an improved lookup table for wirelength estimation.

3.1 Introduction

VLSI global placement seeks to minimize routed wirelength (WL) along with timing path delays, dynamic power and other design metrics, subject to the constraint that placeable instances do not overlap. Because signal nets are routed as Steiner trees, their routed wirelengths are ideally modeled as the costs of respective *Rectilinear Steiner Minimum Trees* (RSMTs) over pin locations. Since the RSMT problem is NP-hard, placement tools typically minimize the sum over all nets of the bounding box half-perimeter of pin locations – i.e., the *half-perimeter wirelength* (HPWL) objective [46]. An important element of efficient placer implementation is the fast estimation of

RSMT costs, e.g., by weighting HPWL according to a lookup table of scaling factors [40][44].

Our present work focuses on the qualitative difference between *real* pointsets corresponding to pin locations of placed nets, and *random* pointsets that have often been used to characterize the performance and relative merits of RSMT heuristics or RSMT cost estimators. As discussed below, placement tools will tend to move the pins of a net up against two adjacent edges of the net bounding box, as shown in Figure 3.4 below. This phenomenon is due to the HPWL objective in conjunction with each placeable instance having multiple incident nets. By contrast, with random pointsets, all point locations inside the pointset bounding box are equiprobable. We define the *L-ness* of a placed net's pin locations to capture how close they are to two adjacent edges of the net bounding box:

Definition: Given a pointset P, the *bounding box* of P is the minimum-area rectangle that contains all points of P; we use $B(P)$ to denote the bounding box area. The *L-ness* of P is measured as $R(P)/B(P)$, where $R(P)$ is the area of the *largest empty (isothetic) rectangle* that (i) is contained in the bounding box of P, (ii) contains one corner of the bounding box of P, and (iii) contains no points in P.

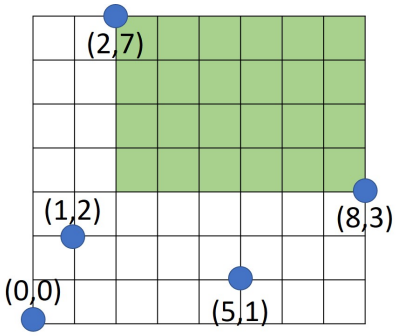


Figure 3.1: Illustration of largest empty (isothetic, i.e., with axis-parallel edges) rectangle. The L-ness of this 5-pin pointset is 24/56.

High $R(P)/B(P)$ ratio corresponds to large L-ness. If $B(P) = 0$, then we consider the L-ness of P to be 1. Figure 3.1 shows a pointset with $R(P)/B(P) = \frac{24}{56}$.

3.1.1 Motivation: Non-uniformity of Net Pin Placements

As a motivating study, we first confirm the non-uniform distribution of real placed pointsets (i.e., pins of signal nets). We use the leon3mp [65] and theia [80] design blocks mapped to a 28nm LP foundry enablement, with place-and-route performed using Cadence Innovus Implementation System v15.2 [75]. Two types of placements are studied: *pseudo-1D* and *2D*. To obtain a pseudo-1D placement, we create a floorplan with width/height *aspect ratio* (AR) of 10:1 following the methodology described in [42]. We collect the point (pin) location distribution for each net along the x-axis, within a normalized range of 0 (left boundary of each given net) to 1 (right boundary of each given net). We categorize nets into three types – L, R, and O, defined as follows. A net n is of type L if, for each cell c of the net, no fanin/fanout net of c has a pin to the right of the rightmost pin of the net n, and at least one has a pin to the left of the leftmost pin of the net n. A net n is of Type R if for each cell c of the net, no fanin/fanout of c has a pin to the left of the bounding box (BBox) of net n, and at least one has a pin to the rightmost pin of net n. A net n is otherwise of type O. For example, in Figure 3.2, nets A and D are of type L; net B is of type R; and net C is of type O.

Figure 3.3 shows results of this empirical study on the designs mentioned above. We see that a "real" placement tool will clearly push cells (pins) of a type L net (respectively, a type R net) toward the left (respectively, right) boundary. There are virtually no cells in the middle, and only a few cells are pushed to the opposite boundary. From our study, we believe that there are two explanations for cells occurring at the opposite boundary: (i) we plot cell locations according to the center of the cell, which has error with respect to exact pin locations; and (ii) nets with short x-span can exhibit this behavior since the placer does not see a significant wirelength penalty for doing this. For a type O net, the cell distribution still shows preference to the bounding box boundary, indicating non-uniform distribution.

We have also performed the above experiment for 2D placements with floorplan height = width, i.e., aspect ratio = 1. In the y direction, "bottom" and "top" are respectively equivalenced to

"left" and "right" in the x direction. Then, we sum up the pointset distribution in both directions. The results look similar to Figure 3.3. We see a very strong deviation from the uniform distribution that is seen with random pointsets.[1]

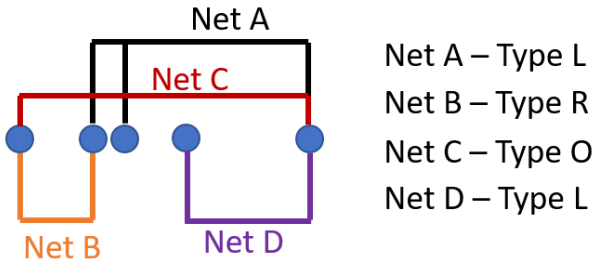


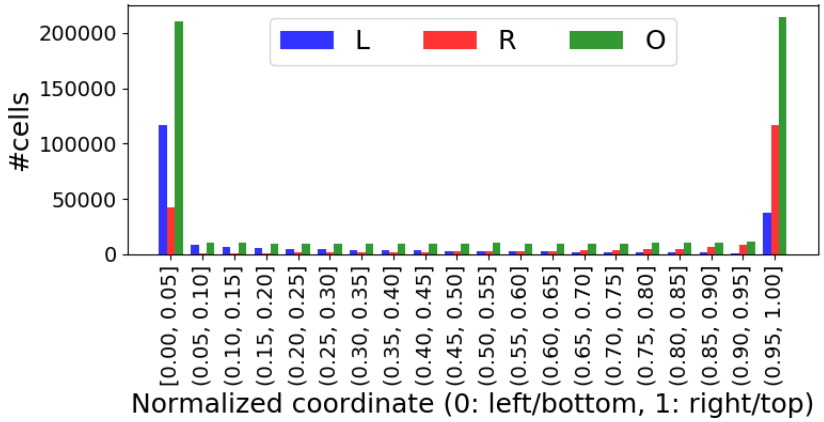
Figure 3.2: Illustration of L, R, and O types of nets.

3.1.2 Related Works

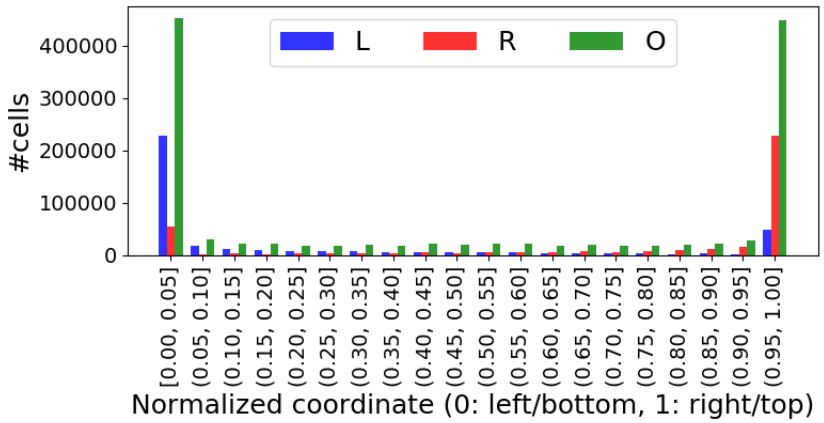
Previous works have estimated RSMT cost based on characteristics of placed pin locations. Caldwell et al. [40] demonstrate how RSMT cost depends on both the cardinality and the aspect ratio of a pointset. This improves upon the earlier work of Cheng [44], which estimates RSMT cost based only on the pointset cardinality. Quite notably, Cheng [44] appears to point out the concept of L-ness in real placed pin locations when discussing the modeling of routing resource demand. However, this observation does not seem to have been followed up in the RSMT estimation or placement literatures.[2] In the computational geometry literature, Chazelle et al. [43] present an $O(n \log^3 n)$ divide-and-conquer algorithm which calculates the area of the largest empty (isothetic) rectangle in a set of n points. By using a semi-dynamic heap, Naamad et al. [23] calculate the largest empty rectangle in a set of n points in $O(s \log n)$ time where s is the number of possible empty rectangles.

[1] While this motivating study uses the Cadence Innovus placer, Section 3.3 below shows similar non-uniformity across multiple academic and commercial placers' outputs.

[2] Section 3 of [44] states, "The high wiring probability at the top and bottom boundaries comes from the following two facts: (1) the probability of having two pins located at the same boundary is high because of bounding box. (2) when finding an optimal Steiner tree, either a left-L or a right-L is used to reduce the wire length of a minimum spanning tree."



(a) 1D



(b) 2D

Figure 3.3: Empirical results from pseudo-1D and 2D placements.

With regard to RSMT heuristic constructions, Chu and Wong [33] give a well-known $O(n \log n)$ RSMT heuristic, FLUTE, which is the most accurate of the RSMT heuristics that we study in Section 3.4.1. The Prim-Dijkstra heuristic of Alpert et al. [1] "blends" classic minimum spanning tree and shortest-paths tree constructions using a weighting factor α to obtain a heuristic "shallow-light" spanning tree. Below, in our experimental studies, we augment the Prim-Dijkstra construction with the edge-overlapping method of Ho et al. [7] to obtain a heuristic RSMT from the Prim-Dijkstra spanning construction.

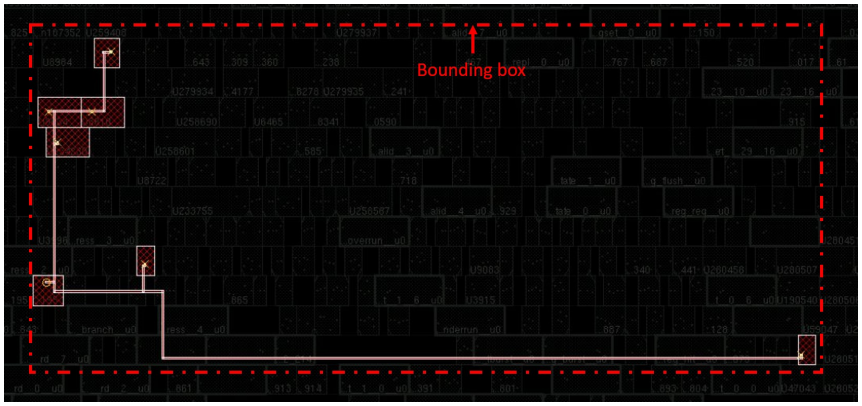


Figure 3.4: Pins of a net from an industrial placer, clustered towards the left and bottom edges of the bounding box.

3.1.3 Contributions

The main contributions of this work are as follows.

- We propose a formal definition of *L-ness* of a pointset in the Manhattan plane.
- We empirically characterize a qualitatively significant difference in L-ness between real placed net pins and random pointsets. As seen in Section 3.3.1, real placed pointsets have significantly higher L-ness than random pointsets.
- We describe a pointset generator which can be used to generate more realistic pointsets with prescribed L-ness distribution. This can be used to assess RSMT heuristics and cost estimators with randomly generated pointsets that match AR and $R(P)/B(P)$ distributions (as well as RSMT costs - see Subsection 3.4.2) of real placed pointsets.
- We give a new lookup table-based RSMT cost estimator which improves over the method of [40] by adding L-ness as a parameter. Our implementation of this lookup table gives a non-dominated (speed, accuracy) option for RSMT cost estimation.

In the following, Section 3.2 presents notation and analyses of L-ness in planar pointsets. Section 3.3 describes empirical characterizations of real placed pointsets, contrasted with random

pointsets. Section 3.4 discusses the impact of L-ness on the relative performance of various RSMT heuristics. Section 3.5 presents a new lookup table-based RSMT cost estimate that improves upon [40] by adding an L-ness dimension. Section 3.6 summarizes our results and concludes.

3.2 Preliminaries

In this section, we first give notations and facts used in this work. We then analyze different properties of pointsets, and discuss the relationship of L-ness to other pointset characteristics. Last, we provide methods to generate realistic pointsets, and an algorithm to compute $R(P)$ in $\Theta(n\ log\ n)$ time.

3.2.1 Notations

Notations that we use in this chapter are summarized in Table 3.1. The layout region is assumed to have lower-left corner $(0,0)$ and upper-right corner (H,W). A *random p*-pin pointset consists of p points chosen randomly from a uniform distribution in the $H \times W$ layout region. As noted above, the *bounding box* of pointset P is the minimal isothetic (axis-parallel) rectangle that contains all points of P. The *half-perimeter* of a given bounding box is half the perimeter of the bounding box. For example, the half-perimeter of the bounding box in Figure 3.1 is 15, and its *AR* is $8/7$.

Our discussion furthermore assumes that points of a random pointset are in *general position*, i.e., all x-coordinates and all y-coordinates are distinct. To validate this assumption, we extract placed pin coordinates from the placements of seven design blocks, including leon3mp and netcard from [65]; theia, jpeg, aes and mpeg from [80]; and an ARM Cortex A53 [71]. The placements are obtained using two leading commercial place-and-route tools, Cadence Innovus Implementation System v15.2 [75] and Synopsys IC Compiler L-2016.03-SP4 [82] with foundry enablements at 28nm and 16nm. We also extract the placements of the DAC-2012

Table 3.1: Notations.

Notation	Meaning
p	net degree (# pins of a signal net) ($p \geq 2$)
P	a net (pointset), $P = (x_1, y_1), ..., (x_p, y_p)$
$B(P)$	the area of the minimum bounding box of P
$R(P)$	the area of the largest empty rectangle of P
$RSMT(P)$	the rectilinear Steiner minimum tree over P
(H, W)	chip dimensions, i.e., height and width of the chip
AR	aspect ratio (W/H) of the bounding box
$R(P)/B(P)$	L-ness, the ratio of $R(P)$ divided by $B(P)$

Table 3.2: Probability that any two points in a pointset share the same x- or y-coordinate.

p	ICC/Innovus	Capo	ePlace
2	9.88%	7.48%	7.65%
3	10.98%	7.90%	7.46%
4	7.57%	6.84%	6.01%
5	8.03%	7.99%	6.32%
6	7.50%	7.69%	7.49%
7	7.45%	8.27%	5.28%
8	7.68%	4.86%	3.90%
9	8.48%	6.13%	4.37%
10	7.46%	4.35%	3.81%
11	6.78%	4.51%	3.59%
12	6.18%	4.27%	3.50%

benchmark suite [37] from two well-known academic placers, i.e., Capo [41] and ePlace [38]. These placements are collectively referred to as *real pointsets* in the rest of this chapter. Table 3.2 shows that the percentage of any two points in a real pointset sharing the same x-coordinate or y-coordinate is less than 11%, supporting our assumption of distinct x- and y-coordinates.

We define *L-ness* of P as the ratio of $R(P)$ to $B(P)$, where $R(P)$ is the area of the *largest empty (isothetic) rectangle* that (i) is contained in the bounding box of P, (ii) contains one corner of the bounding box of P, and (iii) contains no points in P. High $R(P)/B(P)$ ratio corresponds to large L-ness.

