UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

TIAGO JOSÉ REIMANN

Cell Selection to Minimize Power in High-Performance Industrial Microprocessor Designs

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Microeletronics

Advisor: Prof. Dr. Ricardo A. da Luz Reis

Porto Alegre
January 2017

CIP — CATALOGING-IN-PUBLICATION

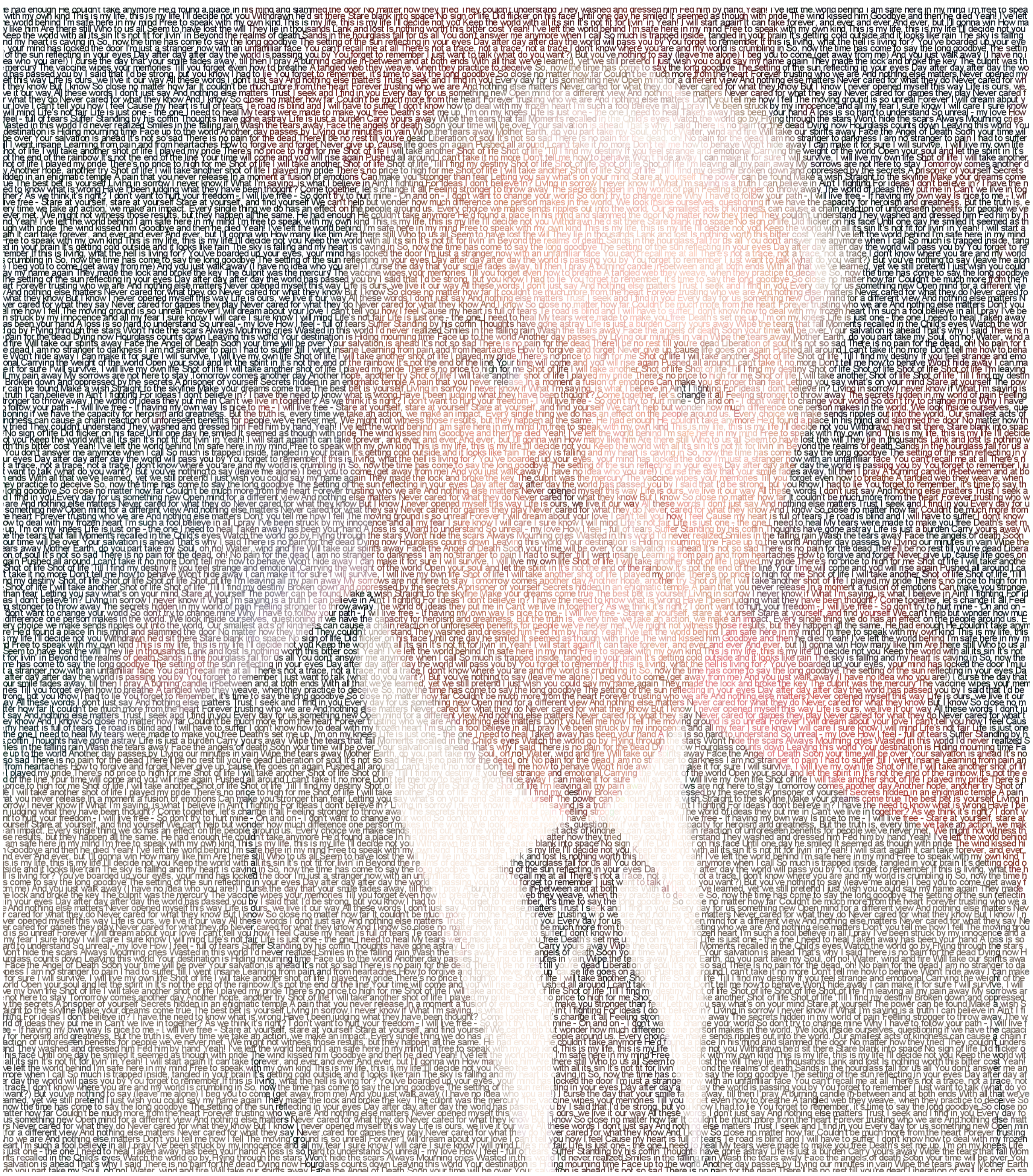
Reimann, Tiago José

Cell Selection to Minimize Power in High-Performance Industrial Microprocessor Designs / Tiago José Reimann. – Porto Alegre: PGMICRO da UFRGS, 2017.

130 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2017. Advisor: Ricardo A. da Luz Reis.

1. Gate Sizing. 2. Threshold Voltage Assignment. 3. Lagrangian Relaxation. 4. EDA. 5. Microelectronics. I. Reis, Ricardo A. da Luz. II. Título.



“Seja você quem for, seja qual for a posição social que você tenha na vida, a mais alta ou a mais baixa, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá."

— Ayrton Senna

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

CAD	Computer Aided Design
DAG	Directed Acyclic Graph
DRC	Design Rules Checking
EDA	Electronic Design Automation
HDL	Hardware Description Language
ISPD	International Symposium on Physical Design
IC	Integrated Circuit
KKT	Karush–Kuhn–Tucker
LR	Lagrangian relaxation
NLDM	Non-Linear Delay Model
RTL	Register Transfer Level
SPICE	Simulation Program with Integrated Circuit Emphasis
SA	Simulated Annealing
STA	Static Timing Analysis
TNS	Total Negative Slack
TTNS	True Total Negative Slack
TR	Timing Recovery
V_t	Threshold Voltage
VLSI	Very Large Scale Integration
WMIS	Weighted Maximum Independent Set
WNS	Worst Negative Slack

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This work addresses the gate sizing and V_t assignment problem for power, area and timing optimization in modern integrated circuits (IC). The proposed flow is applied to the Benchmark Suites of the International Symposium on Physical Design (ISPD) 2012 and 2013 Contests. It is also adapted and evaluated in the post placement and post global routing stage of an industrial IC design flow using a sign-off static timing analysis engine. The proposed techniques are able to generate the best solutions for all benchmarks in the ISPD 2013 Contest (in which we were the winning team), with on average 8% lower leakage with respect to all other contestants. Also, after some refinements in the algorithms, we reduce leakage by another 10% on average over the contest results.

The focus of this work is to develop and apply a state-of-the-art cell selection algorithm to further improve already optimized high-performance industrial designs after the placement and routing stages of the industrial physical design flow. We present the basic concepts involved in the gate sizing problem and how earlier literature addresses it. Several problems found when applying global optimization techniques in real-life industrial designs, which are not fully covered in publications found in literature, are presented and discussed. Considering the industrial application, the proposed techniques reduce leakage power by up to 18.2%, with average reduction of 10.4% without any degradation in timing quality.

Keywords: Gate Sizing. Threshold Voltage Assignment. Lagrangian Relaxation. EDA. Microelectronics.

Seleção de Portas Lógicas para Minimização de Potência em Projetos de Microprocessadores de Alto Desempenho

RESUMO

Este trabalho aborda o problema de dimensionamento portas lógicas e assinalamento de V_t para otimização de potência, área e temporização em circuitos integrados modernos. O fluxo proposto é aplicado aos conjuntos de circuitos de teste dos Concursos do *International Symposium on Physical Design* (ISPD) de 2012 e 2013. Este fluxo também é adapatado e avaliado nos estágios pós posicionamento e roteamento global em projetos industriais de circuitos integrados, que utilizam uma ferramenta precisa de análise estática de temporização.

As técnicas propostas geram as melhores soluções para todos os circuitos de teste do Concurso do ISPD 2013 (no qual foi a ferramenta vencedora), com em média 8% menos consumo de potência estática quando comparada com os outros concorrentes. Além disso, após algumas modificações nos algoritmos, nós reduzimos o consumo em mais 10% em média a pontência estáticas com relação aos resultados do concurso.

O foco deste trabalho é desenvolver e aplicar um algoritmo estado-da-arte de seleção portas lógicas para melhorar ainda mais projetos industriais de alto desempenho já otimizados após as fases de posicionamento e roteamento do fluxo de projeto físico industrial. Vamos apresentar e discutir vários problemas encontrados quando da aplicação de técnicas de otimização global em projetos industriais reais que não são totalmente cobertos em publicações encontradas na literatura. Os métodos propostos geram as melhores soluções para todos os circuitos de referência no Concurso do ISPD 2013, no qual foi a solução vencedora. Considerando a aplicação industrial, as técnicas propostas reduzem a potência estática em até 18,2 %, com redução média de 10,4 %, sem qualquer degradação na qualidade de temporização do circuito.

Palavras-chave: Dimensionamento de Portas Lógicas, Assinalamento de Tensão de Limiar, Relaxação Lagrangiana, EDA, Microeletrônica.

1 INTRODUCTION

The growing importance of low power designs for portable devices to extend battery life and to reduce cooling costs, the challenges imposed by the power density of new technology nodes and cost of energy for server centers is making both industry and academia focus on algorithms to provide power optimized IC designs.

One way to achieve this goal is through gate sizing and V_t assignment. The process of gate sizing defines the sizes of the transistors inside each logic gate that composes the design. In many technology processes transistors sizes can be continuously assigned, i. e., any size is accepted, or previously determined by a standard cell library. In library-based designs, only discrete gate sizes are available. Gate sizing and V_t assignment optimization methods can be applied in several stages of the design synthesis flow, from the logic synthesis to post-route optimization, with increasing accuracy in the timing analysis models as a result of more accurate physical information.

The discrete gate sizing problem has been the subject of many works in the literature. The techniques proposed are based on different methods like Lagrangian relaxation (LR) (CHEN; CHU; WONG, 1999; TENNAKOON; SECHEN, 2005; OZDAL; BURNS; HU, 2011; FLACH et al., 2013; FLACH et al., 2014), Dynamic Programming (DP)(OZDAL; BURNS; HU, 2011), Linear Programming (CHINNERY; KEUTZER, 2005), greedy iterative sensitivity-based methods (LIN; MAREK-SADOWSKA; KUH, 1990; QIAN; ACAR, 2007; HU et al., 2012; KAHNG et al., 2013), Stochastic optimization (HU et al., 2012; KAHNG et al., 2013), network-flow (REN; DUTT, 2008; REN; DUTT, 2011), and Sequential Quadratic Programming (SQP) (MENEZES; BALDICK; PILEGGI, 1995; MENEZES; BALDICK; PILEGGI, 1997).

Throughout this work we refer to simultaneous gate sizing and V_t assignment as *cell selection*. Despite the many contributions in the field, many of the cell selection problems are not fully handled by existing methods. Also, most of the previous works do not present solutions that are practical for industrial designs. Several simplifications adopted in these works, like simpler delays models and incomplete timing propagation, are not applicable any more or have inaccuracies that prevent the convergence to a near-optimal solution.

Power consumption in integrated circuits has two components: static (leakage) and dynamic. The former is related to the leakage current of the components while the latter is associated with signal transition and short-circuit currents. Both components can be addressed through gate sizing and V_t assignment.

Most recent works focus only on the leakage power optimization in a timing-constrained optimization. This focus is a consequence of the problem definition on the two ISPD Discrete Gate Sizing Contests (2012 and 2013). However, it is important to also ensure that the other design objectives like timing, dynamic power, and area are (at least) not degraded by the power optimization process. Moreover, those parameters may also be an optimization objective. These objectives are conflicting and optimizing only one may be detrimental to the others. For instance, if a cell with a larger area (size) and higher V_t (less leakage power) replaces a smaller but lower V_t cell option in a non-critical path, the leakage power will decrease, but dynamic power may increase because of the increase in the capacitance of the new cell option. Also, the area in this case will increase. Since leakage power grows exponentially with respect to reduction in V_t (as detailed later), the area increase in order to reduce power can be excessive.

A good formulation for gate sizing and V_t assignment must enable an appropriate balance of those conflicting objectives to further improve the solution provided by the earlier stages of the physical synthesis flow.

The objective of this work is to develop and apply a state-of-the-art cell selection algorithm to further improve already optimized high-performance industrial designs after the placement and global routing stages of the industrial physical design flow, where wiring parasitics are available and timing models are more accurate.

This work is organized as follows. First, given the importance of timing analysis for the cell selection algorithms, we introduce some basic concepts of Static Timing Analysis (STA) and the relevant methods for timing analysis in Chapter 2. Chapter 3 presents the cell selection problem and gives several options for how it can be inserted in a general physical design flow with some examples of the different methodologies and applications of algorithms. In Chapter 4, we revisit the related works and the state-of-the-art prior to this work. Next, Chapter 5 presents the proposed methodologies to address the cell selection problem in the ISPD contests. Chapter 6 details the industrial applications proposed, including the differences in formulations and the respective experimental validation. The conclusions are presented in Chapter 7.

2 STATIC TIMING ANALYSIS

The quality of results given by cell selection algorithms in post-route optimization strongly relies on the accuracy of the timing models used during the optimization. The presence of wiring parasitics and the non-linearity of cell delay models undermines the performance of analytical optimization methods that required closed-form models, which may present high inaccuracies when compared to the sign-off timing analysis. More accurate timing enables better optimization results, as presented in recent publications (KAHNG et al., 2013; FLACH et al., 2014).

In that context, it is important to present the relevant concepts involved in static timing analysis (STA) that are taken into account for the development of this work.

The concept of static timing analysis (STA) is presented in (HITCHCOCK R.B., 1982) and other publications, including (HITCHCOCK; SMITH; CHENG, 1982; ABATO et al., 1996; KIRKPATRICK; CLARK, 1966; GUNTZEL, 2000). Details can be found in (SAPATNEKAR, 2004; BHASKER; CHADHA, 2009)

The idea of STA is to perform a block-based analysis of the circuit to obtain the fastest and slowest path delays between all the timing start and endpoints. Also, the slacks at each cell pin and input–output (I/O) pins are calculated. This is done by propagating arrival times in topological order and required times in reverse topological order.

Timing startpoints can be either a primary input (PI) or a sequential element (latch, flip-flop) output. Timing endpoints are the primary outputs (PO) and the data input of sequential elements.

The blocks in the circuit are usually represented by the gates. The timing information for each gate is found in the standard cell library. Gates are modeled with different delays and slew rates for each transition (rise and fall), depending on input slew and output load capacitance.

In this work we follow the definitions found in (LEE; GUPTA, 2012) with some small modifications. The definitions are:

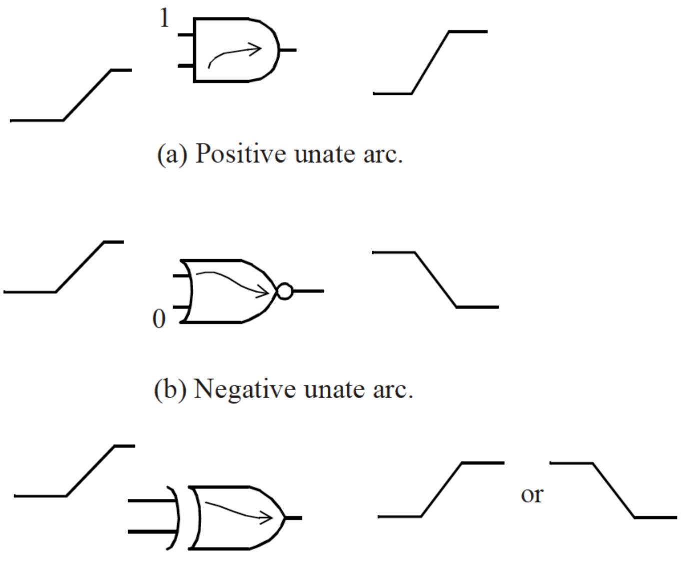
- Primary inputs (PI): input ports in the design that are driven by external sources.
- Primary outputs (PO): output ports in the design.
- Cell rise time or rise delay: time from when the input crosses 50% voltage to when the rising output crosses 50% voltage.
- Cell fall times or fall delay: time from when the input crosses 50% voltage to when the falling output crosses 50% voltage.

- Cell rise transition or slew: the time elapsed from when the output signal crosses the 20% voltage to when it crosses 80%.
- Cell fall transition or slew: the time elapsed from when the output signal crosses the 80% voltage to when it crosses 20%.
- Arrival time (t_a): the time that the signal crosses the 50% voltage threshold at a given point in the circuit. The time associated with a rising and falling signal is called the rise arrival time and the fall arrival time, respectively.
- Required arrival time (t_r): the time a signal needs to cross the 50% voltage threshold at a given point in the circuit to be timing feasible. The required arrival time associated with a rising and falling signal are called the rise and fall required arrival time, respectively.
- Slack (s): the difference of the required arrival time and the arrival time, represents the amount of timing slack available at that point in the circuit.
- Timing arc: a concept used to relate the delay between two adjacent nodes in the circuit.
- Positive unate: when a rising input to a gate causes a rising output, or a falling input causes a falling output.
- Negative unate: when a rising input to a gate causes a falling output, or a falling input causes a rising output.
- Non-unate: when there is no direct relation between the type of transition (rise/fall) in the input and the type of transition in the output, e.g. XOR gate where a rising or falling output transition can be triggered by either rising or falling input transition.
- β ratio: the ratio between the width of the PMOS transistors and the width of the NMOS transistors in a CMOS gate.

The unateness concept is shown in Figure 2.1. Sign-off timers handle rising and falling delays and transitions separately, and the unateness of a cell will determine the possible combinations for signal propagation through its timing arcs. The unateness also determines the timing information available in the cell library. Positive and negative unate arcs only require two timing tables (rise-to-rise, fall-to-fall or rise-to-fall, fall-to-rise) for delay and transition times for each input-output pin pair. Non-unate cells require four timing tables (rise-to-rise, fall-to-fall, rise-to-fall and fall-to-rise) for delay and transition times for each input-output pin pair.

STA is used in the design flow at several stages, with different levels of accuracy,

Figure 2.1: Examples of input/output timing arcs for gates of different unateness.



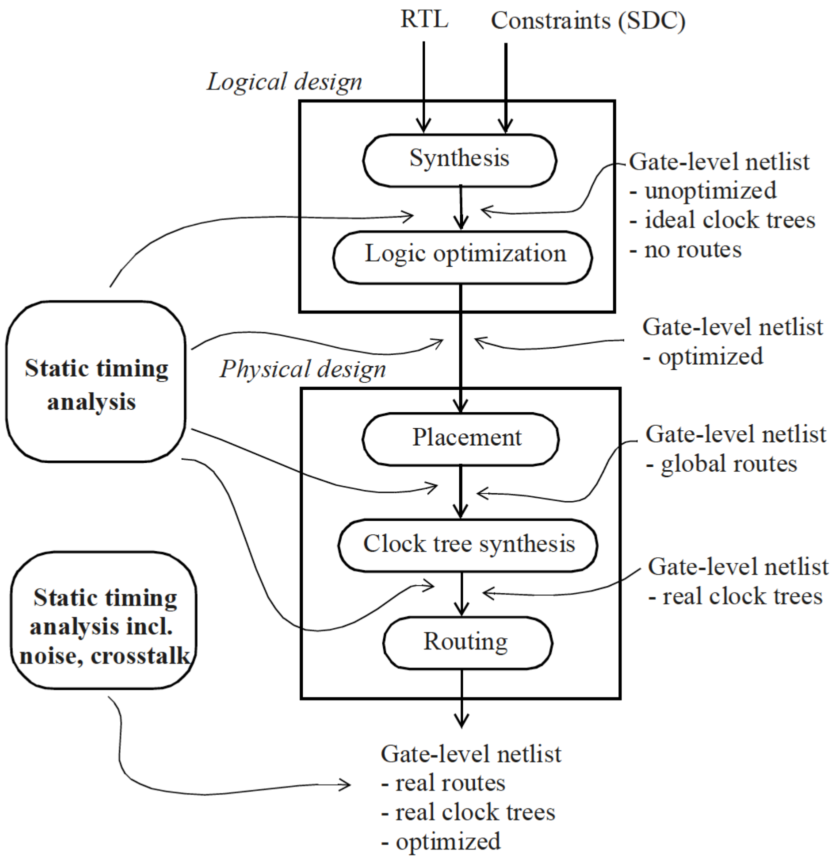
(a) Positive unate arc.

(b) Negative unate arc.

(c) Non-unate arc.

Source: Bhasker and Chadha (2009)

Figure 2.2: STA applications in the design flow.



Source: Bhasker and Chadha (2009)

as shown in Figure 2.2.

In this work we apply STA with library-based technologies. Although cell selection algorithms do not require a specific timing analysis model to work with, global cell selection methods in post global routing optimization perform better with sign-off timers, taking advantage of the actual positive slacks existent in the design and the accurate trade-offs between timing and power. Simplified timing models, which are linear, convex, or differentiable introduce inaccuracy that will reduce the ability of the algorithm to find the actual best combinations of sizes and threshold voltages for a single cell in the design due to reduced timing slack for the rest of the design caused by timing propagation.

Next we discuss the standard cell library model and the interconnect models commonly used in modern designs.

2.1 Standard Cells

Most functions in a chip are designed using basic building blocks which implement simple boolean logic functions such as AND, OR, NAND, NOR, AND-OR-INVERT, OR-AND-INVERT, XOR, MUX, inverters, and sequential elements (latch, flip-flop and its variants). There are also utility cells, such as filler cells, antenna cells, and buffer cells, which are used to help with the physical implementation of the design. These basic building blocks are referred to as standard cells. The functionality and timing of the standard cells is pre-designed and pre-characterized and made available to the circuit designer. Also, larger functional blocks, known as macros, may be in the design and have similar treatment as standard cells, with timing properties assigned to their input and output pins. Designers use Electronic Design Automation (EDA) tools to implement the required functionality of the design using the standard cells as the building blocks.

Usually, several options are available in the library for each logic function. Despite having the same logic function, each cell has different characteristics regarding: transistor widths and folding (that changes the electrical properties of the transistor), transistor channel length, threshold voltage (V_t), and β ratios (that affect the timing performance for different rising and falling output transitions). Those differences imply a change in timing, area, power, and parasitic characteristics. With those options available, the design can be optimized for one or more objectives.

The cell library uses table models to specify the delays and timing checks for each timing arc of the cell. The models used may vary between libraries and technologies depending on the level of accuracy needed. The table models used for delay, output slew, or other timing checks are referred to as NLDM (Non-Linear Delay Model). Newer libraries for nanometer technologies may use more accurate current source based timing models. CCS (Composite Current Source) and ECSM (Effective Current Source Model) are two examples of these models . Current source models are not used throughout this work and their characteristics and properties are not relevant to the application of the methods developed here.

The NLDM table is usually a discretization of simulation data or real prototype measurements. It contains the delay through the cell for several combinations of input transition time at the cell input pin and total output capacitance at the cell output. The models for delay are typically presented in a two-dimensional form, with the two independent variables being the input transition time and the output load capacitance, and the entries in the table denoting the delay. The same is applied to the output transition time (slew rate) calculation.

The following equation shows the bilinear interpolation to the nearest four data points in the lookup table used for NLDM calculation.

$$z = a + b \times x + c \times y + d \times x \times y \tag{2.1}$$

where z is the calculated value for delay or slew, x is the input slew, and y is the output load capacitance.

The timing tables contained in a cell library are:

- Rise delay
- Fall delay
- Rise slew
- Fall slew
- Clock hold time (data input of sequential elements only)
- Clock setup time (data input of sequential elements only)

An example of a delay lookup table is shown in Table 2.1. It represents the delay for the smallest and slowest inverter in the ISPD Contest 2013 library. The calculation is done by interpolating/extrapolating the two-dimensional data in the lookup table. Knowing the input transition time $slew_{input}$ and the output load C_{load} the value from the lookup table is obtained as follows. The two nearest table indices in each dimension are chosen for the table interpolation. Assuming that we are calculating the delay D_A for a given input transition and output load which the closest indices are x_1, x_2 and y_1, y_2, respectively:

Table 2.1: NLDM lookup table from the ISPD Contest 2013. Transition times in ps and load capacitances in fF.

