Incremental Timing-Driven Placement with Displacement Constraint

by

Jude Arun Selvan Jesuthasan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2015

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In the modern deep-submicron Very Large Integrated Circuit(VLSI) design flow interconnect delays are becoming major limiting factor for *timing closure*. Traditional placement algorithms such as routability-driven placement (improves routability) and wirelength-driven placement (reduces total wirelength) are no longer sufficient to close timing. To this end, timing-driven placement plays a crucial role in reducing the interconnect delay through timing critical paths (paths with timing violations/negative slacks) of the design and thereby achieving specific performance/clock frequency.

In the placement flow, timing information about the design can be incorporated during *global placement* and/or *incremental/detailed placement*. Although, over the years, there has been significant advances in the quality of the global placement, there is a growing need for high performance incremental timing-driven placement due to the lack of accurate interconnect information during global placement. Moreover, incremental timing-driven placement is essential to recover timing while preserving the other optimization objectives such as total wirelength, routing congestion, and so forth which are optimized at the early stages of the design flow.

This thesis proposes a simple, yet efficient, incremental timing-driven placement algorithm that seeks to find optimized locations for standard cells so that the *total negative slack* of the design can be maximized. Our algorithm consists two stages: (1) *Global Move* which positions standard cells inside a *critical bounding box* to eliminate timing violations on timing critical nets; and (2) *Local Move* which provides further timing improvement by finely adjusting the current locations of the standard cells within a local region.

We evaluate our algorithm using ICCAD-2014 timing-driven placement contest benchmarks. The results show that, on average, our technique eliminates 94% and 30% of the

late and early total negative slacks, respectively, and, 82% and 27% of the late and early worst negative slacks, respectively, under short and long displacement constraints. The 1st-place team of the contest improves late and early total negative slacks by 90% and 39%, respectively, and improves late and early worst negative slack by 76% and 32%, respectively. Taking into account both timing violation improvement and the placement quality (i.e., other objectives), on average, we outperform the 1st-place team by 3% in terms of the ICCAD-2014 contest quality score and our technique is 4.6× faster in terms of runtime.

Acknowledgements

I am extremely grateful to my supervisor Dr. Andrew Kennings for his support over the course of my degree. Without his help, advice and guidance, this thesis would not have come into existence. I thank Dr. Hiren Patel and Dr. George Freeman for reviewing this thesis. I also thank Jucemar Monteiro for providing the binary of his placer.

I would like to thank my brother, Kalai, and sisters, Sarumathy and Lathanky, for their love and support throughout my academic carrier. I am deeply indebted to my brother in law, James, for his support and the help he provided me since I came to Canada. I would also like to thank my girl friend, Princy, for her love, support, patience and prayers during the last two years of my studies.

Last but not the least, I thank my parents for their love, support and encouragement throughout my studies and for standing by my side during the difficult times of my life.

Dedication

To my parents,

Jesuthasan and Varonicamma,

Without whom none of my success would be possible

Contents

List of Tables

List of Figures

Glossary

VLSI Very Large Scale Integrated circuit

RTL Register Transfer Level

CAD Computer Aided Design

STA Static Timing Analysis

TNS Total Negative Slack

WNS Worst Negative Slack

HPWL Half-Perimeter WireLength

RSMT Rectilinear Steiner Minimal Tree

ABU Average Bin Utilization

RC Tree Resistance-Capacitance Tree

at arrival time

rat required arrival time

DAG Directed Acyclic Graph

LP Linear Programming

DNF Did Not Finish

LCB Local Clock Buffer

Chapter 1

Introduction

1.1 Motivation

Timing closure is a crucial task in the Very Large Scale Integrated circuit(VLSI) design flow. By timing closure, we mean that the design meets the timing constraints, namely setup and hold constraints, and the design achieves a specific clock frequency. Figure 1.1 illustrates a typical VLSI design flow. Even though, according to Figure 1.1, timing closure is mentioned as the last stage of the physical design, timing optimization can be performed throughout the physical design flow using several techniques. Such techniques include buffer insertion, gate sizing, timing-driven routing and timing-driven placement [4]. This thesis focuses on timing closure at the placement stage of the design; specifically during detailed placement.

Placement is an important and challenging step in the physical design of an integrated circuit. The quality of the placement impacts the wirelength, routability and the performance/timing of the design. Although wirelength-driven and routability-driven placement

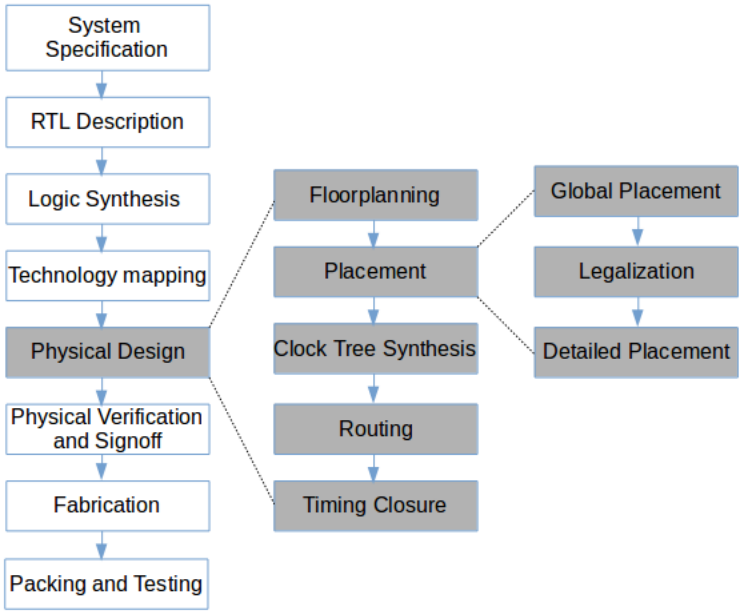

Figure 1.1: Typical VLSI CAD flow.

can help improve the timing (e.g., keeping nets short, in general, will reduce the delay of the nets), timing critical nets may still be present in the final placed design. Incorporating the timing information inside the placement flow to identify timing critical nets and then guide the placement of standard cells which are included in the path of such nets is know as timing-driven placement.

Timing information during placement can be incorporated either globally or incrementally. While global timing-driven placement moves cells freely from one location to another to optimize timing on specific nets, incremental timing-driven placement often limits the displacement of cells from their original locations so that the quality of the global placement can be preserved. Although, at different points in the flow, one can choose different degrees of cell movement, there is a growing need for high performance incremental timing-driven placement in the modern timing closure flow, as it is important to recover timing from

early steps of the placement flow while maintaining its solution quality in terms of total wirelength, routing congestion, placement density and etc. [5]. The two major challenges faced in obtaining high quality incremental timing-driven placement solutions are:

- *Convergence of timing improvement.* Timing violations of a design are quantified using a metric knows as *slacks*, which is obtained by performing Static Timing Analysis (STA) (section 2.3.3) on the design. During STA each input/output pin in the circuit is annotated with slack. As such, the *Total Negative Slack* (TNS) and/or *Worst Negative Slack* (WNS) obtained using STA indicates the amount of timing violation in a circuit. There might be an exponential number of paths with negative slack in a given design/circuit. Eliminating only a few paths may result in marginal improvement. Therefore, an efficient technique is required to a reduce a large number of critical paths or near critical paths [6].

- *Preserving the global placement quality in terms of wirelength and routability.* Initially, incremental timing-driven placement is given a legal placement that might have been optimized for wirelength and/or routability. Altering the initial landscape of cells to improve timing on critical nets may have a non-trivial impact on wirelength and routability of the design. Hence, any movement of cells must preserve the global placement quality in terms of wirelength and congestion [7]. The cell movements must also ensure that the final placement is legal.

In this work, we propose an efficient, yet simple, incremental timing-driven placement algorithm that addresses the aforementioned challenges. Our technique is a greedy path-based algorithm, which improves the circuit timing while maintaining the initial placement quality. Our specific contributions are provided in the next section.

1.2 Thesis Contributions

The key contributions of this work are as follows:

1. Our algorithm is simple but effective. It is based on a greedy path-based technique. The main flow of the algorithm consist of two stages: *Global Move* and *Local Move*. In the global move stage, we globally move cells to a site inside a *critical bounding box*. The global move is composed of two sub-stages: *sequential cell pass* and *combinational cell pass*. During sequential cell pass, only sequential cells are allowed to move, whereas in the combinational cell pass, movement of combinational cells are considered. In the local move stage, we locally move combinational cells to nearby sites for further improvement of timing.

2. To preserve the global view and the quality of the initial placement, incremental timing-driven placement should only consider moving cells within the range of *maximum displacement limit*, which is defined as Manhattan distance from the cell's original location to the target location. Our global move technique can move cells far away from their original placement location. As a result, it might violate maximum displacement limit constraint. Therefore, we provide a methodology which supports *displacement-aware* cell movement during the global move stage of the algorithm.

3. Unlike many timing-driven placements that uses inaccurate, often crude, timing models, during the cell move, our technique relies on accurate timing information to evaluate the impact of a cell move on the design timing [8]. To reduce the runtime overhead due to performing STA on the design during every cell move, we parallelize our static timing analyzer.

4. We evaluate the performance of our technique using ICCAD-2014 contest benchmarks, and compare our results with top three teams of the competition. The experimental study shows that our results are competitive in terms of timing improvement and runtime.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides background on placement and timing analysis. Previous work on timing-driven placement is also presented in Chapter 2. Chapter 3 presents our incremental timing-driven placement algorithm with several key operations described in detail. In chapter 4, results from the experimental study are presented and compared. Chapter 5 details additional experimentation with the focus on reducing the runtime of our timing-driven placement algorithm. Finally, conclusions and future work directions are offered in Chapter 6.

Chapter 2

Background

This chapter provides some background on placement. We describe common placement objectives and timing-driven placement. We also describe static timing analysis procedure. This includes timing models of circuit elements such as combinational cells, sequential cells, interconnect delay, and propagation of timing information such as delays and slews.

2.1 Placement

The process of designing a VLSI circuit is done in multiple stages due to its complexity. In the VLSI design flow (Fig 1.1), placement is performed after logic synthesis and technology mapping, but before routing. Placement is the process of determining the locations of standard cells or logic elements on a die surface. It takes a set of cells/macros, a netlist, and a chip outline as its input and produces legal locations for those cells/macros as its output.

Placement is a complex problem. All modern placers solve placement in several man-

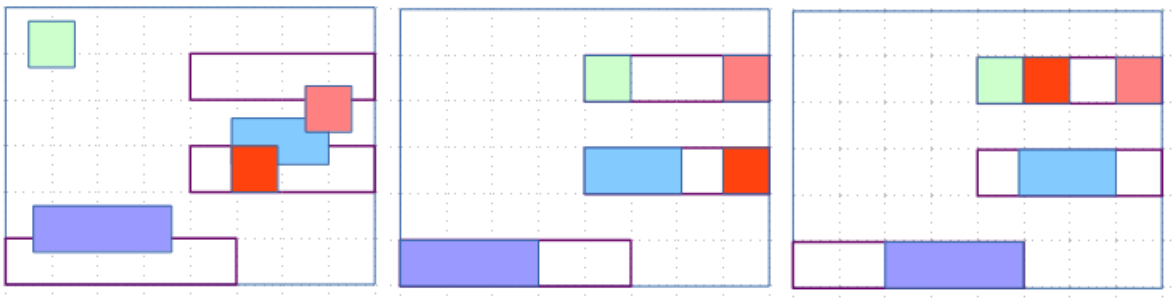

Figure 2.1: Placement of standard cells on a chip using three commonly used stages: from left to right (1) Global placement; (2) Legalization; and (3) Detailed placement.

ageable steps. One commonly used placement flow is as follows:

1. **Global placement:** Given the netlist, global placement seeks to find a rough solution to the problem. Though the obtained solution might violate some placement constraints, such as no overlap among cells/macros, it is an important step in determining the overall placement quality and runtime [2]. The most commonly used global placement algorithms can be classified into three categories: (1) Partitioning-based methods recursively divides (e.g., through recursive bipartitioning) a circuit into several subcircuits and place those subcircuits in the placement sub-region [9]; (2) Simulated-annealing methods applies probabilistic heuristic search to obtain a desired place for cells/macros such that particular cost function can be optimized [10]; (3) Analytical methods approximate some cost function, typically wirelength of a net using a quadratic cost function, such that the optimal solution can be obtained using efficient numerical methods [11, 12].

2. **Legalization:** Given a rough global placement solution, legalization perturbs cells or macros locally so that no placement constraints are violated (i.e., overlap free) and the characteristics of the input global placement solution is preserved [13, 14].

3. **Detailed placement:** The legalized placement is further improved by moving cells/macros around iteratively while maintaining the global view of the solution obtained in the previous steps [15, 16].

Figure 2.1 illustrates an example of standard cell placement using the three steps above described.

2.2 Placement Objectives

During placement one or more cost functions, such as wirelength, timing, routing congestion, power consumption and thermal issues are optimized. Since timing optimization is the focus of this thesis, we only provide an overview of other common placement objectives, namely wirelength and routability.

2.2.1 Total Wirelength

Minimization of total wirelength is the most widely used objective in the placement problem formulation to indirectly optimize timing, routability and power consumption [2]. The main idea is to minimize the wirelength of each net in the design so that net delays, routing demands, and load capacitances can be reduced to improve performance, routability and power consumption, respectively. Hence, a placement formulation based on total wirelength has been the focus of most prior research works and has resulted in several high quality wirelength-driven placement algorithms such as SimPL [12], NTUPlace3 [17], FastPlace3 [18], MAPLE [19] and so on. It is worth noting that total wirelength minimization is only a heuristic to optimize these other objectives, and hence nets in the most congested region, along the timing critical paths, or with the highest switching activities may not be shorter.

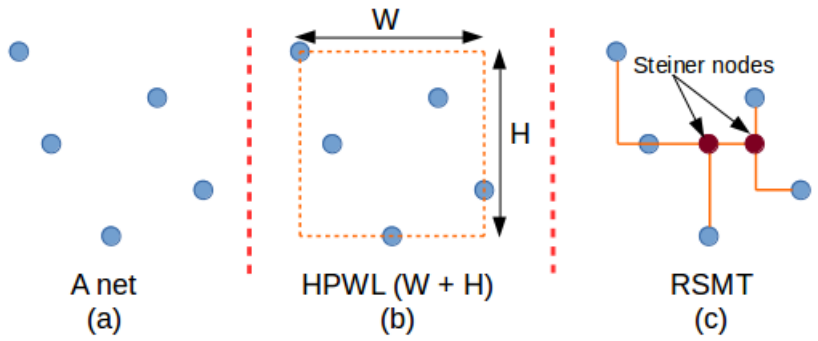

Figure 2.2: An example of HPWL(b) and RSMT(c) wirelength estimation technique for a five pin net(a) [2].

During placement, wirelength of the nets are predicted, because the actual wirelength of nets are not known until they are routed, which is performed following the placement phase of the VLSI design flow (Figure 1.1). To estimate the wirelength of a net, among other approaches, half-perimeter wirelength (HPWL) is the most widely used. The HPWL of a net is half the circumference of the smallest bounding rectangle that encloses all the pins in the net (Figure 2.2-b). To emphasize the importance of HPWL in the wirelength-driven placement, ISPD-2005 placement contest [20] can be considered an example where HPWL is the metric used to measure the total wirelength of the final placement solution.