3.2.2 Probability that k Points Define the Bounding Box

A bounding box can be represented by four extreme coordinate values, i.e., x_{min}, x_{max}, y_{min} and y_{max}. Given unique x- and y-coordinates, at most four points of a pointset can define the pointset's bounding box, where each of the points provides exactly one of the four extreme coordinates. Further, at least two points define the bounding box, where each of the points contains one extreme x-coordinate and one extreme y-coordinate. We use $Pr(p,k)$ to denote the probability that the bounding box of a pointset P (having cardinality p) is defined by k points ($k \in \{2,3,4\}$).

For $k = 2$, assume that points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ define the bounding box. Then, x_1 (resp. y_1) must be x_{min} or x_{max} (resp. y_{min} or y_{max}) out of the p x-coordinates (resp. y-coordinates), and x_2 (resp. y_2) can only be the other extreme x (resp. y)-coordinate out of $p-1$ x-coordinates (resp. y-coordinates). Thus, Equation (3.1) gives the probability $Pr(p,2)$.

For $k = 4$, each of four points can define only one extreme coordinate of the bounding box. Assume that these points are $p_1 = (x_{min}, \neg(y_{min} \vee y_{max}))$, $p_2 = (x_{max}, \neg(y_{min} \vee y_{max}))$, $p_3 = (\neg(x_{min} \vee x_{max}), y_{min})$, and $p_4 = (\neg(x_{min} \vee x_{max}), y_{max})$. Then, the probability that four points define the bounding box is as given in Equation (3.3). Supplemental equations using chain rules to derive probabilities are given in Equations (3.4)–(3.7). For example, $Pr(p_1)$ is computed by finding the probability that a point has the minimum x-coordinate and not an extreme y-coordinate. These probabilities are each computed separately and are then multiplied together since they are independent. The remaining probabilities in Equations (3.4)–(3.7) are computed in a similar fashion.

Table 3.3: $Pr(p,k)$ for $p \in [3,10]$ and $k \in \{2,3,4\}$.

p	$Pr(p,k=2)$	$Pr(p,k=3)$	$Pr(p,k=4)$
3	0.3333	0.6667	0.0000
4	0.1667	0.6667	0.1667
5	0.1000	0.6000	0.3000
6	0.0667	0.5333	0.4000
7	0.0476	0.4762	0.4762
8	0.0357	0.4286	0.5357
9	0.0278	0.3889	0.5833
10	0.0222	0.3556	0.6222

$$Pr(p,k=2) = \binom{p}{2}\left(\frac{2}{p}\right)^2\left(\frac{1}{p-1}\right)^2 \tag{3.1}$$

$$Pr(p,k=3) = 1 - Pr(p,k=2) - Pr(p,k=4) \tag{3.2}$$

$$\begin{aligned} Pr(p,k=4) &= 4!\binom{p}{4} Pr(p_1\ p_2\ p_3\ p_4) \\ &= 4!\binom{p}{4} Pr(p_1)Pr(p_2|p_1)Pr(p_3|p_1p_2)Pr(p_4|p_1p_2p_3) \\ &= \binom{p}{4}\left(\frac{4!}{(p^2)(p-1)^2}\right) \end{aligned} \tag{3.3}$$

For the remaining case of $k = 3$, we can calculate the probability $Pr(p,3)$ using Equation (3.2). Table 3.3 provides a lookup table for $Pr(p,k)$ for $k \in \{2,3,4\}$ and $p \in [3,10]$.

$$Pr(p_1) = \left(\frac{1}{p}\right)\left(\frac{p-2}{p}\right) \tag{3.4}$$

$$Pr(p_2|p_1) = \left(\frac{1}{p-1}\right)\left(\frac{p-3}{p-1}\right) \tag{3.5}$$

$$Pr(p_3|p_1 \cdot p_2) = \left(\frac{p-2}{p-2}\right)\left(\frac{1}{p-2}\right) \tag{3.6}$$

$$Pr(p_4|p_1 \cdot p_2 \cdot p_3) = \left(\frac{p-3}{p-3}\right)\left(\frac{1}{p-3}\right) \tag{3.7}$$

We use this probability in Algorithm 2 to determine the parameter k. Subsequently, we use Algorithm 2 to create the *real′* pointsets, as described in Section 3.4.

3.2.3 Independence of AR and $R(P)/B(P)$

To justify the experimental methodology used below, we prove the intuitive claim that $R(P)/B(P)$ is preserved when a 2D pointset P is "stretched" (by scaling of x-coordinates and of y-coordinates) into a pointset P' that has a different aspect ratio. We refer to this property of pointsets as *independence* of AR and $R(P)/B(P)$. We show this independence of AR and $R(P)/B(P)$ by (i) exhibiting an appropriate 1-1 correspondence between pointsets P with bounding box area $B(P)$ and pointsets P' with bounding box area $B(P')$, then (ii) showing that the ratio $R(P)/B(P) = R(P')/B(P')$ is preserved by this correspondence. In Subsection 3.3.1, we measure $R(P)/B(P)$ without considering the effect of AR on L-ness of pointsets. Hence, we prove the independence of $R(P)/B(P)$ with AR below.

Fact 1. Scaling of x- and y-coordinates provides a (bidirectional) 1-1 mapping between pointsets P having unit square bounding box $B(P)$, and pointsets P', with $|P| = |P'|$ and bounding box B' having an arbitrary aspect ratio.

Fact 1 is established as follows. Denote the width and height of B' are w and h, respectively. We obtain pointset P' from P by scaling x- and y-coordinates of points in P by w and h, respectively. As a result, the x- and y-coordinates of the bounding box edges of P' are also scaled by w and h. The inverse scaling procedure can be applied to restore any such P' to the original P. The scaling of coordinates thus provides a 1-1 correspondence between pointsets having the same cardinality but different bounding box ARs.

Next, we say that the point (x_i, y_i) in P *corresponds* to a point (x'_i, y'_i) in P' if $(x'_i, y'_i) = (w \cdot x_i, h \cdot y_i)$. A bounding box-edge of P analogously *corresponds* to a scaled bounding box-edge of P'. For example, Figure 3.5(b) shows a pointset P' obtained by scaling P (in Figure 3.5(a)) by $(w,h) = (w,1)$. From our definitions, we say that the point (x_i, y_i) in P corresponds to the point

(x'_i, y'_i) in P', and the edge $x = x_{sp}$ of P corresponds to the edge $x = x'_{sp}$ of P'.

The following Fact 2 holds for pointsets (i) P with its largest empty rectangle defined by two edges of the bounding box, $x = x_{sp}$ and $y = y_{sp}$, and two points, (x_1, y_1) and (x_2, y_2); and (ii) P', which is created by scaling the x- and y-coordinates of points in P by w and h, respectively.

Fact 2. Given P and P', the edges and points that define $R(P)/B(P)$ correspond to edges and points that define $R(P')/B(P')$.

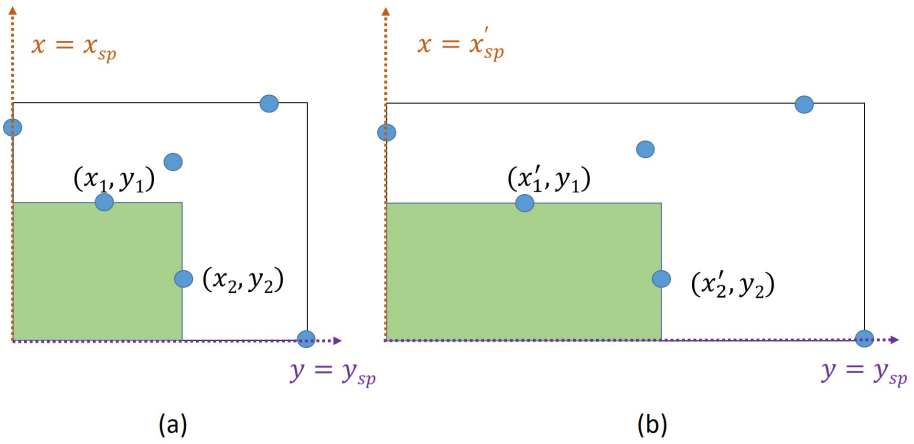


Figure 3.5: The pointsets (a) P and (b) P' with the largest empty rectangle colored green.

Fact 2 is established as follows. P contains p points, i.e., $P = \{(x_1, y_1), (x_2, y_2), \cdots, (x_p, y_p)\}$. Without loss of generality, we assume that the bounding box of P has $AR = 1$, and we only scale points in pointset P in the x-direction by w (i.e., $w > 0$, $h = 1$) to obtain P'. P' also contains p points, $P' = \{(x'_1, y_1), (x'_2, y_2), \cdots, (x'_p, y_p)\}$, where $(x'_j, y_j) = (w \cdot x_j, y_j)$ for $k \in [1, p]$. The following treats the case illustrated in Figure 3.5, namely, the case with the empty rectangle at the lower-left corner of the bounding box, i.e., $x_{sp} < x_1 < x_2$, and $y_{sp} < y_2 < y_1$. The other three cases are similarly analyzed. $R(P)$ is defined as

$$R(P) = (x_2 - x_{sp}) \cdot (y_1 - y_{sp}) \tag{3.8}$$

Assume toward a contradiction that the edges $x = w \cdot x_{sp}$, $y = y_{sp}$, and the points (x'_1, y_1) and (x'_2, y_2) do not form the largest empty rectangle of P'. Then, there must exist an empty rectangle of P' such that

$$R(P') > w \cdot R(P) \tag{3.9}$$

Suppose that the largest empty rectangle of P' is defined by the edges $w \cdot x_{sp}$ and y_{sp} and two points (x'_m, y_m) and (x'_n, y_n), with $\{n, m\} \neq \{1, 2\}$ and $x'_{sp} < x'_m < x'_n$ and $y_{sp} < y_n < y_m$. Then, $R(P')$ is calculated as

$$R(P') = (x'_n - x'_{sp}) \cdot (y_m - y_{sp}) \tag{3.10}$$

$$= (w \cdot x_n - w \cdot x_{sp}) \cdot (y_m - y_{sp}) \tag{3.11}$$

$$= w \cdot (x_n - x_{sp}) \cdot (y_m - y_{sp}) \tag{3.12}$$

According to the definition of $R(P)$, $(x_n - x_{sp}) \cdot (y_m - y_{sp}) \leq R(P)$. Therefore,

$$R(P') \leq w \cdot R(P) \tag{3.13}$$

which contradicts Equation (3.9). This establishes Fact 2.

3.2.4 Efficient Calculation of $R(P)$

We now describe an efficient method to obtain $R(P)$. Each of the four corners of the bounding box may be the intersection of the two edges that form $R(P)$. For simplicity, we only describe our algorithm for the corner (x_{min}, y_{min}). The final result can be obtained by invoking the

algorithm on each corner of the bounding box of P with small modifications and then returning the largest value, at the cost of a constant-factor complexity increase.

Algorithm 3 describes the calculation of $R(P)$. The algorithm begins with pointset P sorted in ascending order of x-coordinates. Lines 1 and 2 perform initializations. In Lines 3 – 8, we check whether the current point has a smaller y-coordinate than the stored value of y_0. If so, the lower-left corner will form an empty rectangle, and we update the maximum known rectangle area. The same procedure is followed to compute the largest empty rectangle at the remaining corners. We step through the sorted list of points, check if each pair of points forms an empty rectangle, and if so, update the maximum known rectangle area. The time complexity of the algorithm is lower-bounded by the implied sorting step, which gives a $\Theta(p\ log\ p)$ time complexity.

Algorithm 3 CalcRP (Assuming lower-left corner is selected)

Input: P with x-coordinates in ascending order
Output: $R(P)$

1: $R(P) = 0$
2: $y_0 = y_1$
3: **for** $i = 2$ to p **do**
4: **if** $y_i \leq y_0$ **then**
5: $R(P) \leftarrow \max\left(R(P), (x_i - x_{min}) \cdot (y_0 - y_{min})\right)$
6: $y_0 \leftarrow y_i$
7: **end if**
8: **end for**
9: **return** $R(P)$

3.3 Real vs. Random Pointsets

In this section, we empirically demonstrate the significant difference in L-ness between real and random pointsets. We then present a method for generating pointsets with prescribed L-ness and aspect ratio.

3.3.1 L-ness of Real vs. Random Pointsets

We experimentally compare the $R(P)/B(P)$ distribution of 100K random pointsets with the $R(P)/B(P)$ distribution of real (placed net pins) pointsets. Figure 3.6 shows the distributions of $R(P)/B(P)$ in random and real pointsets. In each plot, the x-axis denotes the $R(P)/B(P)$ ratio and the y-axis denotes the fraction of nets for each $R(P)/B(P)$ value. From the figure, we see that the placements from commercial and academic placers result in pointsets with significantly larger L-ness (i.e., larger $R(P)/B(P)$ ratio) than random pointsets.[3] We also observe that the qualitative difference from random pointsets holds across academic and commercial placers.

We believe that this large L-ness arises due to the following reasons. Given a large chip area and a relatively small bounding box (b_0) area for any net n_0, it is intuitive that the other nets incident to the cells of net n_0 have their bounding boxes outside b_0. This causes the cells to get pulled towards the boundary, and extend the boundaries of net n_0 due to multiple inter-related nets (i.e., intersecting hyperedges of the netlist). Further, low net degrees usually result in a geometrically asymmetrical cell distribution, increasing the L-ness of a particular net.

To confirm the statistical difference for $p \in [3,12]$, we perform two tests: (i) bootstrapping the mean with a 95% confidence interval [39], and (ii) Two-Sample Kolmogorov-Smirnov (KS) Test [47]. The bootstrap test provides a 0.95 confidence interval on the average of $R(P)/B(P)$ for 10000 random pointsets. We compare the means of $R(P)/B(P)$ values for real pointsets with the 0.95 confidence interval. Figure 3.7 shows that the means of real pointsets do not lie within the confidence intervals of random pointsets for any $p \in [3,12]$. Hence, real pointsets have a statistically significant larger $R(P)/B(P)$ compared to random pointsets.

The Two-Sample Kolmogorov-Smirnov (KS) Test [39] states that for a confidence interval of 95%, we have a statistically significant difference if the KS statistic $D_{nm} > 1.36$. The KS

[3] We have separately extracted net pin locations from an advanced processor design from a leading semiconductor company, and confirmed that the $R(P)/B(P)$ distributions follow the same trend as in Figure 3.6.