	Input transition time							
Output load	5.00	30.00	50.00	80.00	140.00	200.00	300.00	500.00
0.00	11.72	18.22	22.67	27.61	34.82	40.44	48.21	61.24
1.00	16.93	23.43	28.60	34.95	44.15	51.18	60.65	75.83
2.00	22.13	28.63	33.83	41.24	52.23	60.57	71.67	89.00
4.00	32.55	39.05	44.25	52.05	66.05	76.76	90.85	112.34
8.00	53.38	59.88	65.08	72.88	88.48	103.09	122.40	151.42
16.00	95.05	101.55	106.75	114.55	130.15	145.75	171.59	213.31
32.00	178.38	184.88	190.08	197.88	213.48	229.08	255.08	307.08

$$\begin{aligned} D_A = \ & (1 - w_x) \times (1 - w_y) \times D_{11} \\ & + (1 - w_x) \times w_y \times D_{12} \\ & + w_x \times (1 - w_y) \times D_{21} \\ & + w_x \times w_y \times D_{22} \end{aligned} \tag{2.2}$$

where D_{xy} is the table entry for indices x, y and

$$\begin{aligned} w_x &= \frac{x_0 - x_1}{x_2 - x_1} \\ w_y &= \frac{y_0 - y_1}{y_2 - y_1} \end{aligned} \tag{2.3}$$

For the example in Table 2.1, assuming $slew_{input} = 220ps$ and $C_{load} = 4.5fF$:

$$\begin{aligned} w_x &= \frac{220 - 200}{300 - 200} = 0.2 \\ w_y &= \frac{4.5 - 4}{8 - 4} = 0.125 \end{aligned} \tag{2.4}$$

$$\begin{aligned} D_A = \; & (1-0.2) \times (1-0.125) \times 76.76 \\ & + (1-0.2) \times 0.125 \times 103.09 \\ & + 0.2 \times (1-0.125) \times 90.85 \\ & + 0.2 \times 0.125 \times 122.40 \end{aligned} \tag{2.5}$$

$$D_A = \; 83.00ps$$

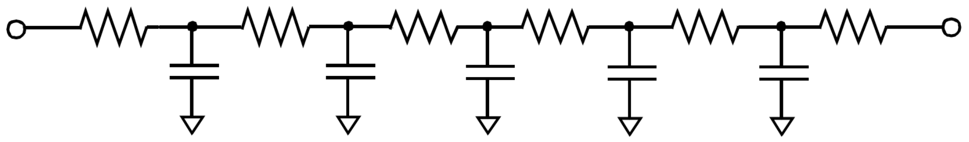
As aforementioned, the same calculation is done to obtain the output transition time from its lookup table. When the input transition time and/or the output load are out of the indices' ranges, an extrapolation is made with the last two indices in both dimensions. The calculation process is the same. However, since the cell is not characterized for such parameters, the resultant delay and output transition time may be quite inaccurate. Maximum load and maximum transition time are two electrical constraints specified in the library to guarantee the accuracy of times obtained from the lookup tables.

Each timing arc inside a cell has its respective tables. Timing arcs relate the input and output pins for both rising and falling transitions and both delay and output slew. In sequential cells, timing arcs relate the clock pin with the synchronous input and output pins. The data input is a timing endpoint (sometimes also called a time barrier). Sequential cells also usually have scan-related pins for testing – scan enable and scan input. Scan pins and scan timing are omitted in this work for simplicity. They are either not present (ISPD Contest) or are not setup timing critical (industrial high-performance designs) in this work. Sequential cells may also have asynchronous inputs (like the clear data pin).

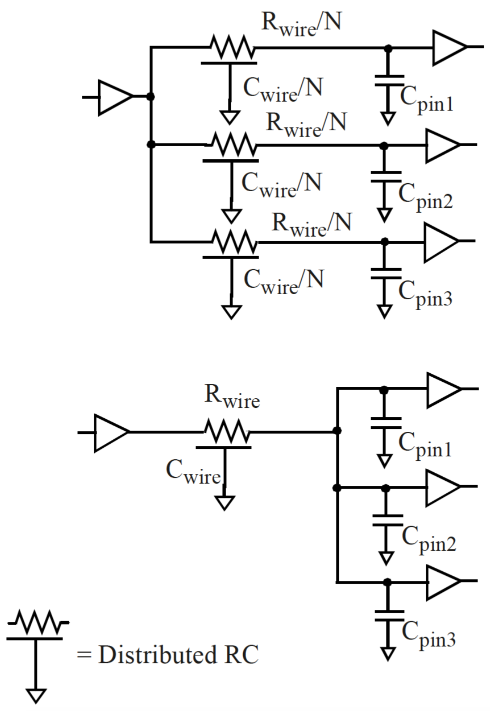
2.2 Interconnect Model

Interconnect models have a significant importance in modern designs where the interconnect delay has become a considerable percentage of total delay. Also, the slew propagation through wires has significant impact in timing calculations. Accurate interconnection delay and slew propagation metrics must be used in the late stages of physical design, where parasitics information is available. Interconnect parasitics are extracted from the circuit layout to form RC (resistance and capacitance) networks. They are commonly represented as RC trees or RC meshes – Figure 2.3.

Figure 2.3: (a) Distributed RC model. (b) Examples of parasitics networks.



(a)



(b)

Source: Bhasker and Chadha (2009)

Lumped capacitance models are inaccurate[1] for timing analysis in post global routing stages.

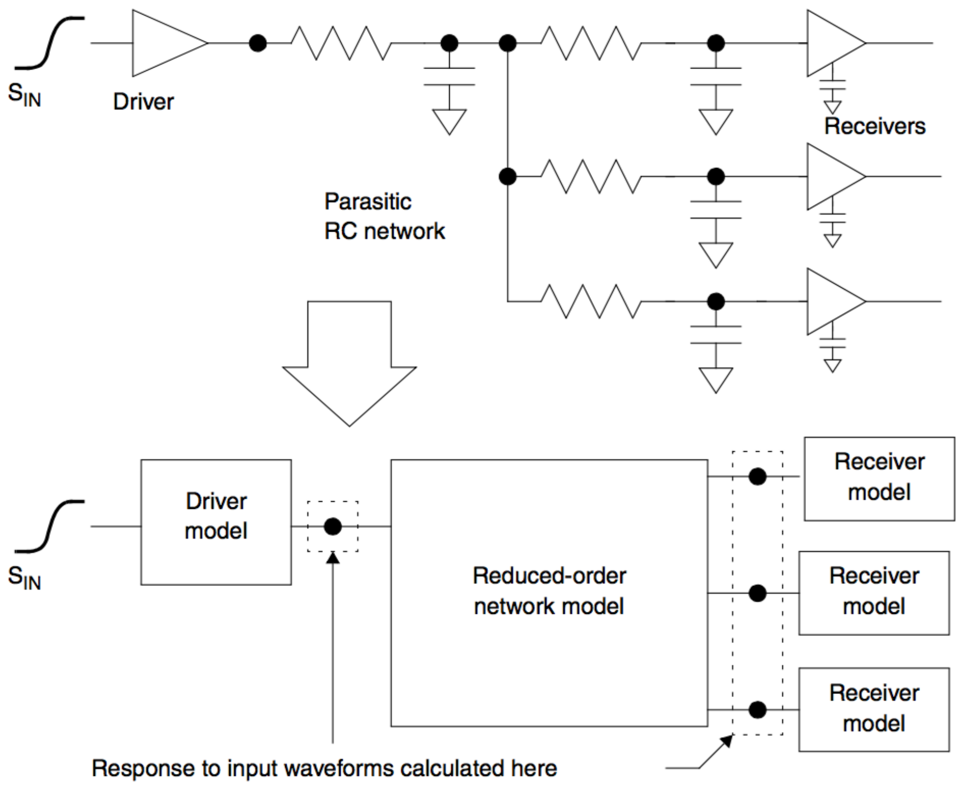
Electrical simulation (like SPICE – Simulation Program with Integrated Circuit Emphasis (NAGEL; PEDERSON, 1973)) can be used to accurately calculate timing by solving the differential equations related to the currents, charges and voltages. However, the runtime for this approach is prohibitive in large or even mid-size designs. The runtime of timing analysis is a key aspect for gate sizing and threshold voltage assignment, where fast delay estimates are needed for optimization.

The Elmore delay (ELMORE, 1948) is a simple method to estimate delays. This method calculates the first moment of the impulse response applied to the RC network. The Elmore delay is an upper bound on the actual delay (GUPTA et al., 1995).

However, modern timers employ more accurate methods that utilize higher order

[1]The lumped capacitance model for interconnects is only accurate when the driver resistance overwhelms the wire resistance (SAPATNEKAR, 2004)

Figure 2.4: Example of reduced order model.



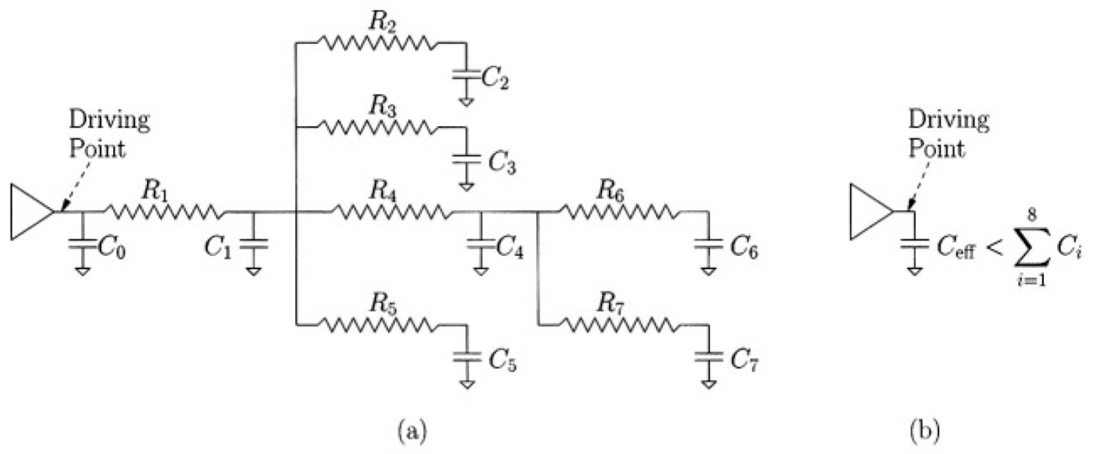
Source: Synopsys (2004)

moments to improve the accuracy. The Asymptotic Waveform Evaluation Method (PILLAGE; ROHRER, 1990; PILLAGE; HUANG; ROHRER, 1989; ODABASIOGLU; CELIK; PILEGGI, 1997; RATZLAFF; PILLAGE, 1994) and Arnoldi-based methods (SILVEIRA; KAMON; WHITE, 1996; ELFADEL; LING, 1997; SILVEIRA et al., 1996) are examples of higher order models commonly used to model the parasitics network – Figure 2.4. More details can be found in (SAPATNEKAR, 2004; CELIK; PILEGGI; ODABASIOGLU, 2002).

The propagation of the output transitions (slews) of cells is also crucial for timing calculations. Signal slews in the input pins play a major role in determining the output delay. It is not possible to ignore slew effects in post global route timing analysis. Especially in cell selection algorithms, ignoring the slew misses the potential in increasing gate size or decreasing the threshold voltage to improve the slew in its fanout gates, and the corresponding gate delays.

For simplicity in timing analysis in industrial STA tools (i.e. Synopsys PrimeTime®) the propagated output slew is usually the slower slew calculated at the output, regardless of the arrival time associated. However, most sign-off timers can also propagate the slew associated with the greatest arrival time at the output if specified (e.g.

Figure 2.5: The effective capacitance model for cell delay calculation in the presence of RC loads.



Source: Sapatnekar (2004)

path-based static timing analysis).

2.2.1 Effective Capacitance

Interconnect models may be simplified to use only a lumped capacitance to represent the wires. However, the lumped capacitance model is not accurate when the driver resistance and the wire resistance are comparable. The phenomenon of *resistive shielding* causes the delay at the driver timing arc to be equivalent to a delay where the output load is smaller than the total capacitance in the RC network. This smaller capacitance is called *effective capacitance*. Figure 2.5 shows an example of effective capacitance (C_{eff}) equivalence.

The effective capacitance models used in accurate STA tools calculate an equivalent capacitance that has the same timing arc delay as the cell with RC tree load. Using an effective capacitance model is useful because cells may continue to be characterized in terms of a load capacitance (Section 2.1).

Many methods for effective capacitance calculation can be found in the literature by O'Brien and Savarino (1989), Dartu et al. (1994), Dartu, Menezes and Pileggi (1996), Puri, Kung and Drumm (2002), Qian, Pullela and Pillage (2006). The driver model used for effective capacitance calculations is usually the same used for the parasitics network analysis (see Figure 2.4). Dartu, Menezes and Pileggi (1996) propose the use of a Thevenin model (NILSSON; RIEDEL, 2005) to represent the driver.

The effective capacitance is calculated based on the current necessary to charge the equivalent RC π-model derived from the complete RC network (O'BRIEN; SAVARINO, 1989). The gate is characterized by the determination of the values of a driver resistance (R_d), transistion delay (t_0) and transistion time (Δt), according to the library timing information (DARTU et al., 1994; DARTU; MENEZES; PILEGGI, 1996).

3 THE CELL SELECTION PROBLEM

In this chapter we define the cell selection problem, showing how it can be inserted in a general physical design flow and give some examples of the different methodologies and applications of algorithms related to the cell selection problem.

First we talk about the physical design flow and how cell selection algorithms can be applied in several stages of the flow.

3.1 Physical Design Flow

Cell selection algorithms can be applied in many parts of the design flow, from the logic synthesis to the post-route stages. Figure 3.1 shows an example of a general Electronic Design Automation (EDA) flow. For each stage in the flow, the actions expected can be summarized as follows:

The *Logic Synthesis* step starts with a Register Transfer Level (RTL)/Hardware Description Language (HDL) design description, a standard cell library information and timing constraints for the design. A gate level netlist mapped to the standard cell library is generated to meet the constraints and optimized different objectives.

The *Floorplanning* stage creates a floorplan with rows for standard cell placement, I/O pads and locations for macro blocks.

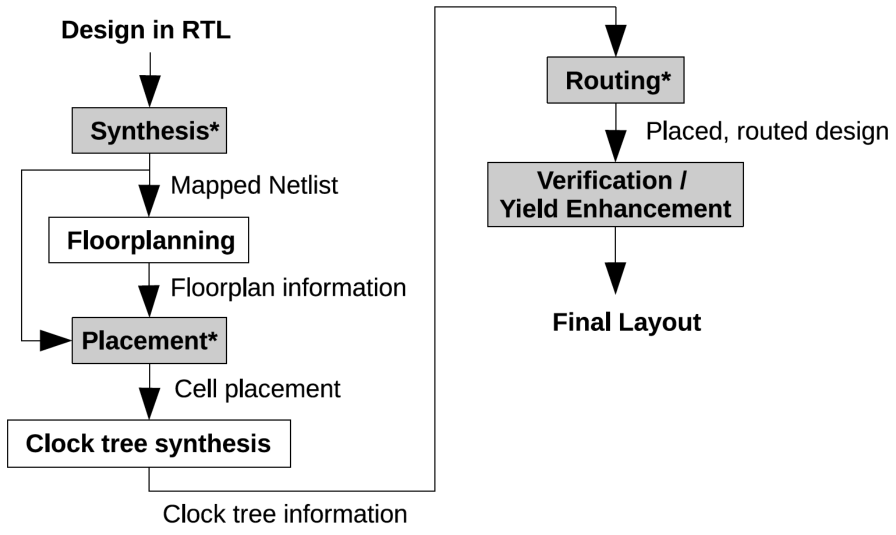
A *Placement* step defines the positions for the cells into the rows created by the floorplanning algorithm, flipping and rotating cells as necessary for the optimization objectives. The main objective is usually to minimize the wirelength in the resulting layout. Another common target is routing congestion mitigation and improve timing in critical paths (timing-driven placement).

The *Clock Tree Synthesis* stage will create a clock tree of buffers to distribute the clock signal to all sequential elements of the design. The main goals are to minimize the size of the clock tree (to minimize clock power and latency), and the skew at each of the outputs of the clock tree.

Routing connects the pins from the placed cells using wires on the available metal layers and vias between layers. The objective is to minimize the total wiring needed (reduce congestion and delay), while avoiding design rule violations and meeting timing constraints.

Cell selection methods can be used throughout the design flow to:

Figure 3.1: General Electronic Design Automation flow.



Source: Lee and Gupta (2012)

- fix timing violations, and to optimize the design.
- fix maximum fanout rule violations or maximum slew violations.
- fix design rule violations.
- optimize the design for timing/area/power/variability/yield.

3.2 Transistor Sizing

The transistor sizing problem has been studied for decades and is applied to handle many different problems. The reader is referred to the following publications Kao, Movahed-Ezazi and Sabiers (1984), Glasser and Hoyte (1984), Kao, Fathi and Lee (1985), Fishburn and Dunlop (1985), Pincus and Despain (1986), Matson and Glasser (1986), Hedlund (1987), Cirit (1987), Marple and Gamal. (1987), Shyu et al. (1988), Marple (1989), Dai and Asada (1989), Hoppe et al. (1990), Sapatnekar, Rao and Vaidya (1991), Sapatnekar et al. (1993), Borah, Owens and Irwin (1995), Borah, Owens and Irwin (1996), Sirichotiyakul et al. (1999), Kasamsetty, Ketkar and Sapatnekar (2000), Sundararajan, Sapatnekar and Parhi (2002), Santos et al. (2003), Kursun, Ghiasi and Sarrafzadeh (2004), Chou, Wang and Chen (2005), Santos et al. (2005/a), Boyd et al. (2005), Beece et al. (2010), Kasamsetty and Sapatnekar (2000), Nikoubin et al. (2010), Liao and Hu (2011), Marranghello et al. (2011), Yoshida and Fujita (2011), Posser et al. (2012) for more details about the problem and several formulations and algorithms to handle it.

Transistor sizing can be applied for analog designs and custom digital IC designs.

Both cases are not covered in this work since the focus here in on cell selection in designs using standard cell libraries.

Most formulations for the transistor sizing problem assume a direct and linear relationship between the delay and the transistor width. However, modern designs have several sources of non-linearities that make such models inaccurate.

As mentioned before, with the use of standard libraries, digital IC designs rarely make use of transistor sizing algorithms. One of the major problems is the lack of fast and accurate characterization tools to provide good models and layout parasitic extraction for new cell sizes. Tools for library characterization can take weeks, especially with turn around to correct the generated cells.

The typical formulation for the transistor sizing problem is:

$$\begin{aligned} \textbf{minimize} \quad & f(x) \\ \textbf{subject to} \quad & Delay \leq T_{spec} \\ & L_i \leq x_i \leq U_i \end{aligned} \tag{3.1}$$

where x is the variable available for optimization (usually transistor width), $f(x)$ is the objective function (usually area or power), T_{spec} is the maximum allowed delay and L_i, U_i are the lower and upper bounds for the transistor size.

3.3 Gate Sizing

The gate sizing problem assumes a fixed set of transistor widths for a logic gate in order to model the gate size. Then, the whole gate is sized, instead of sizing separately each one of its transistors.

Gate sizing algorithms can be divided in two categories: continuous gate sizing and discrete gate sizing. We detail both methodologies next.

3.3.1 Continuous Gate Sizing

The continuous gate sizing approach assumes that any size can be implemented in the layout. This approach is suitable for custom digital designs that do not use a cell library.

Continuous gate sizing is not in the scope of this work. However, many discrete sizing algorithms use the continuous sizing formulation to find a discrete solution, by discretizing the continuous solution or using it to guide another discrete approach. Therefore, this work describes some continuous sizing approaches to explain the concepts behind such algorithms that can be applied in the discrete problem as well.

Continuous sizing methods model the delay, area and power as continuous functions of the design parameters similarly to the transistor sizing problem. This model is used to formulate an optimization problem to find a set of continuous sizes that optimize an objective.

With the use of convex delay models, it can be optimally solved by Lagrangian relaxation algorithms. Examples of approaches applied to the continuous gate sizing problem are presented in Chapter 4.

3.3.2 Discrete Gate Sizing and Threshold Voltage Assignment

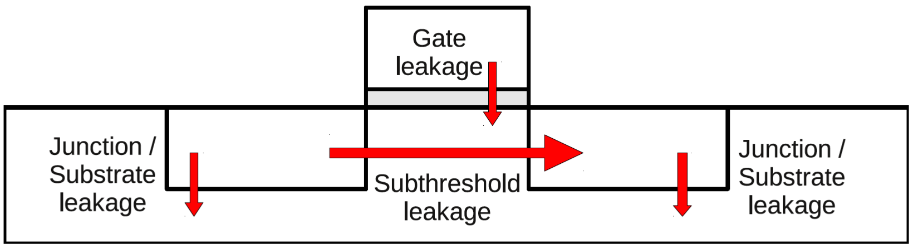
Discrete gate sizing is a known NP-hard combinatorial optimization problem (LI, 1993). It has been extensively studied in the last couple of decades with several different focuses. Many approaches have been proposed to handle a variety of objectives. Sizing algorithms for power (COUDERT; HADDAD, 1996), timing yield (DUTT; REN, 2010), low power standard cell library generation (RAHMAN et al., 2010; AFONSO et al., 2009), placement (CHEN; HSIEH; PEDRAM, 1999; REN; DUTT, 2011), glitch reduction (HASHIMOTO; ONODERA; TAMARU, 1998), variation-aware optimization (SINGH et al., 2005; SINGH; LUO; SAPATNEKAR, 2008) and others have been published. More details about related literature can be found in Chapter 4.

Most works use the following problem definition. Given a design, a standard cell library $\mathcal{L}$ and a timing analysis engine:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{i=1}^{n} x_i & \\
\textbf{subject to} \quad & a_0 \leq T & \text{for outputs} \\
& a_j + D_i \leq a_i & \text{for } j \text{ connected to input of } i \\
& D_n \leq a_n & \text{for inputs} \\
& i \in \mathcal{L} &
\end{aligned} \tag{3.2}$$

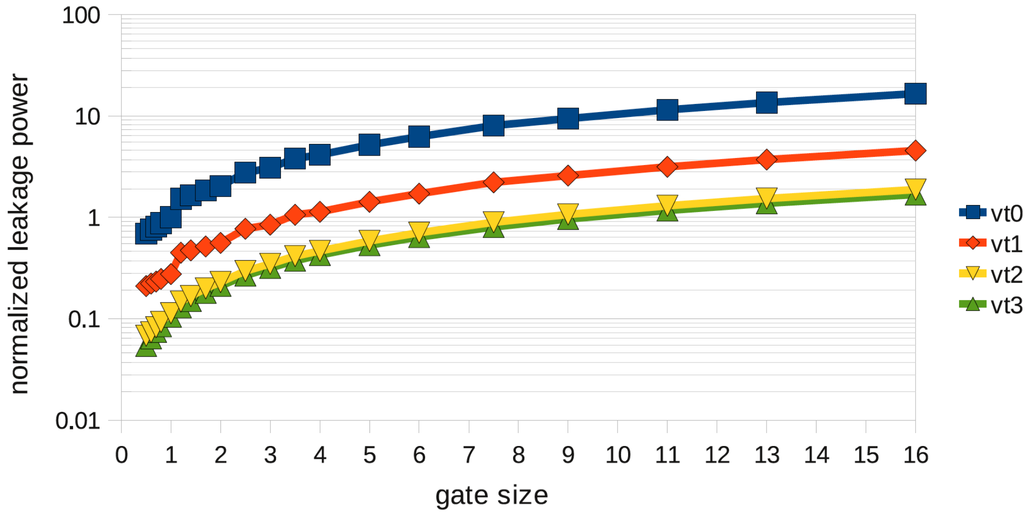
where x_i is the objective associated with cell i (usually area or power) , a_i is the arrival

Figure 3.2: Sources of leakage power in a MOS transistor.



Source: Lee and Gupta (2012)

Figure 3.3: Normalized inverter gate leakage powers in a commercial 45nm library.



Source: Lee and Gupta (2012)

time at i, T is the maximum delay allowed (clock period) and D_i is the delay associated with cell i.