Even though HPWL is popular due to its linear computational time, it can significantly underestimate the wirelength of a net as it only provides the lower bound of the actual wirelength. A better approach to the wirelength estimation is based on rectilinear Steiner minimal tree (RSMT). In RSMT, nodes (pins) of a net are possibly connected through some extra (i.e., Steiner) nodes to minimize the total edge length in Manhattan distance between connected nodes (Figure 2.2-c). Although RSMT, for nets with at least four pins, provides more accurate estimation of wirelength than HPWL, it is computationally more expensive and hence, traditionally, rarely used during placement [2].

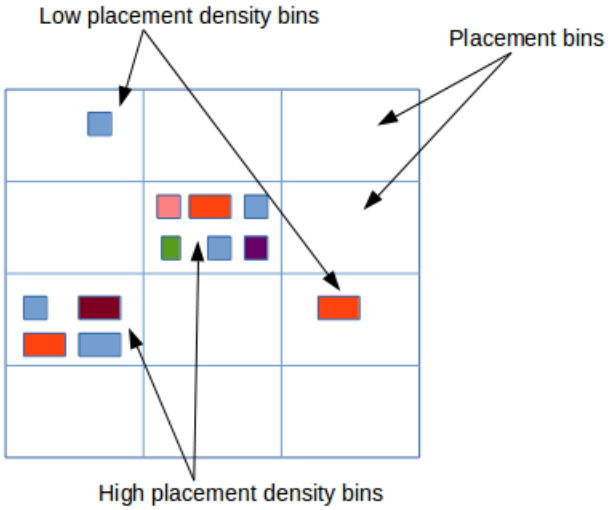

Figure 2.3: Placement bins and their utilization for measuring placement density.

2.2.2 Routability

A placement solution is useless unless it is routable. Therefore, during wirelength-driven placement, it is essential to estimate routing demands in arbitrary neighborhood of placement regions to tradeoff the wirelength to routability. To estimate the routability, ISPD-2006 placement contest [21] introduced density target constraint to force a placer to reserve a specified amount of white space (free space) in any subregions of the placement area for routing and other optimizations. Given that a placement area is divided into equal sized bins as shown in Figure 2.3, the density of a placement bin is defined as the sum of the area of movable cells divided by the total available area in that placement bin. A placement bin with higher density than the target density must consider spreading cells to the nearby bins with low placement density bins, if possible, to achieve the specified target density.

The main shortcoming of the aforementioned approach is that it often fails to capture the uneven distribution of cells in the placement bins. To this end, in [19], a placement den-

sity metric, know as *Average Bin Utilization* (ABU), is proposed. ABU_γ is defined as the average area utilization for top γ% bins excluding the bins fully occupied by fixed macros, and is effective in capturing uneven distribution of standard cells. Thus, minimizing the ABU_γ can uniformly distribute the cells across the placement region.

Since the routabilty of a design is router-dependent, white space allocation based on target density or ABU can underestimate the actual routing congestion of the design. Though the actual routing congestion can be measured by performing rough routing (e.g., global routing), a router is rarely used during global placement. However, performing global routing is considered feasible in routability-driven placement. ISPD-2011, ISPD-2014 and ISPD-2015 rotutabilty-driven placement contests [22, 23, 24] are the examples where a global router is used to measure the routing congestion of a placement.

2.3 Timing-Driven Placement

In addition to wirelength and routability, timing (performance) of a design is important. To this end, timing-driven placement seeks to find locations for standard cells in the placement region so that interconnect delay can be optimized at the expense of wirelength and routability. The main objective of timing-driven placement is either to satisfy all timing constraints or to achieve maximum clock frequency possible.

In this section, we describe the process of STA, including cell and interconnect delay models, timing propagation and the procedure of STA itself. Since we rely on ICCAD-2014 incremental timing-driven placement contest benchmark infrastructure [3] for our experimental study, the timing models are same as the one used in the contest. Finally, we also provide the problem formulation of incremental timing-driven placement.

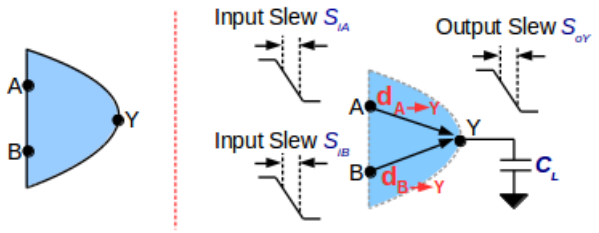

Figure 2.4: Combinational AND gate (left) and its timing model (right) [3].

2.3.1 Timing Models

This section presents pin-to-pin connection delay and output slew computation for combinational, sequential cells and interconnects. Here, the delay defines the amount of time needed for the signal to propagate from pin-to-pin, whereas output slew defines the amount of time needed for signal to switch from high-to-low or low-to-high. Typically, a low (high) signal is defined as 10% (90%) of the voltage.

1. **Combinational cells:** For a combinational cell (Figure 2.4), its pin-to-pin delay, d, and output slew, s_o, can be modeled as a linear combination of load capacitance and input slew [3] given by

$$d = a + bC_L + cs_i \tag{2.1}$$

$$s_o = x + yC_L + zs_i \tag{2.2}$$

where a, b, c, x, y, and z are cell dependent constants determined during standard cell library characterization, C_L is the load capacitance at the output pin and s_i is the slew (propagates via interconnect) at the input pin. The value of C_L is the downstream capacitance seen from the output pin of the cell. In this model, C_L is

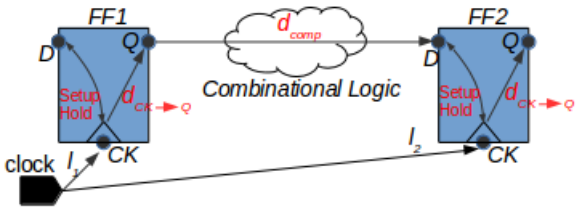

Figure 2.5: Two interconnected flip-flops and their timing model [3].

given by

$$C_L = \sum_{k=1}^{N} C_k \tag{2.3}$$

where C_k is the parasitic capacitances of the RC tree nodes of the interconnect driven by the output pin of the cell and N is the set of nodes in the RC tree network including the cell pins of the fanout.

2. **Sequential cells:** Sequential cells decompose a circuit into several stages and their outputs (inputs) acts as starting (ending) points of timing propagation. In a sequential circuit, the sequential cells are implemented using one or more flip-flops. The operation of flip-flops are synchronized by clock signals generated from one or more clock sources. Due to the distinct positions of flip-flops and clock sources, the arrival time of clock signal from the clock source to a flip-flop will encounter a delay know as *clock latency* and it depends on the routing characteristic of the clock interconnect/net.

   There are three important timing parameters pertaining to a (D) flip-flop (Figure 2.5): clock-to-output delay ($d_{CK \to Q}$), setup time (t_{setup}) and hold time (t_{hold}). $d_{CK \to Q}$ is the amount of time a flip-flop takes to propagate the value at its input pin (D) to output pin (Q) upon the detection of capturing clock edge at its clock pin (CK). Proper operation of a flip-flop requires signal at the input pin (D) to be stable for

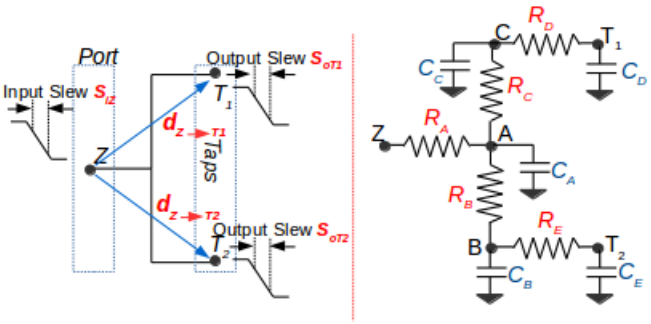

Figure 2.6: Interconnect timing model (left) and its RC tree network (right).

a specific amount of time before and after the capturing clock edge. The former is know as setup time and the latter is the hold time. The setup time and hold time can be modeled as a linear combination of input slews at the clock and data pins of the sequential cell [3] and are given by

$$t_{setup} = g + hs_{iCK}^{early} + js_{iD}^{late} \tag{2.4}$$

$$t_{hold} = m + ns_{iCK}^{early} + ps_{iD}^{late} \tag{2.5}$$

where, $g, h, j, m, n,$ and p are flip-flop specific constants determined standard cell library characterization and s_{iCK}, s_{iD} are input slews at clock pin CK and input pin D, respectively.[1]

3. Interconnect modeling

A net is used to connect output pin (source/port) of a cell to one or more input pins (sinks/taps) of other cells. Figure 2.6 illustrates a parasitic RC tree containing only grounded capacitors and floating resisters, used to model the delay and the output slew of an interconnect.

[1]In an early-late split timing model early(late) denotes the lower (upper) bound on the timing characterization.

Delay. The computation of a port-to-tap delay can be approximated by Elmore delay model [25]. Given a node e in a RC tree network, the Elmore delay at node e is given by

$$d_e = \sum_{k=1}^{N} R_{k-1,k} C_k \tag{2.6}$$

where $R_{k-1,k}$ is the resistance of a wire segment between the nodes $k-1$ and k in path from port to node e, C_k is the downstream capacitance at node k and N is the number of segments in the path from Port to node e.

Output slew. The output slew, s_o, at tap node T is computed using two step process. First, the well-approximated output slew, $\hat{s}_{oT}$, of the impulse response on T is given by

$$\hat{s}_{oT} \approx \sqrt{2\beta_T + d_T^2} \tag{2.7}$$

where d_T is the Elmore delay from equation 2.6 and β_T is second moment of the input response at node T, computed by

$$\beta_T = \sum_{k=1}^{N} R_{kT} C_k d_k \tag{2.8}$$

where R_{kT} is the resistance of common path between port to node k and port to node T, and C_k and d_k are the lumped capacitance and Elmore delay at node k, respectively. Second, the output slew, s_{oT}, of the response to the input ramp is computed by

$$s_{oT} \approx \sqrt{s_i^2 + \hat{s}_{oT}} \tag{2.9}$$

where s_i is the input slew at the port.

Table 2.1: Table of notation for timing propagation equations.

at_Y	-	arrival time at pin Y
rat_Y	-	required arrival time at pin Y
$d_{X \to Y}$	-	combinational cell/port-to-tap interconnect delay from pin X to pin Y
l	-	clock latency
$d_{CK \to Q}$	-	sequential cell delay from pin CK to pin Q

2.3.2 Timing Propagation

Profiling the timing characteristic of a sequential circuit involves performing late and early mode STA on the circuit. As a result, the late and early slacks obtained at each primary outputs and input pins of sequential cells are computed to quantify the timing violation of the design. While a design with positive slacks indicates all timing constraints are met, a design with negative slacks exhibits timing violations. To be specific, a design with late negative slacks and early negative slacks indicates setup time and hold time violations, respectively, in the circuit. Slack is a function of arrival time and required arrival time. In the following, definitions of arrival time, required arrival time and slack are provided. To help aid the discussion, notation of key variables are listed in Table 2.1.

Arrival time (at). The late(early) arrival time at timing point t [2] of a standard cell is the latest(earliest) instant the signal reaches the t. For combinational cells (Figure 2.4), the late and early arrival times at the output pin Y are given as

$$at_Y^{late} = \max(at_A^{late} + d_{A \to Y}^{late}, at_B^{late} + d_{B \to Y}^{late}) \tag{2.10}$$

$$at_Y^{early} = \min(at_A^{early} + d_{A \to Y}^{early}, at_B^{early} + d_{B \to Y}^{early}) \tag{2.11}$$

According to Figure 2.6, given that Z is the source and A is the tap of an interconnect,

[2] Timing point t is the input/output pins of a standard cell.

the arrival times at the input pin A of a combinational cell (Figure 2.4) are given by

$$at_A^{late} = at_Z^{late} + d_{Z\to A}^{late} \tag{2.12}$$

$$at_A^{early} = at_Z^{early} + d_{Z\to A}^{early} \tag{2.13}$$

For sequential cells (Figure 2.5), the arrival times at the data pin(D) of capturing sequential cell (FF2) is given as

$$at_D^{late} = l_1^{late} + d_{CK\to Q}^{late} + d_{comp}^{late} \tag{2.14}$$

$$at_D^{early} = l_1^{early} + d_{CK\to Q}^{early} + d_{comp}^{early} \tag{2.15}$$

Required Arrival time (rat). The late(early) required arrival time at timing point t is the latest(earliest) instant the signal is allowed to reach t. Consider the situation in Figure 2.6, where a net driven by output pin Z of a combinational cell drives input pins *T1* and *T2* of another cells. Thus, the required arrival time at the output pin Z of the combinational cell is given as

$$rat_Z^{late} = \min(rat_{T1}^{late} - d_{Z\to T1}^{late}, rat_{T2}^{late} - d_{Z\to T2}^{late}) \tag{2.16}$$

$$rat_Z^{early} = \max(rat_{T1}^{early} - d_{Z\to T1}^{early}, rat_{T2}^{early} - d_{Z\to T2}^{early}) \tag{2.17}$$

The required arrival times at an input pin A of a combinational cell (Figure 2.4) are given by

$$rat_A^{late} = rat_Y^{late} - d_{Y\to A}^{late} \tag{2.18}$$

$$rat_A^{early} = rat_Y^{early} - d_{Y\to A}^{early} \tag{2.19}$$

For sequential elements, consider the FF2 of Figure 2.5 with setup time t_{setup} and hold time t_{hold}. Given a clock period of P, the required arrival time at the data pin D of FF2 is given as

$$rat_D^{late} = P + l_2^{early} - t_{setup} \tag{2.20}$$

$$rat_D^{early} = l_2^{late} + t_{hold} \tag{2.21}$$

Slack. Given the arrival time and required arrival time at each timing point t of the design, the late(early) slack at timing point t is computed as

$$slack_t^{late} = rat_t^{late} - at_t^{late} \tag{2.22}$$

$$slack_t^{early} = at_t^{early} - rat_t^{early} \tag{2.23}$$

Given the slacks at each timing point t of the circuit, the timing metrics TNS and WNS are used to quantify the amount of timing of the circuit. TNS and WNS can be defined as follows:

(1) Total Negative Slack (TNS):

$$TNS^{late} = \sum_{j \in PO} slack_j^{late} \tag{2.24}$$

$$TNS^{early} = \sum_{j \in PO} slack_j^{early} \tag{2.25}$$

where PO is the set of all primary outputs and data pins of sequential cells with negative slacks.

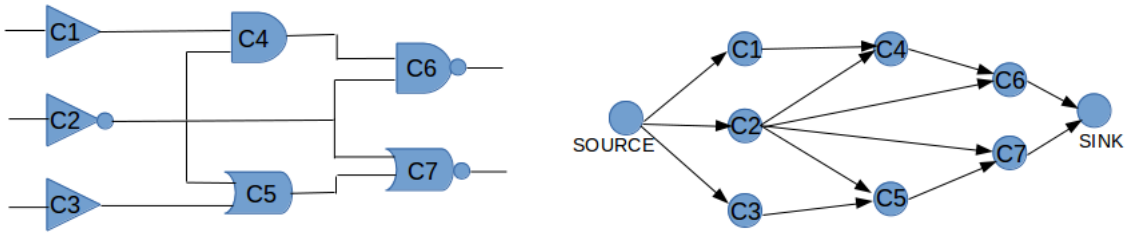

Figure 2.7: Example circuit (left) and its timing graph (right). Except SOURCE and SINK nodes, each node in the timing graph represents a cell. The edges in the timing graph represent the port-to-tap delays of the interconnect computed by Elmore delay model. The SOURCE and SINK nodes act as timing start and end points, respectively.