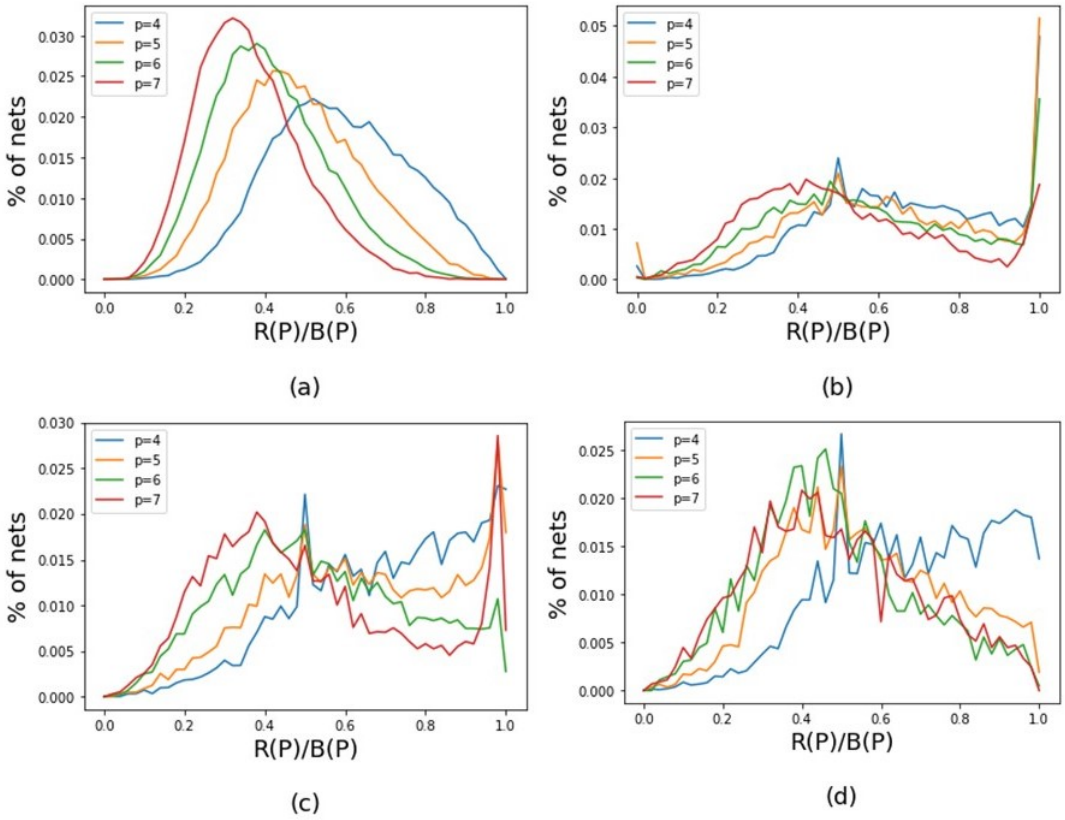


Figure 3.6: Distribution of $R(P)/B(P)$ from (a) random pointsets, (b) ICC [82] and Innovus [75] placements, (c) Capo [41] placements, and (d) ePlace [38] placements.

statistic is computed as

$$D_{nm} = \sqrt{nm/(n+m)} \cdot sup|F(x) - G(x)| \tag{3.14}$$

where n and m are sample sizes of random and real pointsets respectively, F and G are cumulative distribution functions (CDFs) (with 100 bins of width 0.01) of $R(P)/B(P)$ values of random and real pointsets respectively. sup is the maximum distance between F and G for $0 \leq x \leq 1$. Table 3.4 shows the KS statistics. We see that $D_{nm} > 1.36$ for all random versus real pointsets, again confirming the statistically significant difference between $R(P)/B(P)$ distributions of random and real pointsets.

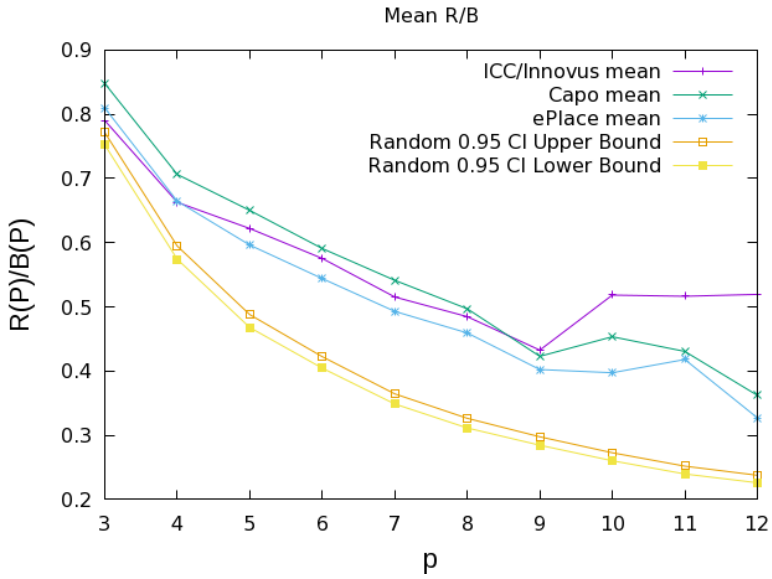


Figure 3.7: 95% confidence interval for $R(P)/B(P)$ in random pointsets and mean $R(P)/B(P)$ for real pointsets.

3.3.2 Pointset Generation

Since random pointsets differ significantly from real placed pin locations, and since it is challenging to obtain real placement data, there is a need to generate pointsets with prescribed L-ness. Here, we present an algorithm (Algorithm 4) to generate a random pointset with prescribed $R(P)/B(P)$ (L-ness) and aspect ratio (AR). The inputs include #pins p, intended number of points k that define the bounding box (see Section 3.2.2), intended L-ness range $[RPBP - \Delta_{err}, RPBP + \Delta_{err}]$, and aspect ratio AR. The output is a pointset P that satisfies the L-ness range constraint.

Lines 1 and 2 perform initializations. In Line 3, we generate k points on the bounding box. Since $R(P)/B(P)$ will monotonically decrease as we add one more point to an existing pointset, the function $getBBoxPts$ comprehends the desired L-ness range and always gives k points with $R(P)/B(P) \geq RPBP - \Delta_{err}$. These k points form a bounding box with area 1M x 1M and aspect ratio AR. In Lines 5 – 9, we iteratively add one point with random location strictly inside the bounding box and check L-ness. If we do not meet the L-ness lower bound, the last added point is removed and reselected. The points are added with unique x- and y-coordinates, following

Table 3.4: D_{nm} for $p \in [3, 12]$ using ICC/Innovus, Capo and ePlace placers.

p	ICC/Innovus	Capo	ePlace
3	3.363	6.160	3.256
4	3.788	6.204	3.926
5	5.159	6.913	4.240
6	4.641	5.605	3.341
7	3.658	5.150	2.884
8	3.219	4.500	2.481
9	1.737	2.953	2.747
10	4.754	3.413	3.777
11	5.790	3.987	3.162
12	7.106	4.028	4.708

the assumption of points in general position in Section 3.2.1. In Lines 11 and 12, we return the pointset satisfying the L-ness range constraint and discard the result otherwise. In our actual implementation, we can reuse discarded pointsets when generating for a different L-ness range – e.g., during the process of reproducing a distribution such as in Figure 3.6(b)-(d).

Algorithm 4 is qualitatively equivalent to randomly generating a pointset and checking if the pointset is valid, i.e., having $R(P)/B(P) \in [RPBP - \Delta_{err}, RPBP + \Delta_{err}]$. If we assume towards a contradiction that it does not, then at least one of the points we remove in Line 8 would contribute to a valid pointset. Since adding points within the bounding box cannot increase the $R(P)/B(P)$ value of a pointset, the points in this pointset cannot be part of a pointset with $R(P)/B(P)$ within the prescribed L-ness range. Hence, Algorithm 4 returns qualitatively the same pointsets as repeated generation of a pointset and checking whether the pointset has $R(P)/B(P)$ within the prescribed L-ness range. However, Algorithm 2 is much more efficient, e.g., we can produce 100K pointsets targeted to match the distribution of Figure 3.6(d) with $p = 7$ in 75.54 seconds with a 2.7 GHz Intel Xeon server.

Algorithm 4 GenRandPointset

Input: p, k, $RPBP$, Δ_{err}, AR
Output: P with $R(P)/B(P) \in [RPBP - \Delta_{err}, RPBP + \Delta_{err}]$

$P \leftarrow \varnothing$
$R(P)/B(P) \leftarrow 0$
$P \leftarrow getBBoxPts(P, k, RPBP, \Delta_{err}, AR)$
while $|P| < p$ **do**
  $P \leftarrow AddPoint(P)$
  if $calcRP(P) < RPBP - \Delta_{err}$ **then**
    $RemovePoint(P)$
  end if
end while
if $calcRP(P) \leq RPBP + \Delta_{err}$ **then**
  return P
else
  return -1
end if

3.4 Implications for RSMT Heuristics

In this section, we perform experiments to analyze the impact of L-ness on the performance (tree cost / wirelength estimation) of various RSMT heuristics. We first show how wirelength changes with different L-ness. Then, we show the RSMT cost difference between random and real pointsets.

3.4.1 Impact of L-ness on RSMT Heuristics

In this subsection, we study the change in wirelength with $R(P)/B(P)$ (L-ness). We generate 10K pointsets for each $R(P)/B(P)$ from 0.2 to 0.8, with a step of 0.1 and $\Delta_{err} = 0.02$. We use a fixed $B(P)$ size of 1M$\times$1M. We evaluate the wirelength cost of four heuristics: (i) rectilinear MST implementation by Kahng et al. [48] using Prim's algorithm, (ii) Prim-Dijkstra (PD) [1] with $\alpha = 0.3$ (PD 0.3) and with $\alpha = 1.0$ (PD 1.0 constructs a shortest path tree, and is equivalent to Dijkstra's algorithm [6]), (iii) HVW [7] algorithm as a post-processing of PD 0.3 (HVW 0.3) and PD 1.0 (HVW 1.0), and (iv) FLUTE [33].

Figure 3.8 shows the wirelength values. The x-axis denotes the $R(P)/B(P)$ ratio and the y-axis represents the total wirelength for all 10K pointsets per each $R(P)/B(P)$. Each row of figures represents a particular value of $AR = \{1,2,4\}$; each column represents a value of $p = \{4,5,7\}$. We see that wirelength decreases as $R(P)/B(P)$ increases, indicating that we should expect lower wirelength for real pointsets which tend to have larger $R(P)/B(P)$ than random pointsets. Also, difference in wirelength among heuristics decreases with increase in $R(P)/B(P)$. Although PD 0.3 and HVW 0.3 have different optimization objectives (i.e., radius and wirelength balance) from FLUTE, wirelength follows the same trend with $R(P)/B(P)$ for all heuristics. These results suggest that assessments of cost or accuracy benefit versus runtime overhead when using these heuristics may have been misguided by the use of random pointsets in experimental studies, and that random pointsets might not give sufficient insight into the benefits of RSMT heuristics. We also observe that crossovers between heuristics tend to decline as AR increases.

3.4.2 RSMT Cost on Real Pointsets

Previous works [1][7] use random pointsets to verify the accuracy of RSMT heuristics. However, we reevaluate their accuracy and show their performance difference considering L-ness in real pointsets. We first generate *real' pointsets* with $R(P)/B(P)$ and AR distributions of *real pointsets* from academic and commercial placements, and show that our Algorithm 4 generates statistically similar pointsets to real placements. We then use *real' pointsets* to analyze the accuracy of heuristic WL estimation.

To generate real' pointsets, we extract the distributions of $R(P)/B(P)$ and AR from real pointsets for $p \in [3,12]$ and use these distributions to create 10K real' pointsets for each p. We run FLUTE on all pointsets and perform the two-sample Kolmogorov-Smirnov test (KS) test on the wirelength distributions with a 95% confidence interval, using 50 bins to generate the CDFs. Table 3.5 shows that eight of nine values are smaller than the minimum D_{nm} value in Table 3.4. This shows that real' pointsets give a good representation of real pointsets for most

cases. Figure 3.9 shows one case with a Kolmogorov-Smirnov failure. However, the probability distributions of wirelengths from real' and real pointset distributions are still similar in appearance.

Table 3.5: D_{nm} for wirelengths on real and real' pointsets.

p	4	5	6	7	8	9	10	11	12
D_{nm}	1.189	1.063	1.402	1.788	1.690	1.621	1.026	1.086	1.601

We use the above real' pointsets to evaluate the accuracy of each heuristics. Tables 3.6 and 3.7 report the errors of these heuristics versus FLUTE[4], and compare the differences in errors for real and random pointsets. A positive value in Table 3.6 means a larger wirelength is given compared to FLUTE.

Table 3.7 reports the percentage difference for each heuristic between real and random pointsets as $Error_{real} - Error_{random}$. A negative value means a smaller error when using real pointsets, and a positive value means a larger error, compared to using random pointsets. Hence, Tables 3.6 and 3.7 show that the errors of heuristics HVW 0.3, RMST and PD 0.3 are overestimated, whereas the errors of heuristics HVW 1.0 and PD 1.0 are underestimated. Since FLUTE is the most accurate of these heuristics and wirelength can only be overestimated when constructing spanning trees, all values in the tables are positive.

Table 3.7 can also be seen as a lookup table to improve the accuracy of existing RSMT cost estimators. For a given heuristic, more accurate wirelength values can be obtained by subtracting the errors reported in Table 3.7 from the wirelength of random pointsets.

3.5 An Improved WL Estimation Lookup Table

In this section, we present a lookup table (LUT) for improved wirelength estimation. Previously, Caldwell et al. [40] constructed a lookup table indexed with p and AR. We build upon this table and add $R(P)/B(P)$ as a third parameter dimension for improved accuracy of

[4]Errors are calculated relative to FLUTE, since FLUTE is optimal for $p \leq 9$ and introduces on average 0.16% RSMT error for $p \in [10, 17]$ [33].

Table 3.6: Percent error of heuristics vs. FLUTE for random and real pointsets.

p	Percent error of heuristics vs. FLUTE on random pointsets				
	HVW 0.3	RMST	PD 0.3	HVW 1.0	PD 1.0
4	1.93%	10.41%	12.35%	13.57%	44.05%
5	2.76%	11.15%	13.94%	16.14%	51.22%
6	3.38%	11.46%	14.96%	19.00%	56.94%
7	3.91%	11.52%	15.44%	21.08%	61.72%
8	4.47%	11.68%	16.02%	23.06%	65.29%
9	4.80%	11.77%	16.44%	24.72%	68.69%
10	5.07%	11.72%	16.71%	26.04%	71.06%
11	5.49%	11.80%	17.20%	27.34%	73.57%
12	5.57%	11.73%	17.24%	28.55%	75.81%
p	Percent error of heuristics vs. FLUTE on real pointsets				
	HVW 0.3	RMST	PD 0.3	HVW 1.0	PD 1.0
4	1.54%	8.96%	10.43%	15.29%	50.04%
5	1.92%	9.03%	10.90%	18.09%	58.06%
6	2.35%	9.31%	11.64%	20.37%	63.56%
7	2.99%	9.86%	12.77%	22.52%	68.10%
8	3.37%	10.19%	13.42%	24.42%	72.17%
9	4.01%	10.75%	14.58%	26.05%	74.38%
10	3.93%	10.38%	14.18%	28.00%	78.88%
11	4.19%	10.46%	14.44%	29.78%	82.00%
12	4.57%	10.60%	15.05%	30.89%	83.70%

wirelength estimation, as shown in Section 3.4. We use FLUTE to obtain the RSMT wirelength (see Footnote 4).