The V_t assignment problem has a formulation similar to the discrete gate sizing problem. In fact, the formulation is the same if there are no specific rules/constraints to the use of different V_t levels.

While the delay of a gate decreases linearly[1] with increasing width, it decreases super-quadratically with decreasing V_t (SAPATNEKAR, 2004). This makes V_t decreasing more effective for reducing the delays of gates.

Figure 3.2 shows the sources of leakage power. The leakage power due to subthreshold conduction is exponentially related to the threshold voltage. This is the main reason for the presence of multiple-V_t cells in standard cell libraries. Since the relationship between leakage and gate size is linear, high-V_t cells enable much bigger leakage power reduction than using sizing alone. Figure 3.3 shows the normalized leakage power as a function of the gate size.

[1]Less than linear if the increase in capacitance for the previous stage of logic is accounted for – internal capacitances also go up.

Most modern technologies allow the use of two or more V_t levels in order to provide cell options with less leakage power than the standard or fastest V_t. When combined with gate sizing, V_t assignment adds another dimension to the solution space, making the use of simplified delay models more complicated. Therefore, many works in literature deal with the V_t assignment problem separated from the gate sizing, using specific algorithms to solve it. Also, it is common to let the designer choose which V_t are available for the algorithms, limiting or expanding the number of options.

In this work, we simultaneously address discrete gate sizing and V_t assignment in order to provide power-optimized solutions. In this way we are able to explore a larger solution space to find a near-optimal solution.

4 RELATED WORKS AND STATE-OF-THE-ART

This chapter presents the previous literature that addresses the cell selection problem. The first section of this chapter briefly summarizes the early literature and publications that have relevant contributions to the problem and especially to the development of this work. The second section summarizes the most recent publications that delimit the state-of-the-art and the two ISPD contests held in 2012 and 2013.

The titles of subsections reflect the respective paper titles or the name of the proposed tool.

4.1 Early Literature

This section details the most relevant related works. The publications are presented in a chronological order to better relate the research to the challenges imposed by the evolution of fabrication process and the design methodologies to which the algorithms can be applied to.

We briefly describe the methods present in some of the most relevant works in the literature that have contributed to the development of cell selection algorithms, especially the algorithms developed and presented in this work.

4.1.1 TILOS

Fishburn and Dunlop (1985) present a transistor sizing algorithm using posynomial programming and convex optimization techniques with a distributed RC model. The capacitances are proportional to transistor size and the resistance is inversely proportional to transistor size.

TILOS starts with minimum transistor sizes for all transistors. A static timing analysis is performed, assigning the latest times (to go low/high) to each node. Then, paths violating the setup timing constraint are traversed in reverse topological order. Transistors that are too slow are examined and the transistor with the highest sensitivity is increased.

The sensitivity function is as follows.

$$D'(x) = RC_u - \frac{R_u C}{x^2} \tag{4.1}$$

where x is the transistor width, C_u and R_u are capacitance and resistance of a unit-sized transistor and C and R are the wire capacitance and resistance, respectively.

Results show that TILOS can improve circuit timing and reduce area. However, the runtime is proportional to the number of timing paths, which is exponential with the number of elements, making it impractical to apply such an algorithm to medium size (10K+) CMOS circuits.

4.1.2 Gate Sizing in MOS Digital Circuits with Linear Programming

References published by Berkelaar and Jess (1990), Berkelaar, Buurman and Jess (1994), Berkelaar, Buurman and Jess (1996) present an algorithm where the gate sizing optimization problem is mapped into a linear program, which is then solved by the Simplex algorithm. A simple (nonlinear) delay model is proposed. A piece-wise linear approximation is performed to reduce linearization error. Any convex delay model can be used with the algorithm without loss of optimality.

The approach guarantees to find the global optimum (considering the inaccurate timing model used), and has proven feasible for circuits of up to several thousand cells. However, this comes with considerable increase in area of up to 172%.

4.1.3 Delay and Area Optimization in Standard-Cell Design

Lin, Marek-Sadowska and Kuh (1990) apply a sensitivity-based heuristic to the discrete cell selection problem. Sensitivities are defined as the ratio between the variation of delay and area. This process of choosing cells based on local cost calculations is the base of several algorithms, with different objectives and different formulations to include, or not, components that estimate the impact on the global solution.

Another feature used together with sensitivities is the "criticality". The criticality is the ratio between the cell delay and its weighted slack. The weighted slack is the slack divided by the path length.

The proposed algorithm starts from all cells set to the smallest option. Then an iterative phase that increase the sizes is applied to solve timing violations. Next a size decreasing phase recovers area on cells on positive slack paths. The two phases are repeated until convergence or stop criteria are met.

The complexity of the proposed algorithm is $O(n^2)$. The results show area improvements ranging from -2%[1] to 29% for a set of nine benchmarks obtained from misII technology mapping tool (DETJENS et al., 1987).

4.1.4 On the Circuit Implementation Problem

References (LI et al., 1992; LI et al., 1993) present a pseudo-polynomial time algorithm, applying suitable decomposition techniques and dynamic programming that obtain optimal solutions for basic series-parallel circuits[2].

Six heuristics are presented to obtain minimal area circuit implementations given a delay constraint. The authors observe that more sophisticated heuristics handle the problem better than the simpler ones. Also, proofs that both basic circuit implementation and general circuit implementation problems are NP-hard are presented.

Another publication by Li (1993) proves that the discrete gate sizing algorithm is strongly NP-hard, implying that for arbitrary DAGs there is no pseudo-polynomial time algorithm to obtain the exact solution unless P=NP.

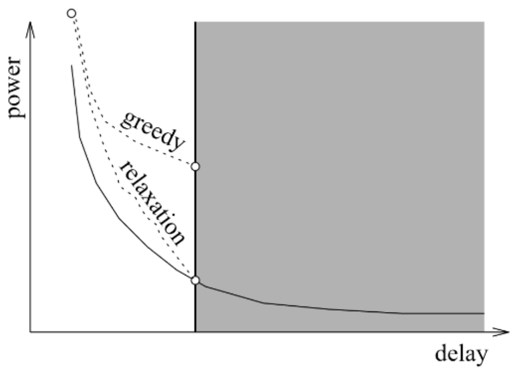
4.1.5 Gate sizing for constrained delay/power/area optimization

The works from Coudert (1996), Coudert, Haddad and Manne (1996), Coudert (1997) present a general purpose gate sizing algorithm GS that is oriented to a pure combinatorial problem optimization, enabling the use of complex cost models. To the best of our knowledge, Coudert (1996) is the first work to address the challenges of gate sizing in modern real-life designs using a standard cell library and an accurate table lookup based nonlinear delay model. With focus on real-life designs, the proposed algorithm uses accurate delay and power models during the optimization process.

The proposed method first evaluates all cells with the calculation of gradients (variation of the cost function when resizing a node). These gradients are used to choose the new sizes for all cells based on the cost function. Then, a new iteration is performed only with cells that have an updated gradient. The process repeats until some convergence criterion is met.

[1]Degradation in area caused by improvement in delay.
[2]McNall and Casavant (1990) apply similar techniques (dynamic programming) for the synthesis of pipelined architectures.

Figure 4.1: Comparison of the behaviors of the proposed method and a greedy method for power optimization.



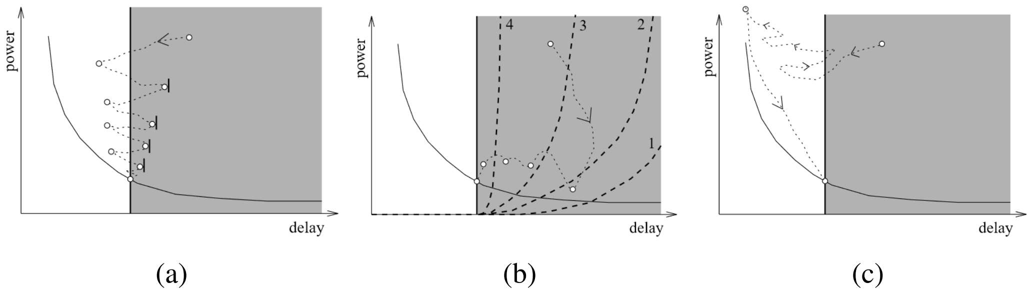
Source: Coudert (1997)

The proposed algorithm can be applied to delay, area and power minimization. In the first case, the algorithm will iteratively process the netlist to choose the best (discrete) sizes that minimize delay. The second and third cases start from a delay optimized solution (obtained with GS) and then proceeds to area/power optimization for the following reasons: 1) Optimizing the delay gives plenty of alternatives for area/power optimization, i.e., going far away from the infeasible region makes power minimization less likely to be trapped in a local minimum. 2) The power optimization is done within the feasible region by relaxing the delay constraints using a penalty/benefit function.

This penalty/benefit function will balance the delay and power through the whole path, as opposed to a greedy method that resizes as many noncritical nodes as possible to their minimal power. Such a greedy approach would deliver low quality results by creating critical paths without power savings proportional to the increase in path delays, preventing most of the other nodes from being resized and saving more power. This behaviour is exposed in Figure 4.1, showing the superiority of the proposed method over the greedy method in scenarios with different timing constraints. The shaded areas in the charts represent solutions with timing constraint violations.

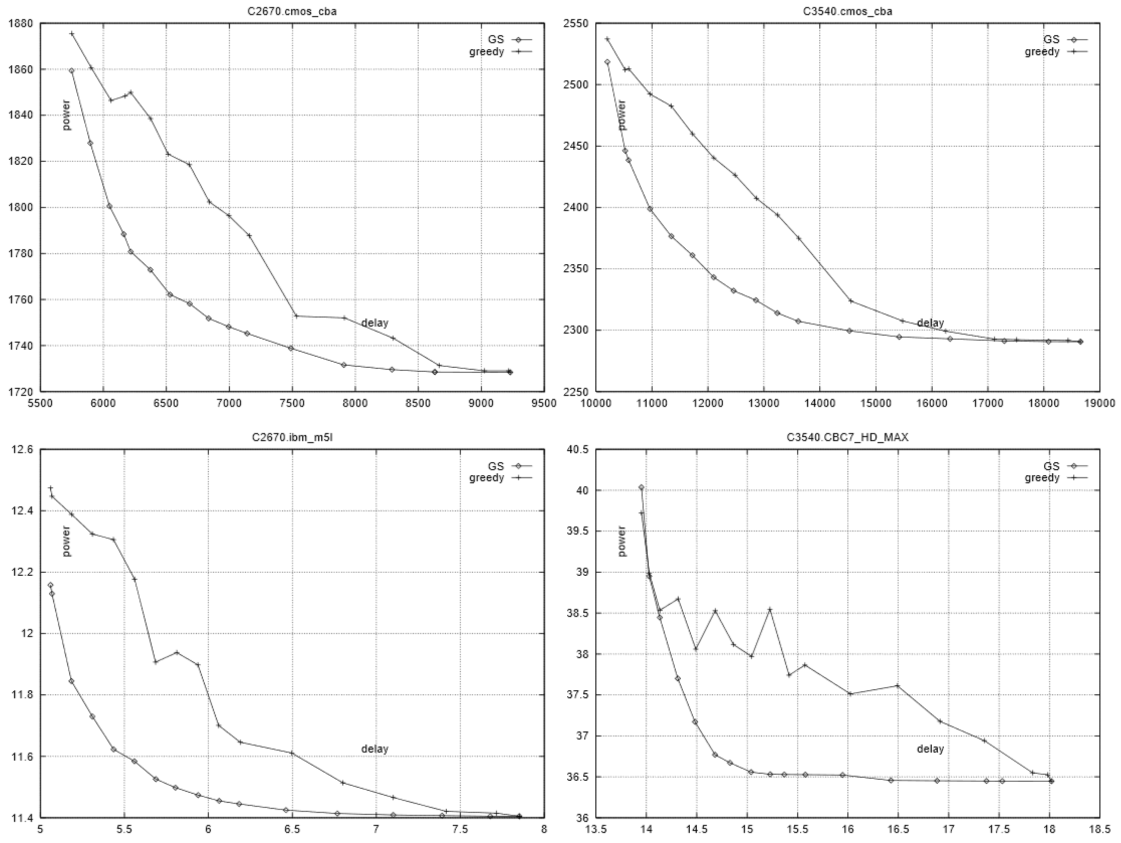
The work also shows a very important analysis of different methods to find a feasible solution with minimal power. Figure 4.2 shows (a) ping-pong, (b) penalty function, and (c) relaxation based constrained optimization. The case shown in Figure 4.2 (b) resembles the behavior of the simulated annealing method presented in this work (Section 5.1, Chapter 5), where the power reduction is obtained while creating timing violations until the point where violations are penalized in detriment of power. The main Lagrangian relaxation algorithm proposed in this work has an hybrid behavior between Figure 4.2 (a) and (c), where the algorithm finds a feasible solution in terms of timing and

Figure 4.2: Behavior of three different methods for power optimization.



Source: Coudert (1997)

Figure 4.3: Power-delay curves for different designs for GS and greedy method.

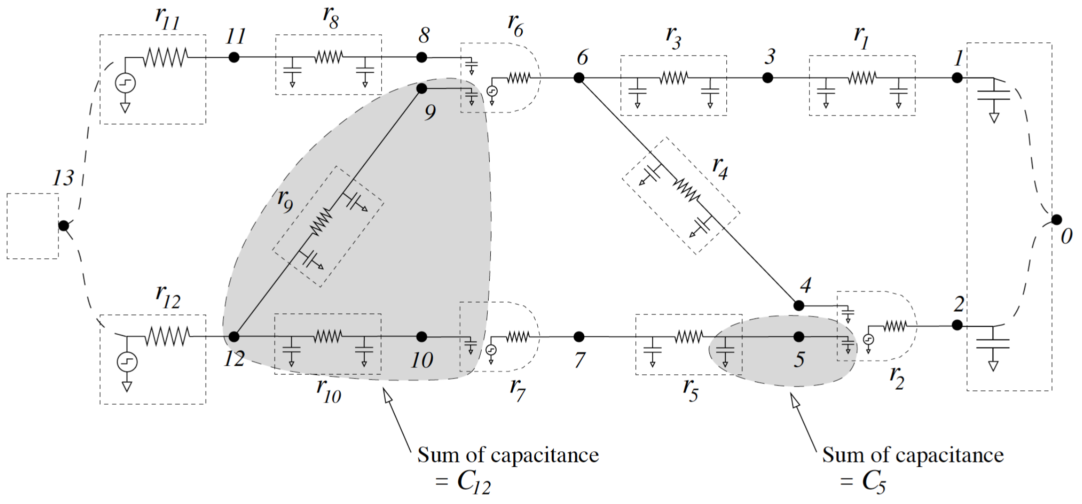


Source: Coudert (1997)

then reduces power, but allowing unfeasible solutions with small violations (ping-pong effect).

Figure 4.3 shows four power-delay curves for different designs for GS and a greedy method. The curves clearly show the superiority of GS over the greedy method in terms of power. The smooth behavior of power along with different target delays is important to measure the quality of the proposed methods because it shows the stability of the algorithm in different scenarios where delay and power have different weights in the input design.

Figure 4.4: Circuit representation after replacing the components by their models (dashed lines).



Source: Chen, Chu and Wong (1999)

Both GS and greedy results show good improvements in both delay and power. Comparing GS with the greedy method, there is only a small difference for delay values. On the other hand, GS results for power are in average 5.2% better than the greedy method.

4.1.6 Fast and Exact Simultaneous Gate and Wire Sizing by Lagrangian Relaxation

One of the most relevant Lagrangian relaxation works in gate sizing is presented by Chen, Chu and Wong (1999). It was the first work to prove convergence and to guarantee theoretical optimality for the continuous gate sizing problem using convex delay model. An exact algorithm for gate and wire sizing based on Lagrangian Relaxation and "one-gate/one-wire-at-a-time" local optimization is presented. The timing constraints are defined on the gates rather than on the signal paths. This will generate a linear number of timing constraints, allowing time complexity of $O(n \times i)$, where n is the number of gates and i is the number of iterations.

The proposed method uses the Elmore delay model (ELMORE, 1948) and the gate is modelled as a switch-level RC circuit with a resistance proportional to the gate size. Figure 4.4 shows how the circuit is represented after replacing the components by the their models (dashed lines).

The Primal Problem (PP) in the Lagrangian relaxation is formulated as follows:

$$\begin{aligned}
&\mathcal{PP}: \\
&\textbf{minimize} \quad && \sum_{i=1}^{n} \alpha_i x_i \\
&\textbf{subject to} && a_j \leq A_0 \quad j \in input(0) \\
& && a_j + D_i \leq a_i \quad i \in \mathcal{G} \cup \mathcal{W} \wedge \forall j \in input(i) \\
& && D_i \leq a_i \quad i \in \mathcal{D} \\
& && L_i \leq x_i \leq U_i \quad i \in \mathcal{G} \cup \mathcal{W}
\end{aligned} \tag{4.2}$$

where α is a user-specified constant weight for each component, x represents the area (or the leakage power) of the component, a is the arrival time, D the delay, L and U are the lower and upper limits for x. '0' is the sink node of the circuit and A_0 the required arrival time at the output components. $\mathcal{G}$, $\mathcal{W}$ and $\mathcal{D}$ are respectively the set of component indexes of gates, wires segments and input drivers in the circuit.

Introducing the Lagrange multipliers for each constraint:

$$\begin{aligned}
L_\lambda(x, a) = &\sum_{i=1}^{n} \alpha_i x_i + \sum_{j \in input(0)} \alpha_{j0}(a_j - A_0) \\
&+ \sum_{i \in \mathcal{G} \cup \mathcal{W}} \sum_{j \in input(i)} \lambda_{ij}(a_j + D_i - a_i) + \sum_{i \in \mathcal{D}} \lambda_{mi}(D_i - a_i)
\end{aligned} \tag{4.3}$$

The Lagrangian relaxation subproblem then becomes:

$$\begin{aligned}
&\mathcal{LRS}/\lambda: \\
&\textbf{minimize} \quad && L_\lambda(x, a) \\
&\textbf{subject to} && L_i \leq x_i \leq U_i \quad i \in \mathcal{G} \cup \mathcal{W}
\end{aligned} \tag{4.4}$$

Using the Karush-Kuhn-Tucker (KKT) conditions to optimality, the problem is greatly simplified, eliminating the arrival time from the equation for the dual problem. This transformation allows only delay calculation during the $\mathcal{LRS}$ solving. The simplified equation becomes:

$$L_\mu(x) = \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n+s} \mu_{ij} D_i \tag{4.5}$$

where, $\mu_i = \sum_{j \in input(i)} \lambda_{ij}$ for $0 \leq i \leq n + s$ (any node in the circuit).

This equation only depends on the gate delays. Using a convex delay model (like the Elmore delay model) the solution is guaranteed to be optimal, although such model is inaccurate.

In order to solve the Lagrangian relaxation subproblem, a new set of sizes for all components is calculated for a fixed set of multipliers. When $i \in \mathcal{G}$, let x_i be the gate size, r_i be the output resistance of the gate and c_i be the input capacitance of a pin of the gate. Let $\hat{r}_i$ and $\hat{c}_i$ be respectively the unit size output resistance and the input capacitance per unit size of gate i. The optimal local sizing of component i is given by equation 4.6.

$$x_i^* = min\left(U_i, max\left(L_I, \sqrt{\frac{\mu_i \hat{r}_i C_i}{R_i \hat{c}_i + \alpha_i}}\right)\right) \tag{4.6}$$

where

$$\begin{aligned} R_i &= \sum_{j \in input(i)} \mu_i \frac{\hat{r}_i}{x_j} \\ C_i &= \sum_{k \in output(i)} \mu_i \frac{\hat{c}_k}{x_k} \end{aligned} \tag{4.7}$$

The proposed iterative algorithm to solve the Lagrangian relaxation subproblem $SOLVE\_LRS/\mu$ is shown in Figure 4.5a. This algorithm performs the sizing (based on the calculated values of capacitance and resistance) with fixed Lagrange multipliers.

Algorithm $SOLVE\_LDP$ shown in Figure 4.5c finds the optimal set of multipliers. Initially, an arbitrary set of Lagrange multipliers respecting the Karush–Kuhn–Tucker (KKT) conditions ($\lambda \in \Omega_\lambda$, where Ω_λ is the set of multipliers that respect the flow conservation property) is set (line 1). Then, the iterative sizing algorithm calculates the new sizes until convergence (line 2). After convergence of $SOLVE\_LRS/\mu$, a new set of Lagrange multipliers is calculated (line 3) and projected into Ω_λ (line 4).

The algorithm $SGWS-LR$ shown in Figure 4.5b solves optimally the gate sizing problem. Line 1 is the the iterative method to find the optimal set of Lagrange multipliers λ. Then, $SOLVE\_LRS$ performs the optimal sizing for each component. Algorithm $SOLVE_LDP$ in Figure 4.5c finds the set of Lagrange multipliers for the new solution, using an user-defined step size ρ_k.

Those algorithms are the main inspiration for the Lagrangian relaxation-based algorithm presented in this work. In the discrete case, the optimality properties are lost, but the algorithm is still able to converge to an optimized solution.

Our work also presents algorithms to circumvent the use of an arbitrary initial set of multipliers in $SOLVE\_LDP$.

Figure 4.5: Algorithms (a) $SOLVE\_LRS/\mu$, (b) $SOLVE\_LDP$, and (c) $SGWS-LR$.

```
ALGORITHM SOLVE_LRS/$\boldsymbol{\mu}$:
Output: $\boldsymbol{x} = (x_1, \ldots, x_n)$ which minimizes $L_\mu(\boldsymbol{x})$
1. for $i := 1$ to $n$ do $x_i := L_i$
2. /* Finding $C'_i$ for $1 \leq i \leq n$ by traversing
       the circuit in a reverse topological order */
   for $i := 1$ to $t$ do
       $C'_i := \begin{cases} C_i^L & \text{if } i \in \mathcal{G} \\ C_i^L + f_i/2 & \text{if } i \in \mathcal{W} \end{cases}$
   for $i := t+1$ to $n$ do
       $C'_i := \begin{cases} 0 & \text{if } i \in \mathcal{G} \\ f_i/2 & \text{if } i \in \mathcal{W} \end{cases}$
       for all $k$ s.t. $i \in input(k)$ do
           $C'_i := \begin{cases} C'_i + \widehat{c}_k x_k & \text{if } k \in \mathcal{G} \\ C'_i + \widehat{c}_k x_k + f_k/2 + C'_k & \text{if } k \in \mathcal{W} \end{cases}$
3. /* Finding $R_i$ and $x_i$ for $1 \leq i \leq n$ by traversing
       the circuit in a topological order */
   for $i := n$ downto $1$ do
       $R_i := 0$
       for all $j \in input(i)$ do
           $R_i := \begin{cases} R_i + \mu_j \widehat{r}_j / x_j & \text{if } j \in \mathcal{G} \\ R_i + \mu_j \widehat{r}_j / x_j + R_j & \text{if } j \in \mathcal{W} \\ R_i + \mu_j R_{j-n}^D & \text{if } j \in \mathcal{D} \end{cases}$
       $x_i := \min\left(U_i, \max\left(L_i, \sqrt{\mu_i \widehat{r}_i C'_i / (\widehat{c}_i R_i + \alpha_i)}\right)\right)$
4. Repeat step 2 and 3 until no improvement.
```

(a)

```
ALGORITHM SGWS-LR:
Output: the optimal gate and wire sizing solution $\boldsymbol{x}$
1. Call SOLVE_LDP to find the optimal $\boldsymbol{\lambda}$.
2. $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{n+s})$ where $\mu_i = \sum_{j \in input(i)} \lambda_{ji}$
3. Call SOLVE_LRS/$\boldsymbol{\mu}$ to find the optimal $\boldsymbol{x}$.
```

(b)

```
ALGORITHM SOLVE_LDP:
Output: $\boldsymbol{\lambda}$ which maximizes $\mathcal{LRS}/\boldsymbol{\lambda}$
1. $k := 1$ /* step counter */
   $\boldsymbol{\lambda} :=$ arbitrary initial vector in $\Omega_\lambda$
2. $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_{n+s})$ where $\mu_i = \sum_{j \in input(i)} \lambda_{ji}$
   Solve $\mathcal{LRS}/\boldsymbol{\lambda}$. (Solve $\mathcal{LRS}/\boldsymbol{\mu}$ by SOLVE_LRS/$\boldsymbol{\mu}$,
       and then calculate $a_1, \ldots, a_{n+s}$ as in Section 3.2).
3. /* Move to a new $\boldsymbol{\lambda}$ by adjusting multipliers $\lambda_{ji}$ */
   for $i := 0$ to $n+s$ do
       for all $j \in input(i)$ do
           $\lambda_{ji} := \begin{cases} \lambda_{ji} + \rho_k(a_j - A_0) & \text{if } i = 0 \\ \lambda_{ji} + \rho_k(a_j + D_i - a_i) & \text{if } i \in \mathcal{G} \cup \mathcal{W} \\ \lambda_{ji} + \rho_k(D_i - a_i) & \text{if } i \in \mathcal{D} \end{cases}$
4. Project $\boldsymbol{\lambda}$ onto the nearest point in $\Omega_\lambda$.
5. $k := k + 1$
6. Repeat step 2–5 until $(\sum_{i=1}^{n} \alpha_i x_i - Q(\boldsymbol{\lambda})) \leq$ error bd.
```

(c)

Source: Chen, Chu and Wong (1999)

Another important difference in our work is how the new sizes are selected. The algorithm $SOLVE\_LRS/\mu$ uses a closed form to calculate the new widths based on the convex delay model to determine the new optimal resistance and the capacitance. In contrast, our proposed model evaluates in one step a cost function for all discrete options available, choosing the lowest cost solution as the new assigned option.