(2) Worst Negative Slack (WNS):

$$WNS^{late} = \max_{j \in PO}(slack_j^{late}) \tag{2.26}$$

$$WNS^{early} = \max_{j \in PO}(slack_j^{early}) \tag{2.27}$$

where PO is the set of all primary outputs and data pins of sequential cells with negative slacks.

2.3.3 Static Timing Analysis

STA on a circuit is performed using its *timing graph*, a directed acyclic graph (DAG) given that the circuit doesn't have any combinational loop. The nodes and edges in the DAG represent the combinational cells in the circuit and port-to-tap delays of interconnects, respectively. The computation of arrival times (required arrival times) at cells input/output pins are performed in *forward (backward) topological ordering* of the corresponding nodes in the DAG.

For example, Figure 2.7 presents an example circuit and its timing graph. The computa-

tion of arrival times at cells input/output pins are computed (by equations 2.12, 2.13, 2.10, 2.11) in the forward topological ordering of the nodes given by {SOURCE, C1, C2, C3, C4, C5, C6, C7, SINK}. On the other hand, the required arrival times at at cells input/output pins are computed (by equations 2.18, 2.19, 2.16, 2.17) in the backward topological ordering of the nodes given by {SINK, C6, C7, C4, C5, C1, C2, C3, SOURCE}. Given the arrival times and required arrival times for each input/output pins in the circuit, the slacks can be computed using equations 2.22 and 2.23.

2.3.4 Problem Formulation

In this section, incremental timing driven placement is formulated as a mathematical optimization problem. Given initial location $(x_j^o,\ y_j^o)$ for each moveable standard cell $c_j \in C$, incremental timing-driven placement seeks to find a new location $(x_j^n,\ y_j^n)$ such that timing violations of the initial placement can be eliminated or reduced. In doing so, it must respect the following physical constraints:

1. **Legality.** The new location of the cell c_j must be within the chip region, denoted by $(X_{left},\ Y_{bottom})$ - $(X_{right},\ Y_{top})$. Also, the new location must fit within a row and be site aligned, meaning that given the width of the site (sitespacing) W_{site} and the hight of the row H_{row}, $(x_j^n,\ y_j^n)$ must be multiples of W_{site}, H_{row}, respectively. In addition, the cell c_j must be "overlap free" with its nearby cells.

2. **Displacement limit.** One of the main objective of incremental timing driven placement is to preserve the quality of the initial placement given as input. To achieve this, timing optimizations are performed under *maximum displacement limit constraint*, which imposes an upper bound on the amount of displacement that a cell

can be moved from its initial location. In other words, given the initial position (x_j^o, y_j^o) and the newly found position (x_j^n, y_j^n) of a cell c_j, the amount of displacement of the cell which is measured as the Manhattan distance between the old position and new position must be less than or equal to the maximum displacement limit D_{max}.

Considering the objective and the constraints, the incremental timing-driven placement can be formulated as the following optimization problem:

$$\textbf{Maximize:} \quad TNS^{late} + TNS^{early} \tag{2.28}$$

$$\textbf{Subject to:} \quad X_{left} \leq x_j^n \leq X_{right} - W_j \quad \forall j \tag{2.29}$$

$$Y_{bottom} \leq y_j^n \leq Y_{top} - H_j \quad \forall j \tag{2.30}$$

$$x_j^n = \left\lfloor \frac{x_j^n}{W_{site}} \right\rfloor W_{site} \quad \forall j \tag{2.31}$$

$$y_j^n = \left\lfloor \frac{y_j^n}{H_{raw}} \right\rfloor H_{raw} \quad \forall j \tag{2.32}$$

$$x_j^n + W_j \leq x_{j+1}^n \quad \forall y_j^n = y_{j+1}^n \tag{2.33}$$

$$|x_j^n - x_j^o| + |y_j^n - y_j^o| \leq D_{max} \quad \forall j \tag{2.34}$$

Equations 2.29 and 2.30 ensure that the newly found location for the cell is within the boundary of the chip. The site alignment constraint and the requirement that the cell must be placed within a row is handled by equations 2.31 and 2.32, respectively. Equation 2.33 make sure that cells do not overlap with nearby cells of the same row. Finally, equation 2.34 describes the upper bound on movement of cells from their original locations.

The objective function of the above-formulated optimization problem is a non-linear function as it depends on the Elmore delay model for the interconnects delay computation. In addition, the constraints given by equations 2.31, 2.32 and 2.34 are non-linear too. Thus,

this optimization problem is non-trival and it cannot be solved by any direct methods. Therefore, timing-driven placement problem requires effective heuristic techniques to solve it efficiently in terms of timing optimization and computational runtime. In the next section, we consider some previous heuristic ideas to address timing optimization during placement.

2.4 Previous Work

In the literature, *net-based*, *path-based* and the *hybrid-approaches* are the three categories of techniques used for timing-driven placement [4]. Net-based algorithmic techniques transform the timing information into net-weights or delay budgets(net-constraints), and then minimize the weighted wirelength instead of the traditional total wirelength. Net weights can be assigned statically or dynamically. Static net weights are usually computed using slacks and do not change during placement. [26] proposes a sensitivity guided net-weighting scheme to increase(decrease) the net-weights from one placement iteration to another, to minimize the total negative slack. In [27], slack-based path counting scheme was proposed to assign weights to nets. In dynamic net weighting, net weights are computed using a timing profile that is updated during placement. In [28], net-weighting is used to solve Lagrangian Dual Problem of the Lagrangian Relaxation Subproblem, to minimize the total negative slacks in the design.

Generally, path-based methods directly optimizes the design's timing by capturing the timing violations as set of constraints, and then minimizing the total negative slack or worst negative slack using linear programming (LP). Since there can be exponential number of possible paths with timing violation, [29] adopted target timing to reduce the number of such paths during net constraint generation of the LP formulation.[30] proposed a differ-

ential timing analysis technique in the LP formulation, in which circuit elements and the interconnects are modeled through variations in cell delay, slew propagation and interconnect delay with respect to accurate timing information. [31] uses simulated-annealing to reducing the timing violation of the design by defining the cost to minimize as a function of wirelength and timing penalty.

Hybrid techniques uses features from both net-based and path-based approaches to eliminate or reduce timing violations in the design.[32] evaluates the critically of nets using a slack-based net weighting scheme, and then uses simulated-annealing to trade-off wirelength of non-critical paths to delay improvement of the critical paths. [7] employs iterative net weighting scheme, which assign high net weights to two pin nets connecting modules passing through critical paths, to smooth critical paths.

Most existing timing-driven placement algorithms that uses net weighting scheme are based on global placement, in which cells are allowed to move freely. This approach is not suitable towards the end of the placement flow (e.g., detailed/incremental placement), because respecting the maximize displacement limit constraint and maintaining the cell distribution of the initial placement are important objectives at this stage. Although the works proposed in [7, 28, 29] are incremental timing-driven placement techniques, they restrict the movement of cells to the local regions of their current locations due to relying on inaccurate linear timing models. Consequently, timing improvement obtained using such techniques may not be optimal. Furthermore, most existing incremental timing-driven placements are based on computationally expensive LP techniques. As the modern deep-submicron circuits are large, consisting billions of transistors, LP based incremental timing-driven placements are not scalable due to the exponential amount of paths with timing violations in a design. Besides, most incremental timing-driven placement techniques do not concern about other placement objectives such as placement density constraints. This

might lead to excessive packing of cells within a neighborhood of placement region causing routing congestion issues.

In the next chapter, we address these issues by proposing a new incremental timing-driven placement algorithm based on greedy path-based technique which supports both global and local cell movements under the constraint of maximum displacement limit to maximize the *total negative slack* of the design. In chapter 4, we empirically show that our technique also provide fast timing convergence compared to other existing incremental timing-driven placement techniques.

Chapter 3

Timing-Driven Placement Algorithm

In this chapter, we provide the implementation of our timing-driven placement algorithm in detail. In section 3.1, an overview of the algorithm is provided. In section 3.2, some key operations are introduced to facilitate the detailed description of our algorithm discussed in sections 3.3 and 3.4.

3.1 Algorithm Overview

The abstract flow of our incremental timing-driven placement is given in Algorithm 1. The proposed algorithm reduces the total negative slack (late) of a circuit by moving critical cells to specific locations of the chip. A cell (combinational/sequential cell) is critical if and only if its input/output pins have late negative slacks annotated to them during STA. We decide the criticality of a cell based only on late mode STA for the following reasons:

- Since a design's performance (i.e., frequency) is only limited by the late timing violations, our focus is on reducing the TNS^{late} as much as possible and as fast as

possible. Generally, the effective way to reduce early timing violations can be done by buffer insertion technique at a later stage in the physical synthesis flow.

- Our experimental study (chapter 4) shows that, we would not perform worse in reducing TNS^{early} by choosing cells based on late slacks. In fact, the improvement we obtained in reducing the early timing violations are comparable to the results obtained in [28], which targets both early and late timing violations.

As such, Algorithm 1 uses a two stage approach to reduce the total negative slack of a circuit. In the first stage, known as *Global Move*, critical cells in the design are moved inside a *critical bounding box*, which is defined in section 3.2. This stage comprises two phases: in phase 1, only critical sequential cells movement are evaluated in an effort to reduce the total negative slack. We identify this phase as *sequential cell pass*. In the second phase, know as *combinational cell pass*, combinational cells found on critical paths are considered

Algorithm 1 Incremental Timing-Driven Placement

```
1:  procedure IncrementalTDP
2:      Input: Legal placement solution with timing violation
3:      Input: Maximum allowed cell displacement D
4:      Output: Legal placement solution with optimized timing violation
5:      Stage 1: Global Move
6:      Perform sequential_cell_pass
7:      (subject to cell displacement D)
8:      repeat
9:          Perform combinational_cell_pass
10:         (subject to cell displacement D)
11:     until timing improvement < threshold_g
12:     Stage 2: Local Move
13:     repeat
14:         Perform combinational_cell_pass
15:         (subject to cell displacement D)
16:     until timing improvement < threshold_l
```

to move inside the critical bounding box to maximize the total negative slack of the design. It must be noted that while phase 1 is performed only once, we iteratively repeat phase 2 until no further improvement in total negative slack is observed above a certain threshold.

The second stage of Algorithm 1 is identified as *Local Move*. During this stage, critical sequential cells of the design will not be moved. Instead, only critical combinational cells are moved locally. The way the algorithm selects combinational cells for the movement is the same as phase 2 of the *Global Move*, but the movement of each chosen combinational cell is limited to the rows above and below vertically, to the current location of the cell. This procedure is repeated iteratively until no further improvement in total negative slack of the circuit is observed above a certain threshold. In both stages of the algorithm the displacement of the cell movement from its original location is constrained by the maximum displacement limit D_{max} given as an input to Algorithm 1.

3.2 Key Operations

In this section, we introduce three key operations, namely *legalization of a cell move*, *parallel static timing analysis* and *incremental Steiner-tree computation*, needed to expand Algorithm 1 in detail.

3.2.1 Legalization of a Cell Move

In any incremental placement algorithm, legalization is an essential step to be implemented to avoid overlap among cells during/after the proposed cell movements. In the literature, there are two approaches used for legalizing the placement following the cell movements. In the first approach, all candidate cells are moved to their desired locations disregarding

the overlap among cells that might have occurred during the optimization process. This approach requires subsequent legalization step to remove overlap among cells [29, 30, 33]. However, this legalization is unaware of timing and placement density. This can lead to degradation in the timing optimization [7] as cell displacement with respect to the chosen location can be very high and can have negative impact on the placement density [34]. The other approach is to perform an instant legalization, in which placement of cells are kept legal after every cell move [34]. In this approach, cells can be placed as close as possible to their desired location while preserving the placement density. Since, while performing timing optimization, maintaining the initial placement density is essential in incremental placement flow, we adopt an instant legalization procedure whose steps are given in Algorithm 2.

In Algorithm 2, a cell is characterized by its physical coordinate of its center and by its width. Given the cell whose move to be evaluated, and its new location as input to Algorithm 2, it first seeks to find a legal position for the cell as close as possible to the new location as described from line 7 to line 22. Following that, any further overlap with nearby cells are removed iteratively by shifting operation as expressed by line 25 to line 34 of Algorithm 2. During this iterative overlap removal phase, we use a control parameter called $MOVE_{limit}$ to limit the number cells moved/shifted for the following reasons:

- The more the cells are shifted, the higher the chance of producing an overlap free cell movement. This, however, means more perturbation to the initial placement solution or moving a cell into high placement density region. Recall that maintaining the input placement density is one of the objective of incremental placement algorithms.

- Shifting more cells means longer runtime to legalize a cell movement. Furthermore, it would also increase the runtime of incremental Steiner tree computation (section

Algorithm 2 Legalization of a cell move

1: **procedure** LEGALIZECELLMOVE
2: **Input**: Cell C_j and its new position $(x_j,\ y_j)$
3: **Output**: True if legalization is successful, False otherwise
4: $R_j \leftarrow$ row of the location $(x_j,\ y_j)$
5: $C_l \leftarrow$ first cell at the left of $(x_j,\ y_j)$
6: $C_r \leftarrow$ first cell at the right of $(x_j,\ y_j)$
7: **if** $C_l = 0$ and $C_r = 0$ **then**
8: $x_{low} \leftarrow R_j.x_{left} + C_j.w/2$
9: $x_{high} \leftarrow R_j.x_{right} - C_j.w/2$
10: $x_j \leftarrow max(x_{low}, min(x_j, x_{high}))$
11: **else if** $C_l \neq 0$ and $C_r = 0$ **then**
12: $x_{low} \leftarrow C_l.x + (C_l.w + C_j.w)/2$
13: $x_{high} \leftarrow R_j.x_{right} - C_j.w/2$
14: $x_j \leftarrow min(x_{high}, max(x_j, x_{low}))$
15: **else if** $C_l = 0$ and $C_r \neq 0$ **then**
16: $x_{low} \leftarrow R_j.x_{left} + C_j.w/2$
17: $x_{high} \leftarrow C_r.x - (C_l.w + C_j.w)/2$
18: $x_j \leftarrow max(x_{low}, min(x_j, x_{high}))$
19: **else**
20: $x_{low} \leftarrow C_l + (C_l.w + C_j.w)/2$
21: $x_{high} \leftarrow C_r.x - (C_l.w + C_j.w)/2$
22: $x_j \leftarrow max(x_{low}, min(x_j, x_{high}))$
23: $(C_j.x, C_j.y) \leftarrow (x_j, y_j)$
24: $nMoved \leftarrow 1$
25: **while** $C_l \neq 0$ and $C_r = 0$ and $C_l.x + C_l.w/2 > C_j.x - C_j.w/2$ **do**
26: $C_l.x \leftarrow C_j.x - (C_l.w + C_j.w)/2$
27: $C_j \leftarrow C_l$
28: $C_l \leftarrow$ first cell at the left of C_l
29: $nMoved \leftarrow nMoved + 1$
30: **while** $C_r \neq 0$ and $C_r.x - C_r.w/2 < C_j.x + C_j.w/2$ **do**
31: $C_r.x \leftarrow C_j.x + (C_r.w + C_j.w)/2$
32: $C_j \leftarrow C_r$
33: $C_r \leftarrow$ first cell at the left of C_r
34: $nMoved \leftarrow nMoved + 1$
35: **if** $nMoved >= MOVE_{limit}$ or any cell in illegal position **then**
36: **return** False
37: **else**
38: **return** True

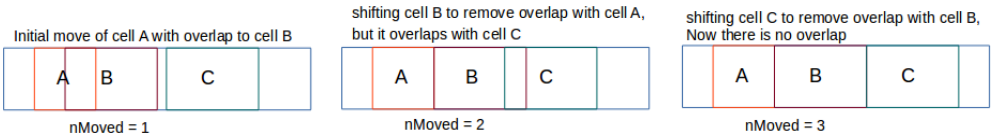

Figure 3.1: Example of legalization steps described in Algorithm 2.