Table 3.8 shows a portion of our lookup table.[5] In the table, we report three sets of values for each p. The first row ($W1$) shows the FLUTE wirelength value by generating and averaging the wirelength over 1000 pointsets with $AR = \{1,2,4\}$. These values are equivalent to the wirelength values reported by Caldwell et al. [40]. The second row ($W2$) shows the FLUTE wirelength with specific $R(P)/B(P) = \{0.2,0.4,0.6,0.8\}$ (generated using Algorithm 4), averaged over 1000 pointsets. The third row ($W3$) is the percent error between $W1$ and $W2$, i.e., $\frac{W2-W1}{W1} \cdot 100\%$. For example, with $p = 6$ and $AR = 1$, we see that the $W1$ row contains the value 2.39; this is the single value for estimated RSMT cost given by [40]. The $W2$ row contains four values, 2.71,

[5]The entire lookup table is available at `http://vlsicad.ucsd.edu/~sriram/Final_WL_estimate_LUT.htm`

Table 3.7: Difference in % error between heuristics and FLUTE for real and random pointsets.

p	Difference in % error between real and random pointsets				
	HVW 0.3	RMST	PD 0.3	HVW 1.0	PD 1.0
4	-0.39%	-1.45%	-1.92%	1.72%	5.99%
5	-0.84%	-2.12%	-3.04%	1.95%	6.84%
6	-1.03%	-2.15%	-3.32%	1.37%	6.62%
7	-0.92%	-1.66%	-2.67%	1.44%	6.38%
8	-1.10%	-1.49%	-2.60%	1.36%	6.88%
9	-0.79%	-1.02%	-1.86%	1.33%	5.69%
10	-1.14%	-1.34%	-2.53%	1.96%	7.82%
11	-1.30%	-1.34%	-2.76%	2.44%	8.43%
12	-1.00%	-1.13%	-2.19%	2.34%	7.89%

2.37, 2.22 and 2.10; these are our estimated RSMT costs with $R(P)/B(P)$ ratios of 0.2, 0.4, 0.6 and 0.8, respectively. The $W3$ row gives the four corresponding percentage differences between the L-ness dependent estimates and the single estimate of [40]. We omit estimates for $p \in [2,3]$ since these RSMT costs are the half-perimeter wirelengths of the bounding boxes.

Runtime. We compare the runtime using different wirelength estimators: (i) FLUTE, (ii) our LUT, and (iii) rectilinear MST (RMST) implementation by Kahng et al. [48] using Prim's algorithm.[6] All algorithms are implemented using C and are executed on a 2.7 GHz Intel Xeon server with 8 threads. We evaluate using 500K real and random pointsets. Table 3.9 shows that our improved lookup table runs significantly faster than FLUTE for all values of $p \in [2,12]$ and faster than RMST except for $p = 4$. We believe that this significant speedup ($\sim$10$\times$), at the cost of small loss of accuracy of WL estimation, can be beneficial in modern-day contexts that involve very large designs, highly iterative methods, and a requirement for reduced tool turnaround times.

Accuracy. Table 3.10 reports the percent error, along with standard deviation and maximum error in wirelength estimates compared to FLUTE, using our lookup table (LUT), Caldwell LUT [40] and RMST implementation [48]. Percent error is calculated as $Error = \left(\frac{WL_{heur} - WL_{FLUTE}}{WL_{FLUTE}}\right) \cdot 100\%$. Our lookup table dominates that of [40] in all error metrics evaluated.

[6] We use an $O(n^2)$ implementation since it runs much faster than other $O(n\ log\ n)$ algorithms for small p.

Table 3.8: Wirelength lookup table using aspect ratio and $R(P)/B(P)$.

AR		1				2				4			
$R(P)/B(P)$		0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8	0.2	0.4	0.6	0.8
p=4	W1	2.14				2.25				2.60			
	W2	2.66	2.23	2.10	2.04	2.63	2.32	2.22	2.17	2.86	2.66	2.58	2.54
	W3	24.37	4.24	-2.03	-4.51	16.89	3.31	-1.48	-3.77	9.93	2.20	-0.76	-2.24
p=5	W1	2.27				2.36				2.69			
	W2	2.66	2.30	2.16	2.08	2.63	2.39	2.27	2.20	2.86	2.71	2.63	2.58
	W3	17.02	1.22	-4.80	-8.57	11.41	1.25	-3.67	-6.84	6.38	0.66	-2.12	-4.19
p=6	W1	2.39				2.48				2.78			
	W2	2.71	2.37	2.22	2.10	2.70	2.45	2.34	2.23	2.93	2.77	2.69	2.61
	W3	13.58	-0.90	-7.21	-12.00	8.95	-1.13	-5.81	-10.28	5.39	-0.30	-3.17	-6.21
p=7	W1	2.52				2.59				2.87			
	W2	2.78	2.44	2.27	2.13	2.77	2.53	2.39	2.26	3.00	2.82	2.73	2.64
	W3	10.13	-3.32	-9.84	-15.42	7.12	-2.38	-7.76	-12.70	4.56	-1.58	-4.73	-8.13
p=8	W1	2.63				2.69				2.96			
	W2	2.83	2.50	2.32	2.16	2.86	2.59	2.44	2.29	3.06	2.89	2.79	2.67
	W3	7.57	-4.81	-11.62	-17.96	6.18	-3.72	-9.30	-15.01	3.50	-2.33	-5.86	-9.96
p=9	W1	2.73				2.81				3.03			
	W2	2.90	2.57	2.37	2.18	2.92	2.67	2.49	2.32	3.13	2.95	2.83	2.69
	W3	6.35	-5.72	-13.07	-20.02	3.99	-5.14	-11.30	-17.51	3.34	-2.72	-6.49	-11.14
p=10	W1	2.84				2.91				3.13			
	W2	2.98	2.65	2.42	2.21	2.99	2.73	2.54	2.34	3.19	3.01	2.88	2.72
	W3	4.77	-6.80	-14.68	-22.07	2.79	-6.24	-12.71	-19.53	1.99	-3.70	-8.10	-13.16

However, our improved lookup table does give a higher standard deviation and maximum absolute error for higher values of p when compared to RMST. We note that the LUT errors reported in Table 3.10 are the averages of absolute errors, whereas RMST error is always positive. Figure 3.10 shows error distributions for the LUT and RMST estimators for $p = 9$.

WL estimation for pointsets with $p \in [2,3]$ using our LUT has no error. (In our studies, 68% of the nets in a 16nm implementation of ARM Cortex A53 have $p = 2$ or $p = 3$.) The WL estimation error using our LUT for $p \in [4,12]$ is $1-2\%$ lower than the error using RMST as an estimate. Thus, in terms of speed and accuracy, the new LUT provides a non-dominated wirelength estimate.[7]

[7]For $p \in [10,12]$ our LUT is approximately 10 times faster than FLUTE and twice as fast as RMST.

Table 3.9: Execution time (seconds) for 0.5M pointsets with $p \in [2, 12]$.

p	Random pointsets			Real pointsets		
	FLUTE	Impr. LUT	RMST	FLUTE	Impr. LUT	RMST
2	0.051	0.003	0.012	0.050	0.003	0.012
3	0.185	0.004	0.023	0.254	0.006	0.040
4	0.229	0.047	0.045	0.295	0.065	0.050
5	0.262	0.060	0.061	0.240	0.061	0.063
6	0.299	0.077	0.095	0.328	0.116	0.109
7	0.352	0.093	0.135	0.318	0.089	0.130
8	0.431	0.111	0.173	0.368	0.104	0.171
9	0.576	0.127	0.216	0.492	0.119	0.223
10	1.192	0.146	0.258	1.248	0.134	0.259
11	1.761	0.164	0.303	1.241	0.152	0.314
12	1.804	0.184	0.378	1.607	0.166	0.360

3.6 Conclusion

In this chapter, we have given a formal definition of the concept of *L-ness*, that is, the phenomenon that a net's pin locations within a real placement tend to be clustered towards two adjacent edges of the net's bounding box. We have provided empirical data showing the extent to which real pointsets have larger L-ness values than random pointsets. This data suggests at least the possibility that previous usage of random pointsets may have led to inaccurate assessments of RSMT heuristics and RSMT cost estimators. With this in mind, we describe a pointset generation function which can produce artificial pointsets that are similar to real placed pointsets. We furthermore present an improved lookup table for RSMT cost estimation that is sensitive to L-ness of a pointset; its implementation gives a speed-accuracy tradeoff point between FLUTE [33] and a fast rectilinear MST implementation [48].

Our ongoing and future works seek ways to exploit the L-ness attribute to achieve better estimates of routed WL or FLUTE heuristic RSMT costs – e.g., after placement and without any running of global/detailed routers. We are also exploring the direct optimization of an L-ness-aware wirelength estimate during placement. A high-fidelity wirelength predictor, congestion- and DRC-aware wirelength predictor, as well as hierarchical placement-based predictors are

Table 3.10: Error for $p \in [2, 12]$ with real pointsets.

p	Absolute Error			Std. Dev. of Abs. Error			Max. Absolute Error		
	Impr. LUT	Cald-well	RMST	Impr. LUT	Cald-well	RMST	Impr. LUT	Cald-well	RMST
3	0.00%	0.00%	6.13%	0.00%	0.00%	7.81%	0.00%	0.00%	33.31%
4	4.06%	5.61%	6.01%	3.62%	3.67%	6.49%	24.51%	28.19%	46.17%
5	4.47%	7.14%	6.20%	3.76%	4.48%	6.01%	24.94%	23.44%	42.73%
6	4.70%	8.07%	6.48%	3.95%	5.44%	5.66%	25.53%	25.24%	36.02%
7	4.93%	8.75%	6.82%	4.04%	6.41%	5.36%	28.20%	25.71%	34.60%
8	5.17%	9.85%	7.15%	4.21%	7.66%	5.14%	27.56%	31.46%	32.25%
9	5.28%	9.81%	7.73%	4.21%	7.88%	4.90%	30.95%	37.03%	32.13%
10	5.75%	11.38%	7.35%	4.69%	9.39%	4.76%	32.94%	42.16%	28.06%
11	6.00%	12.47%	7.14%	4.85%	10.37%	4.59%	37.01%	46.94%	27.46%
12	6.46%	12.62%	7.18%	5.32%	10.82%	4.58%	40.35%	52.94%	28.25%

also of interest. Other future directions include tree topology generation considering L-ness and objectives such as timing or power, as well as comprehension of driver vs. sink pin locations.

3.7 Acknowledgments

Chapter 3 is in part a reprint of A. B. Kahng, C. Moyes, S. Venkatesh and L. Wang, "Wot the L: Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, 2018, pp. 2-9.

I would like to thank my co-authors Professor Andrew B. Kahng, Christopher Moyes and Lutong Wang. I also thank the authors of ePlace [38] for providing the executable that we used in our experiments.

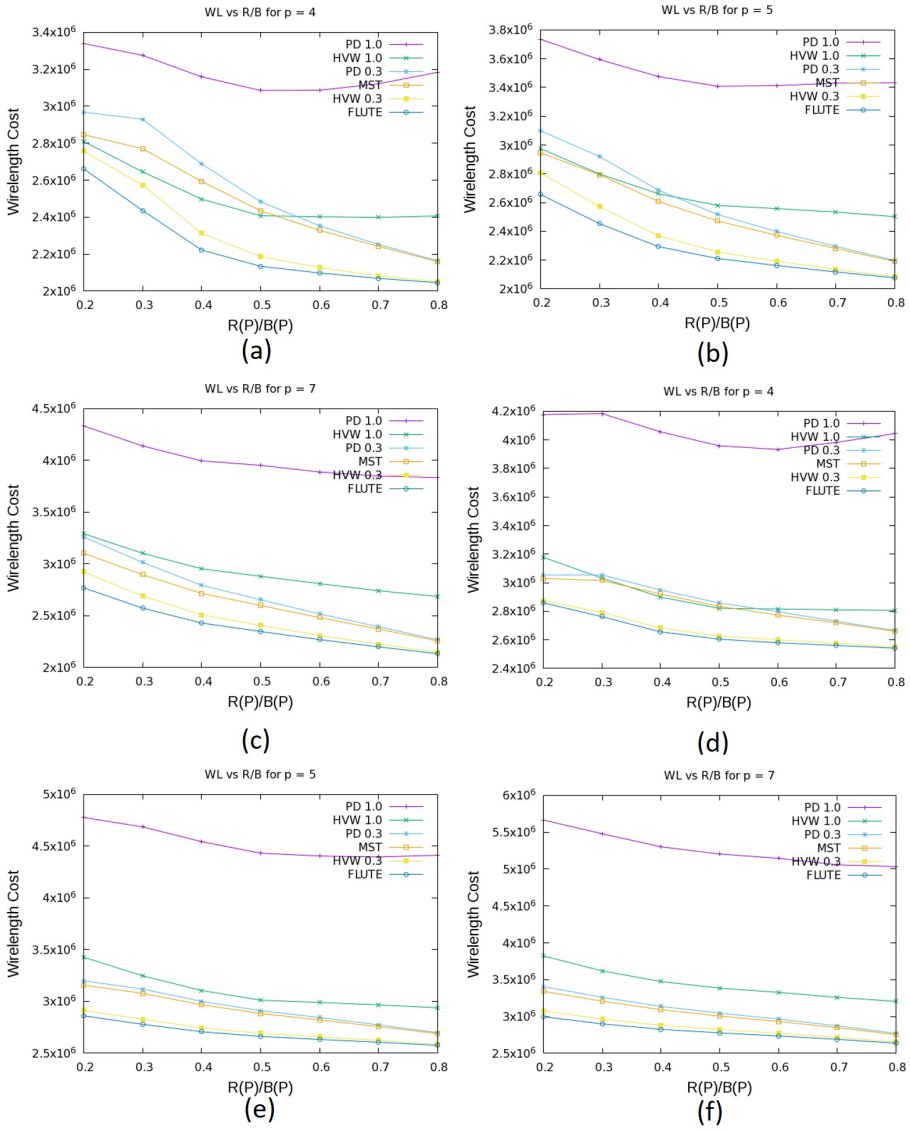


Figure 3.8: Change in wirelength with $R(P)/B(P)$ for nets with $AR = 1$ (a, b, c) and $AR = 4$ (d, e, f) for $p \in \{4, 5, 7\}$.

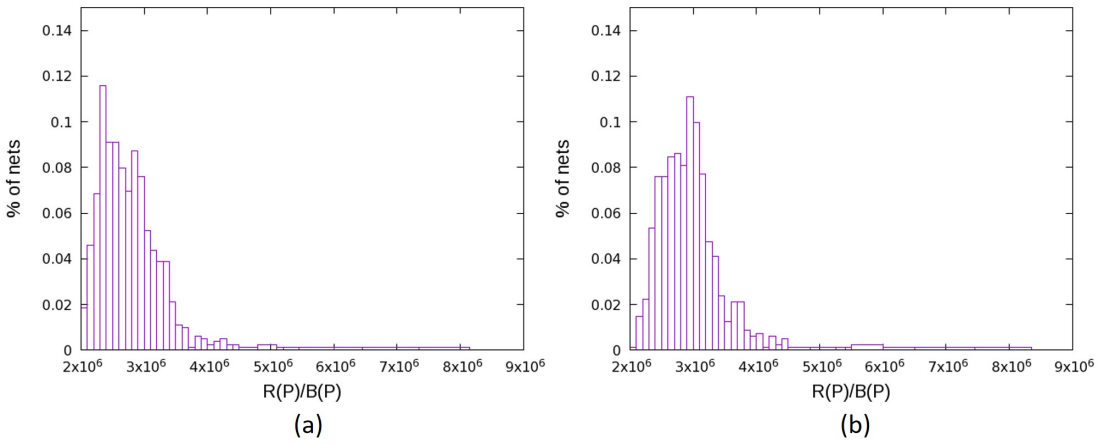


Figure 3.9: WL distribution functions for (a) real and (b) real' pointsets for $p = 12$.