4.1.7 Forge

Tennakoon and Sechen (2002) presented a new fast gradient-based pre-processing step to provide an effective set of initial Lagrange multipliers for the continuous gate sizing problem.

The Lagrangian relaxation method is formulated following the same steps and simplifications proposed by Chen, Chu and Wong (1999).

The work proposes a method to find an initial set of Lagrange multipliers using a steepest-descent, gradient-based approach. The goal is to arrive at the desired delay contour. The steepest descent search is applied in topological order to minimize the delay, using equation 4.8[3]. Re-sizing is repeated until the delay is less than the target delay.

$$x_i^* = \sqrt{\frac{\hat{r}_i C_i}{R_i \hat{c}_i}} \tag{4.8}$$

The equation to find the multiplier based on the size of i and its input j is

$$\lambda = \frac{\alpha_i x_i}{\left(\hat{r}_i/x_i\right) C_i - \left(\hat{r}_j/x_j\right) \hat{c}_i x_i} \tag{4.9}$$

Deriving this equation to handle n-input gates and assuming $\lambda_{ij} = \mu_i/n$,

$$\mu_i = \frac{\alpha_i x_i}{\left(\hat{r}_i/x_i\right) C_i - \frac{1}{n} R_{s_i} \hat{c}_i x_i} \tag{4.10}$$

where Rs_i is the sum of the upstream resistances.

Rearranging and solving for x:

$$x_j = \sqrt{\frac{\lambda_i \hat{r}_j C_j}{(\alpha_j + \lambda_i Rs_j \hat{c}_j)}} \tag{4.11}$$

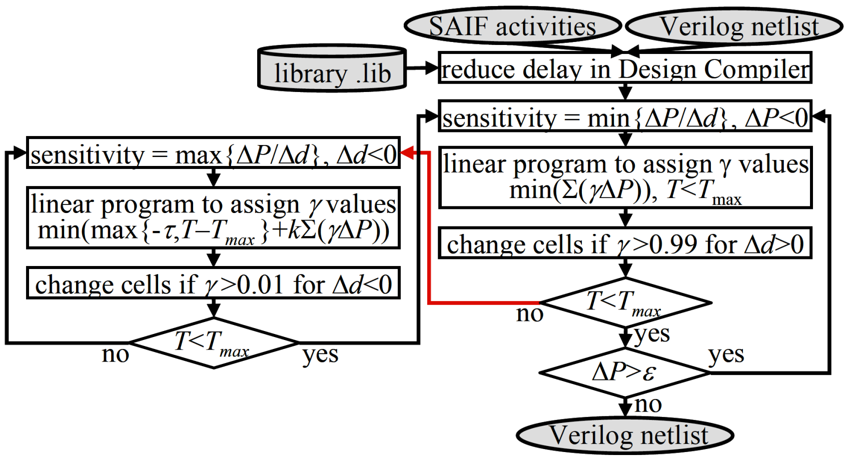
This proposed approach shows the importance of the initial set of multipliers and the effect of such choice in the final convergence even in methods with guaranteed convergence. $Forge$ takes advantage of the closed form calculation that relates delay and widths (resistance and capacitance). In our proposed lambda initialization we also try to approximate the multipliers to correspond with delays in the current solution, but without a closed form equation.

The authors propose a modified subgradient search method for multiplier update $LagrangeM$. The idea is to avoid the difficulty in determining the proper factors used to control the step size adjustments to the multipliers in a subgradient-based method.

The work from Tennakoon and Sechen (2005) extends $Forge$ to handle a novel piecewise convex delay model for static CMOS gates. The model handles distinct rise/-fall delays and rise/fall slew rates. The method improves the accuracy of the model by subdividing the available simulation data in to small regions, fitting one convex function per region. An overlap between regions is defined in order to prevent the solution getting trapped between two regions. Results comparing with a commercial transistor sizing tool show an average 28.6% area reduction, showing the superiority of LR methods over common greedy methods.

[3]Variables and constants follow definitions of previous reference (CHEN; CHU; WONG, 1999).

Figure 4.6: Optimization flow.


SAIF activities
Verilog netlist
library .lib
reduce delay in Design Compiler
sensitivity = min{ΔP/Δd}, ΔP<0
linear program to assign γ values min(Σ(γΔP)), T<T_max
change cells if γ>0.99 for Δd>0
T<T_max
no
yes
ΔP>ε
yes
no
Verilog netlist
sensitivity = max{ΔP/Δd}, Δd<0
linear program to assign γ values min(max{-τ,T−T_max}+kΣ(γΔP))
change cells if γ>0.01 for Δd<0
T<T_max
no
yes

Source: Chinnery and Keutzer (2005)

4.1.8 Linear Programming for Sizing, V_{th} and V_{dd} assignment

Chinnery and Keutzer (2005) expose the problems of using sensitivity-based greedy sizing as in (SRIVASTAVA; SYLVESTER; BLAAUW, 2004; SHAH et al., 2005), proposing linear programming for a global sizing approach. The linear program is similar to Nguyen et al. (2003) but modeling each timing arc and including wire loads.

Figure 4.6 shows the flow for the linear programming (LP) problem formulation. The flow is divided into two steps, both using linear programming to assign which cells will change. The first step reduces power from the initial solution (obtained from a commercial design tool) while trying to still meet the timing constraints. Since there is no guarantee that the timing constraint will be satisfied, the second step reduces the delay for the cases where timing constraint is violated. The process iterates until power improvements are less than a defined threshold.

Experimental results show 10% to 16% power improvements over commercial tool solutions when two threshold voltages are available. A greedy post-optimization method is applied to further reduce sizes of gates in paths with some positive slack left after LP optimization. Nevertheless, the method presents power increase in one case where only one threshold voltage is available. Moreover, the LP method used is very timing consuming when compared to LR methods.

4.1.9 Timing-aware Power Minimization via Extended Timing Graph Methods

Qian and Acar (2007) discuss the practical issues in applying gate sizing, and the experimental validation of the proposed methods with real circuits from high performance microprocessor designs. The work presents a sensitivity-based method that allows fast incremental and concurrent gate sizing and V_t assignment.

The method starts with a design meeting the timing requirements and then minimizes the leakage power without creating any timing violations. A graph-based iterative approach is used to look for an optimal set of gates to modify in each iteration.

The work also addresses constraints of industrial design flows usually not addressed in literature works, such as *Notouch* designation and cell *Grouping*, presenting how the proposed methods handle such constraints.

4.1.10 Gate Sizing for Cell-Library-based Designs

The work presented in (HU; KETKAR; HU, 2007; HU; KETKAR; HU, 2009) proposes a *continuous solution guided dynamic programming algorithm* to directly solve the discrete gate sizing problem. The algorithm is a combination of continuous sizing and discrete sizing methods. The search space of the dynamic programming (discrete) is significantly narrowed down under the guidance from a good continuous solution. At each gate node, instead of every discrete gate size, only those close to the continuous solution will be investigated, avoiding excessive runtime.

The continuous problem is solved using the Lagrangian relaxation formulation from (CHEN; CHU; WONG, 1999) with Elmore delay model. Partial discrete solutions are selected for investigation using a simulated annealing-like randomized process based on the proximity to the continuous solution. More gate assignments are investigated for timing critical nodes.

Two types of pruning are used. The first is called node pruning which is performed when a node is processed. Solutions with known inferiority based on delay and area are pruned. Only the solution with either a smaller maximum delay or smaller total area survives. The second type is called solution-set pruning which is performed after a node is processed and when the size of the solution set is greater than a threshold. Similar solutions are grouped and then the representative one from each group is selected for further propagation. The solutions which have not been selected are pruned to save runtime, enabling the LR-based method to achieve runtimes similar to other simpler techniques. In our work, pruning techniques are also employed to make the LR methodology runtime feasible.

Results are compared with an implementation of (COUDERT, 1997). 1%–21% of area-cost reductions are obtained. The runtime including computing the continuous solution is, on average, about 50% higher than that of (COUDERT, 1997).

4.1.11 Gate Sizing for Large Cell-based Designs

Reference (HELD, 2009) uses slew targets instead of delay budgets to guide the sizing process. Gates are sized (chosen from cell library) to meet the slew target on its successors input pins. Similar to our work, it is applied to industrial designs.

First, the algorithm sets the slew targets. Then, an iterative approach repeatedly chooses the cell sizes to meet the slew targets, updates timing and computes new slew targets based on an estimate of the slew gradient. The gradient guides the cell to a locally optimum solution. The algorithm avoids expensive incremental timing updates. Instead, the timing is updated for the complete design by a timing oracle once per iteration.

After the fast global sizing, a local search (greedy) sizing is applied to further improve the result. The greedy algorithm is applied to gates in the most timing critical paths and their fanout gates (bounded to 0.2% of total circuit size) to improve timing.

Results comparing to a industrial tool show an 60% average improvement in TNS with similar WNS. Area is increased by 7% on average. The total runtime is 5X faster than the industrial tool for the chosen set of benchmarks (ranging from 64K to 5879K cells).

4.1.12 A New Algorithm for Simultaneous Gate Sizing and Threshold Voltage Assignment

References (LIU; HU, 2009) and (LIU; HU, 2010) propose two new techniques which enable a DP-like solution search for gate sizing and Vt assignment. One is based on relaxing the presented history consistency constraint. The other is an iterative bi-directional search that seeks solutions along both the forward and the backward topological directions.

Figure 4.7: Timing optimization gate sizing algorithm (LIU; HU, 2010).

```
Input  : combinational circuit $G$ and cell library $L$
Output: size and $V_t$ assignment for all gates in $G$

// Initialize the option sets at
   multi-fanin gates
$X_i \leftarrow$ {all implementaion options of $v_i$}, $\forall v_i \in V$;
repeat
    // PHASE I: Initial Optimization
      $ConsistencyRelaxation(G)$;
      $ConsistencyRestoration(G)$;
    // PHASE II: Iterative Refinement
      $InterativeRefinement(G)$;
    // Update the option sets at
       multi-fanin gates
    $X_i \leftarrow$ {$v_i$'s used options in this iteration}, $\forall v_i \in \mathcal{V}$;
until no improvement ;
```

Compared to the sensitivity driven heuristics, the algorithm is more systematic and therefore can lead to improved solution quality. The algorithm can be directly applied on DAG topology. This is achieved using two techniques: *consistency relaxation* and *coupled bi-directional search*. Those techniques are combined in the timing optimization flow as shown in Figure 4.7.

The solution found by one iteration of relaxation and restricted bi-directional search may be a local optimum. Therefore, multiple iterations of relaxation and restriction are used. This approach is thus called iterative joint relaxation and restriction (IJRR).

The disadvantages of Phase I are compensated by an iterative refinement procedure in Phase II. Each iteration of Phase II consists of a backward search followed by a forward search.

The authors also present a timing-constrained power optimization flow based on Lagrangian relaxation. As in our work, the approach follows the methods proposed in (CHEN; CHU; WONG, 1999).

A parallel GPU-based implementation of the techniques proposed in this work is presented in reference (LIU; HU, 2011), which provides the same power and timing results with an average runtime speedup of 39x.

4.1.13 Lagrangian Relaxation for Gate Implementation Selection

Huang, Hu and Shi (2011) present a Lagrangian relaxation-based method for discrete gate sizing using a projection-based descent method for solving the Lagrangian dual problem. The method predicts the timing constraint vs. Lagrangian multiplier behavior based on the history of previous iterations and then speculates the direction and step size of the descent move.

First, the expected arrival time for all the fan-in gates is calculated such that the sum of multipliers distributing to them is equal to what the gate received from its fan-in gates. The equation used is

$$\sum_{v_j \in fanin(v_i)} \frac{a_{exp} - a_j}{\eta'(T_j, \mu_j)} = \sum_{v_k \in fanout(v_i)} \Delta\lambda_{ki} = \Delta\mu_i \qquad (4.12)$$

where $\eta'(T_j, \mu_j)$ is based on the history of previous iterations, with more emphasis on recent history. This approach was tested in our flow but did not present valid results (LR could not converge).

Then multipliers are calculated for each fan-in gate and also KKT is guaranteed to be satisfied. Authors claim better convergence in the discrete sizing problem for the proposed method when compared to sub-gradient method from (CHEN; CHU; WONG, 1999).

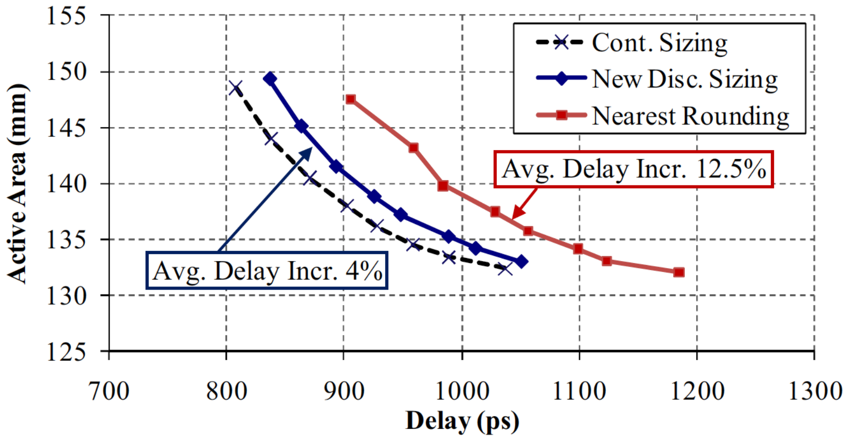
4.1.14 Power Reduction Via Near-optimal Library-based Cell-size Selection

Rahman, Tennakoon and Sechen (2011) extend the continuous sizing formulation in (CHEN; CHU; WONG, 1999) to handle separate rise and fall delays.

The discrete cell sizing is a branch-and-bound methodology guided by continuous solution. Each cell is assigned a candidate set of discrete sizes that are “close” in some sense to the continuous size, and which preserve the continuous beta ratio (β) as much as possible, since PMOS and NMOS transistors’ widths are chosen separately. Upper bounds (bounding box) in both active area and delay based on the continuous solution are set to minimize the number of retained partial solutions. Candidate partial solution points are discarded if they fall outside these bounds.

In the new discrete cell size selection algorithm, the decision on the discretization of a particular cell is delayed until it is clearer how it will globally impact active area and delay for the whole circuit. Each generated partial solution, with one additional discretized gate, requires a call to STA to evaluate its delay. This is the most CPU intensive part of the algorithm. Authors exploit parallelism to reduce the total number of required STA runs as well as the CPU time for each STA run.

Figure 4.8: Active area and delay results for continuous and discrete sizing methods.



Source: Rahman, Tennakoon and Sechen (2011)

The effectiveness of the proposed discretization method is shown in Figure 4.8 where it is shown that the average delay difference from continuous solution is around 4%.

Results show that the active area is reduced by an average of 40% and leakage power by an average of 40% compared to leading commercial tools as used in a specific design flow.

Reference (RAHMAN; SECHEN, 2012) proposes a V_t selection algorithm that is performed before sizing using the method proposed in (RAHMAN; TENNAKOON; SECHEN, 2011). In that algorithm, the threshold voltages are raised as much as possible while strictly maintaining the delay goal. A cost function that is globally aware of the entire circuit is used to rank the solutions.

The algorithm iteratively globally ranks all of the cells based on a cost function which is the total slack elimination (in the whole circuit) divided by the leakage reduction (for that cell). The lowest cost cell is swapped to the next highest available a V_t, as long as the delay is not increased. The procedure iterates until no cell can feasibly be swapped to a higher a V_t.

4.1.15 Gate Sizing and Device Technology Selection Algorithms for High-performance Industrial Designs

Ozdal, Burns and Hu (2011) and Ozdal, Burns and Hu (2012) list the main optimization challenges in modern industrial designs. The main issues are: discrete cell sizes, cell timing models, complex timing constraints, interconnect timing models, slew effects, many near-critical paths, and large design sizes.

The timing related challenges can be handled by an accurate commercial timer. However the formulation must enable the use of such a tool, since accurate timing analysis is very expensive in terms of runtime.

The authors propose a Lagrangian relaxation formulation that decouples timing analysis from the optimization engine without resulting in loss of accuracy. Also, a graph model that captures the delay costs of discrete cells accurately based on timing tables in the cell library is presented. A delta-delay cost metric is proposed to alleviate the suboptimalities due to double counting in directed acyclic graph (DAG) optimization by combining fanin and fanout costs in the subnode costs. A dynamic programming (DP) algorithm based on critical tree extraction is used to solve the LRS optimization problem for discrete cells.

Following a formulation similar to Chen, Chu and Wong (1999) the authors propose the decoupling of timing engine and optimization, through the use of slack values to model the timing constraints. These slacks can be obtained from any commercial sign-off timer. The use of slacks makes the optimization relevant to multiple clock domains and false path calculations, since they are provided by the timing engine. The LRS then becomes

$$\alpha power + \sum_{po} \mu_{po} m_{po} + \sum_{u \to v} \mu_{u \to v}(m_{u \to v} - m_v) \tag{4.13}$$

where α is the scaling factor that trades-off power and timing, $m_{u \to v} = a_u + d_{u \to v} - r_v$ and $m_v = a_v - r_v$. a_i and r_I are the arrival and required time at node i, respectively.

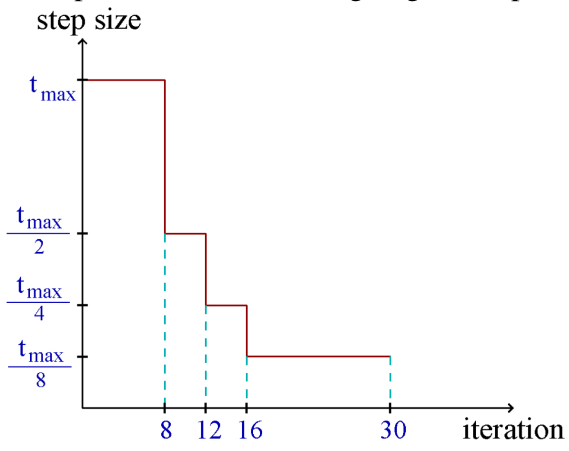
The method for the Lagrange multiplier update is the same as defined by Chen, Chu and Wong (1999). The step size function for the subgradient method is shown in Figure 4.9.

Results show a good convergence for around 30 iterations of LR – see Figure 4.10. This number of iterations is similar (but larger) than the number of iterations of our industrial version of LR-based algorithm. However, it is smaller than the iterations needed in our implementation for the ISPD contest.

The authors present comparisons with respect to the use of an internal (faster but less accurate) timer, showing great improvements when a sign-off timer is used – Figure 4.11.

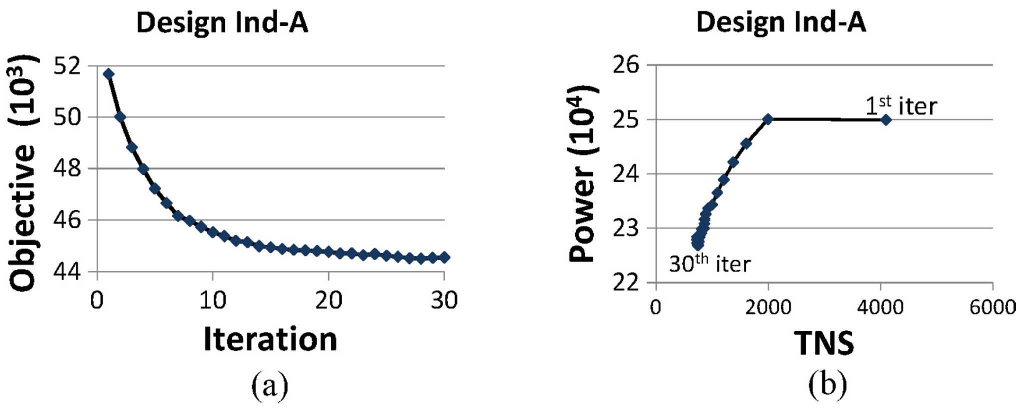
Another point made by the authors is the considerable improvement in quality seen when comparing the proposed DP-based method against a configuration where LR

Figure 4.9: Step size function for Lagrange multipliers update.



Source: Ozdal, Burns and Hu (2012)

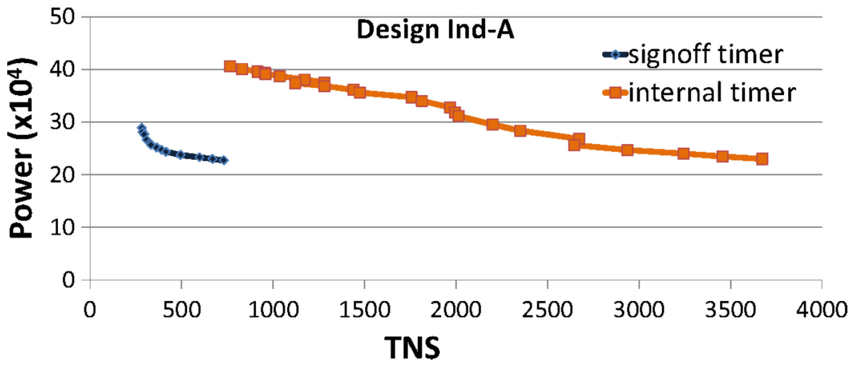
Figure 4.10: Lagrangian relaxation convergence for (a) objective function and (b) TNS.



Source: Ozdal, Burns and Hu (2012)

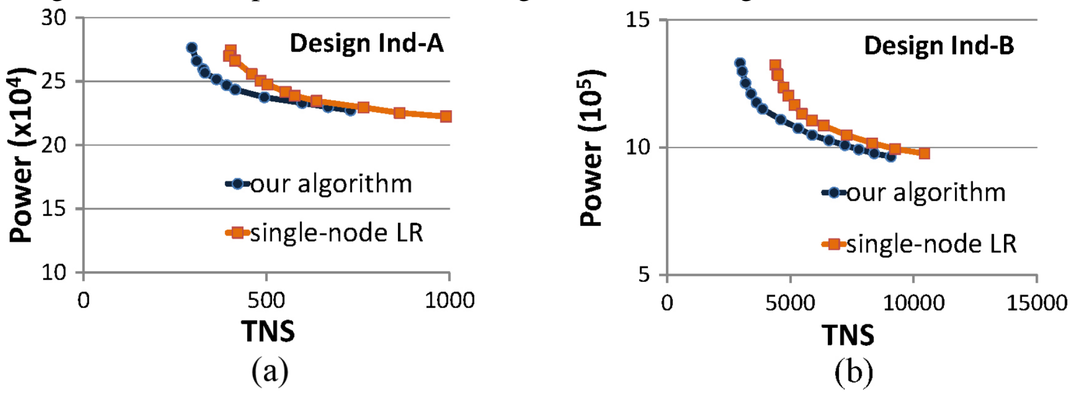
processes gates one-by-one – Figure 4.12.