3.2.3) phase of our placement algorithm.

As such, in our implementation, we choose $MOVE_{limit}$ to 5 as a tradeoff between increasing the chance of legalizing a cell move and avoid degrading the placement density as well as improving the runtime of the placer.

Figure 3.1 presents an example of our legalization technique of a cell move. Initially, cell A is moved to its chosen location. As a result, cell A overlaps with cell B. By shifting cell B so that it no longer overlaps with cell A has resulted in cell B overlaps with cell C. Thus, cell C is shifted to remove overlap with cell B. Following the move of cell C, there is no other overlap remains. Since the total number of cells moved during the legalization process is 3, which is less than $MOVE_{limit}$ (5), the procedure *LegalizeCellMove* would return true. If the number of cells moved surpass $MOVE_{limit}$ or any other violations such as a cell going beyond the boundary of the row, the procedure *LegalizeCellMove* would return false, indicating that all moved cell positions must be restored.

3.2.2 Parallel Static Timing Analysis

STA is an integral part of any timing-driven placement whether it is incremental or global. How often the STA engines are called during the placement varies depending on the implementation. In this work, the impact of timing on the circuit upon every legalized cell move is evaluated by performing STA on the new circuit.

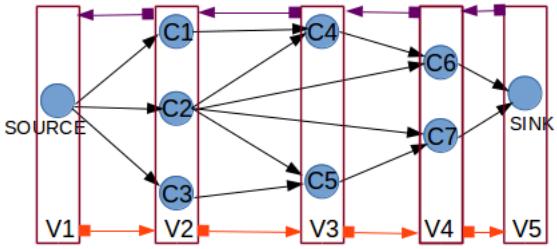

Figure 3.2: Capturing of nodes in the timing graph as a list of vectors during forward and backward topological sorting for the example circuit presented in Figure 2.7. The cells in vectors V1, V2, V3, V4 and V5 can be processed independently in that order (orange arrows) for arrival times computation and in reverse order (violate arrows) for required arrival times computation.

Even though the runtime of performing STA grows linearly on size of the circuit, calling STA engines for every move throughout the execution of the timing-driven placement algorithm can be computationally expensive. Thus, to reduce the runtime of the STA engine and hence to improve the overall runtime of our incremental timing-driven placement algorithm, we parallelize the STA.

Our OpenMP implementation of the STA is given in Algorithm 3. In Line 5 and 6 of the procedure *ParallelSTA*, the forward and backward topological sorting of the circuit are captured as a list of vectors where each cell in a vector can be processed for the computation of arrival times and required arrival times independently. For instance, as illustrated in Figure 3.2, for arrival times computation, the cells C1, C2, and C3 can be processed independently once the SOURCE node has been processed. As such, the forward and the backward list can be given as {{SOURCE}, {C1, C2, C3}, {C4, C5}, {C6, C7}, {SINK}} and {{SINK}, {C6, C7}, {C4, C5}, {C1, C2, C3}, {SOURCE}}}, respectively. Given the vectors of cells that can processed simultaneously, from line 10 through 16, arrival times of input/output pins for each cell is computed in parallel by calling *compute_at* method.

Algorithm 3 Static Timing Analysis

```
1: procedure ParallelSTA
2:     Input: Timing graph of the circuit, timing parameters of standard cells
3:     Output: at and rat of each input/output pin of the circuit
4:     Let LF, LB be list of vectors where each vector contains cells that can be processed
       independently
5:     LF ← perform forward topological sort of the circuit
6:     LB ← perform backward topological sort of the circuit
7:     #pragma omp parallel
8:     {
9:     for i ← 1 to | LF | do
10:        F ← LF[i]
11:        #pragma omp for
12:        {
13:        for j ← 1 to | F | do
14:            cell c ← F[j]
15:            compute_at(c)
16:        }
17:    #pragma omp barrier
18:    for i ← 1 to | LB | do
19:        B ← LB[i]
20:        #pragma omp for
21:        {
22:        for j ← 1 to | B | do
23:            cell c ← B[j]
24:            compute_rat(c)
25:        }
26:    }
```

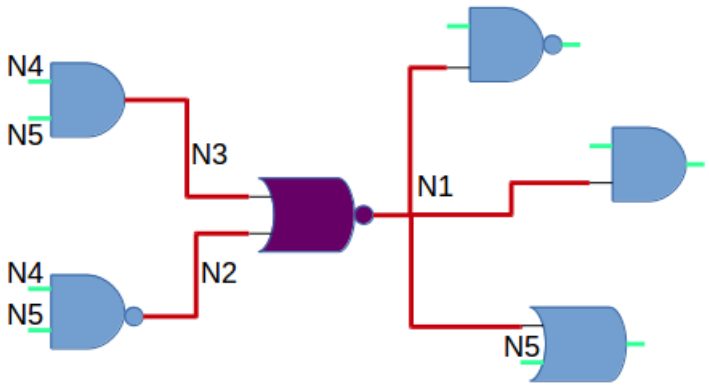

Figure 3.3: Incremental Steiner tree computation for cell A, where regeneration of Steiner tree is needed only for nets N1, N2 and N3.

Before the computation of required arrival time of input/output pin using *compute_rat* method in the similar way (from line 19 to 25), barrier synchronization is essential (line 17) as the slew propagation obtained during the arrival time computation of cells is required during the computation of required arrival time.

3.2.3 Incremental Steiner Tree Computation

To perform STA on the design, the interconnects must be modeled as RC tree networks, as discussed in Section 2.2.1. This, in turn, needs the topology of the routing tree of each net in the design. In this work, FLUTE [35], a rectilinear Steiner minimal tree algorithm, is used to generate routing tree for a net. During a cell move, there is no need to call FLUTE to regenerate Steiner-tree for each net in the circuit. Instead, it only has to be regenerated for nets associated with the moving cells (i.e., nets connected to the input and output pin of the cell) as illustrated in Figure 3.3. This incremental computation of Steiner tree can significantly reduce the overall runtime of Algorithm 1, as there is only, at most,

$MOVE_{limit} - 1$ extra cells needs to be moved (due to legalization) during a cell move.

3.3 Global Move

Most existing incremental timing-driven placement techniques limits the cell movement to a local window of its current location [28, 29, 30, 33]. This is because of their reliance on inaccurate or crude timing models of gate delay, interconnect delay, and etc. which might break down when cells are moved by large distance [8]. Consequently, this might lead to suboptimal solution as there can be a better solutions if the cells were allowed to move by larger distances. Therefore, we propose a global move technique which would relax the constraint of moving the cell only within a local region of its current location. We also rely on accurate timing information from STA engine due to allowing a cell to move by large distance from its current location.

3.3.1 Critical Bounding Box

During the global move of a cell, we require a better positioning of the cell. To this end, inspired by the median idea of [36] [1] for a cell, we define critical bounding box as a region where the candidate cell would be moved. The basic idea behind finding the critical bounding box for a critical cell i is that, to find a site for the cell i as close as possible to the center of the critical bounding box. This, in turn, can reduce the wirelength of nets that are critical to the cell i and thereby leads to improvement in the design timing. The critical bounding box for a cell i can be defined as follows: given that a cell i is critical, at least one of its input pins or output pins must have negative slacks associated with them.

[1] FastDP[16] uses this idea during its global swap stage to reduce HPWL of nets connecting the candidate cell.

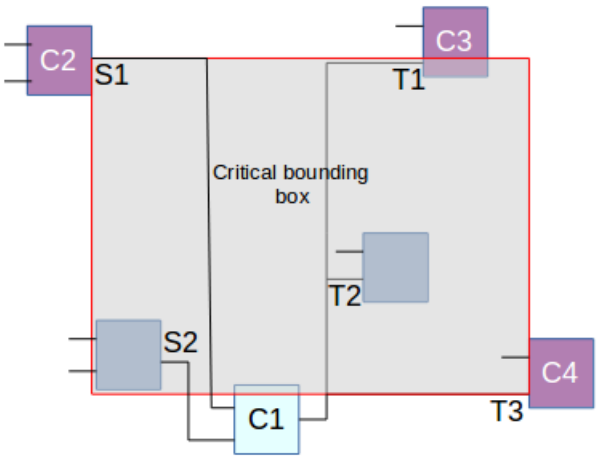

Figure 3.4: The critical bounding box of cell C1 is shown as the shaded region. It is defined by the critical pins T1, T2 of net driven by cell C1 and the critical pin S1 which is the source pin of the net that drives one of the input pin of cell C1. Pins T2 and S2 are ignored as they are not critical.

Let N_o be the set of nets driven by the critical output pins of the cell i. Also, Let N_i be the set of nets connected to the critical input pins of the cell i. For a net $p \in N_o$, we find the critical sink pins location (x_s^p, y_s^p), and for a net $p \in N_i$, we find the driver pin location (x_d^p, y_d^p). Upon finding the locations of the critical sink pins and critical driver pins, we find the optimal bounding box, denoted by (x_l^c, x_r^c, y_b^c, y_t^c - the left, right ,bottom and top boundaries), for the cell as follows:

$$x_l^c = \min(\{x_s^p\}, \{x_d^p\}) \tag{3.1}$$

$$x_r^c = \max(\{x_s^p\}, \{x_d^p\}) \tag{3.2}$$

$$y_b^c = \min(\{y_s^p\}, \{y_d^p\}) \tag{3.3}$$

$$y_t^c = \max(\{y_s^p\}, \{y_d^p\}) \tag{3.4}$$

Figure 3.4 shows the critical bounding box for cell C1. Net1 has critical sinks T1, T3, and Net3 has critical driver pin S1. We ignore pins T2, S2 as they are not critical. Therefore the bounding box construction involves coordinates (x_{T1}, x_{T3}, x_{S1}, y_{T1}, y_{T2}, y_{S1}). The resulting bounding box region is shaded in the Figure 3.4. Given the bounding box, the new site (new location) of the cell i is given by

$$x_i = \left\lfloor \frac{0.5(x_l^c + x_r^c)}{W_{site}} \right\rfloor W_{site} \tag{3.5}$$

$$y_i = \left\lfloor \frac{0.5(y_b^c + y_t^c)}{H_{row}} \right\rfloor H_{row} \tag{3.6}$$

The reason for choosing the new location of cell i as the center of the critical bounding box is to move cell i close to critical cells connecting the cell i. For example, according to Figure 3.4, when cell C1 is moved inside the center of critical bounding box, the cell C1 becomes close to the critical cells C2, C3 and C4, thereby reducing the interconnect lengths connecting cell C1.

3.3.2 Displacement-Aware Cell Move

Typically, in incremental placements, cells are moved under maximum displacement constraint D, to limit degradation in the quality of the input (initial) placement solution by moving cells by far distance [1, 7, 34]. As such, any cell move that violate the maximum displacement constraint must be aborted. But, disregarding a cell move, due to maximum displacement constraint violation, may lead to suboptimal timing improvement in the design. Such degradation in the timing improvement can be minimized by the proposed *displacement-aware cell move*. The displacement-aware cell move is defined as follows: upon finding that a cell C1 violates the maximum displacement constraint D, we construct

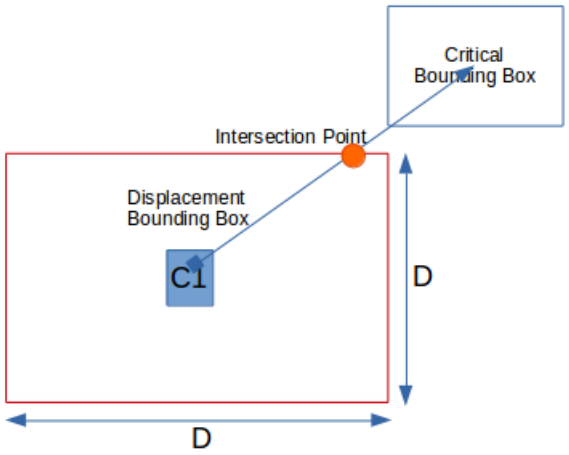

Figure 3.5: Displacement-aware cell move of cell C1: since the new location found by critical bounding box for cell C1 violates the maximum displacement limit D, the intersection point (orange) in the displacement bounding box is the new location for cell C1.

a $D \times D$ *displacement bounding box* around cell C1 as illustrated in Figure 3.5. Then, we draw a line from the cell's current location to the center of the critical bounding box of cell C1 and determine the intersection point $(x_p,\ y_p)$ of the line and displacement bounding box. Given the intersection point $(x_p,\ y_p)$, the new location of the cell C1 is given by

$$x_{c1} = \left\lfloor \frac{x_p}{W_{site}} \right\rfloor W_{site} \tag{3.7}$$

$$y_{c1} = \left\lfloor \frac{y_p}{H_{row}} \right\rfloor H_{row} \tag{3.8}$$

This new positioning of the cell $C1$ would help avoid aborts due to maximum displacement constraint violation, and would still move the cell C1 closer to other critical cells connecting the cell C1.

3.3.3 Global Move Algorithm

In Global Move, cells are moved inside critical bounding box under the constraint of maximum cell displacement to optimize the timing violations of the design. Our global move consist of two phases, namely, sequential cell pass and combinational cell pass.

(1) Sequential cell pass: In this phase, only critical sequential cells are considered to move. Sequential cell moves can provide large improvement in the timing [30], because (1) it allows trade-off of slack between path ending and starting at the sequential cell, (2) changes to clock latency values at the clock input pins (due to changing routing characteristics of the clock net), allows data arrival times be modified at the input (data) pins of other sequential cells, and thereby have an impact on the timing violations of the design. As such, the purpose of this phase is to eliminate the number of sequential cells with negative slack at their input pins, and then perform further timing optimization using combinational cell pass, given that the design remains with timing violations.

The steps of sequential cell pass is shown in Algorithm 4. Although we find critical bounding box for a critical sequential cell, it may not be possible to move it inside the critical bounding box due to maximum displacement limit constraint (lines 11 to 13). In that case, a new location that respects the displacement limit is found for the sequential cell, using the procedure described subsection 3.3.2 (line 14). It is possible that the newly found location for the cell may not improve the design's timing violations. Therefore, we accept or reject a move based on the *benefit* function B obtained by equation (3.7)

$$B = TNS_{prev} - TNS_{curr} \tag{3.9}$$

where TNS_{prev} and TNS_{curr} are total negative slack before and after the move. If $B < 0$, we accept the move as it indicates total negative slack of the design has improved from

Algorithm 4 Sequential Cell Global Move

```
1: procedure SequentialCellPass
2:     Input: Circuit with timing violation, Maximum Displacement limit D_max
3:     Output: Timing optimized circuit using sequential cell global move
4:     Build Steiner-Routing-Tree for each net
5:     ParallelSTA()
6:     Update slack for each pin
7:     tns_prev ← total negative slack
8:     V ← set of all sequential cells in the circuit
9:     for i ← 1 to | V | do
10:        if V[i] is critical then
11:            CBB ← Find critical bounding box
12:            Find (x_j, y_j) using equations 3.5 and 3.6
13:            if cell_disp > D_max then
14:                Find new (x_j, y_j) using equations 3.7 and 3.8
15:            if LegalizeCellMove(V[i], (x_j, y_j)) then
16:                Incrementally update Steiner-Routing-Trees for all moved cells
17:                ParallelSTA()
18:                tns_curr ← total negative slack
19:                if tns_prev − tns_curr < 0 then
20:                    Update slack for each pin
21:                    tns_prev ← tns_curr
22:                else
23:                    Restore all cell positions moved during legalization
24:                    Incrementally update Steiner-Routing-Trees for restored cells
```

the move, as well as update the TNS_{prev} to TNS_{curr}. Otherwise, we reject the move and leave TNS_{prev} unchanged (line 15 to line 24).