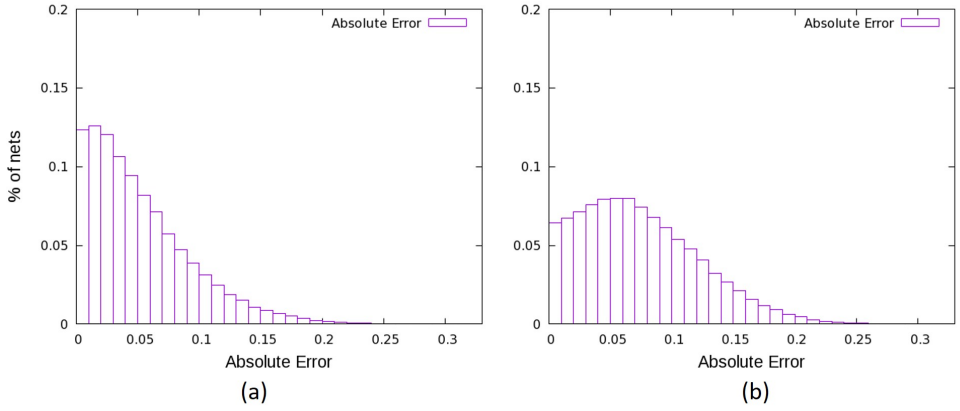


Figure 3.10: Error distributions with (a) lookup table and (b) RMST estimators for $p = 9$.

Chapter 4

Improving Model-Hardware Correlation by Layer-Balancing Clock Trees

This chapter presents two methods for improving model-hardware correlation in the back-end-of-line stack used for clock trees.

4.1 Introduction and Motivation

Manufacturing yield has been a major challenge in recent nodes, particularly for cost-driven SoC products. Significant yield losses can result from inaccurate modeling of process variations, and hence model-hardware correlation (MHC) is now a major focus for design and technology enablement. A particular challenge is created by back-end-of-line (BEOL), i.e., metal layer, variations that are not captured by modeling or signoff criteria. Such variations inevitably exist because not all combinations of layer variations can be examined in signoff, and because pessimism in signoff criteria harms design power, performance and area (PPA) as well as design convergence (tapeout schedule). When such variations occur, imbalanced metal layer usage in the clock paths of a launch-capture (LC) pair of flip-flops can result in hold violations in silicon,

leading to decreased yields. This challenge is unlikely to ease in coming nodes, since (i) clock paths and delay variations are dominated by wiring segments on layers that will not benefit from new patterning (extreme ultraviolet, EUV) technologies, and (ii) metal variation worsens due to the increased variety and tuning of layer geometries and adjacencies (e.g., in a foundry 7nm BEOL stack, a 64nm-pitch layer can be between similar layers, or on top of a 48nm- or 44nm-pitch layer, or under an 80nm-pitch layer, etc. [72]; see also [67]).

4.1.1 Benefit of Layer Balancing

In Figure 4.1(a), the flip-flops f_1 and f_2 have layer imbalance in their clock routes. D1, D2 and D3 are different metal layers, and the numerical labels give respective lengths of each metal segment. For the (ordered) LC pair (f_1, f_2), the difference in the lengths of metal segments on D1 and D2 can cause different delay variations in the clock paths, possibly leading to hold violations. The modified tree in Figure 4.1(b) is more robust since both launch and capture paths will experience similar insertion delay variations when manufacturing variation occurs.

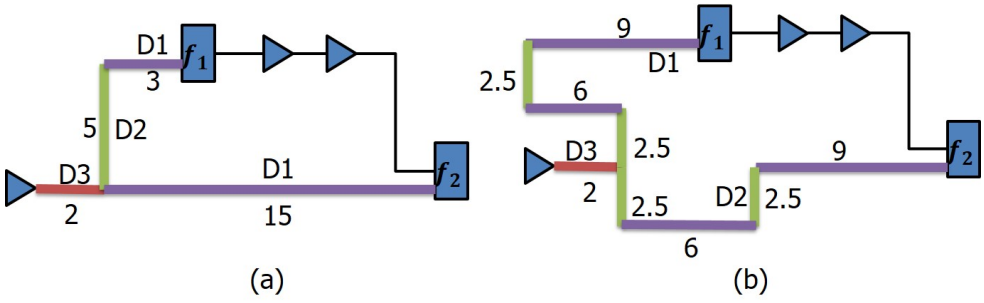


Figure 4.1: Illustration of (a) metal layer imbalance for a launch-capture (LC) pair of flip-flops, and (b) improved layer balance.

Our present work develops a new *layer-balancing*, *engineering change order* (*ECO*) clock optimization to mitigate model-hardware miscorrelation-induced hold timing failure and associated yield loss. Our optimization avoids costly increase in model guardbands, and reduces the number of hold fixes (e.g., buffer insertions which harm PPA and design schedule, especially for congested designs) that are needed when design robustness to model-hardware miscorrelation

(MHM) is validated before tapeout. Improving the balance of layer usage across hold-critical launch-capture pairs addresses the root cause of yield loss due to hold failures and – as we demonstrate below – can be accomplished with negligible PPA or schedule impact.

4.1.2 Design Methodology Context

Today's advanced hold signoff methodologies face severe challenges. As noted above, standard BEOL signoff corners cannot capture all combinations of layer variations, and by their nature do not capture model-hardware **mis**correlation. P&R tools do not analyze, and therefore cannot close, the design in light of *individual layer* variation impacts on clock paths (see also Section 4.2).[1] And, simply adding pessimism in the form of margin is too costly in terms of product PPA [57].

A world-leading semiconductor/SoC company tackles this combination of challenges today with a two-stage methodology (see Figure 4.2). In the first stage, traditional post-route optimization achieves hold closure. Then, model-hardware miscorrelation is simulated to find the set of LC pairs, denoted by P_h, that violate hold constraints under miscorrelation,[2] by applying an additional hold margin to the LC pairs in P_h, before performing another round of signoff STA and ECO fixing. The additional hold margin is determined by modeling the additional layer variation as a modified delay per unit length of wire, specific to each metal layer. The details of the hold margin calculation can be found in Section 4.4.2.

At the semiconductor/SoC company, the additional hold margin typically exposes new, "surprise" hold violations at approximately 2% of all flip-flops in the design. (Thus, the first-stage conventional hold fix substantially reduces the scale of the hold closure problem: it removes from consideration timing paths that have sufficient hold margins.) The "surprise" violations (which

[1] Applying resistance and capacitance derating factors is not straightforward when attempting to derate per metal layer, as opposed to per net.

[2] I.e., this additional simulation captures the possibility of unrelated RC variations on the metal layers of interest. It exposes scenarios where, e.g., one stage of register pipelining is vulnerable to an M4 wire resistance excursion, while another stage is vulnerable to an M5 excursion.

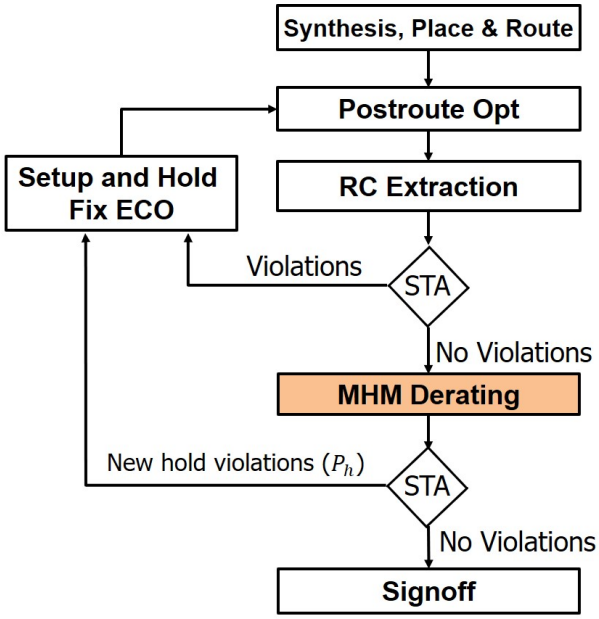


Figure 4.2: Two-stage methodology followed by a leading semiconductor company.

occur under model-hardware miscorrelation) must then be fixed using standard mechanisms such as hold buffer insertion. The central goal of our work is to reduce the number of flip-flops at which such hold violation "surprises", and required hold fixes, occur.[3]

4.1.3 Contributions of This Work

Our work mitigates the above-described model-hardware miscorrelation and yield loss due to hold failure, by balancing the metal layer usage of the clock routes to each hold-critical LC pair. Importantly, given that leading P&R tools perform highly integrated co-optimizations of clock routes, useful skews, datapaths and detailed routing, we must restrict our scope to "repair" of the clock tree solution generated by a commercial EDA tool. That is, we achieve improved layer

[3]Particularly in highly congested, cost- and schedule-driven product designs, it is extremely desirable to avoid addition of hold buffers. This is because even if the number of hold-violating endpoints is small, fixing them is not easy. To our understanding, the first stage of "conventional" hold fixing (i.e., based on nominal BEOL signoff criteria), often uses up the reserved area for hold-fixing buffers which was estimated as part of the design methodology. (This step can take multiple weeks of schedule.) Then, the second-stage simulation of model-hardware miscorrelation due to wire variation exposes the "surprise" hold violations. If there are not sufficient cell sites and routing area to achieve the new hold fixes, additional weeks of schedule can be consumed.

balance only by changing layer assignments and rerouting the affected clock tree wire segments. Our optimization thus minimizes overhead and preserves convergence for methodologies based on commercial EDA tools. Our main contributions are (i) we propose a linear program (LP) to find the optimum layer usage and segment length for all LC pairs in a given skew group; (ii) we develop a full methodology to *realize* the LP-generated solution as a layer-balanced clock tree; and (iii) we show how the realized LP solution reduces variation between clock paths of launch-capture flip-flops. Overall, our approach improves the layer balance between LC pairs by 14% while incurring only 0.2% clock wirelength and 0.08% clock power overheads.

The following sections of this chapter are organized as follows. Section 4.2 briefly summarizes related works. Section 4.3 formally states our layer-balancing ECO clock routing problem, and the commercial EDA tool/flow context for our optimization. Section 4.4 describes our experimental setup and results, and we conclude in Section 4.6.

4.2 Related Work

Synthesis of low-skew, balanced clock trees has been studied by dozens of authors in both academia and industry, as exemplified by [51][53][54][56][58]. The clock tree synthesis (CTS) literature is broadly reviewed in Chapters 42 and 43 of [50]. CTS has been studied in the 3D IC context, e.g., [60] balances clock tree skews across different thermal profiles. Non-tree constructions such as [66] and [68] propose techniques to create a skew-balanced clock tree and reduce variations by link-based buffer insertion. More generally, [69] and others have studied the use of clock meshes to reduce variations across the clock network. Mesh topologies can reduce skew between launch-capture flip-flops, but incur higher dynamic power and reduced tunability of launch-capture skews. Hence, tree constructions are still dominant, especially in cost- and power-driven mobile SoCs at advanced nodes.

The problem of variation in the metal layers used for clock routing has been understood

for decades. For example, Liu et al. [59] note the impact of metal variations on clock skew, and [61] modify the DME algorithm [51] to achieve clock topologies that are more robust to variation. The well-known ISPD contests [78] of 2009-2010 acknowledge metal variation in their respective problem formulations. Such works are complemented by variation-aware interconnect analysis approaches, e.g., [70] models clock skew as a function of device, interconnect, and system parameter variations. However, we are aware of only limited previous literature that directly attacks a *layer-balancing* formulation as we do here. Agnihotri et al. [49] propose a method to achieve specified routing utilizations on different metal layers by assigning costs to layers. Carothers et al. [52] propose a method for layer-balancing of nets on a printed circuit board (PCB). They perform redundant routing on multiple layers and iteratively remove routes to achieve a layer-balanced routing solution. The layer-balancing formulation that we study is new, particularly in light of its ECO and hold failure-motivated context.

There is an obvious tight coupling between clock network design (which defines clock paths to flip-flops) and datapath design (which defines minimum- and maximum-delay combinational paths between each sequentially-adjacent pair of launch and capture flip-flops). Especially as design tools must contribute more to power, performance and area (PPA) improvements in advanced nodes, the leading P&R tools cited by [77], including those of [73][74][79][81], have all seen tight integration of clock delivery, datapath logic, and design closure optimizations. Rationales for this tight integration are detailed in, e.g., [55].

This tight integration of clock distribution and skew optimization with datapath optimization presents a challenge for physical design teams: it is difficult to exit and re-enter the P&R tool in order to improve the balance of CTS metal layer usage on hold-critical LC pairs (i.e., for robustness to yet-unknown model-hardware miscorrelations). Yet, as noted above, simple guardbanding of metal parasitics (e.g., in the RC models used within the design closure loop) would incur unacceptable PPA overheads. For these reasons, we must address the layer-balancing requirement *in an ECO context*, without disrupting the P&R tool's high-quality co-optimization

of clock distribution and datapaths.

Last, in light of our use of linear programming in this work, we note that Oh, Pyo and Pedram [62][63][64] have previously used linear programming to construct bounded-delay routing trees. Our formulation, described in the next section, differs in that it aims to minimize layer imbalance for each LC pair. Furthermore, our LP solution gives both layer information, as well as locations of Steiner points and buffers.

4.3 Our Approach

In this section, we first describe the overall flow of our optimization. Then, we propose a linear programming (LP)-based optimization for layer balancing of clock trees. Last, we show our methodology to *realize* a well-formed clock tree routing according to our LP solution.

4.3.1 Overall Flow

Figure 4.3 shows the overall flow of our optimization. Conventional physical implementation performs placement, CTS, post-CTS optimization, routing and post-routing optimization in sequence. We insert our layer balancing clock tree modifications between CTS and post-CTS optimization, as follows. We first extract the clock tree topology and perform the LP optimization. We then realize the clock tree according to the LP solution and the methodologies described in Section 4.3.3. Given the new, layer-balanced clock tree, we go back to the conventional flow until we get the post-routing optimized design. The evaluation method is detailed in Section 4.4.2.

4.3.2 LP Formulation

The notations we use in our work are given in Table 4.1. Figure 4.4 shows a clock tree with two LC pairs (f_1, f_2) and (f_2, f_3). Specifically, flip-flop f_2 is both on a launch path and

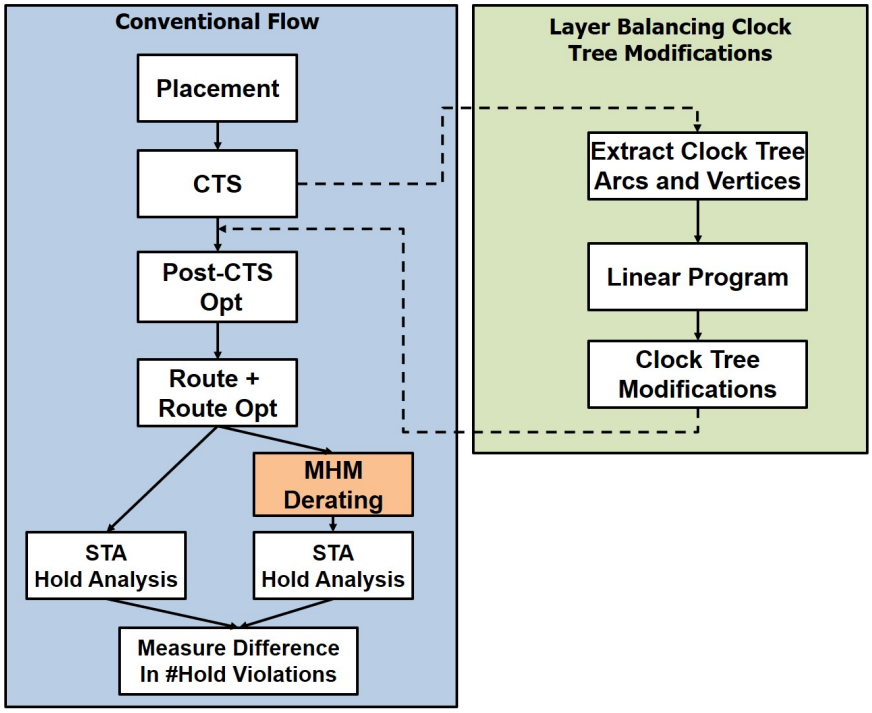


Figure 4.3: Our flow.

a capture path, hence layer balancing needs to consider both in a proper way. We define layer imbalance (LI), total imbalance (TI) and maximum imbalance (MI) for a LC pair as follows.