Figure 4.11: Results when using internal or sign-off timer.



Source: Ozdal, Burns and Hu (2012)

Figure 4.12: Comparison for the DP algorithm and a single node method in LR.



Source: Ozdal, Burns and Hu (2012)

4.2 State-of-the-Art

4.2.1 The ISPD Contest 2012

The ISPD Discrete Gate Sizing Contest 2012 brought attention back to the discrete gate sizing problem (OZDAL et al., 2012). The Contest focuses on simultaneous gate sizing and V_t assignment to optimize static (leakage) power under performance constraints.

The simplified problem formulation comprises a standard cell library and a set of netlists with fixed timing constraints and interconnect parasitics. As the output of the problem, cell sizes and V_t should be provided. The first goal is to satisfy all performance constraints: maximum slew, maximum load capacitance, and setup timing constraints. The second goal is to minimize total leakage power.

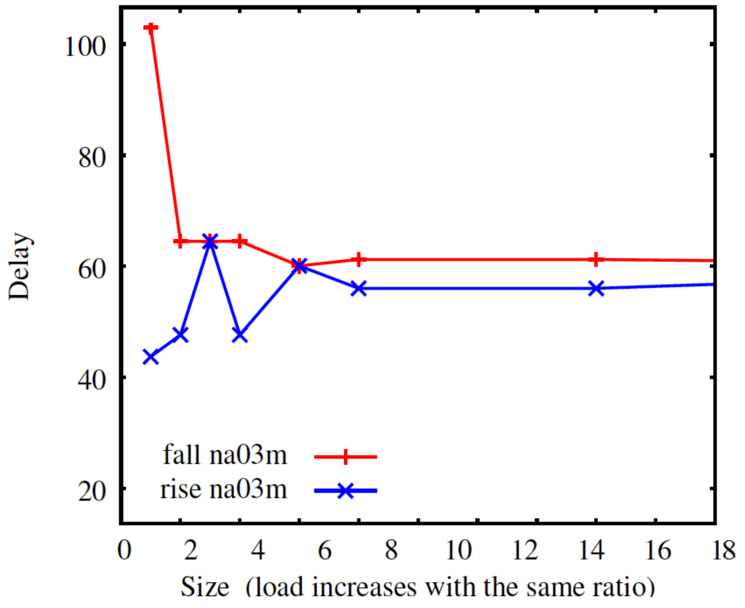
An industrial timing engine is used as the reference timer. All benchmarks provided are guaranteed to have a feasible solution with all constraints being satisfied.

The main objective of the contest was to expose industrial challenges in the gate sizing problem to academia. The most common industrial challenges pointed out by the contest are: discrete cell sizes, continuous optimization and rounding, typically suboptimal non-convex cell timing models due to transistor folding in the layout, slew dependencies and max slew constraints, and large design sizes. This contest was not covering other challenges as: multiple clock domains, false paths, interconnect models.

The cell library was created specifically for the contest, with realistic non-convex timing models (Figure 4.13) and realistic cell sizes and V_t levels. Timing tables were generated based on a simple current source model.

There were two main sources of non-convexities in the models: transistor folding

Figure 4.13: Non-convex gate delays for a 3-inputs medium V_t NAND gate.



Source: Ozdal et al. (2012)

in the layout and p/n transistor size ratios not always constant due to discreteness.

A runtime limit is imposed for the execution of each benchmark. It depends linearly on the number of gates in the design, as follows:

$$Runtime = 5h + 1h \times \left\lceil \frac{\#gates}{35K} \right\rceil \tag{4.14}$$

For the primary ranking, the metric used prioritizes the minimization of the total number of violations. Since all benchmarks had violation-free solutions, the second metric (total leakage power) becomes the real goal in the contest.

In the secondary ranking, violations are still the primary metric, but all the solutions with the same number of violations are ranked based also on the runtime necessary to generate a solution, as defined in 4.15. This metric trades quality by runtime improvement with respect to a reference power and runtime values.

$$cost = \frac{Power}{Power_{REF}} + \gamma \times \frac{Runtime}{Runtime_{REF}} \tag{4.15}$$

$Power_{REF}$ and $Runtime_{REF}$ are the measurements from the best quality solution (according to primary ranking metric) for each benchmark, and γ is set to 0.05, e.g., 1% degradation in the solution quality can be compensated by a 20% runtime reduction with respect to reference values.

Different techniques were presented by the contestant teams (only one methodology published by Reimann et al. (2013)) and a new set of benchmarks together with a

Table 4.1: Number of combinational gates and leakage power (W) on ISPD 2012 benchmarks.

Benchmark	# of Gates	Clock (ps)	Leakage Power (W) 1st	2nd	3rd	**Best**
DMA_slow	25K	900	0.205	0.158	**0.147**	**0.147**
DMA_fast		770	0.511	0.323	**0.312**	**0.312**
pci_bridge32_slow	33K	720	0.203	**0.115**	0.116	**0.115**
pci_bridge32_fast		660	0.512	**0.168**	0.226	**0.168**
des_perf_slow	111K	900	**0.674**	0.884	0.697	**0.674**
des_perf_fast		735	2.390	3.520	**2.320**	**2.320**
vga_lcd_slow	165K	700	0.415	**0.378**	0.391	**0.378**
vga_lcd_fast		610	0.758	**0.580**	0.773	**0.580**
b19_slow	219K	2500	0.627	**0.614**	0.736	**0.614**
b19_fast		2100	2.710	-	4.490	**1.040**
leon3mp_slow	649K	1800	**1.420**	1.790	2.960	**1.420**
leon3mp_fast		1500	-	-	4.940	**2.020**
netcard_slow	959K	1900	**1.770**	1.970	1.940	**1.770**
netcard_fast		1200	**2.010**	2.300	2.970	**2.010**

cell library based on recent technologies were publicly released for the evaluation of the submitted tools. The results are presented in Table 4.1.

However, the results of the contest could still be improved by considerable margin (OZDAL et al., 2012). Lee and Gupta (2012) also stated their concern with the quality of results presented in the contest, highlighting that no method was able to establish as the “best" method. The authors also urged that more work is needed to properly develop the state-of-the-art.

That led to the publication of much more improved results after the contest (LI et al., 2012; HU et al., 2012) using the same set of benchmarks and cell library.

New results were presented further improving the best results published after the contest (FLACH et al., 2013). These state-of-the-art results are part of this work, as detailed in next chapter.

Figure 4.14: (a) Initial sizing to fix electrical violations, (b) Min-Clock LR, and (c) Lagrange multiplier update algorithms.

Algorithm 2 Initial Sizing

```
1: for each gate $g$ in reverse topological order of the circuit do
2:     Find the smallest-size alternative cell-type $c_1$ for $g$ that satisfies
       $\alpha * max\_capacitance(c_1) > output\_load(g)$.
3:     Find the smallest-size alternative cell-type $c_2$ for $g$ that satisfies
       $output\_slew(g) < max\_transition$ assuming that the input
       slews of $g$ are $max\_transition$.
4:     Set $g$ to cell-type $max(c_1, c_2)$.
5: end for
```

(a)

Algorithm 1 Min-Clock LR

```
1: Find an initial sizing without capacitance and slew violations.
2: while clock period improved in last iteration is larger than a
   threshold do
3:     for each gate $g$ in design do
4:         if topological_level($g$) % 3 == iteration_count % 3 then
5:             Resize $g$ to minimize $L_\mu(\mathbf{a})$.
6:         end if
7:     end for
8:     Update global timing.
9:     Update $\mu$.
10: end while
```

(b)

Algorithm 3 Updating μ

```
1: for each timing endpoints $po$ in design do
2:     $\mu^r_{po} = \mu^r_{po}(a^r_{po} + d_{po})^\beta$.
3:     $\mu^f_{po} = \mu^f_{po}(a^f_{po} + d_{po})^\beta$.
4: end for
5: for each timing arc $uv$ in design do
6:     $\mu^r_{uv} = \mu^r_{uv}(a^{s(uv)}_{uv} + d^r_{uv})^\beta$.
7:     $\mu^f_{uv} = \mu^r_{uv}(a^{\overline{s}(uv)}_{uv} + d^f_{uv})^\beta$.
8: end for
9: for each timing endpoints $po$ in design do
10:     $cur\_sum = \sum_{po} \mu^r_{po} + \sum_{po} \mu^f_{po}$.
11:     $\mu^r_{uv} = \mu^r_{uv} / cur\_sum$.
12:     $\mu^f_{uv} = \mu^f_{uv} / cur\_sum$.
13: end for
14: for each gate $g$ in reverse topological order of the circuit do
15:     $cur\_sum^r = \sum_{uv \in g} \mu^r_{uv}$. $cur\_sum^f = \sum_{uv \in g} \mu^f_{uv}$.
16:     $opt\_sum^r$ = Equation (4), $opt\_sum^f$ = Equation (5), where
        $n$ is output net of $g$.
17:     for each timing arc $uv$ in gate $g$ do
18:         $\mu^r_{uv} = \mu^r_{uv} * opt\_sum^r / cur\_sum^r$.
19:         $\mu^f_{uv} = \mu^f_{uv} * opt\_sum^f / cur\_sum^f$.
20:     end for
21: end for
```

(c)

Source: Li et al. (2012)

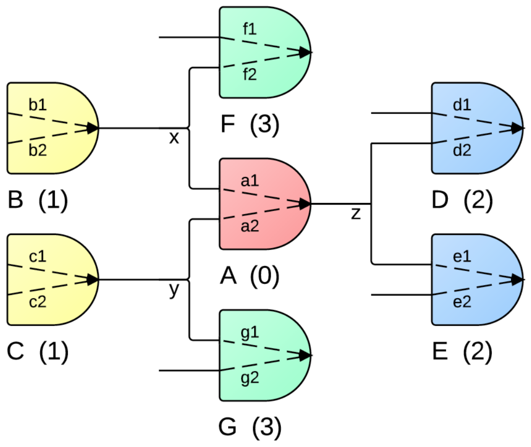
4.2.2 An Efficient Algorithm for Library-based Cell-type Selection in High-performance Low-power Designs

Authors of reference (LI et al., 2012) propose a flow to take advantage of the emerging many-core systems to effectively reduce the design cycle time for modern designs. Another concern is that the designer should have a clear picture of the performance (timing) limit of a design, as well as the possible power-performance trade-off.

The work is based on the ISPD Contest 2012 formulation. The proposed flow includes a Minimum Clock Period Lagrangian Relaxation (Min-Clock LR) method to generate the fastest design with all possible cell-types. After finding the highest-performance design, a min-cost network flow based method is used to optimize the power consumption while maintaining the timing feasibility of the design. Further improvement in power is achieved by a greedy heuristic to prune power by maximally utilizing any remaining timing slacks.

An initial sizing is performed in order to provide a electrical violation-free design to the Min-Clock LR algorithm. The algorithm is shown in Figure 4.14a. $\alpha \leq 1$ is an empirical parameter to control the load ratio between a driver cell and its fanout cells. $\alpha = 1$ produces the slowest design without any capacitance or slew violation. For the

Figure 4.15: Resizing effect example.



Source: Li et al. (2012)

experiments, the authors use $0.6 \leq \alpha \leq 0.8$.

The algorithm of *Min-Clock LR* is shown in Figure 4.14b. The LRS is approximately solved in the inner *for* loop. The outer *while* loop is to solve the LDP. In order to avoid the excessive use of incremental timing updates with an accurate timing model, the authors present a heuristic for the LRS which avoids the incremental timing update. A novel parallel-friendly approach to the LR subproblem (LRS) is presented. It is important to note that these procedures cause timing inaccuracies and sub-optimality – such methods are not employed in our work. Nevertheless, our approach can achieve similar runtimes – see Section 5.4.

The example in Figure 4.15 is used to show the comprehensive effect of resizing a cell. When resizing cell A, while all other cells remain unchanged, all cells connected to input and output pins of A are considered for timing analysis. Cell resizes that create electrical violations are not considered. The delays on timing arcs b1, b2, c1, c2 need to be re-estimated due to load changes. With the slew change in nets x and y, f2 and g1 also need a timing update. The same is true for timing arcs a1 and a2 for the new size of cell A. Then timing on net z and arcs d1 and e1 is updated. The authors claim that no further propagation is needed for accuracy, and even cells F and G could be skipped in some cases (if input pin capacitances of cell A are reduced by a resize).

The LR convergence speed is boosted by a modified subgradient method. The algorithm to update the Lagrange multipliers is shown in Figure 4.14c. Authors propose a new multiplier update method that adjusts sub-gradients according to local delay and arrival time. The parameter β defines the step size. A larger β implies a faster but less controlled convergence. The authors found that $\beta = 2$ is a proper trade-off between stability and fast convergence.

The min-cost flow algorithm is modified to handle discrete non-convex timing models. The network flow formulation is based on the work by Ma and Young (2008). In order to speed up the heuristic, a fast and coarse-grained timing update technique is developed to replace the accurate but expensive timing update. After the min-cost flow algorithm, the solution is further improved by a sensitivity-based greedy method called *Power Pruning*.

Experimental results show that the algorithm can achieve 13% more power savings on designs with fast timing constraints compared to the best ISPD Contest 2012 results – those results are outperformed by our methods (Section 5.4). The algorithm exhibits a near-linear empirical runtime, with processing rate from 1 to 1.5 hours per million cells.

4.2.3 Trident

(HU et al., 2012) presents a sensitivity-guided heuristic approach based on sequential importance sampling (ALDOUS; VAZIRANI, 1994) that integrates power and timing optimization, and handles several types of constraints.

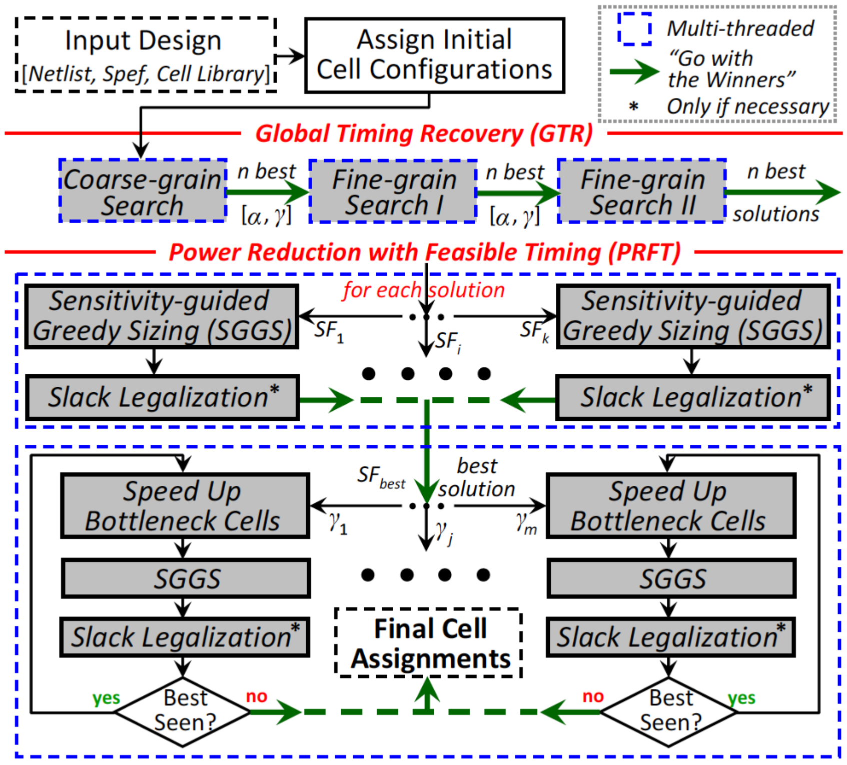
Total Negative Slack (TNS) estimation is used in the sensitivity functions rather than just the worst slack. A parameterized space of sensitivity functions for gate sizing is defined. The method traverses this space using a multistart technique that naturally lends itself to efficient parallelization on multi-core and shared memory CPU architectures, and distributed systems.

The proposed heuristic has two stages – Global Timing Recovery (GTR), and Power Reduction with Feasible Timing (PRFT). GTR first seeks violation-free (feasible) solutions, and then PRFT iteratively reduces total leakage power of sizing solutions by local search. The overall flow is presented in Figure 4.16.

GTR starts with minimum-leakage cell configurations that are incrementally refined by increasing/decreasing gate sizes or decreasing/increasing threshold voltages. Both cell upsizing and decreasing V_t are performed in the smallest possible increments; the ordering of these actions is determined by their sensitivities, which are calculated by the impact on TNS and leakage power.

$$sensitivity_{GTR} = \frac{\Delta TNS}{\Delta leakage\_power^{\alpha}} \tag{4.16}$$

Figure 4.16: Overall gate sizing flow.



Source: Hu et al. (2012)

Figure 4.17: Sensitivity functions used.

Acronyms	Sensitivity functions
$SF1$	$-\Delta leakage\_power/\Delta delay$
$SF2$	$-\Delta leakage\_power \times slack$
$SF3$	$-\Delta leakage\_power/(\Delta delay \times \#paths)$
$SF4$	$-\Delta leakage\_power \times slack/\#paths$
$SF5$	$-\Delta leakage\_power \times slack/(\Delta delay \times \#paths)$

Source: Hu et al. (2012)

All cell modifications are evaluated assuming other cells are fixed. In order to prevent inaccuracies in sensitivities calculation, only the first $\gamma\%$ of modifications are committed between consecutive STA invocations. The variables $0 \leq \alpha \leq 3.0$ and $0 < \gamma \leq 60\%$ determine specific multistart configurations.

The sensitivity-guided greedy sizing (SGGS) method downsizes cells according to the sensitivity while avoiding timing violations. The sensitivities used in SSGS are presented in Figure 4.17.

ISTA(c_i) is an incremental STA operation after cell c_i is changed. The SGGS algorithm starts with STA and initializes all timing nodes. Sensitivities are computed for all downsizable cells. Both gate downsizing and increasing V_t for the sensitivity calculation.

In order to reduce leakage power, the algorithm selects a cell c_i with maximum sensitivity, and downsizes c_i or increases its V_t. Incremental timing analysis and checking for violations is performed to avoid inserting electrical violations. The loop continues until M becomes empty. The slack legalization phase fixes timing violations present after the execution of SGGS. Furthermore, bottleneck cells are identified and sped up to create more space for downsizing and V_t increase in other cells belonging to the same paths. PRFT repeats until no more improvements are found.

The experimental results presented in the paper show considerable improvements over the best results in the ISPD Contest 2012 and similar quality to the results in (LI et al., 2012). Again, the quality of our results outperforms these improvements.

4.2.4 The ISPD Contest 2013

The second contest presented more challenging benchmarks and also a more realistic delay model (OZDAL et al., 2013). For static timing analysis, interconnecting wires are represented as an RC tree, different from the lumped capacitance model used in 2012. This model makes the correct assessment of timing and electrical violations much more challenging, since the algorithms used in the reference commercial sign-off timer are not publicly available.

The introduction of distributed wire delays and slew degradation in the contest makes the contestant's internal STA tools inaccurate with respect to the commercial accurate timer used for the solution evaluation. This inaccuracy will vary based on the chosen timing models used in each tool.

As in the first contest, there are two rankings, the primary and the secondary. For the first ranking the metrics are the same from previous contest. The new runtime limit for the primary ranking is set as:

$$Runtime_{PRIMARY} = 3h + 1h \times \left\lceil \frac{\#gates}{40K} \right\rceil \tag{4.17}$$

For the secondary ranking a tighter runtime is defined:

$$Runtime_{SECONDARY} = \left\lceil \frac{Runtime_{PRIMARY}}{5} \right\rceil \tag{4.18}$$

In the secondary ranking, the quality metric considers the runtime following the equation:

$$cost = Power + \left((1 - \gamma) + \gamma \times \frac{Runtime}{Runtime_{REF}} \right) \tag{4.19}$$

Table 4.2: Number of combinational gates, leakage power (W) and runtime (min) on ISPD 2013 benchmarks.

Benchmark	# of Gates	Leakage Power (W) 1st	2nd	3rd	Runtime (min) 1st	2nd	3rd
usb_phy_slow	609	0.001	0.001	0.001	9.90	0.60	0.79
usb_phy_fast		0.002	0.002	0.007	0.58	0.42	6.75
pci_bridge32_slow	31K	0.058	0.059	0.077	14.28	6.60	225.37
pci_bridge32_fast		0.097	0.107	-	87.03	11.62	-
fft_slow	32K	0.090	0.095	0.107	36.63	23.17	121.07
fft_fast		0.226	0.321	0.638	52.15	35.25	229.38
cordic_slow	42K	0.324	0.444[1]	1.078[1]	94.7	92.42[1]	244.73[1]
cordic_fast		-	-	-	-	-	-
des_perf_slow	113K	0.353	0.380	2.392	96.05	69.17	350.20
des_perf_fast		-	-	-	-	-	-
edit_dist_slow	131K	0.447	0.468	-	116.23	107.93	-
edit_dist_fast		0.596	0.639	-	185.50	166.07	-
matrix_mult_slow	155K	0.470	0.513[1]	1.381	243.40	215.78[1]	416.28
matrix_mult_fast		-	-	-	-	-	-
netcard_slow	982K	5.302	5.371	5.246[1]	549.40	1680.27	1655.42[1]
netcard_fast		5.318	-	19.152	613.25	-	1655.03

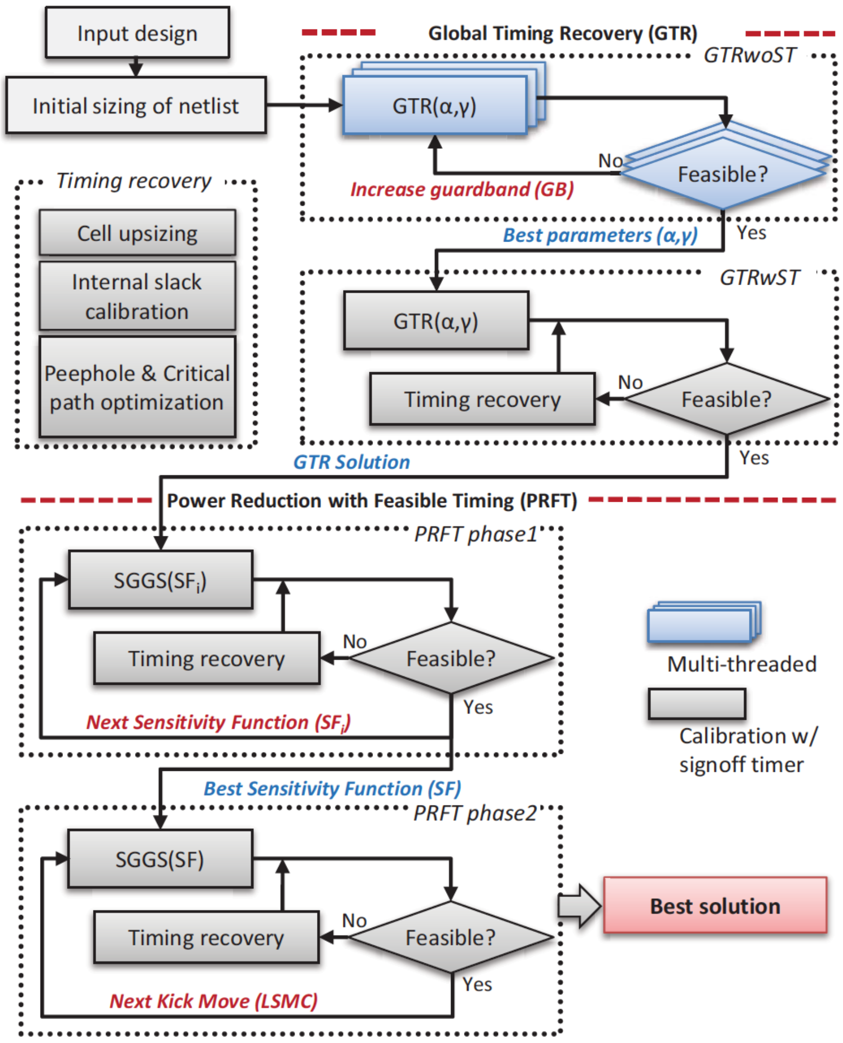
[1] Solutions had very small violations ($<$ 0.1 – ps and/or pF).