(2) Combinational cell pass: As the name suggest, in this phase only combinational cells are moved. We observed that moving every critical combinational cell in the design does not contribute to improvement in the timing of the design. In other words, moving combinational cells that doesn't belong to the most critical nets provide zero impact in the timing of the design. Therefore, to speed up the timing convergence, we only consider moving combinational cells those pass through critical paths between two adjacent sequential cells and between sequential cells and primary outputs.

Algorithm 5 Combinational Cell Global Move

```
1:  procedure CombinationalCellPass
2:      Input: Circuit with timing violation, Maximum Displacement limit D_max, tns_prev
3:      Output: Timing optimized circuit using combination cell global move
4:      V ← empty
5:      for all sequential cells with critical input pins and critical primary outputs P do
6:          n ← find the net that drives pin P
7:          C ← find the cell that drives net n
8:          if C is combinational cell then
9:              push(V, C)
10:     for i ← 1 to | V | do
11:         Perform Line 11 to 24 of Algorithm 4
12:         P ← most critical input pin of cell V[i]
13:         n ← find the net that drives pin P
14:         C ← find the cell that drives net n
15:         if C is combinational cell then
16:             push(V, C)
```

Algorithm 5 provides the description of the combinational cell pass. The candidate combinational cells are selected in reverse topological order, because critical path between two adjacent sequential cells or between a primary output and a sequential cell initiates

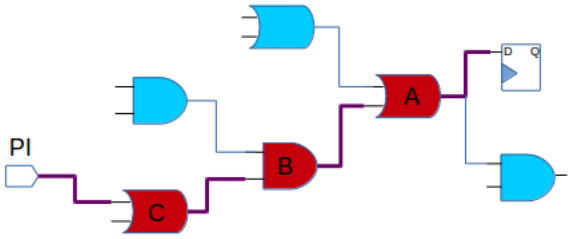

Figure 3.6: An example of combinational cell passing procedure between a flip-flop and primary input pin. Starting from the flip-flop first combinational cell will be selected in the following order: A, B and C. Note that these combinational cell pass through critical nets (violet).

from the primary output or the input pins of a sequential cell. Thus, starting from the input pin D of a critical sequential cell or primary output PO, we find the most critical net n that drives D or PO. We then pick the combinational cell C that drives the net n as the new candidate to be considered for moving into critical bounding box, to improve timing of the circuit (lines 5 to 11). Following that, to find new critical net n, we consider the driver net of most critical input pin of the combinational cell C. By finding the combinational cell C that drives the newly found critical net n, we have selected a new combinational cell candidate to be move inside the critical bounding box (lines 12 to 16). The aforementioned process will continue until the output pin of a sequential cell or primary input pin is found. Figure 3.6 provides an example of this process, where we will consider moving combinational cell in the following order: A, B and C.

3.4 Local Move

The objective of the local move is to provide finer improvement in timing by locally moving the cells from their current locations. In global move, for a cell, there might not be enough

space in the allocated site of the critical bounding box region. In that case, the cell move may have be aborted during the legalization process. We also suspect that the site found inside the critical bounding box for a cell may not be optimal. Thus, the idea here is to search for the less congested row above/below from the current position of a cell to reduce vertical wirelength and thereby further improve timing.

In this stage, for a combinational cell i, the row above and blow are considered to be candidate locations. Similar to the *vertical move* of [34], the x-position of the cell is not changed to ensure that the timing improvement is produced by allocating the cell in that row. The candidate combinational cells for this stage of the algorithm is selected in the same the way as described in Algorithm 5. A combinational cell i is first tried in the upper row from its current row. If the move is accepted (i.e, timing is improved), the next combinational cell $i + 1$ is tried, otherwise the cell i is tried again in the lower row from its current row, before moving onto next combinational cell $i + 1$.

Chapter 4

Experimental Study

In this chapter we present benchmarking methodology and numerical results to compare our algorithm to previous works. We also provide empirical results to justify our algorithm flow previously presented.

4.1 Benchmarking Methodology

This section provides details of benchmarking circuits and evaluation metrics used for experimental process.

4.1.1 Benchmarking Circuits

The experiments are conducted using the benchmark infrastructure provided by the ICCAD-2014 contest (problem B: incremental timing-driven placement) [1]. It comprises 7 benchmark circuits, with number of circuit elements ranging from 130k to 959k. Each benchmark contains the following files:

Table 4.1: Details of ICCAD-2014 incremental timing-driven placement contest benchmarks [1].

benchmark	**#Cells**	**#Nets**	**Target Utility**	**Clock Period** (ns)	**Short/Long Displacement limit** (um)
b19	219268	219290	0.76	5	20/200
vga_lcd	164891	164976	0.70	4	10/200
leon2	794286	794901	0.70	64	40/400
leon3mp	649191	649445	0.70	35	30/300
netcard	958792	960616	0.72	42	50/400
edit_dist	130674	133223	0.75	5	30/200
matrix_mult	155341	158527	0.70	4.4	30/200

- .lef (Cadence Library Exchange Format): provide definitions of design unit, sites, routing layers and available macros in the library.
- .def (Cadence Design Exchange Format): specifies chip dimension, placeable regions of the chip as row sites, locations of fixed/movable cells and pins and interconnect information.
- .sdc (Synopsys Design Constraints): specifies initial timing conditions such as input drivers and slews, load capacitance at primary outputs and clock period asserted on the design.

The original placement solution provided in the .def file is legal. To discourage any significant distribution from the original placement, maximize cell displacement constraint is imposed. For each circuit, there are two maximum cell displacement constraint (short and long) is provided. Table 4.1 shows all essential information about the benchmark circuits being used in this experiments.

4.1.2 Evaluation Metrics

The ICCAD-2014 contest proposed the following evaluation metrics to measure the quality of the results of the incremental timing-driven placement:

(1) Slack Improvement: With respect to initial timing result of the original placement, it is measured as the weighted average of improvements in TNS^{late}, WNS^{early}, TNS^{early} and WNS^{early}. The *slack improvement* is given by

$$\begin{aligned} slack\_improv. = w_t \times (w_l \times TNS\_improv.^{late} + w_e \times TNS\_improv.^{early}) \\ + w_w \times (w_l \times WNS\_improv.^{late} + w_e \times WNS\_improv.^{early}) \end{aligned} \tag{4.1}$$

where w_t, w_w, w_l, w_e is set 2.0, 1.0, 5.0, 1.0, respectively. From the weights, it is worth noting that great important is given to TNS than WNS and late slack than early slack improvements. Please note that $TNS_{improv.}$ and $WNS_{improv.}$ are measured in %.

(2) ABU Penalty: Given a placement solution and it target utilization Γ_{target}, $\gamma\_over\_utilization$ is defined as

$$\gamma\_over\_utilization = \max(ABU_\gamma/\Gamma_{target} - 1, 0) \tag{4.2}$$

When γ_over_utilization greater than zero, it indicates that the average utilization of top γ% density bins, excluding bins fully occupied by fixed macros, are tightly packed such that it is even greater than the design's target utilization. In [37], to measure the quality of a placement solution, a metric known as $ABU\_penalty$ is introduced in terms of γ_over_utilization and is defined as

$$ABU\_penalty = \frac{\sum(K_\gamma \times \gamma\_over\_utilization)}{\sum K_\gamma} \tag{4.3}$$

where $\gamma = \{2, 5, 10, 20\}$ and $\{K_2, K_5, K_{10}, K_{20}\} = \{10, 4, 2, 1\}$. The higher ABU_penalty, the lesser the quality of the placement and thus requires more cells spreading from highly dense placement regions.

With respect to ABU_penalty of the original placement, the impact on ABU_penalty is defined as

$$ABU\_penalty_{impact} = 1 - (ABU\_penalty^{final} - ABU\_penalty^{initial}) \tag{4.4}$$

where $ABU\_penalty^{final}$, $ABU\_penalty^{initial}$ are the ABU_penalties of final and initial placements.

(3) Quality Score: The quality score of the final placement is a function of equations 4.1 and 4.4. It is given by

$$quality\ score = slack\_improv \times ABU\_penalty_{impact} \tag{4.5}$$

If the placer increases the timing violations of the initial placement, the slack_improv would be negative, and so is the quality score.

4.1.3 Hardware/Software Environment

We implemented the algorithm in C++, and conducted experiments with quad-core Intel(R) Core(TM) i7-2620M, running at 2.70GHz with 10GB of RAM. The operating system is Ubuntu 14.04 LTS.

4.2 Sequential Cell Pass or Combinational Cell Pass?

During global move, it is possible to perform combinational cell pass, followed by sequential cell pass, or conversely. To determine the best flow (i.e.,Combinational/Sequential pass, Sequential/Combinational pass), we conducted an experiment where all benchmarks are run using each flow. As an experimental setup, for each flow, we provide maximum *1 hour runtime limit* per benchmark. Also, the experiments were conducted under short displacement constraint. The flow that performed better in terms of quality of the result and runtime is eventually selected as the feasible flow for the rest of the experimental study.

Table 4.2 summarizes the results for each flows. The first row of the each benchmark provides the timing results of the initial placement. The second and third row provides the timing results for our timing-driven placement with combinational/sequential cell pass flow and sequential/combinational cell pass flow during the global move stage, respectively. Columns 5 and 6 provide the overall slack improvement and runtime, respectively. A DNF entry in the Table 4.2 specifies that the results are unavailable for that benchmark, because the placer needs more than the 1 hour runtime limit imposed.

In terms of runtime, all benchmarks except *leon2* and *leon3mp*, were able to finish within 1 hour for both flows. Only sequential/combinational pass flow finished benchmarks *leon2* and *leon3mp* within an hour. This shows that allowing sequential cells to move at the start of the algorithm increases the speed of the timing convergence. Recall that, a sequential cell move can result in large timing improvement for the reasons described in the last chapter. It is worth mentioning that, during a combinational cell pass phase of the combinational/sequential pass flow, sequential cells can be moved during the legalization step of the algorithm. This implicit sequential cell move can also result in large improvement in timing. The *netcard* benchmark is an example of such situation

Table 4.2: Results for combinational/sequential (Com-Seq), sequential/combinational (Seq-Com) cell pass during *global move* with short displacement constraint.

benchmark	**solution**	**late wns** (*sec.*)	**late tns** (*sec.*)	**early wns** (*sec.*)	**early tns** (*sec.*)	**slack improv.**	**runtime** (*sec.*)
b19	initial	-1.16e-9	-1.58e-8	-3.76e-9	-1.14e-5	—	—
	Com-Seq	-2.43e-10	-6.42e-10	-1.99e-9	-4.65e-6	1521.17	196.34
	Seq-Com	-7.54e-11	-1.42e-10	-1.87e-9	-4.31e-6	1634.01	210.53
vga_lcd	initial	-1.33e-9	-6.39e-7	-4.58e-9	-4.75e-5	—	—
	Com-Seq	-4.02e-10	-2.19e-9	-3.34e-9	-3.32e-5	1425.50	383.84
	Seq-Com	-2.96e-10	-1.07e-9	-3.34e-9	-3.49e-5	1467.13	105.38
leon2	initial	-1.07e-8	-2.00e-5	-1.25e-7	-1.05e-2	—	—
	Com-Seq	DNF	DNF	DNF	DNF	DNF	DNF
	Seq-Com	0	0	-9.09e-8	-8.91e-3	1557.35	522.96
leon3mp	initial	-7.61e-9	-2.82e-5	-6.85e-8	-3.62e-3	—	—
	Com-Seq	DNF	DNF	DNF	DNF	DNF	DNF
	Seq-Com	0	0	-5.30e-8	-2.99e-3	1557.61	1273.99
netcard	initial	-7.51e-9	-7.55e-6	-1.08e-7	-7.35e-3	—	—
	Com-Seq	0	0	-8.69e-8	-5.96e-3	1557.49	284.75
	Seq-Com	-4.33e-10	-4.33e-10	-8.73e-8	-6.16e-3	1522.81	282.63
edit_dist	initial	-8.08e-10	-9.41e-8	-1.45e-9	-3.45e-6	—	—
	Com-Seq	-6.50e-10	-2.36e-8	-8.64e-10	-1.63e-6	992.68	1051.13
	Seq-Com	-6.77e-10	-2.79e-8	-9.17e-10	-1.75e-6	920.01	746.83
matrix_mult	initial	-4.41e-10	-2.61e-9	-3.59e-10	-1.10e-7	—	—
	Com-Seq	-2.91e-10	-1.30e-9	-3.41e-10	-9.17e-8	709.93	275.70
	Seq-Com	-3.01e-10	-1.29e-9	-3.32e-10	-8.16e-8	722.08	306.91
Avg. Improv.	Com-Seq	60.54%	84.13%	27.88%	35.55%	1241.35	438.35
	Seq-Com	62.73%	83.95%	28.20%	36.00%	1253.21	330.46

where a sequential cell move during the legalization of combinational cell pass provided large improvement in the design timing.

Nevertheless, excluding benchmarks *leon2* and *leon3mp*, sequential/combinational pass flow, on average, runs 1.32× faster than combinational/sequential pass flow and they both provides similar improvement in timing. This lead us to the conclude that performing sequential cell pass before combinational cell pass can result in faster timing convergence.

4.3 Empirical Validation

In this section, we analyze the results of applying our technique on ICCAD-2014 contest benchmarks [1], in terms of timing improvement, quality of the placement and computational runtime.

4.3.1 Timing Improvement

Table 4.3 and Table 4.4 presents the complete results of timing optimization obtained for ICCAD-2014 contest benchmarks using short and long displacement constraints, respectively. The late and early worst negative slacks and the late and early total negative slacks (LWNS, EWNS, LTNS, and ETNS) are shown is columns 3, 5, 4, and 6, respectively. The 1^{st} and 2^{nd} row of each benchmark provides the timing metrics obtained from input (global) placement and the timing metrics obtained after applying our optimization technique on the input placement, respectively.