Layer Imbalance. For a critical LC pair, for a specific metal layer j, layer imbalance (LI) is the absolute difference in routed wirelength on metal layer j between launch and capture clock paths.

Total Imbalance. For a critical LC pair, total imbalance (TI) is the sum of layer imbalance across all metal layers for clock tree routing.

Maximum Imbalance. For a critical LC pair, maximum imbalance (MI) is the maximum of layer imbalance across all metal layers for clock tree routing.

Layer Balancing Problem (LBP). Given a fixed routing tree topology and fixed flip-flops, relocate Steiner points and buffers, and reroute the clock tree to minimize the weighted sum of total and maximum imbalance for all critical LC pairs.

Table 4.1: Notations.

Notation	Meaning
V	set of $n+1$ vertices in the clock tree, $V = \{v_0, ...v_f, v_{f+1}, ..., v_n\}$
v_0	the root of the clock tree
f	number of flip-flops (FFs) in the design
e_k	edge between adjacent vertices
$P_{i,i'}$	set of clock tree edges on the unique path from v_i to $v_{i'}$.
$LCA(i,i')$	lowest common ancestor (vertex) for $(v_i, v_{i'})$.
S_{LC}	set of critical LC pairs (v_i, v_i')
$m_j(e_k)$	total routed wirelength for edge e_k on metal layer j, $1 \leq j \leq \|M\|$, $\|M\|$ is the total number of clock routing layers
α	weighting parameter, $(0 \leq \alpha \leq 1)$
BL, BU	lower and upper bounds on clock path length
BD	upper bound on displacement of each vertex v_i
$plen_x(i,i')$	sum of arc length (x direction) from v_i to $v_{i'}$
x_i, y_i	the x- and y-coordinates of v_i
x_{i_0}, y_{i_0}	the initial values of the x- and y-coordinates of v_i
$dist_x(i,i')$	$x_i - x_{i'}$ (can be negative)
$D_{ii'}$	total displacement of vertices on the unique path from v_i to $v_{i'}$
P_h	set of LC pairs with hold violation due to model-hardware miscorrelation

$$\text{Minimize} \sum_{\forall (v_i, v_i') \in S_{LC}} TI_{ii'} + \alpha \cdot MI_{ii'} + \beta \cdot D_{ii'} \tag{4.1}$$

where $TI_{ii'}$ is the total imbalance for critical LC pair (v_i, v_i'), $MI_{ii'}$ is the maximum imbalance for critical LC pair (v_i, v_i') and $D_{ii'}$ is the total displacement of vertices in the launch-capture paths of critical LC pair (v_i, v_i'). α and β are weighting factors that we study in Section 4.4.3. We include the total displacement in the objective function to minimize the disruption to the initial clock tree, since the insertion delay, skew and congestion are all comprehended by the "black-box" tool.

The constraints used in our formulation are as follows. Constraints C5 to C10 are given for x directions, and should be applied to y directions accordingly.

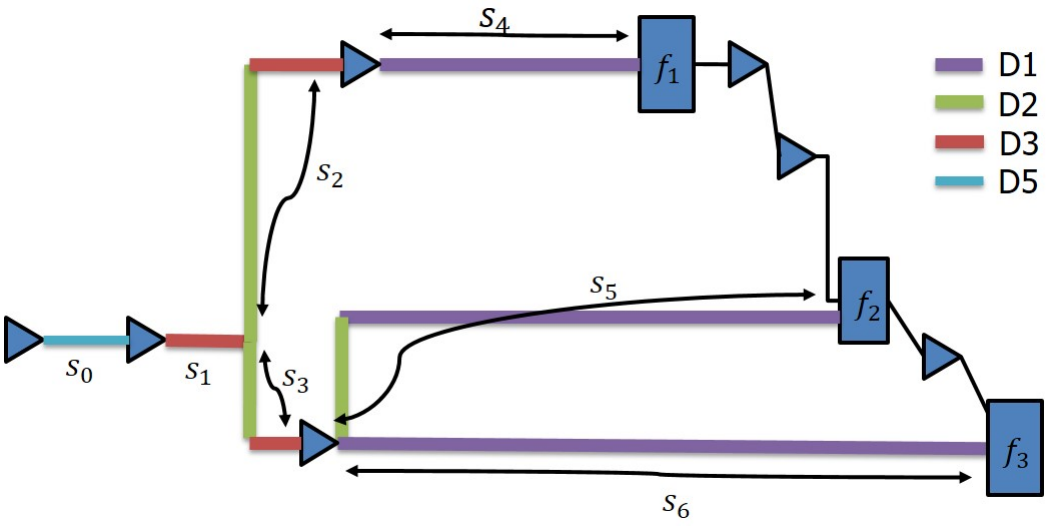


Figure 4.4: Metal layer imbalance among multiple launch-capture pairs.

$$\text{C0: } LI_{j,ii'} = |\sum_{e_k \in p_{ii^\dagger}} m_j(e_k) - \sum_{e_{k'} \in p_{i'i^\dagger}} m_j(e_{k'})|, \quad v_{i^\dagger} = LCA(i,i') \tag{4.2}$$

$$\text{C1: } TI_{ii'} = \sum_j LI_{j,ii'} \tag{4.3}$$

$$\text{C2: } MI_{ii'} = \max_j LI_{j,ii'} \tag{4.4}$$

$$\text{C3: } \forall_{j,k}\, m_j(e_k) \geq 0 \tag{4.5}$$

$$\text{C4: } BL_i \leq plen_x(0,i) + plen_y(0,i) \leq BU_i, \quad \forall i \in (0,...,f] \tag{4.6}$$

$$\text{C5: } BL_{ii'} \leq plen_x(i,i') \leq BU_{ii'}, \quad \forall (i,i') \tag{4.7}$$

$$\text{C6: } -plen_x(i,i') \leq dist_x(i,i') \leq plen_x(i,i'), \quad \forall (i,i') \tag{4.8}$$

$$\text{C7: } plen_x(i,i') = \sum_{M_x} \sum_{e_k \in P_{ii'}} m_j(e_k), \quad \forall (i,i') \tag{4.9}$$

$$\text{C8: } dist_x(i,i') = x_i - x_{i'}, \quad \forall (i,i') \tag{4.10}$$

$$\text{C9: } BD \leq x_i - x_{i_0} \leq BD, \quad \forall i \in [f+1,...,n) \tag{4.11}$$

$$\text{C10: } D_{ii'} = \sum x_j - \sum x_{j_0}, \ \forall i,i' \in (0,...,f], \forall j \in [f+1,...,n) \tag{4.12}$$

C0 is layer imbalance for a LC pair on a particular layer. C1 is total layer imbalance for a LC pair. C2 is maximum layer imbalance for a LC pair among all layers. C3 is routing segment length constraint. C4 is total linear delay constraint. C5 is arc length constraint, e.g., for each arc, we allow the new total arc length to be within 95% to 105% of original arc length. C6 is feasibility constraint, i.e., distance between each pair of connected vertices should be smaller than its arc length. C7 calculates the arc length. C8 calculates the distance between each pair of vertices. C9 is the bounds for the coordinate of movable vertices, i.e., Steiner points and buffer locations. $\min(i)$ and $\max(i)$ determine the maximum allowed coordinate range for each vertex, e.g., no more than $5\mu m$ from its original location. C10 is the total displacement for all vertices on path (i, i'). We note that x_i where $i \in (0, ..., f]$ is a known, fixed coordinate. However, x_i where $i \in [f+1, ..., n)$ is unknown, and is a variable in LP to be solved.

By solving LP, we can obtain: (i) $m_j(e_k)$, and (ii) x_i and y_i. Thus, unlike [62][63][64] where we need to use deferred-merge embedding (DME) to determine Steiner points and buffer locations, we can get these locations directly from the LP solution. Also, by specifying C5 and C9 according to the original placement and routing solution, we can ensure the feasibility of LP since the original CTS as input is always a feasible solution.

4.3.3 Routed Clock Tree Realization

Given the arc length and vertex locations from an LP solution, we modify the clock tree accordingly, using the following steps:

1. We rip up the clock routes with changes in vertex locations or arc lengths.

2. For all vertices with location changes:

    - We move all segments incident to the vertex to the new vertex location using internal tool commands. We then "complete" the routes using ECO routing commands in the commercial tool.

- For each arc incident at the moved vertex, if the arc length (obtained from the LP solution) is longer than the completed route, we create *U-detours* to achieve the required arc length. Figure 4.5 illustrates the insertion of *U-detours*, with detour lengths (increase) of Δy in vertical direction and Δx in horizontal direction. For each horizontal and vertical detour in the arc, we choose a segment *S*, which is perpendicular to the direction of increase, to make a "cut" and insert a detour. This segment is chosen greedily according to the order of segments stored in the tool internal database, as long as it has a minimum length of $0.4\mu m$ (this minimum length requirement is added to avoid cut spacing violations between the vias added in the U-detour). We then insert *U-detours* by (i) cutting the segment *S* at $1/3$rd and $2/3$rd lengths from the lower left corner of the segment[4], (ii) removing the portion of wire between the two cuts, and (iii) then adding a "U" shape using metal segments on the required layer. We always apply upper or right *U-detours* (as shown in Figure 4.5), and we skip the arc if no legal[5] *U-detours* can be made.

3. For all vertices without location changes:

   - If there is redistribution of lengths between layers in a particular direction (horizontal or vertical), layer swap is performed for the required metal segment length. The segment to swap be two layers down / up the required layer, with a length equal or larger than the required length, and is chosen greedily based on the order of segments stored in the tool database. If segment length is longer than the required length, we cut the segment into three subsegments. The middle segment length is equal to the required length, with the remaining two segments being of equal length. We round the segment lengths to ensure on-grid routing, and swap the middle segment to the required layer with via insertions. We skip the arc if no legal *swaps* can be made.

[4] All wire edits made are snapped to the routing grid using tool internal settings to ensure on-track routing.
[5] We only check for and avoid shorts.

- If there is an increase in length of the arc for a single or multiple metal layers, $U - detours$ are added for each increase/layer, as described above.

4. These modified segments are set with a "fixed" attribute in all of the subsequent P&R stages. Any DRCs created by our clock tree realization method are handled by running ECO routing commands after all modifications are completed.

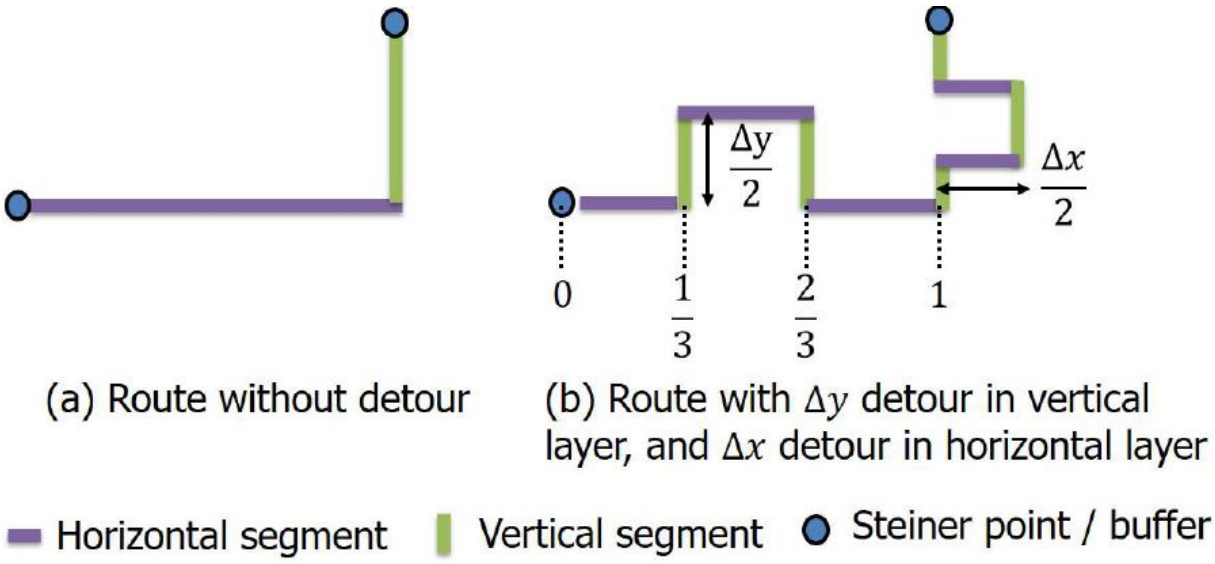


Figure 4.5: Illustration of *U-detours*: (a) original route without detour; and (b) route modification with detour length of Δy in vertical layer, and detour length of Δx in horizontal layer.

4.4 Experimental Setup, Metric and Results

We now present our experimental setup, evaluation metric, and results.

4.4.1 Experimental Setup

Table 4.2 lists the designs used in our experiments and evaluations, and reports the design block, #instances, #flip-flops, clock period, placement utilization and clock wirelength

of each design.[6] We create placements with low utilizations to generate clock trees with large WL, to model large clock trees in commercial SoC designs. We implement these testcases in a commercial foundry 14nm technology with a 9-track cell library and an 11-metal layer stack. Logic synthesis, and P&R flow are performed with 2016 and 2017 releases of industry-leading tools from major EDA vendors, respectively. These vendors have instructed us to not give more specific information regarding tools/commands that we use. Our code is written in Tcl, and runs on top of the clock-routed P&R tool database. The experiments are performed with 8 threads on a 2.7GHz Intel Xeon server.

Table 4.2: Testcases.

Testcase	#Insts	#FFs	Clk. Per.	Util	ClkWL
THEIA [80] I	154K	13K	0.7 *ns*	7%	75746 μm
THEIA [80] II	154K	13K	0.7 *ns*	52%	33522 μm
LEON3MP [65]	378K	108K	2.0 *ns*	32%	338900 μm

In our experiments, we guide the clock tree generated by the commercial P&R tool to be routed in layers M5, M6 and M7, with only pin accesses going below to reach cell pin access layer. Layers M5 and M7 are vertical routing layers, and M6 is a horizontal routing layer. This reflects an industrial design methodology, where trunk-and-branches topologies and the use of higher metal layers help achieve low insertion delays. To set up the LP for layer balance improvement across multiple hold-critical paths, we extract lengths and layer information of each segment in the clock tree. For each LC pair, we gather layer distributions of each arc e_k in the clock paths and annotate each vertex v_i. We also gather initial positions of the vertices. We then construct the LP objective function and constraints. We use CPLEX v12.63 [76] as our LP solver. The Tcl script calls CPLEX internally and a solution file is written out. The Tcl script then parses the solution file, compares the lengths of arc and locations of vertices with the initial clock tree, and implements changes to the clock tree. Our code takes 8 minutes for the entire flow (i.e., clock tree extraction, LP and realization of LP solution) on a design with 154K instances and 13K flip-flops.