$Runtime_{REF}$ is fixed and defined as half of the runtime limit. Also, γ is set to 0.05, e.g., using the full runtime limit will represent an increase in 5% in the solution cost. A runtime close to zero represents a reduction of close to 5% in the final cost.

The winning team – called SOUTH-Brazil, and which algorithms are part of this work – presented a good margin against the second placed tool ($\sim$8%). Contest results are presented in Table 4.2. For three designs, none of the submitted tools were able to find a violation-free solution.

After the contest, the results of the winning tool were further improved by about 10% after changes in the internal timing engine and the use of a commercial accurate timer in some stages of the proposed flow (FLACH et al., 2014). These methodologies are also part of this work. Different from the first contest, the proposed flow finds the best solution for all benchmarks. This method is also the only published work to present results with violation-free solutions for all designs.

Figure 4.18: Optimization flow.



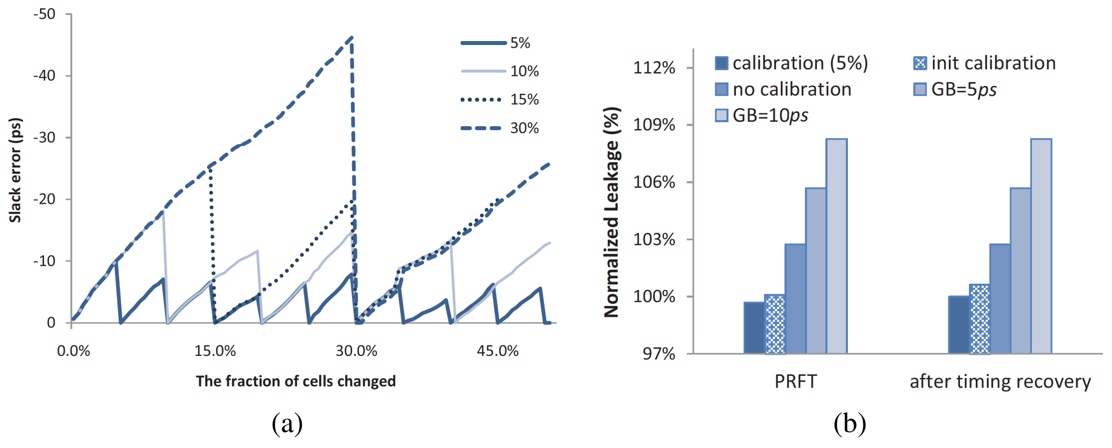
Source: Kahng et al. (2013)

4.2.5 Trident 2.0

Kahng et al. (2013) present a multi-threaded, stochastic optimization tool for cell selection to minimize leakage power subject to capacitance, slew and timing constraints. The overall optimization flow is shown in Figure 4.18.

The first stage of Trident2.0, called *Global Timing Recovery without a sign-off timer (GTRwoST)*, uses a multi-threaded meta-heuristic to optimize individual parameters of lower-level search heuristics with interconnect delay models and constraints, relying only on internal timer calculations. The second stage also seeks to produce a feasible solution, but performing timing calibration with the sign-off timer. The third stage performs power reduction with feasible timing (PRFT), also using timing calibration. A sensitivity-guided greedy downsizing (SGGS) is used with different sensitivity functions. The best

Figure 4.19: (a) Impact of calibration frequency and (b) leakage power results for different calibration strategies.



Source: Kahng et al. (2013)

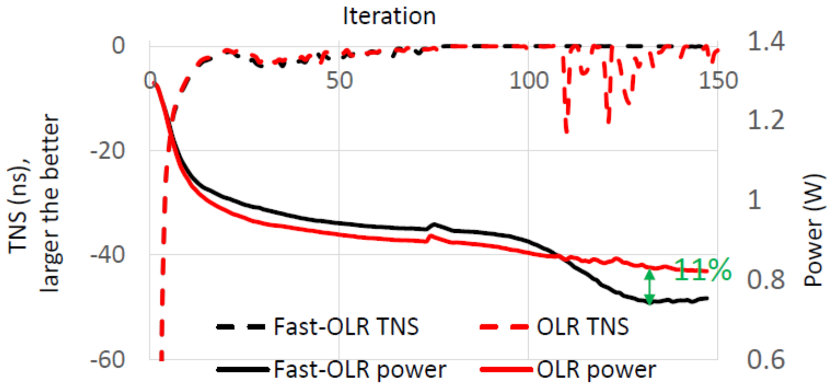
solution is carried to the second phase of PRFT where the best sensitivity function from first phase is used with greedy downsizing as in (HU et al., 2012).

The internal timer applies an offset-based timing calibration method that invokes the sign-off timer periodically and stores the slack differences at every timing endpoint.The accuracy of the internal timer decreases as the number of cell changes accumulates, requiring a new calibration every time the number of cell changes reaches 5% or 10%. Figure 4.19a shows the impact of calibration frequency in the slack error.

Authors also discuss the impact of timing accuracy on leakage power results, showing that frequent timing calibration presents the best leakage power results when compared to less frequent or no calibration results. Figure 4.19b shows the leakage comparison between runs with different calibration strategies. It is shown that methods using calibration are more efficient than using a guardband (GB) to compensate the slack error. The more frequent the calibration (more accuracy), the better are the leakage power results. The D2M (delay with two moments) (ALPERT; DEVGAN; KASHYAP, 2000) and PERI (KASHYAP et al., 2002) models are used for wire delay and slew model, respectively.

Trident 2.0 outperforms the winners in the secondary-metric of ISPD Contest 2013 and places between first and second for the primary metric.

Figure 4.20: TNS and power progression over LR iterations on $b19_fast$ benchmark.



Source: Sharma et al. (2015)

4.2.6 Fast Lagrangian Relaxation Based Gate Sizing using Multi-Threading

Sharma et al. (2015) propose techniques to speedup Lagrangian relaxation-based gate sizing. The proposed flow is tested with the ISPD 2012 Contest benchmark suite.

The authors present two main techniques to explore parallelism in the LR iterations without loosing significant quality in the results of a flow similar to the LR flow described here in Section 5.3. Instead of using the more common leveling and clustering techniques for parallel processing of several gates, it is suggested the use of mutual-exclusion edge (MEE) and directed acyclic graph (DAG) netlist traversal (DNT). These techniques allow better load balancing among threads and a reduction in thread idle time than the former approaches.

Another technique proposed addresses the local resizing during LR iterations. The local search is performed from smallest to largest cell size in the vicinity of current solution and is halted when no improvement in the cost is found in this search window. That approach reduces cell evaluations by 3.3x less and overall LR runtime by 3x.

Figure 4.20 shows the TNS and leakage power convergence for the two optimal local resizing (OLR) implementations. Fast-OLR includes the technique just described. The local search also improves convergence by avoiding disruptions in solution timing quality. We can see in the final iterations of LR that disruptions in timing generate less improved power results.

4.3 Summary and Discussion

Beyond the aforementioned works, many other publications also address the transistor/gate sizing and the device selection problem but are not detailed in this work. The reader can refer to the works from Beeftink et al. (1998), Harris et al. (1997), Sapatnekar (2004), Berkelaar, Buurman and Jess (1994), Berkelaar, Buurman and Jess (1996), Nguyen et al. (2003), Srivastava and Sylvester (2005.), Ghiasi et al. (2004), Santos (2005), Santos et al. (2005/b), Shah et al. (2005), Chopra et al. (2005), Singh et al. (2005), Roy, Chen and Chen (2005), Chinnery (2006), Roy et al. (2007), Singh, Luo and Sapatnekar (2008), Dutt and Ren (2010), Ren and Dutt (2011), Rahman, Tennakoon and Sechen (2013) to find more algorithms and further insights related to the cell selection problem. Also, Wang, Das and Zhou (July 2009) revisit the Lagrangian relaxation gate sizing formulation, correcting misunderstandings and extending it to handle general convex delay models.

Table 4.3 presents a summary of the techniques described in previous section and some characteristics of each one. *Trans./Gate* refers to the application of the method: transistor or gate sizing. *Sign-off* column shows whether the method uses a commercial/industrial sign-off timer during the optimization. *C/D* shows whether the method is applied to the continuous or discrete problem.

Lee and Gupta (2012) present a comprehensive study of the cell selection problem going through the details of several works in the literature and proposed techniques. The work classifies the methods by their techniques and presents useful comparisons between the algorithms.

Many of the issues faced by cell selection algorithms are not new and are extensively studied in literature. However, the difficulty of applying academic algorithms in industry is still one of the major problems. As stated by Coudert (1996), approaches found in literature typically suffer from at least one of the following problems:

1. The cost models, especially for gate delay, slew, wire delay, and power, are not realistic, or are oversimplified to fit an optimization technique.
2. Some methods assume that the gates can be continuously sized, with the idea of solving an easier problem and then projecting the continuous solution on a discrete solution. But projective methods can even fail to find a feasible solution: gate sizing is essentially a combinatorial problem (NP-complete).
3. Some methods make crude assumptions on the optimality criterion, e.g., minimiz-

Table 4.3: Summary of techniques present in references. *Optimal* refers to the optimality claimed for the chosen (inaccurate) models in each work.

Reference	Trans./Gate	Core Method	Optimal	Sign-off	C/D
Fishburn and Dunlop (1985)	Trans.	Posynomial Progr.	No	No	C
Chan (1990)	Gate	Heuristic	Yes	No	D
Berkelaar and Jess (1990)	Gate	Linear Program	Yes	No	C
Lin, Marek-Sadowska and Kuh (1990)	Gate	Sensitivities	Yes	No	D
Li et al. (1993)	Gate	Heuristic	No	No	D
Coudert (1997)	Gate	Heuristic	No	Yes	D
Chen, Chu and Wong (1999)	Gate	Lagrangian	Yes	No	C
Tennakoon and Sechen (2005)	Gate	Lagrangian	Yes	No	C
Chinnery and Keutzer (2005)	Gate	Linear Progr.	No	Yes	D
Qian and Acar (2007)	Both	Sensitivities	No	Yes	C/D
Hu, Ketkar and Hu (2009)	Gate	Lagrangian	No	No	D
Held (2009)	Gate	Slew target	No	Yes	D
Liu and Hu (2010)	Gate	Lagrangian	No	No	D
Huang, Hu and Shi (2011)	Gate	Lagrangian	No	No	D
Rahman, Tennakoon and Sechen (2011)	Gate	Lagrangian	No	No	C/D
Ozdal, Burns and Hu (2012)	Gate	Lagrangian	No	Yes	D
Hu et al. (2012)	Gate	Sensitivities	No	No	D
Kahng et al. (2013)	Gate	Stochastic	No	Calibr.	D

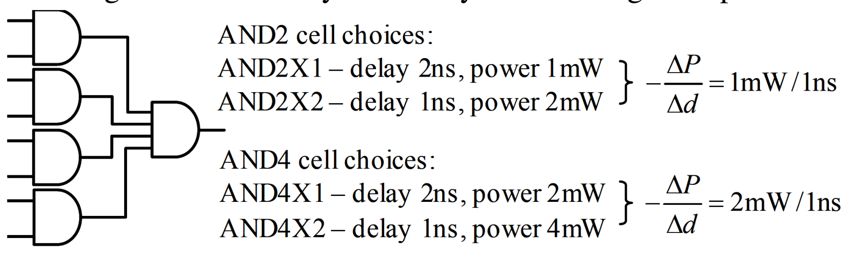
ing a weighted power and delay product is the best power/delay trade-off, while the problem is about constrained optimization.

4. Some methods assume that the objective function or/and the feasible region is convex, which does not hold with accurate delay and power model.
5. Some methods are too runtime expensive to be applied on circuits with more than 1000 nodes.

Also, as detailed by Coudert (1996), several of the industry problems in cell selection are not considered in most publications. The constraints also include maximum fanout load or maximum transition time. Accurate delay models make gate sizing a non-linear, non-convex, constrained, discrete, optimization problem. Moreover, that it is not even unimodal, i.e., several local extrema exist. Because of the delay dependency on output load and input transition time, sizing a gate affects the propagation times and output transition times of its fanin and of its fanout gates, demanding many timing updates. Delay optimization can encounter (and get trapped in) several local extrema because of the non-convexity of the delay model. This makes more important to have a method that can avoid such traps (COUDERT, 1996).

Another industry concern is the application of cell selection when the design is

Figure 4.21: Greedy sensitivity-based sizing example.



AND2 cell choices:
AND2X1 – delay 2ns, power 1mW
AND2X2 – delay 1ns, power 2mW
$-\frac{\Delta P}{\Delta d} = 1\text{mW}/1\text{ns}$
AND4 cell choices:
AND4X1 – delay 2ns, power 2mW
AND4X2 – delay 1ns, power 4mW
$-\frac{\Delta P}{\Delta d} = 2\text{mW}/1\text{ns}$

Source: Chinnery and Keutzer (2005)

infeasible, violating some timing constraints. The objectives become different than most works found in the literature. In industry, a practical objective is to maximize the worst slack, but to also push less critical negative slacks towards zero. This approach reduces the need for other more resource-consuming optimization routines (HELD, 2009).

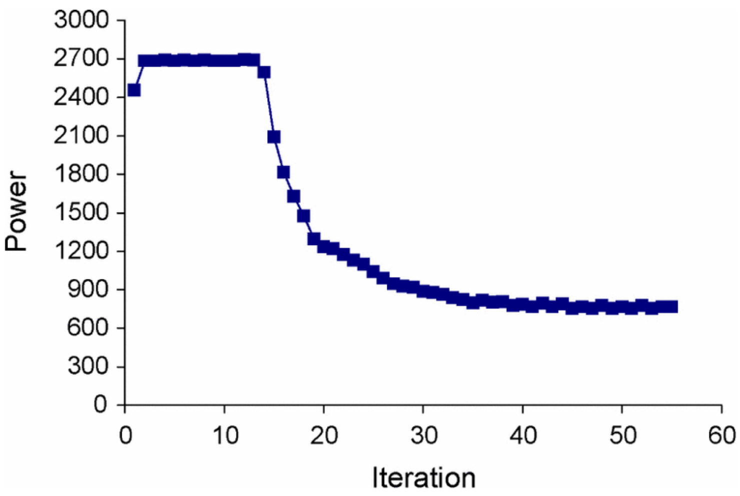
The importance of using cell selection algorithms in incremental optimization tools is examined by Lee and Gupta (2012). Performing gate sizing and changing V_t levels is less disruptive than changing placement and/or routing of tens or hundreds of cells.

References also highlight that known and widely used methods can also fail even in simple cases. Chinnery and Keutzer (2005) present an example of how greedy sensitivity-based methods fail to find local optimal solutions in simple cases (Figure 4.21). Just choosing the gate with the maximum sensitivity is suboptimal. For instance, if all the gates in the above example are initially sized with the X2 option, the critical path delays is 2ns and total power consumption is 12mW. Considering a 3ns delay target, the maximum power_reduction/delay_increase sensitivity choice wiil be to downsize the AND4 gate, resulting in 10mW total power. Nevertheless, downsizing the four AND2 results in 8mW total power.

Considering the objectives of cell selection optimization, the reader can notice a very clear shift from timing/area optimization to power optimization in the literature. This is caused by the increasing challenge of power in modern technologies and designs for portable low power devices. Dynamic power is still dominant in some designs, particularly for process technologies 22nm and below with FinFETs. According to Chinnery and Keutzer (2005), more than 95% of the change in power can be calculated at that gate: switching power due to C_{in}; switching power of the load with V_{dd}; leakage power; and internal power. Slew changes cause small total power changes of typically less than 10%.

From a practical perspective, runtime is always a limiting factor when applying realistic and accurate delays models, as highlighted by Li et al. (2012): Also, it is known

Figure 4.22: Lagrangian relaxation convergence.



Source: Liu and Hu (2010)

that the continuous sizing model may work well for traditional transistor sizing, but it is not a good option when designing with macrocells and standard cells (CHAN, 1990).

However, with standard cell libraries, the delay model is not convex, eliminating the guarantee of finding the optimal solution. Also, the delay directly depends on slew propagation, implying the same effort of propagating arrival times. Those issues will be discussed in more detail in Chapter 5.

Comparing with many other techniques applied to this problem, Lagrangian relaxation algorithms have shown the best results in the recent literature (FLACH et al., 2013; FLACH et al., 2014). However, applying LR is not an easy task. Tennakoon and Sechen (2002) emphasize the difficulties of convergence in LR formulations for cell selection. In practice, it has been seen that the LR convergence guarantee when using subgratient is not easy either (BAZARAA; SHETTY, 1979; LORENA; SENNE, 1999). The subgradient optimization is very sensitive to the initial values for the multipliers and the step size (TENNAKOON; SECHEN, 2002).

More drawbacks of sub-gradient method are also presented by Huang, Hu and Shi (2011). Discrete cell sizing presents either drastic changes in slack, causing the ping-pong effect on solution, or very small changes, when LR needs to waste several iterations to cause a simple change in size/V_t.

Another know problem of applying LR is the convergence. Figure 4.22 shows an example of convergence that takes around 15 iterations to start reducing power. With the use of sign-off timing analysis, it is desirable to speed up convergence as much as possible.

All the aforementioned issues are still faced when solving the cell selection in modern real-life industrial designs. Algorithms to handle such issues and provide highly optimized solutions are needed in industry. The goal of this work is to provide a solid problem formulation to apply a state-of-the-art cell selection algorithm in real industrial designs, considering all the necessary quality metrics involved in a industrial design flow.

The publications Reimann et al. (2013), Flach et al. (2013), Flach et al. (2014), Reimann, Sze and Reis (2015), Reimann, Sze and Reis (2016), Reimann, Sze and Reis (2016) are part of this work and all algorithms, flows, discussions and results contained in them are presented in this work. Flach (2015) also presents parts of the work detailed in this thesis.

5 PROPOSED FLOWS AND TECHNIQUES

In this chapter we detail the algorithms proposed in this work. As previously in the text, the methods are presented in a chronological order for better comprehension of the timeline and the evolution of this work.

5.1 Simulated Annealing-based Algorithm

This is the first method developed for discrete gate sizing and V_t assignment in this work. This method was evaluated and compared with respect to the other contestant teams in the ISPD Contest 2012. Part of this work is already published in (REIMANN et al., 2013).

The proposed methodology is composed of a set of heuristic algorithms to address the cell selection problem for timing-constrained leakage power minimization while satisfying maximum load capacitance and maximum input slew constraints. The cell selection flow combines the Fanout-of-4 (FO4) empirical rule, the Logical Effort (LE) concept, a Simulated Annealing (SA) as the main optimization engine, as well as a new set of specific optimization strategies to solve the problem as formulated in the 2012 ISPD Gate Sizing Contest. No initial solution is provided. Therefore, not only power optimization but also timing closure must be achieved, what is a challenge by itself in designs with thigh constraints.

The main contribution of this work is to show how a sequence of Simulated Annealing runs, starting from a timing-infeasible solution improved by Logical Effort, Fanout-of-4 rule, and employing a set of new techniques can be used together to solve cell selection problems of up to a million gates.

A new dynamic cost function is used. It enables SA to deal with the conflicting objectives during the optimization. The entire flow was able to achieve the second and first ranks in the ISPD 2012 Contest. Here we present a set of different experiments to support design decisions and highlight the quality of the achieved results.

5.1.1 Logical Effort

The logical effort (LE) method is based on a simple model of the delay through a single logic gate (SUTHERLAND; SPROULL; HARRIS, 1999). Each different combinational function in the library has a different logical effort. The logical effort calculation is based on an inverter, which has a logical effort of 1. For a logic gate, it tells how much slower it will drive a load than would do a reference inverter, i.e., how much more input capacitance a gate must present in order to deliver the same output current as an inverter (SUTHERLAND; SPROULL; HARRIS, 1999). It is defined as the ratio of the input capacitance of a gate to the input capacitance of an inverter delivering the same output current.

$$LE = \frac{C_{in_{gate}}}{C_{in_{inv}}} \tag{5.1}$$

For example, if the inverter has three units of input capacitance while the NAND gate has four, the NAND gate has a logical effort of $LE = 4/3$.

5.1.2 Fanout-of-n Sizing

Fanout-of-4 (FO4) is a simple and efficient rule for delay-optimal gate sizing (SUTHERLAND; SPROULL; HARRIS, 1999; RABAEY; CHANDRAKASAN; NIKOLIC, 2002). It is based on the idea that an inverter can drive a load approximately four times larger than its input capacitance. This ratio closely relates to the best delay in an inverter chain that drives a large capacitance. The fanout ratio is given by the following equation:

$$Fanout = \frac{C_{load}}{C_{in}} \tag{5.2}$$

where, C_{load} is the load capacitance (including wire load in our case) and C_{in} is the input capacitance of the gate.

Considering the *fanout* of 4 rule, the input capacitance is given by:

$$C_{in} = \frac{C_{load}}{4} \tag{5.3}$$

In this work we show the impact of different values of "n" (fanout) on the SA convergence and solution quality. The initial solution finds which cell options have fanout ratio equal to or closer to "n". The cells are ordered according to their leakage power, so the algorithm will choose the cell option with smallest leakage that satisfies the fanout ratio rule. The cells are evaluated in reverse topological order, since the output loads are fixed.

If a cell is in a path with negative slack, only the faster cell options, i.e. options with smaller V_t , are considered, resulting in gates with more leakage power. Therefore, to calculate the fanout of each cell we combine the fanout ratio with LE as follows:

$$Fanout = \frac{C_{load}}{C_{in} * LE_{gate}} \tag{5.4}$$

where LE_{gate} is the logical effort of the gate as defined above.

This equation is used to produce the initial solution that is provided to the SA algorithm. Such an initial solution is considered to have a relatively good timing quality and few or none electrical violations. However it is not power-optimized.

5.1.3 Timing Engine

The most important aspect that need to be addressed to make SA scalable (and feasible for the expected runtime) is the process of timing updates after changes in size and V_t of gates. Since SA only performs a single cell change per iteration, the timing engine must do many updates during execution. In order to enable such a methodology, the timing analyser must be highly efficient. To achieve that, logical depths are pre-computed and contiguous data structures are used to enhance cache-obliviousness while executing timing updates, as detailed below.

The timing model defined in the ISPD 2012 Contest (OZDAL et al., 2012) is used for static timing analysis. This model propagates only the worst slews and worst arrival times, using NLDM with delay and slew lookup tables and lumped wire capacitance. Logical depths are useful as they define a proper order to process cells for static timing analysis. A cell with logical depth n should be processed before all cells with logical depth $n + 1$.With this, the arrival times can be updated without the need for an extra loop over the circuit.

Prior to the data structure construction, a pre-processing step is performed where the logical depth of each cell is computed. Logical depths are propagated from path drivers (a primary input or the output of a sequential element) to path sinks (primary outputs and/or data inputs of sequential elements). Primary inputs and sequential elements are defined as having a logical depth of zero. The logical depth of a combinational cell is set to the maximum logical depth among its driving cells plus one.

The netlist is modeled as a directed graph where edges represent the timing arcs and nodes represent circuit nets. The graph structure is stored in a way similar to the Compressed Row Storage (CRS) format (SILVA, 2005) used to store sparse matrices.