Late WNS and TNS: Our technique reduces late timing violations significantly. The amount of late timing reduction for each circuit under short and long displacement constraint is given in Figure 4.1. Under short displacement constraint late timing violations of circuits *leon2*, and *leon3mp* are reduced to zero, whereas under long displacement con-

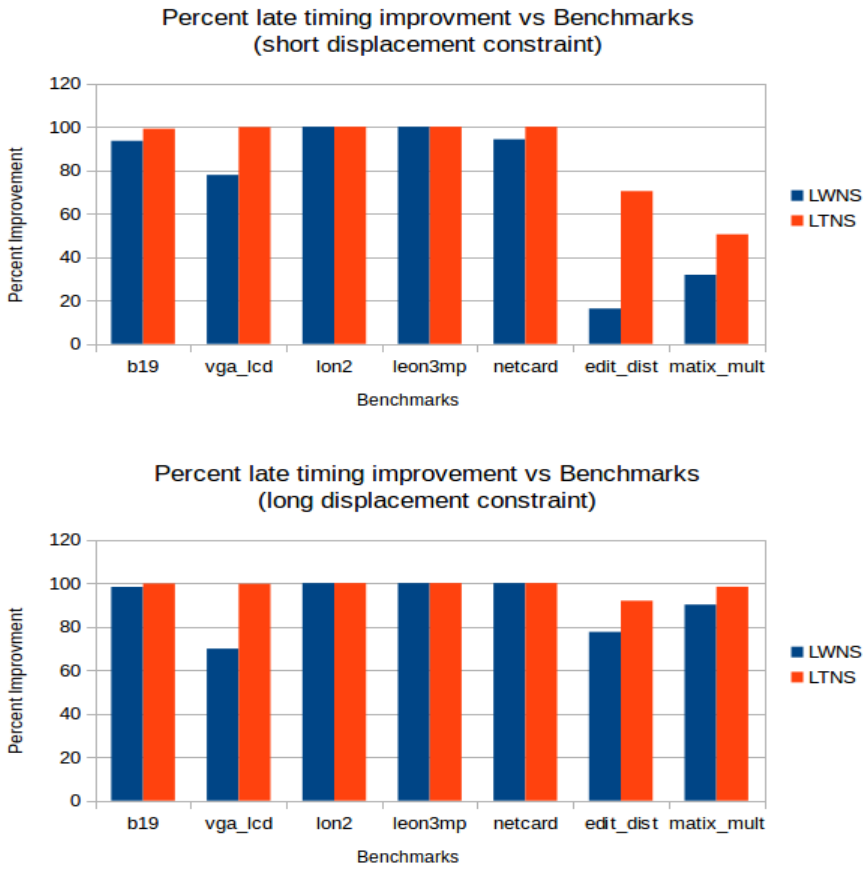

Figure 4.1: Late WNS and TNS improvement for each benchmark circuit with short displacement constraint and long displacement constraint.

Table 4.3: Results for ICCAD-2014 incremental timing-driven placement contest benchmarks using short displacement constraint. For each benchmark, table presents metrics for initial, ours, and contest 1st-place solutions.

benchmark	**solution**	**lwns** (*sec.*)	**ltns** (*sec.*)	**ewns** (*sec.*)	**etns** (*sec.*)	**ABU** (e-2)	**obt./max disp** (*um*)	**Quality score**	**runtime** (*sec.*)
b19	initial	-1.16e-9	-1.58e-8	-3.76e-9	-1.14e-5	2.59	—	—	—
	Ours	-7.54e-11	-1.42e-10	-1.87e-9	-4.31e-6	2.59	18.1/20	**1634.01**	210.53
	1st-place	-2.09e-10	-5.31e-10	-1.95e-9	-4.52e-6	2.59	19.8/20	1546.31	**61.93**
vga_lcd	initial	-1.33e-9	-6.39e-7	-4.58e-9	-4.75e-5	1.25	—	—	—
	Ours	-2.96e-10	-1.07e-9	-3.34e-9	-3.49e-5	1.29	8.7/10	**1467.13**	105.38
	1st-place	-3.30e-10	-2.17e-9	-3.27e-9	-3.18e-5	2.17	10/10	1454.50	**104.40**
leon2	initial	-1.07e-8	-2.00e-5	-1.25e-7	-1.05e-2	2.45	—	—	—
	Ours	0	0	-9.09e-8	-8.91e-3	2.46	34.6/40	1557.35	**522.96**
	1st-place	0	0	-8.22e-8	-7.79e-3	2.46	3.2/40	**1585.57**	1073.95
leon3mp	initial	-7.61e-9	-2.82e-5	-6.85e-8	-3.62e-3	0.78	—	—	—
	Ours	0	0	-5.30e-8	-2.99e-3	0.79	17.7/30	**1557.61**	**1273.99**
	1st-place	0	0	-5.41e-8	3.09e-3	0.78	3.2/30	1550.51	2494.19
netcard	initial	-7.51e-9	-7.55e-6	-1.08e-7	-7.35e-3	1.13	—	—	—
	Ours	-4.33e-10	-4.33e-10	-8.73e-8	-6.16e-3	1.14	29.2/50	1522.81	**282.63**
	1st-place	0	0	-8.05e-8	-5.78e-3	1.13	6.4/50	**1568.42**	7269.10
edit_dist	initial	-8.08e-10	-9.41e-8	-1.45e-9	-3.45e-6	0	—	—	—
	Ours	-6.77e-10	-2.79e-8	-9.17e-10	-1.75e-6	0	29.4/30	**920.01**	**746.83**
	1st-place	-7.02e-10	-3.37e-8	-8.34e-10	-1.48e-6	0	4.8/30	864.55	5798.32
matrix_mult	initial	-4.41e-10	-2.61e-9	-3.59e-10	-1.10e-7	0	—	—	—
	Ours	-3.01e-10	-1.29e-9	-3.32e-10	-8.16e-8	0	27.3/30	**722.08**	**306.91**
	1st-place	-3.96e-10	-1.72e-9	-2.81e-10	-4.39e-8	0	4.8/30	534.46	2751.06
Avg. Red.	Ours	73.38%	88.53%	27.26%	30.37%	-0.88	—	**1340.14**	**492.75**
	1st-place	68.69%	84.94%	31.71%	38.98%	-10.6	—	1300.62	2793.28

straint, late timing violations of circuits *leon2*, *leon3mp* and *netcard* are completely eliminated. On average, the late WNS and TNS reductions are 73% and 89%, respectively, under short displacement constraint, and 91% and 99%, respectively, under long displacement constraint. The late timing improvement of circuits *edit_dist* and *matrix_mult* are significantly better when applying long displacement constraint than short displacement constraint. In all other circuits, the late timing improvements are similar regardless of the type of displacement constraint applied. This shows that cell movement by largest distant can sometimes result in better timing improvement.

Early WNS and TNS: Although, we made no effort in reducing the early slacks, there were reduction in early WNS and TNS. One can observe that, under both displacement

Table 4.4: Results for ICCAD-2014 incremental timing-driven placement contest benchmarks using long displacement constraint. For each benchmark, it presents metrics for initial, ours, and contest 1st-place solutions.

benchmark	**solution**	**lwns** (*sec.*)	**ltns** (*sec.*)	**ewns** (*sec.*)	**etns** (*sec.*)	**ABU** (e-2)	**obt./max disp** (μm)	**Quality score**	**runtime** (*sec.*)
b19	initial	-1.16e-9	-1.58e-8	-3.76e-9	-1.14e-5	2.59	—	—	—
	Ours	-2.04e-11	-2.91e-11	-2.01e-9	-4.75e-6	2.59	135.3/200	**1653.35**	303.54
	1st-place	-1.09e-10	-2.08e-10	-1.95e-9	-4.52e-6	2.66	160.9/200	1608.60	**63.00**
vga_lcd	initial	-1.33e-9	-6.39e-7	-4.58e-9	-4.75e-5	1.25	—	—	—
	Ours	-4.02e-10	-1.92e-9	-3.40e-9	-3.72e-5	1.45	188/200	1412.66	235.22
	1st-place	-2.49e-10	-1.76e-9	-3.27e-9	-3.16e-5	2.99	199/200	**1473.79**	**106.90**
leon2	initial	-1.07e-8	-2.00e-5	-1.25e-7	-1.05e-2	2.45	—	—	—
	Ours	0	0	-9.14e-8	-8.69e-3	2.46	2.4/400	1561.15	1343.74
	1st-place	0	0	-8.22e-8	-7.79e-3	2.46	3.2/400	**1585.57**	**1091.72**
leon3mp	initial	-7.61e-9	-2.82e-5	-6.85e-8	-3.62e-3	0.78	—	—	—
	Ours	0	0	-5.06e-8	-2.71e-3	0.79	116/300	**1576.33**	2497.38
	1st-place	0	0	-5.41e-8	3.09e-3	0.78	3.2/300	1550.51	**2494.19**
netcard	initial	-7.51e-9	-7.55e-6	-1.08e-7	-7.35e-3	1.13	—	—	—
	Ours	0	0	-8.58e-8	-6.20e-3	1.14	318.5/400	1551.90	**379.17**
	1st-place	0	0	-8.05e-8	-5.78e-3	1.13	6.4/400	**1568.42**	7274.96
edit_dist	initial	-8.08e-10	-9.41e-8	-1.45e-9	-3.45e-6	0	—	—	—
	Ours	-1.81e-10	-7.60e-9	-9.16e-10	-1.74e-6	0	195.4/300	**1442.98**	**611.22**
	1st-place	-7.02e-10	-3.37e-8	-8.34e-10	-1.48e-6	0	4.8/300	864.55	5755.20
matrix_mult	initial	-4.41e-10	-2.61e-9	-3.59e-10	-1.10e-7	0	—	—	—
	Ours	-4.35e-11	-4.35e-11	-3.38e-10	-8.70e-8	0	179/300	**1481.38**	**173.63**
	1st-place	0	0	-2.81e-10	-4.39e-8	0	181.4/300	1641.82	2757.76
Avg. Red.	Ours	90.83%	98.54%	26.97%	29.80%	-2.60	—	**1525.68**	**782.64**
	1st-place	83.58%	94.66%	31.71%	39.04%	-20.3	—	1470.47	2791.96

constraint, the average improvement in early WNS and TNS are 27% and 30%, respectively. This result is comparable to the early timing improvement (EWNS: 35%, ETNS: 43%) obtained by techniques such as [28] which targets slack reduction for both early and late timing. The improvement in early timing violation mostly came from global sequential cell phase of our technique. We suspect that the reason for this is as follows: a design has early timing violation due to signal arrives at timing points earlier than expected. To optimize early timing violation, interconnects of cells that lie in the early timing violation path must be increased. During our sequential cell pass phase of Algorithm 1, when the output pin is late timing critical and the input pin is early timing critical, moving the sequential cell inside the critical bounding box causes the interconnect associated with the output pin be

decreased while the interconnect associated with the input pin be increased. Consequently, a sequential cell move can result in improving both early and late timing violations.

4.3.2 Placement Quality

It is essential to avoid major degradation to the initial global placement quality in terms of placement bin density while performing the timing optimization.

The impact on a placement bin density is measured using ABU_penalty metric. In Tables 4.3 and 4.4, column 7 provides the ABU_penalties of placement solutions for each benchmark. An increase in ABU_penalty from initial to timing-optimized solution is penalized. As can been, the overhead in ABU_penalty to perform timing optimization using our technique is marginal. Under short and long displacement constraint, we have only increased the ABU_penalty, on average, by 0.88% and 2.60%, respectively. The largest increase in ABU penalty only came from circuit *vga_lcd*.

Taking both factors (slack improvement and ABU_penalty) into account, the quality score of our timing-driven placer is given in column 9 of the Tables 4.3 and 4.4. On applying our timing-driven placement technique, the obtained average quality score over all benchmarks, is 1340.14 and 1525.68, under short and long displacement constraints, respectively.

4.3.3 Runtime

Table 4.5 presents the runtime breakdown of optimizing the timing of considered benchmarks using our technique. The runtime of global sequential cell pass (gscp) is a function of the number of sequential cells in the circuit (determines the Steiner routing tree for the clock net) and the number of critical sequential cell moves evaluated during the sequential cell pass. This explains the larger runtime of the sequential cell pass phase for

Table 4.5: Runtime breakdown of using our technique for timing optimization on ICCAD-2014 benchmarks. (*gscp* - global sequential cell pass, *gccp* - global combinational cell pass, *lccp* - local combinational cell pass)

	under short displacement constraint				**under long displacement constraint**			
benchmark	**gscp** (*sec.*)	**gccp** (*sec.*)	**lccp** (*sec.*)	**runtime** (*sec.*)	**gscp** (*sec.*)	**gccp** (*sec.*)	**lccp** (*sec.*)	**runtime** (*sec.*)
b19	49.19	68.03	84.46	210.53	53.94	86.57	154.19	303.54
vga_lcd	47.79	22.49	27.51	105.38	29.41	111.81	86.45	235.22
leon2	434.42	0	0	522.96	1255.44	0	0	1343.74
leon3mp	1223.28	0	0	1273.99	2380.73	0	0	2431.93
netcard	64.94	26.57	31.27	282.63	204.60	14.94	0	379.17
edit_dist	47.87	278.71	395.62	746.83	47.52	285.53	253.55	611.22
matrix_mult	5.14	128.08	137.30	306.91	6.11	79.70	52.26	173.63
Avg.	267.52	74.84	76.98	492.75	568.25	82.65	78.06	782.64

circuits *leon3mp* and *leon2*, as they are two of larger circuits (≈650k and ≈800k gates) among the benchmarks. Although the circuit *netcard* is large (≈1M gates), its runtime of sequential cell pass phase is low, as the number of critical sequential cell evaluation is low. Also, for circuits *leon2* and *leon3mp*, runtime of combinational cell pass of both global and local stages is zero, as the global sequential cell pass eliminates all the late timing violations. Overall, global sequential cell pass, global and local combinational cell pass requires 63.45%, 12.88% and 12.80% of the total runtime, respectively.

4.4 Comparison

In this section we compare our results to the top three teams of the ICCAD-2014 contest. The results of the top three teams were obtained from the contest website [38] and it only provides the results in terms of *quality score* and *runtime*. The computation platform used in the contest has the following capabilities: CPU: 32 × 64-bit Intel(R) Xeon 2.60GHz, main memory: 64 GB and OS: CentOS release 6.2. This is slightly different from our

computational platform.

4.4.1 Comparison to Contest Results

The results of the top three teams from the ICCAD-2014 contest and ours are presented in Tables 4.6 and 4.7 for the considered benchmarks under short and long displacement constraints, respectively. In tables 4.6 and 4.7, placer that obtained best quality score per benchmark is highlighted in bold face font.

Table 4.6: Comparison of results for ICCAD-2014 timing-driven placement contest benchmarks under short displacement constraint. The results are presented for top three teams from the contest and ours in terms of quality score and runtime.

	Quality score				**runtime** (*sec.*)			
Benchmark	**1st-place**	**2nd-place**	**3rd-place**	**Ours**	**1st-place**	**2nd-place**	**3rd-place**	**Ours**
b19	1546.31	1580.78	1615.01	**1634.01**	51.45	90.78	166.71	210.53
vga_lcd	1454.50	1394.60	**1492.01**	1467.13	92.54	176.18	135.15	105.38
leon2	**1585.57**	819.50	1253.90	1557.35	1086.26	1476.90	2565.76	522.96
leon3mp	1550.51	1513.95	**1564.71**	1557.61	2649.71	3523.07	253.05	1273.99
netcard	**1568.42**	1279.01	1561.13	1522.81	8044.91	795.59	1611.78	282.63
edit_dist	864.55	1137.57	**991.49**	920.01	4942.80	57.78	94.25	746.83
matrix_mult	534.46	**1032.93**	790.86	722.08	2353.89	84.99	97.32	306.91
AVG.	1300.62	1251.19	1289.64	**1340.14**	2745.94	886.47	703.43	492.75

Table 4.7: Comparison of results for ICCAD-2014 timing-driven placement contest benchmarks under long displacement constraint. The results are presented for top three teams from the contest and ours in terms of quality score and runtime.