[6] THEIA I and II represent the same design with different placement densities.

4.4.2 Evaluation Metrics

Effectiveness of our layer-balancing ECO optimization is ultimately measured by how well it reduces the hold violation "surprises" discussed in Section 4.1. As noted above, saving these additional hold violations will result in better yield, a reduced number of hold buffers needed during final design closure steps, and improved design schedule. Our metrics of interest is evaluated as follows.

Hold Margin for MHM modeling. We calculate additional hold margins to expose the MHM-induced hold violations. The hold margin is calculated as follows. The *increment* of wire delay δ_j for a given metal segment in layer j is calculated as

$$\delta_j \;=\; \tau_j \cdot \lambda_j \tag{4.13}$$

where τ_j is the nominal wire delay per unit length of metal in layer j, and λ_j is the metal segment length. If layer j is impacted due to variation, then we calculate the derated wire delay τ'_j per unit length by applying a blanket derate σ_j to τ_j, as shown in Equation (4.14). We then obtain the derated *increment* of wire delay δ'_j. We obtain the difference in wire delays, Δ_j, according to Equation (4.15).

$$\tau'_j \;=\; \tau_j \cdot \sigma_j \tag{4.14}$$

$$\Delta_j \;=\; \delta'_j - \delta_j = \lambda_j \cdot (\tau'_j - \tau_j) \tag{4.15}$$

If ΔL_{wire}, ΔC_{wire} and ΔD_{wire} denote the change in wire delays in the launch, capture and data paths respectively, the change in hold margin ΔT_{hold} for an LC pair is calculated as

$$\Delta T_{hold} = \Delta L_{wire} + \Delta D_{wire} - \Delta C_{wire} \tag{4.16}$$

$$= \sum_{1 \le j \le l} \Delta_{j,launch} + \sum_{1 \le j \le l} \Delta_{j,data} - \sum_{1 \le j \le l} \Delta_{j,capture} \tag{4.17}$$

This additional margin is then applied to all paths which have clock routes on layers susceptible to variation.

Evaluation. Our metrics are then evaluated as follows.

1. We finalize physical implementation using the tool-generated clock tree, and the flow on the left side of Figure 4.2. We apply STA (left-most path in Figure 4.2) to calculate the total number of hold violations (H_1), and total negative slack (TNS_1) at post-route stage.

2. We apply deratings for specific metal layers of interest (e.g., M6 or M7) using Equations (4.13) - (4.17), as indicated by the red box in Figure 4.2. We then again apply STA to calculate the total number of hold violations (H_2), and total negative slack (TNS_2). The number of hold violations due to model-hardware miscorrelation is $H_i = H_2 - H_1$, and the set of LC pairs with hold violations is P_h. The degradation in TNS due to MHM is $TNS_i = TNS_2 - TNS_1$.

3. In our ECO flow, shown in the green boxes of Figure 4.2, we layer-balance the clock nets of the LC pairs in P_h, and detail-route these modifications before continuing the physical implementation flow with our optimized, layer-balanced clock tree.

4. We perform STA and calculate the number of hold violations (H_3), and total negative slack (TNS_3) at post-route stage.

5. We apply deratings for specific metal layers of interest (e.g., M6 or M7) using Equations (4.13) - (4.17), and calculate total number of hold violations (H_4), and total negative slack (TNS_4). Again, the number of hold violations due to clock MHM is $H_f = H_4 - H_3$. The degradation in TNS due to MHM is $TNS_f = TNS_4 - TNS_3$.

6. We evaluate the benefit of our layer balancing using H_i, H_f, TNS_i and TNS_f.

4.4.3 Results

In this section, we analyze the improvement in layer balance with varying parameters, and show hold violations due to our clock tree modifications with regard to our evaluation metric.

Sensitivity to Parameters. We investigate the sensitivity of layer balance to different optimization parameters, and choose appropriate parameter values for our experiments. We sweep the following parameters: (1) maximum imbalance weighting factor α, (2) total displacement weighting factor β, (3) maximum vertex displacement bound (*BD*) and (4) upper bound on clock path length increase (*BU*, normalized).

Table 4.3 shows the normalized layer imbalance (and maximum imbalance) improvement per LC pair, on M5, M6 and M7 layers as we sweep α from 0 to 15%. For each LC pair, layer imbalance (*LI*) is the absolute difference in routed wirelength between launch and capture clock paths. From Table 4.3, we see that layer imbalance improves around 14/14/11% on average for each LC pair on M5/M6/M7 layers, respectively. Maximum imbalance improves slightly as α increases. Since α does not affect the results much, we use $\alpha = 0.02$ (2%) arbitrarily for our experiments.

Table 4.4 shows the layer balance improvements with different values of β, the weighting factor for vertex displacement. We see that large β restricts the vertex movement and has a negative impact on layer balance improvement. $\beta > 1$ prohibits all vertex movement and results in around 4% less layer balance improvement. Hence, we use $\beta = 0.01$ for our experiments.

Table 4.5 shows the effect of the vertex displacement bound *BD* on normalized layer imbalance improvement per LC pair, #vertices moved, and total vertex displacement. We see that layer imbalance improves by up to 4% on M6 and M7 by enabling vertex movement. We also see that large *BD* does not improve the layer balance significantly. In lieu of minimum perturbation to the original clock tree solution, we use $BD = 5\mu m$ for our experiments. In our studies, we also find that buffer movement does not change the layer balance in any significant way (e.g., allowing buffer movement results in only 4 clock tree buffers being moved and 0.04 μm improvement in

average layer imbalance per LC pair), and is disruptive to the detailed route of the initial CTS solution. Therefore, we allow only the movement of Steiner points, while keeping the buffer location fixed in our experiments.

Table 4.6 shows the sensitivity of layer balance to different values of BU. A larger $BU = 1.20$ results in more than $6\times$ layer balance improvement than $BU = 1.03$, but also increases the average wirelength per clock path by up to 4.45%. Since increased clock WLs lead to higher capacitances and higher clock power, we use $BU = 1.10$ for our experiments.

Table 4.3: Sensitivity to α.

α	Norm. *LI* per layer			Norm. *MI* per layer		
	M5	M6	M7	M5	M6	M7
0.00	-14.27%	-14.04%	-11.51%	-5.16%	-11.39%	-7.02%
0.01	-14.36%	-14.02%	-11.51%	-5.16%	-11.42%	-7.02%
0.02	-14.36%	-14.02%	-11.51%	-5.16%	-11.42%	-7.02%
0.05	-14.36%	-13.94%	-11.49%	-5.16%	-11.85%	-7.02%
0.10	-14.36%	-13.77%	-11.49%	-5.16%	-11.95%	-7.02%
0.15	-14.36%	-13.77%	-11.46%	-5.16%	-11.92%	-7.02%

Table 4.4: Sensitivity to β.

β	Norm. *LI* per layer			Vertex Disp.	
	M5	M6	M7	#	Dist (μm)
0.01	-14.36%	-14.02%	-11.51%	236	258
0.1	-14.52%	-11.47%	-8.76%	74	71
1	-13.63%	-9.77%	-6.82%	0	0
10	-13.63%	-9.77%	-6.82%	0	0
100	-13.63%	-9.77%	-6.82%	0	0

Table 4.5: Sensitivity to *BD*.

BD	Norm. *LI* per layer			Vertex Disp.	
	M5	M6	M7	#	Dist (μm)
0	-13.63%	-9.77%	-6.82%	0	0
3	-14.36%	-13.94%	-11.31%	234	237
5	-14.36%	-14.02%	-11.51%	236	258
10	-14.36%	-14.02%	-11.56%	236	265

Table 4.6: Sensitivity to BU.

BU	Norm. LI per layer			Avg. WL incr. per LC pair (μm, %)
	M5	M6	M7	
1.03	-4.38%	-4.56%	-3.85%	2.72 (+0.52%)
1.05	-7.30%	-7.30%	-6.10%	5.06 (+0.96%)
1.10	-14.36%	-14.02%	-11.51%	11.86 (+2.25%)
1.20	-27.58%	-25.95%	-20.84%	23.42 (+4.45%)

Table 4.7: Experimental results on evaluation metrics.

Testcase	σ_j	M5 derated				M6 derated				M7 derated			
		H_i	H_f (Δ%)	TNS_i	TNS_f (Δ%)	H_i	H_f (Δ%)	TNS_i	TNS_f (Δ%)	H_i	H_f (Δ%)	TNS_i	TNS_f (Δ%)
THEIA I	0.05	0	0	0	0	30	29 (-3.33%)	-0.651	-0.624 (-4.17%)	85	92 (+8.24%)	-1.806	-1.739 (-3.73%)
	0.10	1	1 (0%)	-0.002	-0.006 (+248%)	202	190 (-5.94%)	-6.746	-6.798 (+0.78%)	199	214 (+7.54%)	-10.32	-11.27 (+9.19%)
	0.20	14	15 (+7.14%)	-0.156	-0.173 (+10.4%)	502	493 (-1.79%)	-35.45	-32.90 (-7.22%)	522	515 (-1.34%)	-47.81	-48.63 (+1.72%)
	0.30	62	63 (+1.61%)	-0.932	-0.915 (+1.91%)	783	793 (-1.28%)	-78.69	-72.80 (-7.38%)	737	785 (+6.51%)	-100.0	-101.7 (+1.74%)
THEIA II	0.05	0	0	0	0	0	4	0	-0.024	2	1 (-50.0%)	-0.037	-0.006 (-84.3%)
	0.10	0	0	0	0	29	17 (-41.4%)	-0.250	-0.444 (+77.7%)	32	7 (-78.1%)	-0.671	-0.147 (-78.1%)
	0.20	0	1	0	-0.045	142	84 (-40.8%)	-4.892	-3.548 (-27.5%)	81	103 (-27.2%)	-4.243	-2.691 (-36.6%)
	0.30	0	7	0	-0.112	312	175 (-43.9%)	-14.97	-9.946 (-33.6%)	170	200 (-17.7%)	-10.19	-8.372 (-17.8%)
LEON3MP	0.05	0	0	0	0	935	911 (-2.57%)	-34.41	-30.26 (-12.1%)	1083	1072 (-1.02%)	-35.24	-32.36 (-8.18%)
	0.10	0	0	0	0	2141	1994 (-6.87%)	-134.7	-123.6 (-8.28%)	2623	2575 (-1.83%)	-156.1	-147.3 (-5.65%)
	0.20	2	3 (+50.0%)	-0.002	-0.018 (+862%)	4221	4160 (-1.45%)	-414.3	-388.5 (-6.23%)	4526	4417 (-2.41%)	-493.5	-469.4 (-4.89%)
	0.30	21	42 (+100%)	-0.122	-0.300 (+145%)	6690	6639 (-0.76%)	-749.7	-711.7 (-5.08%)	6527	6426 (-1.55%)	-883.8	-842.2 (-4.70%)

Evaluation results. Table 4.8 shows the layer balance improvement for our testcases with the parameters chosen above. We achieve up to 14% improvement in layer balance (averaged over all LC pairs), with less than 0.5% increase in overall clock tree WL.

Then we perform timing analysis (with and without derates) to study the improvement in hold violations. Table 4.7 reports the hold violations at different stages of our analysis flow, as described in Section 4.4.2. We evaluate the change in number of hold violations and hold TNS (H_i, H_f and TNS_i, TNS_f) as we derate all metal segments of a particular metal layer j by σ_j. We see that the number of hold violations saved differs with different derating and that there is no particular trend with layer balance improvement. Unfortunately, Table 4.7 shows that our present realization of the clock tree modifications in the P&R tool do not yet preserve the benefits of a better layer-balanced clock tree. Even though our LP gives us a better, globally layer-balanced clock tree, the various internal optimization steps performed by the P&R tool mask the benefits. Thus, when the LP solution is realized in DRC-clean final routing, we are unable to consistently save hold violations for all our experiments.

Table 4.8: Normalized layer balance improvement for all testcases.

Testcase	Norm. *LI* per layer			Avg. clock tree WL incr. per path (μm)
	M5	M6	M7	
THIEA I	-14.36%	-14%	-11.5%	0.16%
THIEA II	-14.6%	-15.9%	-18.3%	0.50%
LEON3MP	-13.52%	-13.1%	-10.8%	0.43%

4.5 Feedback-based Methods for Better Yield

As seen above, layer balancing has little impact when restricted, but is disruptive when not restricted. We make the following observations.

- Layer balancing is performed after CTS, but the subsequent optimization and signal routing stages are not layer-balance-aware. The rest of the P&R steps disrupt the impact of layer balancing and hence, the LP solution is unable to consistently save all the MHM hold violations.
- Small tolerance factors (e.g., *BU*) limit the extent of layer balance, and have little impact on the MHM violations. Significant improvement in balance between LC paths requires high tolerance on wirelength increase, which leads to significant increase in clock WL and power.

Hence, we evaluate two feedback-based methodologies to reduce the number of MHM-induced hold violations. After executing the "Conventional Flow" shown in Figure 4.3, we obtain the number of MHM violations and their negative slacks. These slacks are fed back to earlier stages in the design flow as hold margins, and the subsequent stages of optimization target these hold violations.

4.5.1 Hold Margin Calculation

To analyze the MHM hold violations, we follow the same steps as described in Section 4.4.2. However, instead of applying derates for an individual metal layer, we apply derates

for all metal layers of interest (M4 to M8). For each LC pair, we choose the metal layer derate with the most negative slack and set this slack as the hold margin for that register as the end point.

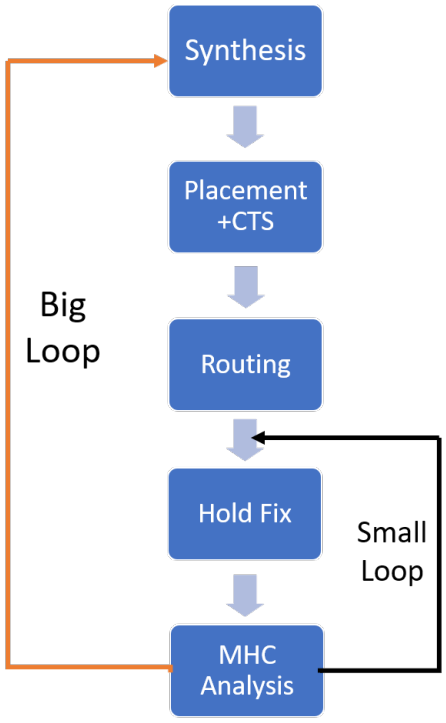


Figure 4.6: The big loop.

4.5.2 Experiments and Results

We feed the hold margins back at two different stages of the SP&R flow: (i) at the post-synthesis stage and (ii) at the post-route stage. We call these the *big loop* and the *small loop*, respectively. Figure 4.6 shows the steps in the two methodologies. After the hold margins are fed back, the remaining stages of the SP&R flow are executed once more and the derating analysis described in Section 4.4.2 is performed.