The necessary data is stored in three vectors as depicted by Figure 5.1:

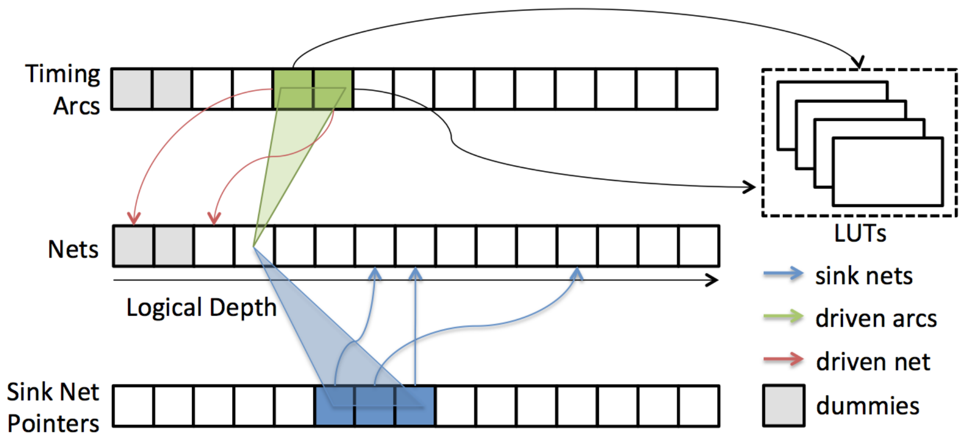
- **Vector of Timing Arcs** - stores slew and delay information of timing arcs computed directly from the library lookup tables, as well as the pointer to the respective lookup table;
- **Vector of Nets** - stores slew and delay information, arrival and required times, capacitive load, as well as pointers to the vector of Timing Arcs and to the vector of Sink Net Pointers. The net vector is sorted by increasing logical depth, with the logical depth of a net being equal to the logical depth of its driving cell;
- **Vector of Sink Net Pointers** - stores pointers to the Vector of Nets. These pointers are used to easily find sink nets being driven by a specific net.

Dummy nets and timing arcs are left at the beginning of vectors to avoid dealing with special cases when calculating timing for zero-logical depth nets. A dummy net is created for each primary input and another one represents the clock net. For each dummy net, a respective dummy timing arc is added. They are used to model the pre-defined primary input delays and account for special timing characteristics of sequential elements. This allows the timing engine to treat both cell types, sequential and combinational, uniformly.

Two modes for the static timing analysis are available in order to enable many timer calls: full timing analysis and incremental timing analysis.

The full timing analysis is used to update timing for the whole circuit. It is required during start up and whenever more than one cell is changed at once without the respective incremental timing update. As the timing data structure keeps nets organized by logical depth, the timing analysis can be performed by simply sweeping nets, without relying on any extra data structure such as a queue. Nets are swept from logical depth zero to the highest logical depth. This ensures that, when a cell is being updated, all required information have already been calculated. Furthermore, cells with the same logical depth may be computed in parallel.

Figure 5.1: Timer Data Structure



Source: from author (2016)

Incremental timing analysis is used to keep timing updated when a single cell is changed. Differently from full timing analysis, the incremental timing update relies on a queue data structure to update only the paths which are affected by the change.

When a cell size is changed, all cells in the fanout cones of its drivers must be updated. These fanout cones may have a large number of cells in common so that updating cells in a simple breadth-first manner may perform a lot of unnecessary work. This issue is simply solved by replacing the queue used in the breadth-first walking by a priority queue where cells with lower logical depth are processed first.

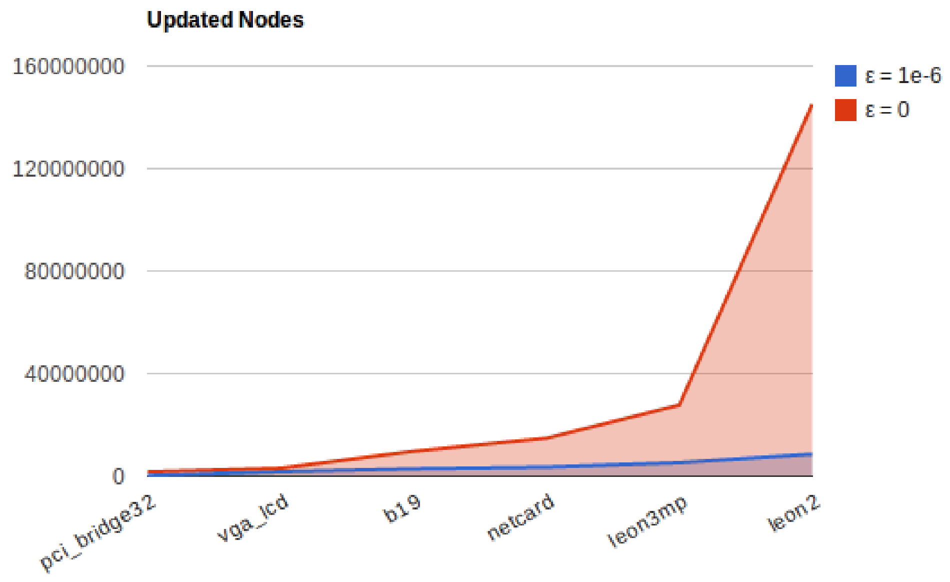
As the timing changes propagate through the netlist away from the changed cell, the timing variations may become less and less significant. Therefore, the proposed methodology uses a threshold limit (ε) for the timing variation that stops further propagation, saving runtime with no impact on the accuracy of final timing results. This threshold to stop propagation leads to up to 17X savings in node updates and an average 3X overall faster timing calculation when executing the proposed flow. Figure 5.2 shows the number of updated nodes for six benchmarks from the ISPD 2012 Contest using $\varepsilon = 0$ and $\varepsilon = 1\text{E-}6$.

5.1.4 Simulated Annealing with Dynamic Cost Function

It is well known that SA requires large runtime and typically does not scale well to large problem sizes. To compensate for this downside of the SA, heuristics combined with a reasonable temperature schedule are needed.

Some empirical tests showed that an appropriated schedule alone would not have a

Figure 5.2: Number of updated nodes for two different threshold values ε.



Source: from author (2016)

reasonable runtime and would also not respect the hard runtime limit set in the ISPD 2012 Contest. In the proposed flow, each SA iteration randomly selects a cell instance over the circuit and randomly chooses its new size. The new cost is then evaluated (incremental timing update) and the new solution is accepted or rejected, as usual in SA-based methods.

The main conflict in cell selection using SA is that the algorithm needs to accept violations to reduce leakage and keep the violations under control at the same time. However, a static cost function would penalize violations with a constant value during the entire annealing process. This approach is extremely inefficient due to single cell changes performed in SA. Changing only a single cell in each SA iteration may lead to violations (to reduce leakage power) that can only be solved in subsequent iterations (for example by reducing other cells, i.e., output load). A cost function that penalizes every violation in the same way would reject several changes that could be fixed later, preventing the algorithm from finding a good solution in terms of leakage, since it loses the “hill climbing” capability.

In this work, a dynamic cost function is proposed to allow a certain amount of violations during initial SA iterations, when temperature is still high, and reject violations when the temperature is reduced. This change in the cost function emulates a relaxation-like methodology.

The dynamic cost function used is defined in equation 5.5. The dynamic feature of the function is given by α and β , both dependent on the current temperature, as shown in equations 5.6 and 5.7.

$$cost = \alpha \times (timing_{viol} + slew_{viol}) + \beta \times load_{viol} + leakage_{tot} \tag{5.5}$$

$$\alpha = temp^{-1} \tag{5.6}$$

$$\beta = temp^{-2} \tag{5.7}$$

The original flow submitted to the ISPD 2012 Contest uses an initial solution with Fanout-of-n, where $n = 2$, but only the larger cells (with lower V_t) are considered to apply the fanout rule in paths with negative slack. Then a sequence of four SA loops iterate over the circuit to find the final power-optimized solution. The runtime limit for each SA loop is empirically defined as 10%, 30%, 50% and 100% of total maximum runtime. As the temperature decay actually gives iteration count, it must be directly related to the circuit cell count (slower decay for bigger circuits).

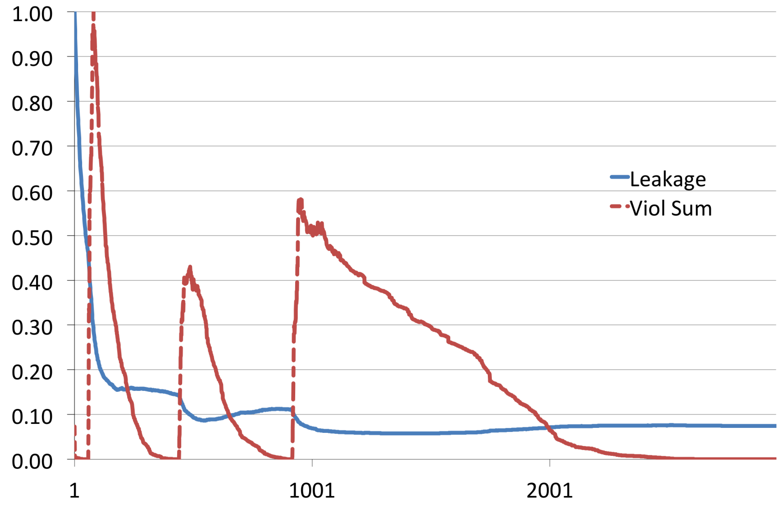
The first SA loop starts with a low temperature in order to generate a solution free from timing violations. The loop is not always executed since some circuits already have no timing violations at this point. Timing critical cells (belonging to paths with negative slack) have the priority when SA randomly chooses the cell to be changed in the design.

The next three loops perform like a typical SA algorithm. The goal of having three loops is to avoid locally optimal solutions, since a good temperature schedule would not be practical due to the runtime limit. With this approach, violations are allowed to increase three times during the SA execution.

Figure 5.3 shows how total leakage power and total violation behave along SA iterations (sampling). The three violations peaks are related to initial iterations of each SA loop, when the higher temperature allows violations to increase while decreasing power (leakage).

During all loops, a max-load violation control heuristic is used to keep this kind of violation as low as possible. This heuristic performs a greedy search and increases the size of a cell with load/slew violation on its output and/or reduces the sizes of its fanout cells (i.e. cells connected to its output). Although this violation control may interfere with timing violations, it avoids excessive searching during SA to solve max-load and

Figure 5.3: *pci_bridge32* total leakage power and total violation along SA iterations.



Source: from author (2016)

slew violations. Electrical violations are difficult to solve since the only way to do so is by changing the cell with the violation and/or its fanout cells would solve the violation. On the other hand, timing violations can be solved by changing any cell in the path that has the negative slack.

5.2 Empirical Validation

The methodology described herein was submitted to the ISPD 2012 Contest and was able to generate the best solution in 5 out of 14 benchmark circuits in the benchmark suite. Despite having the most consistent results over all circuits, the flow did not find violation-free solutions for two circuits: $b19\_fast$ and $leon3mp\_fast$.

Table 5.1 shows the ratios from each tool to best leakage result reported in the ISPD 2012 Contest. Considering only violation free solutions, the proposed flow presents the best average ratio between its solutions and the best solution found. It also presents the lowest maximum deviation (i.e. 52%) and at least half the average deviation of the other tools submitted to the contest.

Additional experiments were performed to determine the most suitable value for n and to evaluate different approaches in SA loops and also omitting the initial solution. All experiments have the same original flow submitted to the ISPD 2012 Contest, with only a few corrections in the code.

The fanout-of-n is an empirical rule, and can be affected by aspects such as technology parameters, or how large the actual capacitances are in a design. The flow submit-

Table 5.1: Leakage power ratio to best solution found for all ISPD'12 circuits.

Benchmark	NTUgs	Ours	PowerValve	Goldilocks	eOPT	CUsizer
DMA_slow	1.39	1.07	1.00	1.46	3.07	2.50
DMA_fast	1.64	1.04	1.00	2.20	2.75	1.57
pci_bridge32_slow	1.77	1.00	1.01	6.05	1.97	2.50
pci_bridge32_fast	3.05	1.00	1.35	5.64	2.43	2.02
des_perf_slow	1.00	1.31	1.03	1.41	3.38	1.68
des_perf_fast	1.03	1.52	1.00	4.23	2.53	1.05
vga_lcd_slow	1.10	1.00	1.03	1.22	1.70	1.99
vga_lcd_fast	1.31	1.00	1.33	-	1.32	1.48
b19_slow	1.02	1.00	1.20	1.23	1.40	8.18
b19_fast	2.61	-	4.32	1.71	1.82	-
leon3mp_slow	1.00	1.26	2.08	1.04	1.32	1.35
leon3mp_fast	-	-	2.45	1.00	1.20	1.03
netcard_slow	1.00	1.11	1.10	1.02	1.19	1.13
netcard_fast	1.00	1.14	1.48	1.02	1.41	1.22
Avg.[a]	1.45	**1.12**	1.53	2.25	1.96	2.13
Avg.[b]	1.36	**1.13**	1.21	2.41	2.11	2.29

[a] Ignoring results with violation.
[b] Ignoring Vga_lcd_fast, leon3mp_fast and b19_fast for all tools.

ted to the contest used fanout-of-2 (*4SA/FO2-LE*), as it was empirically determined at first that it produced the best results in some test circuits. New experiments were performed with $n = 3$ and $n = 4$ (hereafter called *4SA/FO3-LE* and *4SA/FO4-LE*, respectively) to finally check which generates a better initial solution for SA optimization. Choosing a lower value of n means that the cells will be bigger and faster (for the same V_t level). That helps the critical paths to have no timing violations but also increases total leakage power. The better is the power-timing trade-off provided by the initial solution, the better will SA perform.

The results for all these experiments are presented in Table 5.2 and Table 5.3.

The first column of Table 5.2 reports the proposed flow submitted to the ISPD 2012 Contest (*4SA/FO2-LE*). Column *4SA/FO3-LE* shows some better solutions compared to *4SA/FO2-LE* but in average it shows inferior results. It also results in more solutions with timing violations when compared to the other method.

The *4SA/FO4-LE* method mostly generates lower leakage solutions than both other methods. This shows that $n = 4$ has the most balanced trade-off between timing and leakage power. However, this configuration also presents less violation-free solutions when compared to *4SA/FO2-LE*.

Table 5.2: Total violation and total leakage power results with different Fanout-of-n rules and different flows using one or four SA loops and alternating the use of an initial solution.

Benchmark	4SA/FO2-LE		4SA/FO3-LE		4SA/FO4-LE		1SA/FO2-LE	
	Total Viol.	Leakage (W)	Total Viol.	Leakage (%)	Total Viol.	Leakage (%)	Total Viol.	Leakage (%)
DMA_slow	-	0.157	-	1.1	-	-0.8	-	0.6
DMA_fast	-	0.341	-	28.4	-	-3.4	-	-12.5
pci_bridge32_slow	-	0.119	-	-1.0	-	-4.6	-	-4.0
pci_bridge32_fast	-	0.176	-	31.7	-	-5.1	-	-2.4
des_perf_slow	-	0.694	-	-2.6	-	-5.0	5.21E0	6.3
des_perf_fast	-	3.40	-	-20.2	-	-25.6	1.04E5	-50.4
vga_lcd_slow	-	0.395	9.68E2	-1.3	4.45E2	-0.7	3.13E0	3.1
vga_lcd_fast	3.27E3	0.545	5.05E3	-0.6	3.06E3	-3.6	2.37E3	4.5
b19_slow	7.90E2	0.646	3.19E2	-0.7	3.04E3	-5.7	3.77E3	-6.7
b19_fast	1.37E4	0.713	1.25E4	-1.2	1.52E4	-4.5	1.40E4	0.1
leon3mp_slow	2.24E6	2.40	1.52E6	27.8	3.73E3	7.9	9.49E3	31.0
leon3mp_fast	8.21E3	2.58	1.42E5	27.6	6.39E6	37.0	2.68E4	94.5
netcard_slow	2.62E3	2.87	7.73E2	5.6	3.96E2	0.9	7.64E3	12.0
netcard_fast	5.84E3	3.51	7.11E3	-1.2	1.04E5	0.3	8.24E3	4.1
#viol. / avg.	7	-	8	6.7	8	-0.9	10	5.7
Runtime Ratio	1.00		1.08		1.02		1.09	

1SA/FO2-LE represents the flow with a single SA run. The *1SA/FO2-LE* flow is not able to generate as many violation-free solutions as the original flow. This single loop configuration has a slower temperature decay in order to have a similar number of SA iterations and runtime. The difficulty to solve the violations created with high temperature increases the runtime to generate the final solution.

The results in Table 5.3 show how simulated annealing struggles to find violation-free solutions by itself when starting with an arbitrary solution. For such experiments, the initial solution provided to SA is the smallest leakage option for all cells in the design. It is clear that the flow with four fast SA runs – *4SA* – is more effective than a single SA loop – *1SA*. However, both algorithms cannot converge to good solutions without timing violations. *4SA* generates only three violation-free solutions and *1SA* a single one.

4SA and *1SA* are the two flows without the initial heuristic solution. *4SA* keeps the original flow but ignores the sizing based on fanout and logical effort. The last configuration has only one SA loop and no initial sizing with fanout-of-n and logical effort (*1SA*). Table 5.3 presents the results of the proposed configuration that employs no initial solution and four separate SA runs, by comparing it to a single SA algorithm, also without

Table 5.3: Total violation and total leakage power results with different flows using one or four SA loops without an initial solution.

Benchmark	4SA		1SA	
	Total Viol.	Leakage (%)	Total Viol.	Leakage (%)
DMA_slow	-	4.1	-	13.0
DMA_fast	-	4.5	5.21E2	20.7
pci_bridge32_slow	5.14E3	37.3	1.59E3	46.7
pci_bridge32_fast	1.36E2	17.1	2.62E4	40.1
des_perf_slow	-	-1.5	7.70E2	27.5
des_perf_fast	7.63E4	2.9	3.36E5	-58.4
vga_lcd_slow	4.63E3	-3.2	5.17E3	-3.6
vga_lcd_fast	9.38E3	-8.1	1.00E4	-12.9
b19_slow	1.44E4	-4.3	1.63E4	-2.6
b19_fast	1.42E4	8.6	2.06E4	5.9
leon3mp_slow	1.63E6	5.7	1.36E6	29.7
leon3mp_fast	1.61E6	38.2	2.29E7	103.9
netcard_slow	8.10E2	-18.8	6.24E3	-20.5
netcard_fast	3.17E7	53.4	1.15E7	151.7
#viol. / avg.	11	9.7	13	24.4
Runtime Ratio	1.05		1.10	

the initial heuristic solution.

Results show that the flows without initial solution (*4SA* and *1SA*) completely fail to generate a violation free solution for almost all circuits. It becomes clear that the process of solving violations is very hard for the Simulated Annealing even with the proposed heuristics. The single cell change approach would require a large amount of iterations to find the right combination of cells without any heuristic solution.

All runtimes for the experiments described are reported in Table 5.4. These experiments were performed on a machine with two AMD(R) Opteron(R) @ 2.3GHz CPUs[1].

[1]This machine is slower than the one used to run the tools in the ISPD 2012 Contest. Therefore, the quality of results presented in Tables 5.2 and 5.3 are considerably inferior than those reported in the ISPD 2012 Contest.

Table 5.4: Runtime in minutes for the different flows under test.

Benchmark	4SA/FO2-LE	4SA/FO3-LE	4SA/FO4-LE	1SA/FO2-LE	4SA	1SA
DMA_slow	81	80	94	139	99	136
DMA_fast	73	352	192	159	183	360
pci_bridge32_slow	32	53	53	352	106	311
pci_bridge32_fast	69	351	72	352	100	237
des_perf_slow	293	514	324	479	540	541
des_perf_fast	529	529	529	529	543	540
vga_lcd_slow	600	600	600	600	600	600
vga_lcd_fast	600	600	600	600	600	600
b19_slow	720	720	720	720	720	720
b19_fast	720	720	720	720	720	720
leon3mp_slow	1440	1440	1440	1440	1440	1440
leon3mp_fast	1440	1440	1440	1440	1440	1440
netcard_slow	1980	1980	1980	1980	1980	1980
netcard_fast	1980	1980	1980	1980	1980	1980
Ratio	1.00	1.08	1.02	1.09	1.05	1.10

5.3 Lagrangian Relaxation-based Algorithm

The second methodology developed uses Lagrangian relaxation as the main engine for timing-constrained power optimization. This approach is applied in two different situations: the ISPD Contest Benchmark Suites; and inside an industrial design flow for high-performance microprocessor blocks. Both applications share the same core methodologies but differ considerably in several aspects that are detailed in the next sections.

First, we describe the general formulation for the Lagrangian relaxation applied to the cell selection problem. We follow the basic formulation found in (CHEN; CHU; WONG, 1999) for the continuous gate sizing problem, as described in Section 4.1.6. Table 5.5 shows the notation used in this section.

The timing-constrained power[2] optimization problem, here called the Primal Prob-

[2]Here we use power as an example of the optimization objective to present the general formulation. As we show later in this work, other objectives like area can be in the optimization objective.

Table 5.5: Notation.

T	clock period
TNS	total negative slack
$i \rightarrow j$	timing arc from node i to node j
$d_{i\rightarrow j}$	delay of timing arc $i \rightarrow j$
a_i	arrival time at node i
q_i	require time at node i
λ	Lagrangian multiplier

lem (PP), can be defined as follows:

Primal Problem (PP):

$$\begin{aligned} \textbf{minimize} \quad & \sum_i power_i \\ \textbf{subject to} \quad & a_i + d_{i\rightarrow j} \leq a_j, \text{ for each timing arc } i \rightarrow j \\ & a_k \leq T, \text{ for each path output node } k \end{aligned} \tag{5.8}$$

By applying the Lagrangian Relaxation technique we can bring the constraints inside the minimization function by the use of Lagrange multipliers. Thus, we obtain the Lagrangian Relaxation Subproblem LRS shown in Equation (5.9).

LRS:

$$\begin{aligned} \textbf{minimize} \quad & \sum_i power_i + \\ & \sum \lambda_{i\rightarrow j}(a_i + d_{i\rightarrow j} - a_j) + \\ & \sum \lambda_k (a_k - T) \end{aligned} \tag{5.9}$$

Chen, Chu and Wong (1999) show that by applying the Karush–Kuhn–Tucker (KKT) conditions to optimality, the problem in Equation (5.9) can be simplified, as shown in Equation (5.10). This simplification makes the minimization objective dependent only on the delays of timing arcs, reducing the number of calculations required from the timing engine.

LRS (simplified):

$$\textbf{minimize} \quad \sum_i power_i + \times \sum \lambda'_{i\rightarrow j} d_{i\rightarrow j} \tag{5.10}$$

Hereafter, the sum $\sum \lambda_{i\rightarrow j} d_{i\rightarrow j}$ is referred as *lambda-delay*.

The relaxed version of the sizing problem can be viewed as the selection of gate versions which minimizes the objective plus lambda-delay with no explicit information about arrival and/or required times. Finally, LDP is simply the maximization of LRS where λ is also variable as shown in Equation (5.11).

$$\text{LDP:} \quad \underset{\lambda}{\textbf{maximize}} \quad \left(\sum_i power_i + \times \sum \lambda'_{i \to j} d_{i \to j} \right) \tag{5.11}$$

Following this formulation we propose new algorithms to handle the cell selection problem in modern designs.

The new algorithms are developed to handle the set of benchmarks from the ISPD 2013 and 2012 Discrete Gate Sizing Contests. In both cases, no initial solution is provided with the benchmarks. Also, this work later presents an extension of this formulation to adapt the algorithms to optimize real-life industrial high-performance designs in an industrial design flow.

In the next sections and following chapter we present the proposed flows that have LR as the core optimization technique. The empirical results show the effectiveness of the proposed frameworks.

5.3.1 Proposed Flow for the ISPD 2012 and 2013 Contest Benchmarks

The methodology here presented is partially published in (FLACH et al., 2013). This work has received the best paper award in the 2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2013). The extension of the methodology to handle the new set of benchmarks from the ISPD 2013 Contest is partially published in (FLACH et al., 2014).

This is a fast and efficient approach to select the gates in a design producing state-of-the-art results. The gate sizing tool was first built based on the infrastructure provided in the ISPD 2012 Contest Benchmark Suite (OZDAL et al., 2012). It was later adapted to handle the modifications introduced in the ISPD 2013 Contest.