	Quality score				**runtime** (*sec.*)			
Benchmark	**1st-place**	**2nd-place**	**3rd-place**	**Ours**	**1st-place**	**2nd-place**	**3rd-place**	**Ours**
b19	1608.60	**1661.35**	1615.01	1653.35	51.44	91.33	184.10	303.54
vga_lcd	**1473.79**	1403.32	1413.61	1412.66	92.25	254.32	146.57	235.22
leon2	**1585.57**	822.98	266.94	1561.15	1143.68	1447.72	2574.01	1343.74
leon3mp	1550.51	1512.60	1564.71	**1576.33**	2666.14	3502.40	256.95	2497.38
netcard	**1568.42**	1279.28	1464.87	1551.90	8188.74	787.96	1647.60	379.17
edit_dist	864.55	1113.14	906.96	**1442.98**	4947.81	47.74	102.30	611.22
matrix_mult	**1641.82**	1520.62	415.22	1481.38	2343.55	86.89	104.64	173.63
AVG.	1470.47	1330.47	1092.48	**1525.68**	2776.23	888.34	716.59	782.64

(1) Quality Score

According to Tables 4.6 and 4.7, in terms of the quality score, we outperform other teams in benchmark *b19* under short displacement constraint and in benchmarks *leon3mp* and *edit_dist* under long displacement constraint. In other benchmarks, our quality score is very competitive to the team who achieved the best quality score. Also, our performance is consistently high regardless of the type (short/long) of maximum displacement constraint. This can be seen from the quality score comparison graph provided in Figure 4.2. This is the reason why we outperform all other teams in terms of the average quality score under both short and long displacement constraints.

(2) Runtime

Because of the computational platform difference between ours and the contest, a direct runtime comparison between our placer and other teams using Tables 4.6 and 4.7 is impossible. However, we would like to ignore this difference for the average runtime comparison and differ the reason for it to section 4.4.2. According to Tables 4.6 and 4.7, our runtime, on average, could be ranked first under short displacement constraint and second under long displacement constraint. It is also worth noting that, in the contest, the quality score was given more importance than runtime in deciding the winner of the competition.

4.4.2 Comparison to the 1st-place Team

We compare our results to the 1st-place team in greater detail, because we were able to get the binary of their placer and able to collect runtime accurately. The row 3 of the Tables 4.3 and 4.4 provides the results obtained by the 1st-place team for each benchmark. The bold face entries in Tables 4.3 and 4.4 show the placer which produced the best result per benchmark in terms of quality score and runtime.

Figure 4.2: Normalized quality score comparison of the top three teams of the ICCAD-2014 contest and ours under short displacement constraint and long displacement constraint.

Late WNS and TNS: With short displacement constraint, in all but circuit *netcard*, we outperform 1st-place team in improving late timing violations. Also, with long displacement constraint, our technique provides better improvement in late timing in all circuits except *vga_lcd*. On average, we improve late WNS and late TNS by, 6%, 5%, respectively, under short displacement constraint and 8%, 4%, respectively, under long displacement constraint.

Early WNS and TNS: In improving early timing violations, we were outperformed by the 1st-place team in many instances except in circuits *b19* and *leon3mp*. This is expected as our primary focus was on reducing late timing violations. On average, 1st-place team performed better in improving early wns and early tns by, 18%, 30%, respectively, under short and long displacement constraint. It is worth noting that the ICCAD-2014 contest gives twice as much importance for improving late timing violations as for improving early timing violations (equation 4.1).

Quality Score: The overhead of performing timing-driven placement on global placement solutions (measured by ABU_penalty) were kept to minimum in both of our technique, albeit a larger ABU_penalty change in circuit *vga_lcd* by the 1st-place team. Taking into account the slack improvements and the impact on ABU_penalties, our performance is better on 5 out of 7 circuits and 4 out of 7 circuits under short and long displacement constraint, respectively. This is, on average, 3% increase in the quality by our timing-driven placement technique.

Runtime: Regardless of the type of displacement constraint applied, we require significantly less computational time than that of 1st-place team except in circuits *b19* and *vga_lcd*. Under short displacement constraint, our placer is 5.67× faster than 1st-place team, whereas under long displacement constraint our placer is 3.6× faster than 1st-place team.

Given the runtime of 1st-place team for each benchmark in our computation platform (Tables 4.3 and 4.4) and the contest platform (Tables 4.6 and 4.7), we can infer that the runtime difference is insignificant per benchmark. In terms of average runtime, we can observe that our computational platform is slightly inferior to the contest platform. This is the reason why we ignored the runtime difference in section 4.4.1.

Chapter 5

Additional Experimentation

The purpose of this chapter is to explore runtime reduction opportunities of our incremental timing-driven placement algorithm and to outline some of the challenges we faced in doing so.

5.1 Clock Net Routing

The *global sequential cell pass* is the most time consuming phase of our timing-driven placement algorithm, at least for the larger benchmarks such as *leon2*, *leon3mp* and *netcard* due to the presents of large number of sequential cells. The runtime of this phase can be defined as

$$runtime_{scp} \propto N_{sc} \times T_{cnr} \tag{5.1}$$

where N_{sc} is the number of sequential cell moves accepted/rejected and T_{cnr} runtime of a clock net routing tree generation. In this work, nets are routed using FLUTE [35], which is a minimal Steiner routing tree algorithm with $O(n^2)$ runtime. Since a single clock source drives all the sequential cells for the considered benchmarks, routing the clock net using FLUTE is the most time consuming step (i.e.,T_{cnr}) of the sequential cell pass phase of our technique. Table 5.1 gives some perspective as to the amount of time spent on routing the

clock net compared to total amount of time taken by the sequential cell pass.

Table 5.1: Comparison of amount time spent on routing clock net to time spent on sequential cell pass.

benchmark	**Type of disp. const. limit**	**clock net routing** $T_{cnr}(sec.)$	**sequential cell pass** $T_{scp}(sec.)$	$(\mathbf{T}_{cnr}/\mathbf{T}_{scp})\%$
leon2	short	395.99	434.42	91%
leon3mp	short	1008.99	1223.28	82%
leon2	long	1097.57	1255.44	87%
leon3mp	long	1957.12	2380.73	82%
netcard	long	180.56	204.59	88%

In larger circuits, on average, 86% of the time is spent on routing the clock net using FLUTE. Consequently, by reducing the runtime of routing the clock net, the runtime of the sequential cell pass can be improved. On the other hand, according to equation 5.1, any reduction on the time spent on routing the clock net (T_{cnr}) should not increase the amount of sequential cell moves accepted/rejected (N_{sc}) in a way that the gain we obtain in reducing the T_{cnr} be lost by increasing the N_{sc}. Also we have to make sure that the techniques we apply to reduce T_{cnr} result in improving the timing of the initial placement solution, at all times.

5.1.1 Reducing the Runtime of Clock Net Routing

In an effort to reduce the runtime of routing a clock net, we attempted two heuristics based on FLUTE that would route clock nets faster. Unfortunately, the results are not promising and unpredictable. Therefore, the purpose of this section is to explain the reasons for the failure of our heuristics and lead the discussion to the changes proposed to the *netlist* in the benchmarks of ICCAD-2015 contest [5] that would solve this problem.

Here, we consider one heuristic [1] to reduce clock routing time described as follows:

[1] Additional heuristic is not discussed here, because the conclusions we obtained from our first heuristic

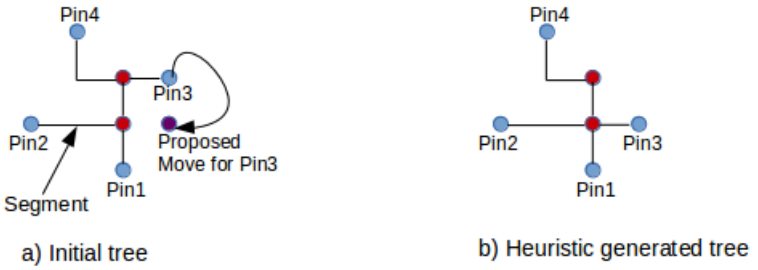

Figure 5.1: Example of heuristic clock routing tree generation for a four pin clock net.

1. Initially, we route the clock net using FLUTE. Since FLUTE is a minimal Steiner routing tree algorithm, it represents the tree using set of connected clock pins and Steiner points. A connection between two Steiner points or between a Steiner point and a clock pin is know as a *segment*. We save this in a data structure.

2. When a sequential cell move is proposed, we remove the segment associated to the moved clock pin from the tree. Then, we search for a Steiner point location that is closer to the new location of the clock pin in terms of Manhattan distance between the new location of the clock pin and a Steiner point.

3. Upon finding the closest Steiner point, we create a new segment between the new locations of the clock pin and the Steiner point. This completes the steps for generation of a heuristic clock routing tree.

The aforementioned procedure attempts to locally modify the clock tree incrementally and it can be performed in linear time on the size of the clock pins plus Steiner points. Figure 5.1 presents an example of generating clock net routing tree using the heuristic for a 4 pin clock net. Figure 5.1(a) illustrate the initial tree generated by FLUTE. When a sequential cell associated with the Pin3 is moved, we apply the step 2 of our heuristic to

applies to the second heuristic as well.

find the closest Steiner point to Pin3 and then apply the step 3 of our heuristic as shown in Figure 5.2(b). This heuristic generated tree is different from the tree would have been generated by FLUTE, if we were to compute it from the scratch. Therefore, we additionally setup a *refresh counter* that would count up on every sequential cell move proposed, and when it reaches a predefined threshold value N_{uf}, a new tree is generated by FLUTE from scratch.

To test our technique, we provide the following experimental setup: since the focus of this study is to speed up the *global sequential cell pass* step, we ignore the combinational cell pass step of the original algorithm. For simplicity, we only provide the quality score and the runtime of the sequential cell pass. Furthermore, we limit the runtime to *1 hour* per benchmark and the N_{uf} is swept with values ∞ (i.e., FLUTE is not used at all), 10, 5 and 0 (i.e., FLUTE is used 100% to route the clock net).

Table 5.2: Quality score and the runtime results for heuristic clock net routing technique using ICCAD-2014 contest benchmarks under short displacement constraint.

	$N_{uf} = \infty$		$N_{uf} = \mathbf{10}$		$N_{uf} = \mathbf{5}$		**Original ($N_{uf} = 0$)**	
benchmark	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)
b19	1416.32	100.90	200.37	90.77	**-33.71**	63.37	1605.01	63.86
vga_lcd	1442.57	370.54	209.62	54.41	434.03	76.79	1467.13	60.34
leon2	DNF	DNF	1551.47	878.13	**-18675.20**	605.26	1557.35	577.15
leon3mp	DNF	DNF	1578.78	894.29	1559.95	2147.26	1557.61	1324.42
netcard	DNF	DNF	1564.45	254.45	1562.42	338.10	1521.57	270.30
edit_dist	575.83	84.14	597.44	62.96	140.34	66.81	710.48	77.22
matrix_mult	69.66	43.33	69.66	42.66	69.66	42.99	87.34	46.20

The results are presented in Table 5.2. When compared to the results with $N_{uf} = 0$ (Table 5.2: columns 8 and 9), the $N_{uf} = \infty$ (Table 5.2: columns 2 and 3) couldn't able to finish before the runtime limit expired for the larger designs such as *leon2*, *leon3mp* and *netcard*. The reason for this is as follows: as illustrated in Figure 5.2, when using FLUTE for clock tree evaluation, we get large variations in the TNS compared to the stable TNS obtained when using heuristic. To explain the variability, we look into example clock

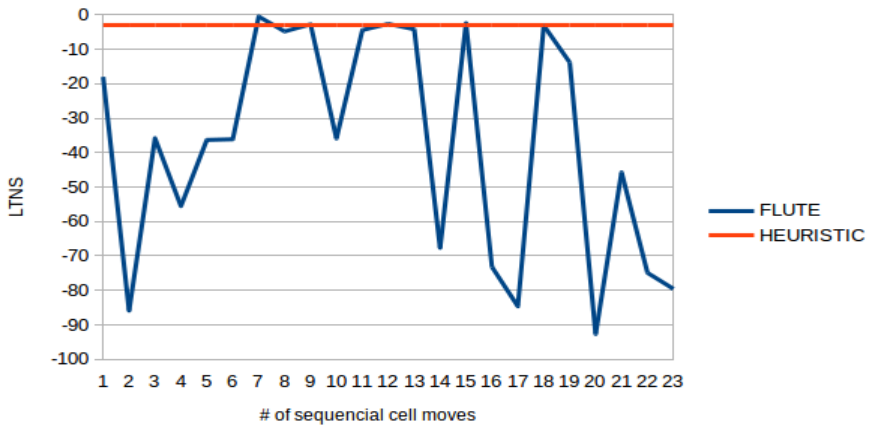

Figure 5.2: The impact on late total negative slack using FLUTE generated clock net versus heuristic generated clock net for benchmark leon3mp.

trees given in Figure 5.3: between two successive moves of sequential cells, changes to the FLUTE generated clock tree can be global. That is, the clock tree can be changed not only in the region around the moved sequential cell, but also in other regions of the tree. Consequently, clock latency values of all sequential cells can change significantly. This, in turn, can have significant impact on arrival times, required arrival times and hence slacks of all timing points of the circuit. As a result, we might observe significant impact on TNS between two successive sequential cell moves as shown in Figure 5.2. But, since our heuristic only modify the clock tree locally around the moved sequential cell, the impact on the timing is negligible, thereby wouldn't be able to provide faster timing convergence.

This limitation of our heuristic is somehow rectified when N_{uf} is less than ∞. With $N_{uf} = 10$ (Table 5.2: columns 4 and 5), we obtain comparable performances to $N_{uf} = 0$ for all benchmarks but *b19* and *vga_lcd* in terms of runtime and performance. Here, it is worth noting that the larger improvement in the timing came from, whenever a sequential cell move is accepted during FLUTE is used to route clock net than heuristic. Although, when

$N_{uf} = 10$, the timing of each benchmark has been improved from their initial timing, the timing improvement cannot always be guaranteed due to the large disagreement between the routing characteristic of clock trees generated by FLUTE and the heuristic. This can be observed from the results obtained for $N_{uf} = 5$ (Table 5.2: columns 6 and 7). Here, we have negative quality score for designs *b19* and *leon2*. By approximating the routing characteristic of the clock net, we are accepting/rejecting sequential cell moves using false timing values. Therefore, a wrong sequential cell move acceptance based on false timing improvement can irretrievably worsen the timing violations of a circuit from their initial timing.

In conclusion, the main challenge in approximating FLUTE generated tree is the unstable behavior of FLUTE as shown in Figure 5.3. Such an unstable clock net routing also leads to unpredictable timing behavior of the circuit. The main problem for the considered benchmarks is that (1) allowing single clock source to drive the entire sequential cells in the circuit and (2) using FLUTE to route large clock nets. These features will be eliminated in ICCAD-2015 contest[5] benchmarks by the introduction of *Local Clock Buffer* (LCB). The idea behind LCBs is to allow each LCB to drive only up to few sequential cells, for example 40 to 50 sequential cells. This would, in turn, facilitate (1) a faster clock net routing tree generation, because the size of the clock net would be equal to as many sequential cell as an LCB drives (2) any reasonable routing approximation would be able to replace FLUTE, as the size of the clock net is too small, on any reasonable placement, actual route wouldn't make that much difference [5].