Table 4.9 shows the results obtained with the big and the small loops. The first row for each design, "Base", reports the values obtained at the end of hold-fixing from a single SP&R run. The second row, "Big Loop", reports the values after following the big loop methodology and the third row, "Small Loop", reports the values from the small loop methodology. All values

Table 4.9: Results from big loop experiments.

Testcase	Technique	Hold WNS (ns)	Hold TNS (ns)	#Violations	Total Power (mW)	Std. Cell Area (um^2)	Buffer & Inverter Area (um^2)
THEIA I	Base	-0.208	-3.403	35	59.89	47594	3793
	Big Loop	0	0	0	59.91	47645	3733
	Small Loop	0	0	0	60.32	47729	3877
THEIA II	Base	-0.343	-5.97	99	68.59	49406	4235
	Big Loop	0	0	0	68.54	49423	4237
	Small Loop	-0.003	-0.004	2	69.53	49742	4478
LEON3MP	Base	-0.143	-19.983	1031	456.0	186374	16436
	Big Loop	0	0	0	458.21	186616	16610
	Small Loop	-0.024	-0.973	47	476.42	205128	34616

reported are after applying the MHM derating.

We see that the big loop methodology solves all MHM hold violations with minimal increase in power and area, but with the added runtime penalty of running the entire P&R flow again. The small loop methodology, however, is unable to resolve all the violations for some designs, with significant increase in buffer area (and consequently, power). Also, the small loop will result in worse timing, power and area when the design is congested and has high utilization. Hence, the big loop methodology is a better strategy for improved PPA, providing significant guardbanding against MHM violations with minimal power and area overheads.

4.6 Conclusion

This work shows that improving the layer balance of clock routes can reduce inter-layer variation significantly, leading to improved yields. Our linear program-based approach improves the layer balance by up to 18% on average, with minimal clock tree WL and clock power overheads. However, our attempts to realize the layer balancing improvements in a commercial tool do not yet show the full benefits of our LP. Guardbanding each endpoint with hold margins throughout the SP&R flow provided a simpler and more effective solution, with minimal impact

on area and power. Future research includes developing (i) a better evaluation metric to quantify the benefits of layer balancing and its effect on MHC, (ii) a CTS algorithm which understands layer imbalance implicitly and minimizes it in concert with skew optimizations, and (iii) an improved routing realization strategy which is DRC-clean by construction.

4.7 Acknowledgments

Chapter 4 is in part a reprint of the submitted draft: S. Hong, A. B. Kahng, S. Venkatesh and L. Wang, 'Layer-Balancing ECO Clock Optimization for Improved Model-Hardware Correlation and Design Closure", *Proc. Design Automation Conf.*, 2018, submitted draft.

I would like to thank my co-authors Mr. Soowan Hong, Professor Andrew B. Kahng and Lutong Wang. I would also like to thank Mr. Jongpil Lee and Dr. Bong-Il Park of Samsung for their guidance and feedback on this work.

Bibliography

[1] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng and D. Karger, "Prim-Dijkstra Tradeoffs for Improved Performance-driven Routing Tree Design", *IEEE Trans. on CAD of ICAS* 14(7) (1995), pp. 890-896.

[2] ITRS 2013 Edition Report - Interconnect, `https://www.semiconductors.org/clie\ \ntuploads/Research\_Technology/ITRS/2013/2013Interconnect.pdf`, 2013.

[3] S. K. Rao, P. Sadayappan, F. K. Hwang and P. W. Shor, "The Rectilinear Steiner Arborescence Problem", *Algorithmica* 7(2) (1992), pp. 277-88.

[4] C. J. Alpert, *Personal Communication*, Nov. 2016.

[5] R. C. Prim, "Shortest Connecting Networks and Some Generalizations", *Bell System Tech. J.* 36 (1957), pp. 1389-1401.

[6] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik* 1 (1959), pp. 269-271.

[7] J. M. Ho, G. Vijayan and C. K. Wong, "New Algorithms for the Rectilinear Steiner Tree Problem", *IEEE Trans. on CAD of ICAS* 9(2) (1990), pp. 185-193.

[8] J. B. Kruskal Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *American Mathematical Society* 7(1) (1956), pp. 48-50.

[9] C. J. Alpert, A. B. Kahng, C. N. Sze and Q. Wang, "Timing-driven Steiner Trees are (Practically) Free", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 2006, pp. 389-392.

[10] J. Cong, A. B. Kahng, G. Robins and M. Sarrafzadeh, "Provably Good Performance-driven Global Routing", *IEEE Trans. on CAD of ICAS* 11(6) (1992), pp. 739-752.

[11] G. Kortsarz and D. Peleg, "Approximating Shallow-light Trees", *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 1997, pp. 103-110.

[12] S. Khuller, B. Raghavachari and N. Young, "Balancing Minimum Spanning Trees and Shortest-path Trees", *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 243-250.

[13] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh and C.K. Wong, "Performance-driven Global Routing for Cell Based ICs", *Proc. IEEE International Conference on Computer Design*, 1991, pp. 170-173.

[14] A. Lim, S.-W. Cheng and C.-T. Wu, "Performance Oriented Rectilinear Steiner Trees", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 1993, pp. 171-175.

[15] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Trans. on CAD of ICAS* 11(7) (1992), pp. 893-902.

[16] M. Borah, R. M. Owens and M. J. Irwin, "An Edge-based Heuristic for Steiner Routing", *IEEE Trans. on CAD of ICAS* 13(12) (1994), pp. 1563-1568.

[17] W. Shi and C. Su, "The Rectilinear Steiner Arborescence Problem is NP-complete", *SIAM J. Computing* 35(3) (2006), pp. 729-740.

[18] J. Cong, K. S. Leung and D. Zhou, "Performance-driven Interconnect Design Based on Distributed RC Delay Model", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 1993, pp. 606-611.

[19] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, 1995.

[20] M. Hanan, "On Steiner's Problem with Rectilinear Distance", *SIAM J. Applied Mathematics* 14(2) (1966), pp. 255-265.

[21] L. Scheffer, Bookshelf RMST code, `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST/`.

[22] L. J. Guibas and J. Stolfi, "On Computing All Northeast Nearest Neighbors in the L1 Metric", *Information Processing Letters* 17(1983), pp. 219-223.

[23] A. Naamad, D. T. Lee and W.-L. Hsu, "On the Maximum Empty Rectangle Problem", *Discrete Applied Mathematics* 8(1984), pp. 267-277.

[24] J. Griffith, G. Robins, J. S. Salowe and T. Zhang, "Closing the Gap: Near-optimal Steiner Trees in Polynomial Time", *IEEE Trans. on CAD of ICAS* 13(11) (1994), pp. 1351-1365.

[25] L. He, S. Yao, W. Deng, J. Chen and L. Chao, "Interconnect Routing Methods of Integrated Circuit Designs", *U.S. Patent 8386984*, Feb. 2013.

[26] S. Bose, "Methods and Systems for Placement and Routing", *U.S. Patent 8332793*, Dec. 2012.

[27] R. F. Hentschke, M. de Oliveira Johann, J. Narasimhan and R. A. de Luz Reis, "Methods and Apparatus for Providing Flexible Timing-driven Routing Trees", *U.S. Patent 8095904*, Jan. 2012.

[28] G. M. Furnish, M. J. LeBrun and S. Bose, "Tunneling as a Boundary Congestion Relief Mechanism", *U.S. Patent 7921393*, Apr. 2011.

[29] G. M. Furnish, M. J. LeBrun and S. Bose, "Node Spreading Via Artificial Density Enhancement to Reduce Routing Congestion", *U.S. Patent 7921392*, Apr. 2011.

[30] P. Saxena, V. Khandelwal, C. Qiao, P-H. Ho, J. C. Lin and M. A. Iyer, "Interconnect-driven Physical Synthesis using Persistent Virtual Routing", *U.S. Patent 7853915*, Dec. 2010.

[31] C. J. Alpert, J. Hu and P. H. Villarrubia, "Practical Methodology for Early Buffer and Wire Resource Allocation", *U.S. Patent 6996512*, Feb. 2006.

[32] C. J. Alpert, R. G. Gandham, J. Hu, S. T. Quay and A. J. Sullivan, "Apparatus and Method for Determining Buffered Steiner Trees for Complex Circuits", *U.S. Patent 6591411*, Jul. 2003.

[33] C. Chu and Y.C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design", *IEEE Trans. on CAD of ICAS* 27(1) (2008), pp.70-83.

[34] M. Elkin and S. Solomon, "Steiner Shallow-light Trees are Exponentially Lighter than Spanning Ones", *SIAM J. Computing* 44(4) (2015), pp. 996-1025.

[35] R. Scheifele, "Steiner Trees with Bounded RC-delay", *Algorithmica* 78(1) (2017), pp. 86-109.

[36] G. Chen, P. Tu and E. F. Y. Young, "SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017.

[37] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li and Y. Wei, "The DAC 2012 Routability-driven Placement Contest and Benchmark Suite", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 2012, pp. 774-782.

[38] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng and C.-K. Cheng, "ePlace-MS: Electrostatics based Placement for Mixed-Size Circuits", *IEEE Trans. on CAD of ICAS* 34(5) (2015), pp. 685-698.

[39] J. Bloom and J. Orloff, *18.05 Introduction to Probability and Statistics*, Cambridge, Massachusetts Institute of Technology: MIT OpenCourseWare, 2014. `https://ocw.mit.edu`

[40] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement", *IEEE Trans. on CAD of ICAS* 18(9) (1999), pp. 1265-1278.

[41] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 2000, pp. 477-482.

[42] W.-T. J. Chan, A. B. Kahng and J. Li, "Revisiting 3DIC Benefit with Multiple Tiers", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2016, pp. 6:1-6:8.

[43] B. Chazelle, R. L. Drysdale and D. T. Lee, "Computing the Largest Empty Rectangle", *SIAM J. Computing* 15(1) (1986), pp. 300-315.

[44] C. L. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1994, pp. 690-695.

[45] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits", *IEEE Trans. on CAD of ICAS* 4(1) (1985), pp. 92-98.

[46] I. L. Markov, J. Hu and M.-C. Kim, "Progress and Challenges in VLSI Placement Research", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 1985-2003.

[47] D. Panchenko, *18.650 Statistics for Applications*, Cambridge, Massachusetts Institute of Technology: MIT OpenCourseWare, 2004. `https://ocw.mit.edu`

[48] RMST-Pack: Rectilinear Minimum Spanning Tree Algorithms [Source code]. `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST`

[49] A. R. Agnihotri and P. H. Madden, "Congestion Reduction in Traditional and New Routing Architectures", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2003, pp. 211-214.

[50] C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, eds., *Handbook of Algorithms for Physical Design Automation*, CRC Press, 2008.

[51] K. D. Boese and A. B. Kahng, "Zero-Skew Clock Routing Trees With Minimum Wirelength", *Proc. IEEE International Conference on ASIC*, 1992, pp. 17-21.

[52] J. D. Carothers, T. Liu and D. Li, "MCM Multilayer Routing with Layer Balancing", *Proc. IEEE International Conference on ASIC*, 1996, pp. 179-182.

[53] J. Cong, A. B. Kahng, C.-K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing Under Elmore Delay", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 66-71.

[54] J. J. Donato, "Routing Balanced Clock Signals", *U.S. Patent 6513149 B1*, 2003.

[55] R. Goering, "Why Cadence Bought Azuro - A Closer Look", `https://community.cadence.com/cadence_blogs_8/b/ii/posts/why-cadence-bought-azuro-a-closer-look`, July 24, 2011.

[56] P. A. Habitz, D. J. Hathaway, J. D. Hayes and A. D. Polson, "Method of Generating Wiring Routes with Matching Delay in the Presence of Process Variation", *U.S. Patent 7865861 B2*, 2011.

[57] K. Jeong, A. B. Kahng and K. Samadi, "Impacts of Guardband Reduction on Design Process Outcomes: A Quantitative Approach", *IEEE Trans. on Semiconductor Manufacturing* 22(4) (2009), pp. 552-565.

[58] S. Lin and C. K. Wong, "Process-Variation-Tolerant Clock Skew Minimization", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1994, pp. 284-288.

[59] Y. Liu, S. Nassif, L. Pileggi and A. Strojwas, "Impact of Interconnect Variations on the Clock Skew of a Gigahertz Microprocessor", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 2000, pp. 168-171.

[60] J. Minz, X. Zha, and S. K. Lim, "Buffered Clock Tree Synthesis for 3D ICs Under Thermal Variations", *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 504-509.

[61] U. Padmanabhan, J. M. Wang and J. Hu, "Robust Clock Tree Routing in the Presence of Process Variations", *IEEE Trans. on CAD of ICAS* 27(8) (2008), pp. 1385-1397.

[62] J. Oh, I. Pyo and M. Pedram, "Constructing Lower and Upper Bounded Delay Routing Trees Using Linear Programming", *Proc. IEEE/ACM/EDAC Design Automation Conf.*, 1996, pp. 401-404.

[63] J. Oh, I. Pyo and M. Pedram, "Constructing Minimal Spanning/Steiner Trees with Bounded Path Length", *CENG Technical Report 94-35*, University of Southern California, 1994.

[64] J. Oh, I. Pyo and M. Pedram, "Constructing Minimal Spanning/Steiner Trees with Bounded Path Length", *Integration, the VLSI Journal* 22(1-2) (1997), pp. 137-163.

[65] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke and C. Zhuo, "ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 161-164. `http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html`.

[66] A. Rajaram and D. Z. Pan, "Variation Tolerant Buffered Clock Network Synthesis with Cross Links", *Proc. ACM International Symposium on Physical Design*, 2006, pp. 157-164.

[67] E. Sperling, "Variation Spreads at 10/7nm", *Semiconductor Engineering*, November 2017. `https://semiengineering.com/variation-spreads-at-107nm/`

[68] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness and C. Alpert, "Practical Techniques to Reduce Skew and Its Variations in Buffered Clock Networks", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 592-596.

[69] L. Xiao, Z. Xiao, Z. Qian, Y. Jiang, T. Huang, H. Tian and E. F. Y. Young, "Local Clock Skew Minimization Using Blockage-aware Mixed Tree-Mesh Clock Network", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 458-462.

[70] P. Zarkesh-Ha, T. Mule and J. D. Meindl, "Characterization and Modeling of Clock Skew with Process Variations", *Proc. IEEE Custom Integrated Circuits Conference*, 1999, pp. 441-444.

[71] ARM Cortex A53 Processor. `https://developer.arm.com/products/processors/cortex-a/cortex-a53`

[72] ASAP: Arizona State Predictive PDK. http://asap.asu.edu/asap/

[73] Avatar Integrated Systems. http://www.avatar-da.com/

[74] Cadence Design Systems, Inc. https://www.cadence.com

[75] Cadence Innovus User Guide.

[76] IBM ILOG CPLEX, www.ilog.com/products/cplex/

[77] Gary Smith EDA. https://www.garysmitheda.com/

[78] International Symposium on Physical Design Contests. http://ispd.cc/contests/

[79] Mentor, A Siemens Business. https://www.mentor.com

[80] OpenCores: Open Source IP-Cores. http://www.opencores.org

[81] Synopsys, Inc. https://www.synopsys.com

[82] Synopsys IC Compiler User Guide.