5.2 Impact of Sequential Cell Ordering

The purpose of this section is to evaluate the impact on the quality score and runtime by the order in which sequential cells are tried. To this end, we propose sequential cells to

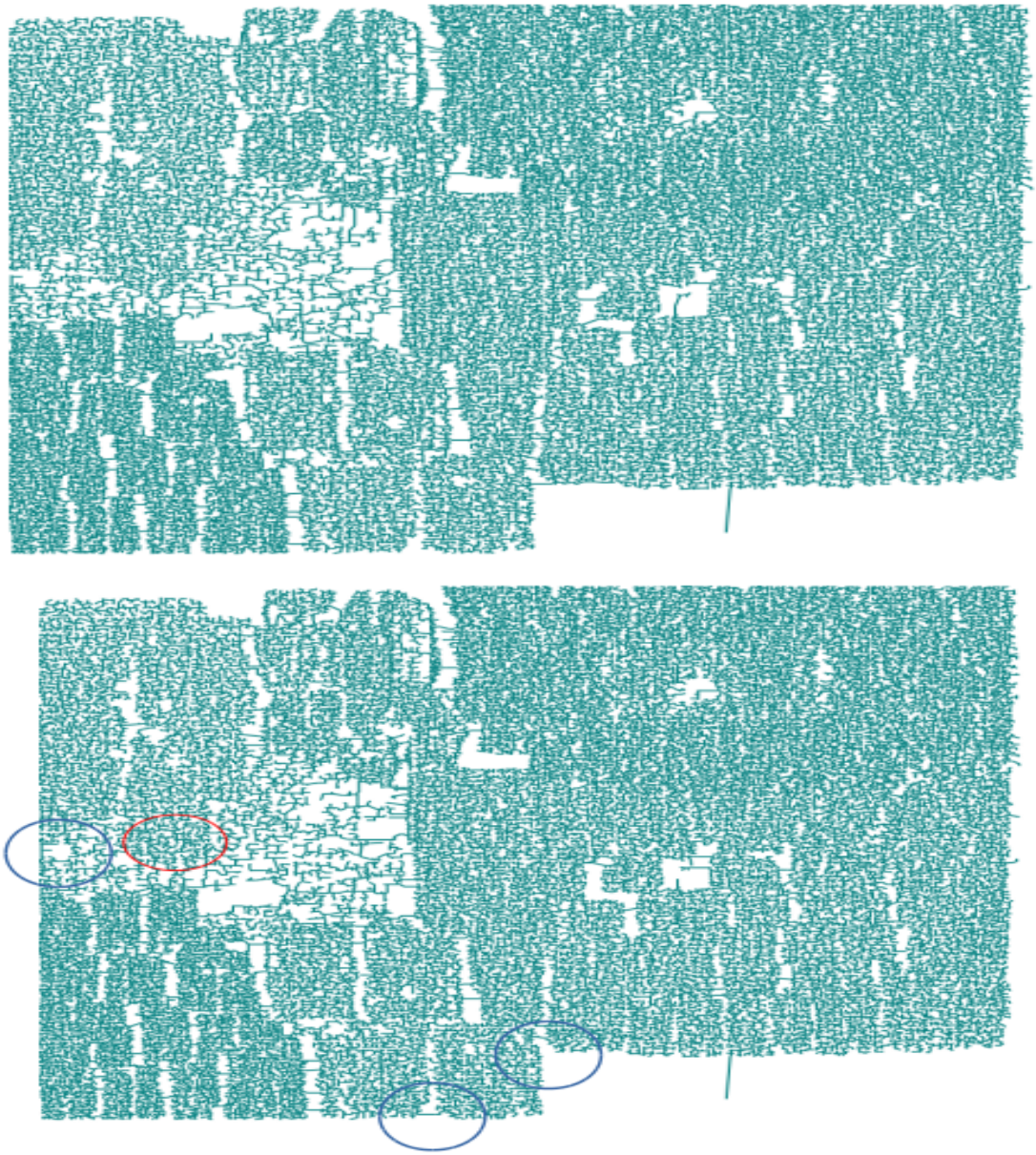

Figure 5.3: FLUTE generated trees during two successive sequential cell moves for benchmark leon3mp. The circles in the bottom tree shows, among other places, the places where the tree got changed from the tree on the top. The red circle indicates where the sequential cell moved.

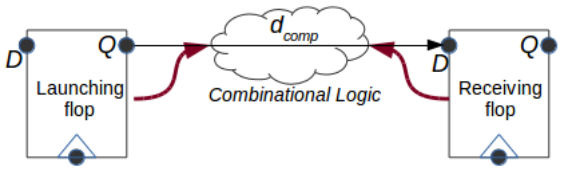

Figure 5.4: Example of timing critical path starting and ending at two different (D)flip-flops. The critical path from Q to D can be eliminated either by moving launching flop or receiving flop.

be tried in the following manner: Recall that sequential cells are the starting and ending points of timing paths. This means that a critical path starts at the output pin of the sequential cell (launching sequential cell) and ends at input pin of another sequential cell (receiving sequential cell) or primary output as illustrated in Figure 5.4. It presents two possible opportunities to eliminate a specific critical path - either by moving launching sequential cell or receiving sequential cell.

As such, to implement the aforementioned idea, we make the following changes to the *sequential cell pass* phase of our Algorithm 1: before we start moving the sequential cells, we first collect all the critical launching sequential cells in a list. Second, for each launching sequential cell, we find its critical receiving sequential cells using *breath first search*. Then, sequential cells are moved to their chosen positions in launching sequential cell followed by its receiving sequential cells order.

The results for the modified algorithm is presented in Table 5.3 with original Algorithm 1. In Table 5.3, the algorithm that provided the best result per benchmark is highlighted in bold face font. Compared to the results of our original Algorithm 1, the proposed modification to Algorithm 1 has resulted in improvement in both quality score and runtime, on average. To be specific, the quality score of all but benchmark *leon3mp* has improved under short displacement constraint, whereas, under long displacement constraint, the

quality score has improved for all benchmarks except *leon3mp* and *vga_lcd*. The runtime has also improved for the modified Algorithm 1, on average. Even though the runtime of smaller designs has slightly increased, it has decreased for larger designs such as *leon2*, *leon3mp* and *netcard*.

Table 5.3: Comparision of results for ICCAD-2014 benchmarks under shor and long displacement constraints with original and modified timing-driven placement algorithm.

	Under short displacement constraint				**Under long displacement constraint**			
	Original Algorithm1		**Modified Algorithm1**		**Original Algorithm1**		**Modified Algorithm1**	
Benchmark	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)	**Quality score**	**runtime** (*sec.*)
b19	1634.01	**210.53**	**1642.26**	349.30	1653.35	**303.54**	**1659.40**	449.15
vga_lcd	1467.13	105.38	**1522.14**	**84.50**	**1412.66**	235.22	1390.91	**170.48**
leon2	1557.35	**522.96**	**1571.88**	839.24	1561.15	1343.74	**1571.88**	**835.47**
leon3mp	**1557.61**	1273.99	1550.18	**309.05**	**1576.33**	2497.38	1550.18	**307.79**
netcard	1522.81	282.63	**1567.43**	**298.90**	1551.90	379.17	**1571.18**	**248.47**
edit_dist	920.01	**746.83**	**987.17**	906.32	1442.98	**611.22**	**1465.98**	973.43
matrix_mult	722.08	**306.91**	**871.01**	456.94	1481.38	**173.63**	**1531.08**	328.14
AVG.	1340.14	492.75	**1387.43**	**398.18**	1525.68	782.64	**1534.37**	**473.27**

In conclusion, compared to the average quality score of the original Algorithm1, the modified Algorithm 1 shows 4% and 1% improvement under short and long displacement constraint limit, respectively. In terms of average runtime, the runtime reduction by the modified Algorithm 1 amounts to 19% and 40% under short and long displacement constraint, respectively.

Chapter 6

Conclusions and Future Work

Timing-driven placement is becoming crucial step of the VLSI CAD flow to close timing, as the circuit performance of the modern deep-submicron technology is largely dominated by interconnect delay. Incremental timing-driven placement is tasked with finding optimized locations for standard cells on the chip under maximum displacement constraint so that paths with negative slacks can be eliminated with minimal distribution to original placement.

In this work, we proposed a simple yet effective incremental timing-driven placement algorithm based on greedy path-based technique. Unlike many traditional timing-driven placement algorithms which adopts cell movement within local fixed-size window region, we provided a methodology that supports displacement-aware global cell movement. We have also divided the cell movement into two separate stages, namely sequential cell pass and combinational cell pass, to provide faster timing convergence. Furthermore, we relied on the actual timing profile from STA engine to guide our placer without much overhead on the runtime.

Experimental results showed that our technique provides significant improvement in reducing the timing violations without degrading the placement quality of the original

solution on the ICCAD-2014 contest benchmarks. On average, we outperform the 1st-place team who won the contest in terms of quality score and runtime.

The following are some future research directions that we can employ to extend our timing-driven placement algorithm.

- Incremental STA can be an attractive feature to have within timing-driven placement algorithm to improve runtime. This means, figuring out the portion of circuit that would be affected by a cell move, and then performing STA on that specific part of the circuit, instead of the whole circuit. We couldn't able to include this feature into our timing-driven placement, because, for the considered benchmarks, entire circuit may be affected due to single clock source driving all sequential cells. The LCB introduction for the ICCAD-2015 contest benchmarks enables us to implement incremental STA.

- To minimize the impact on variability, we can extend our timing-driven placement technique to focus on maximizing timing on paths with slack values between 0 and 1, at the expense of runtime.

- During timing optimization, it essential not to introduce any routing congestion in the design. To this end, we can include a global router to guide timing-driven placement with routing congestion information.

- We can also incorporate fast buffer insertion techniques to solve early timing violations during the timing-driven placement.

Bibliography

[1] M.-C. Kim and J. Hu, "Incremental timing-driven placement." `http://cad_contest.ee.ncu.edu.tw/CAD-Contest-at-ICCAD2014/problem_b/default.html`.

[2] L. Wang, Y. Chang, and K. Chang, *Electronic Design Automation: Synthesis, Verification and Test*. Elsevier Inc., 2009.

[3] M.-C. Kim, J. Huj, and N. Viswanathan, "Iccad-2014 cad contest in incremental timing-driven placement and benchmark suite: Special session paper: Cad contest," in *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pp. 361–366, Nov 2014.

[4] A. B.Kahng, J. Lienig, I. L.Markov, and J. Hu, *VLSI Physical Design: From graph partitioning to timing closure*. Springer, 2011.

[5] M.-C. Kim and J. Hu, "Incremental timing-driven placement." `http://cad-contest.el.cycu.edu.tw/problem_C/default.html`.

[6] C. Alpert, Z. Li, G.-J. Nam, C. Sze, N. Viswanathan, and S. Ward, "Placement: Hot or not?," in *Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on*, pp. 283–290, Nov 2012.

[7] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu, "Itop: Integrating timing optimization within placement," in *Proceedings of the 19th*

International Symposium on Physical Design, ISPD '10, (New York, NY, USA), pp. 83–90, ACM, 2010.

[8] N. Viswanathan, *Placement techniques for the physical synthesis of nanometer-scale integrated circuits*. PhD thesis, Iowa State University, 2009.

[9] C. Alpert, J.-H. Huang, and A. Kahng, "Multilevel circuit partitioning," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, pp. 655–667, Aug 1998.

[10] C. Sechen and A. Sangiovanni-Vincentelli, "Timberwolf3.2: A new standard cell placement and global routing package," in *Design Automation, 1986. 23rd Conference on*, pp. 432–439, June 1986.

[11] N. Viswanathan and C. Chu, "Fastplace: efficient analytical placement using cell shifting, iterative local refinement,and a hybrid net model," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, pp. 722–733, May 2005.

[12] M.-C. Kim, D.-J. Lee, and I. Markov, "Simpl: An effective placement algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, pp. 50–60, Jan 2012.

[13] D. Hill, "Method and system for high speed detailed placement of cells within an integrated circuit design," Apr. 9 2002. US Patent 6,370,673.

[14] H. Ren, D. Pan, C. Alpert, P. Villarrubia, and G.-J. Nam, "Diffusion-based placement migration with application on legalization," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, pp. 2158–2172, Dec 2007.

[15] K. Doll, F. Johannes, and K. Antreich, "Iterative placement improvement by network flow methods," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, pp. 1189–1200, Oct 1994.

[16] M. Pan, N. Viswanathan, and C. Chu, "An efficient and effective detailed placement algorithm," in *Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on*, pp. 48–55, Nov 2005.

[17] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, pp. 1228–1240, July 2008.

[18] N. Viswanathan, M. Pan, and C. Chu, "Fastplace 3.0: A fast multilevel quadratic placement algorithm with placement congestion control," in *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*, pp. 135–140, Jan 2007.

[19] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji, "Maple: Multilevel adaptive placement for mixed-size designs," in *Proceedings of the 2012 ACM International Symposium on International Symposium on Physical Design*, ISPD '12, (New York, NY, USA), pp. 193–200, ACM, 2012.

[20] `http://archive.sigda.org/ispd2005/contest.htm`.

[21] `http://archive.sigda.org/ispd2006/contest.htm`.

[22] "Ispd 2011 routability-driven placement contest and benchmark suite." `http://www.ispd.cc/contests/11/ispd2011_contest.html`.

[23] "Ispd 2014 detailed routing-driven placement contest." http://www.ispd.cc/contests/14/ispd2014_contest.html.

[24] "Ispd 2015 blockage-aware detailed routing-driven placement contest." http://www.ispd.cc/contests/15/ispd2015_contest.html.

[25] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55–63, 1948.

[26] H. Ren, D. Pan, and D. Kung, "Sensitivity guided net weighting for placement-driven synthesis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, pp. 711–721, May 2005.

[27] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pp. 172–176, Nov 2002.

[28] C. Guth, V. Livramento, R. Netto, R. Fonseca, J. L. Güntzel, and L. Santos, "Timing-driven placement based on dynamic net-weighting for efficient slack histogram compression," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, (New York, NY, USA), pp. 141–148, ACM, 2015.

[29] W. Choi and K. Bazargan, "Incremental placement for timing optimization," in *Computer Aided Design, 2003. ICCAD-2003. International Conference on*, pp. 463–466, Nov 2003.

[30] A. Chowdhary, K. Rajagopal, S. Venkatesan, T. Cao, V. Tiourin, Y. Parasuram, and B. Halpin, "How accurately can we model timing in a placement engine?," in *Design Automation Conference, 2005. Proceedings. 42nd*, pp. 801–806, June 2005.

[31] W. Swartz and C. Sechen, "Timing driven placement for large standard cell circuits."

[32] A. Marquardt, V. Betz, and J. Rose, "Timing-driven placement for fpgas," in *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays*, FPGA '00, (New York, NY, USA), pp. 203–213, ACM, 2000.

[33] T. Luo, D. Newmark, and D. Pan, "A new lp based incremental timing driven placement for high performance designs," in *Design Automation Conference, 2006 43rd ACM/IEEE*, pp. 1115–1120, 2006.

[34] W.-K. Chow, J. Kuang, X. He, W. Cai, and E. F. Young, "Cell density-driven detailed placement with displacement constraint," in *Proceedings of the 2014 on International Symposium on Physical Design*, ISPD '14, (New York, NY, USA), pp. 3–10, ACM, 2014.

[35] C. Chu, "Flute: fast lookup table based wirelength estimation technique," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, pp. 696–701, Nov 2004.

[36] S. Goto, "An efficient algorithm for the two-dimensional placement problem in electrical circuit layout," *Circuits and Systems, IEEE Transactions on*, vol. 28, pp. 12–18, Jan 1981.

[37] M.-C. Kim, N. Viswanathan, Z. Li, and C. Alpert, "Iccad-2013 cad contest in placement finishing and benchmark suite," in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pp. 268–270, Nov 2013.

[38] M.-C. Kim and J. Hu, "Incremental timing driven placement." `http://cad_contest.ee.ncu.edu.tw/CAD-Contest-at-ICCAD2014/problem_b/results/ICCAD2014_Contest_P2_Results.pdf`.