As mentioned in Section 4.2.1, the contes benchmarks consist of a modern standard-cell library, including 11 different combinational functions and 1 flip flop, and designs ranging from 23k to 861k cells, For each combinational logic function, 30 different cell types are available (composed by 3 threshold voltages and 10 sizes for each V_t). The benchmarks contain a Verilog netlist, a SDC (Synopsys Design Constraints) file with the timing constraints and interconnect parasitics in IEEE SPEF format. The objective is to

Table 5.6: Notation.

$slew_i$	slew at node i
$\Delta D_{i \to j}$	delay change given a change in input slew of arc $i \to j$
ΔD_n	delay change given a change in slew of net n
$\frac{\delta d_{i \to j}}{\delta slew_i}$	delay sensitivity to input slew of arc $i \to j$
$\frac{\delta slew_j}{\delta slew_i}$	output slew sensitivity to input slew of arc $i \to j$
ϕ	cumulative back-propagated arc delay sensitivity

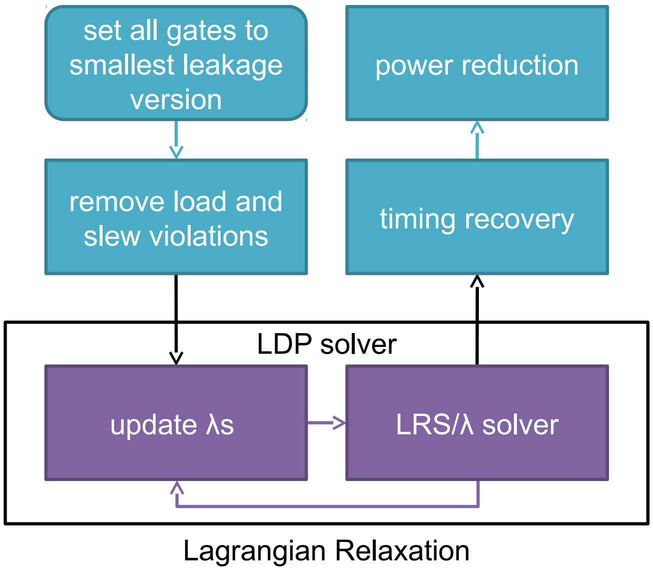
reduce the leakage power while all performance and design constraints are satisfied.

The highlights of the methodology here presented are:

- A fast, effective, and simple methodology to solve the discrete gate sizing problem, that employs a selected set of techniques in a much improved way. The core step models the gate selection problem using Lagrangian relaxation. The relaxed problem is solved by a greedy one-gate-at-time algorithm applying only local modifications while relying on both precise local timing information and fast global timing estimation.
- A fast technique based on sensitivities to estimate the global delay impact of a gate sizing without re-running a complete incremental timing analysis.
- Best results for all ISPD 2012 benchmarks compared to previous state-of-the-art works, with leakage power improvements of up to 27.73%. On average, the our flow reduces leakage power by 9.53% compared with (HU et al., 2012) and by 12.45% compared with (LI et al., 2012). Moreover, the method is, on average, 19X faster than (HU et al., 2012) and 1.18X faster than (LI et al., 2012).
- Best results for all ISPD 2013 benchmarks compared with all contestants (including our previous flow submitted to the contest). Our new flow provides on average 9.62% power reduction compared to the best contestant's results, with up to 30% power reduction in one circuit. It is also the first gate sizing method to report violation-free solutions for all benchmarks of the ISPD 2013 Contest.

In this section, a circuit is described by its logical and memory elements called *gates* and the connections between them called *nets*. Only combinational gates are considered for sizing. Gates are attached to nets at specific points of the net topology named driver and sink *nodes*. A gate is composed by one or more *timing arcs* which describe the timing characteristics of the gate. Table 5.6 summarizes the additional notation for this section.

Figure 5.4: High-level view of our cell selection flow.



Source: Flach (2015)

When connections are modeled using a simple lumped capacitance as in the ISPD 2012 Contest, timing information (arrival, required and slew) at sink nodes is equal to the values at the driver node. In this case, driver and sinks share the same name borrowed from the net which connects them.

To keep the notation simple, rise and fall edges are not represented separately throughout this paper, but the proposed method is developed for considering them separately for timing accuracy.

Figure 5.4 depicts a high-level view of the flow developed in this work. As described in Section 4.2.1, the method developed for the ISPD Contest starts with a timing-infeasible solution. By definition, the proposed methodology will ignore the input solution (since it is not by any means useful), setting all gates to the smallest option available with the highest V_t. Then, a solution without electrical violations is generated by applying the initial sizing algorithm from (LI et al., 2012). This solution is the input to the Lagrangian relaxation method, which is constrained to keep the solution free of electrical violations. Next, any timing violations left are eliminated by a Timing Recovery method. Finally the leakage power is further reduced by a Power Reduction algorithm.

Every method developed in this work has a linear complexity vs. circuit size per iteration. The number of iterations for each iterative method depends on several aspects related to the quality of the solution provided as the input of each method. It is hard to predict the final number of iterations and runtime, but for most cases only a few (tens to hundred) are required.

5.3.2 Eliminating Load and Slew Violations

As mentioned before, load and slew violations have to be removed from the initial minimum leakage solution, which is also needed in order to avoid extrapolations when computing timing from the library lookup tables. Without such an approach, the LR method would be misguided by unrealistic timing information. Since LR convergence highly depends on timing information in each iteration, all kinds of electrical violations must be avoided.

The removal is performed by means of the iterative method presented in (LI et al., 2012) (see Figure 4.14a), with $\alpha = 0.7$. The procedure visits all gates, one at a time, from outputs (where output capacitances are fixed) to the inputs. For each gate, the gate version with least leakage that respects both slew and load constraints is selected. In our implementation, as the gates are visited, timing is updated only locally as explained in detail in this section.

The $\alpha_L \in (0, 1]$ parameter is used to control the ratio between driver strength and the load capacitance. The smaller is the α_L value, the larger should be the driver strength, but also its input capacitance. In all results obtained in this work we use $\alpha_L = 0.7$.

5.3.3 Cell Selection Problem Formulation

This section details timing-constrained leakage power minimization. However, more objectives can also be optimized as explained in Chapter 6.

A concise way to represent timing constraints is by imposing constraints on the arrival times. In this case, arrival times are seen as dependent variables by the optimization problem, allowing a linear number of constraints with circuit size instead of path-based constraints (CHEN; CHU; WONG, 1999).

There are several methods in the literature to solve the Lagrange Dual Problem (LDP). Most share very similar steps when applied to the gate sizing problem. Algorithm 1 presents an overview of the iterative method used to look for a competitive solution to the LDP. The idea of the LDP solver is straightforward. It sequentially combines two algorithms: (1) greedy cell selection and (2) Lagrange multiplier update. The second algorithm requires a previous timing analysis to obtain the updated timing information after the changes made by the first algorithm.

Using the current solution, the Lagrange multipliers are updated to reflect how

Algorithm 1: LDP Solver

```
1  store initial solution
2  set an initial value for $\lambda$s
3  update timing (STA)
4  update $\lambda$s // Alg. 2
5  repeat
6      solve $LRS/\lambda$ // Alg. 4
7      update timing (STA)
8      update $\lambda$s // Alg. 2
9      if new solution is better than stored one then
10         store solution
11     end
12 until convergence;
13 restore best solution found
```

much a constraint, now incorporated to the objective function, is being violated. This generates a new instance of LRS/λ which is then solved again by the greedy cell selection method. A cost function is used to evaluate the best LR solution:

$$cost = totalLeakage + TNS * (\alpha + WNS)^{\beta} \tag{5.12}$$

Different from the simplified convex continuous sizing problem (CHEN; CHU; WONG, 1999), the sub-gradient method does not guarantee the optimality of the LDP maximization with an accurate, i.e. non-convex, library of discrete gate sizes. Considering that, other methods for the λ update process are presented in the literature (TENNAKOON; SECHEN, 2002; LI et al., 2012). Here we also present a new methodology for Lagrangian multiplier update that leads to direct improvement in convergence and quality of the final solution.

The initial multiplier value is set to a constant value for all timing arcs. This value is scaled by the KKT propagation performed in the Lagrange multiplier update method.

The empirically chosen initial multiplier value used for the experiments is 12. A solution is said to be better than another one if its TNS is less than 10% of T and it has smaller leakage.

The updating of Lagrange Multipliers is accomplished in two steps: (1) slack scaling and (2) KKT projection. Algorithm 2 presents the method used to update them.

Initially Lagrange Multipliers are scaled according to the slack of the respective timing arcs. The idea is to increase proportionally the importance (λ value) for timing arcs with negative slack and to decrease the importance for those with positive slack. The

Algorithm 2: Updating Lagrange Multipliers

```
1 for each timing arc $i \rightarrow j$ do
2     $$\lambda_{i\to j} \leftarrow \lambda_{i\to j} \times \begin{cases} \left(1 + \frac{a_j - q_j}{T}\right)^{+1/k} & a_j \geq q_j \\ \left(1 + \frac{q_j - a_j}{T}\right)^{-k} & a_j < q_j \end{cases}$$
3 end
4 KKT projection
```

Algorithm 3: Update Lambdas KKT (CHEN; CHU; WONG, 1999)

```
1  for each net $n$ of the circuit do
2      for each edge (rise and fall) do
3          if sum of driver timing arc lambdas $> 0$ then
4              for each driver timing arc do
5                  $\lambda_{arc} = sum\_sink\_lambdas * \lambda_{arc} / sum\lambda_{drivers})$
6              end
7          end
8          else
9              for each driver timing arc do
10                 $\lambda_{arc} = sum\lambda_{sinks} / num\_sinks$
11             end
12         end
13     end
14 end
```

higher is the λ value the higher is the impact of the respective timing arc delay in the objective function.

KKT projection is performed to ensure that multipliers obey the KKT conditions for optimality. These conditions imply that the sum of multipliers driving a net must be equal to the sum of multipliers being driven by that net (CHEN; CHU; WONG, 1999).

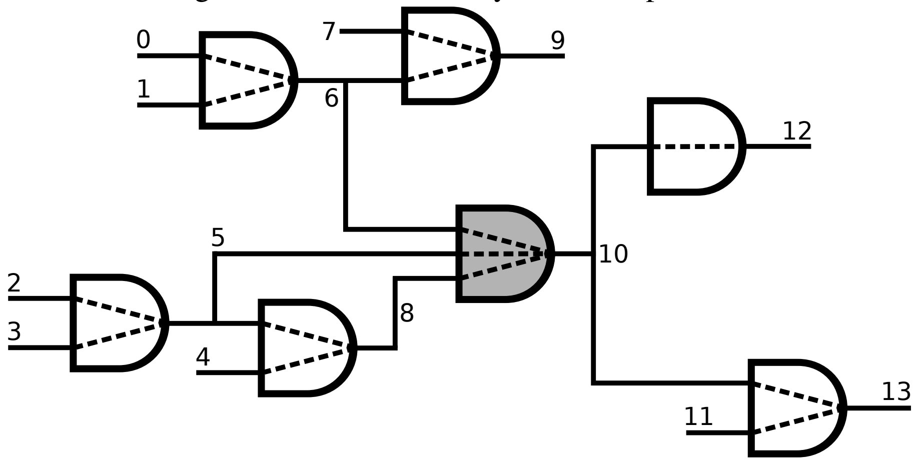
In our flow, the projection is performed by traversing the circuit in reverse topological order distributing proportionally the sum of multipliers being driven by a net to the ones driving the net. The proportion is defined by the updated multiplier value of driving timing arcs. Algorithm 3 shows the pseudo code for KKT projection.

In order to find a solution for each LRS/λ instance, differently from (OZDAL; BURNS; HU, 2011), which uses dynamic programming for multi-gate-at-a-time changes, the one-gate-at-a-time greedy method presented in Algorithm 4 is employed. It works by scanning all gates in topological order, trying to properly select a new version to each of them. The new selected version is the one which locally minimizes leakage power plus the lambda-delay cost.

Algorithm 4: LRS/λ Solver

```
1 compute lambda-delay sensitivities // Eq. 5.19
2 for each gate $g$ in topological order do
3     select a gate version for $g$ // Alg. 5
4 end
```

Figure 5.5: Lambda-Delay Cost Computation



Source: Flach (2015)

As the impact on lambda-delay requires an incremental STA to be performed, it would be infeasible to use updated information every time a gate option is analyzed. Therefore, the greedy method relies on local timing information and global estimation to approximate the global impact on lambda-delay of affected gates.

In most cases, the timing impact of a gate resize is absorbed within a few logic levels. Therefore the global impact on circuit timing can be estimated considering only local information. However, to deal with the cases where a local change greatly affects the overall timing – i.e. a change in a gate belonging to a path highly sensitive to slew changes – a global sensitivity-based lambda-delay function is developed below.

The objective of the LRS problem is the minimization of the lambda-delay plus leakage power. Since a gate resize may affect several other delays, how the total lambda-delay is affected needs to be taken into account together with the leakage change. The lambda-delay cost of a gate option indicates how the gate version impacts on lambda-delay. This is not exact and is computed mostly relying on local information, but the sensitivity-based global estimation helps the calculation of such impact.

The lambda-delay cost for the current version of a gate g (e.g. darker gate in

Figure 5.5), $lambdaDelayCost(g)$, is shown in Equation 5.13.

$$\begin{aligned} &lambdaDelayCost(g) = \\ &\sum_{i\to j \in driverArcs(g) \cup gateArcs(g) \cup sinkArcs(g)} \lambda_{i\to j} d_{i\to j} + \\ &\sum_{i\to j \in sideArcs(g)} \Delta D^{\lambda}_{i\to j} + \sum_{n \in drainNets(g)} \Delta D^{\lambda}_{n} \end{aligned} \tag{5.13}$$

where

- $driverArcs(g)$ is the set of arcs driving the driver nets of gate g (e.g. arcs $\{0,1\} \to 6$, $\{2,3\} \to 5$ and $\{5,4\} \to 8$ in Figure 5.5);
- $sinkArcs(g)$ is the set of arcs which are driven by gate g (e.g. arcs $10 \to \{12,13\}$ in Figure 5.5);
- $sideArcs(g)$ is the set of arcs which are driven by gate g's driver nets but which do not belong to g itself (e.g. arcs $6 \to 9$ and $5 \to 8$ in Figure 5.5);
- $gateArcs(g)$ is the set of arcs which belong to gate g (e.g. arcs $\{6,5,8\} \to 10$ in Figure 5.5);
- $drainNets(g)$ is the set of nets driven by sink gates of g (e.g. nets 12 and 13 in Figure 5.5).

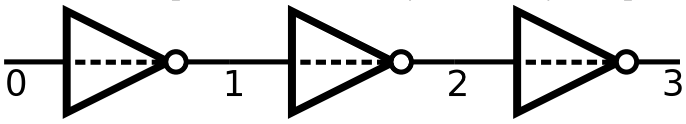
The timing arc sensitivity measures how the delay/slew of an arc changes given a change on its context (i.e. input slew, output load). It linearly approximates the delay/slew from the lookup table at around the current context. Sensitivities are combined and propagated back from path outputs to inputs. This enables the use a single operation to approximate the effect of a local change in the timing of the whole fanout cone.

The cumulative sensitivity of an arc estimates how the total delay of the fanout cone changes given a change on its input slew. For sensitivity computation, a lumped capacitance interconnection model is assumed.

Next follows an example for a better understanding of this process. Consider the simple inverter chain example in Figure 5.6.The Lagrange multipliers λs are omitted to facilitate the comprehension. They are easily accounted for just by multiplying each arc delay sensitivity to input slew by its respective multiplier.

The delay change due to an input slew change of timing arc $2 \to 3$, is simply the

Figure 5.6: Example circuit for delay sensitivity computation.



Source: Flach (2015)

timing arc sensitivity itself times the input slew change, as in Equation 5.14.

$$\Delta D_{2\rightarrow 3} = \Delta slew_2 \frac{\delta d_{2\rightarrow 3}}{\delta slew_2} \tag{5.14}$$

Similarly, for timing arc $1 \rightarrow 2$, the delay change is the input slew change times timing arc delay sensitivity plus the delay change for timing arc $2 \rightarrow 3$ as in Equation 5.15.

$$\Delta D_{1\rightarrow 2} = \Delta slew_1 \frac{\delta d_{1\rightarrow 2}}{\delta slew_1} + \Delta D_{2\rightarrow 3} \tag{5.15}$$

Combining (5.14) and (5.15), and noting that Equation 5.16 holds

$$\Delta slew_2 \approx \Delta slew_1 \frac{\delta slew_2}{\delta slew_1} \tag{5.16}$$

we end up with the Equation 5.17 that depends on only one unknown, $\Delta slew_1$.

$$\Delta D_{1\rightarrow 2} = \Delta slew_1 \left(\frac{\delta d_{1\rightarrow 2}}{\delta slew_1} + \frac{\delta slew_2}{\delta slew_1} \frac{\delta d_{2\rightarrow 3}}{\delta slew_2} \right) \tag{5.17}$$

Finally, the delay change for timing arc $0 \rightarrow 1$ is shown in Equation (5.18).

$$\begin{aligned} \Delta D_{0\rightarrow 1} = \Delta slew_0 \Bigg[\frac{\delta d_{0\rightarrow 1}}{\delta slew_0} + \frac{\delta slew_1}{\delta slew_0} \Bigg(\frac{\delta d_{1\rightarrow 2}}{\delta slew_1} \\ + \frac{\delta slew_2}{\delta slew_1} \frac{\delta d_{2\rightarrow 3}}{\delta slew_2} \Bigg) \Bigg] \end{aligned} \tag{5.18}$$

Note that such propagation could continue on for as many levels as necessary till reaching the path input. Note also that $D_{0\rightarrow 1}$ provides the whole path delay change due to a change in the slew at net 0.

In general terms, the back-propagate lambda-delay sensitivity of a timing arc $i \rightarrow j$ is defined by the recurrence Equation 5.19 for every timing arc $i' \rightarrow j'$ driven by arc $i \rightarrow j$. Note that, differently from the aforementioned example, the Lagrange multiplier

λ associate to the timing arc is now being shown.

$$\phi_{i \to j} = \lambda_{i \to j} \frac{\delta d_{i \to j}}{\delta slew_i} + \frac{\delta slew_j}{\delta slew_i} \begin{cases} \sum \phi_{i' \to j'} & \text{dominant arc} \\ 0 & \text{otherwise} \end{cases} \tag{5.19}$$

In order to handle multiple fanout nets and to avoid counting multiple times the delay change, only the arc with the worst slew rate – that is propagated to the output, here called the dominant arc – propagates back the cumulative sensitivity. The remaining arcs see the cumulative sensitivity as zero since they are likely dominated by the arc with worst slew and hence, they should not affect the timing of gates ahead.

The delay change of timing arc $i \to j$ is then calculated as in Equation 5.20.

$$\Delta D^{\lambda}_{i \to j} = \Delta slew_i \phi_{i \to j} \tag{5.20}$$

For a net, the delay change is equal to the sum of the delay changes of all timing arcs driven by it, as shown in Equation 5.21.

$$\Delta D^{\lambda}_{n} = \Delta slew_n \sum \phi_{i \to j} \tag{5.21}$$

Algorithm 5 presents the method that selects a new gate version. In this method, all options available in the library are tested.

To improve overall convergence and guide our flow to a good solution, there are two conditions that a gate option should obey to be qualified to replace the current option: (1) not increasing any load violation and (2) not impacting too much the local negative slack as we explain below.

In order to replace the current option, the new one must not increase electrical violations (line 6). For the library used in the contest, slew violations only exist at nodes with load violations. Thus, preventing load violations also prevents slew violations at that node. As our flow starts with a solution with no load violations, this implies that no load violations will be ever generated.

Most load violations lead to slew violations which are harder to keep track of because they may be generated at many logic levels after the perturbation. They may also be propagated throughout the circuit. Prohibiting the increase of load violations avoids the method from wandering through solutions with lots of slew violations, which are difficult to recover back and cause lookup table extrapolations. As mentioned before, extrapolations may generate exaggerated delay/slew values that affect the overall convergence of

Algorithm 5: Gate Option Selection

```
1  $originalSlack \leftarrow computeLocalNegativeSlack(g)$
2  $bestCandidate \leftarrow version(g)$
3  $bestCost \leftarrow lambdaDelayCost(g) + leakage(g)$
4  **foreach** *gate option* $t \in options(g)$ **do**
5      $option(g) \leftarrow t$
6      **if** *load violation has increased* **then**
7          go to the next version
8      **end**
9
10     update timing locally
11
12     $slack \leftarrow computeLocalNegativeSlack(g)$
13     **if** $slack < \gamma * originalSlack$ **then**
14         go to the next version
15     **end**
16
17     $cost \leftarrow lambdaDelayCost(g) + leakage(g)$
18     **if** $cost < bestCost$ **then**
19         $bestCandidate \leftarrow t$
20         $bestCost \leftarrow cost$
21     **end**
22 **end**
23 $option(g) \leftarrow bestCandidate$
24 update timing locally
```

the flow.

Similarly to slew violations, timing violations generated due to a single perturbation may spread out to all logic levels until reaching a timing endpoint. This is more apparent when critical paths are passing through the vicinity of the current gate.

In order to keep TNS under control, a gate option must not increase the local negative slack above a certain threshold (empirically defined). This control is performed in line 13.

As keeping track of the actual slack would require running a complete incremental STA, the solution proposed in our method looks only at the slack perturbation in the vicinity of the current gate, so as to minimize runtime. This slack calculation uses the required arrival time from the previous timing update. On the other hand, arrival times used are up-to-date since they are calculated in local timing updates.

Local negative slack is defined simply as the sum of negative slacks (positive slacks are not included) of the driver nets and the sink net of the current gate.

To allow some sort of "hill climbing" like in stochastic methods, the local negative slack is allowed to increase a small amount controlled by the parameter γ as defined in Equation 5.22. The idea is to allow larger changes at the first iterations when the timing violations are likely to be high and to avoid them as the method converges to a low timing violation solution. Not allowing any sort of local TNS degradation restricts too much the search space, reducing the leakage power optimization.

$$\gamma = (-(min(0, worstSlack))/T + 1) \tag{5.22}$$

The local negative slack constraint indirectly controls the trade-off between leakage and lambda-delay in the objective function. It avoids choosing an option which reduces locally the objective function but is likely to cause a large impact on timing violation.

5.3.4 Interconnection Modeling

Another important aspect for designs with realistic wire models (i.e. not the lumped wire load model in ISPD 2012 Contest) is how delay and slew propagation are calculated. The interconnection modeling used in the internal timing analysis engine is based on the Elmore delay (GUPTA et al., 1995). It is fast enough to be used several times during the optimization process. However, it is still slower than the lookup table gate timing propagation ($\sim$10X). The flow submitted to the ISPD 2013 Contest only relies on the method described in (PURI; KUNG; DRUMM, 2002).

It starts by computing the effective capacitance for the driver node of the net using the method presented in (QIAN; PULLELA; PILLAGE, 2006). The effective capacitance is then used to obtain delay and slew information from the lookup table. Next, delay and slew are propagated in topological order to the sink nodes of the net. Algorithm 6 presents the interconnection timing calculation.

However, this modeling presents several inaccuracies when compared to commercial timing analysis tools that employ reduced order models described in Section 2.2. More details about the effect of this inaccuracy are presented ahead in the text.

Algorithm 6: RC Interconnection Model

```
1 compute effective capacitance for driver node
2 obtain delay and slew from lookup table for driver node using effective
    capacitance
3 foreach node $n$ ($\neq$ driver) in topological order do
4     $R \leftarrow$ resistance connecting $n$ to parent node
5     $C \leftarrow$ downstream capacitance
6     $delay_n \leftarrow delay_{parent} + RC$
7     $slew_n \leftarrow \sqrt{slew_{parent}^2 + 1.93 * (RC)^2}$
8 end
```

5.3.5 Improving the Lagrangian Relaxation Solution

It is expected that Lagrangian relaxation generates a solution that does not respect all timing constraints in the discrete problem. This is mainly due to the lost of optimality in the discrete case and also the use of incomplete timing propagation during gate resizing. Thus, after performing several iterations of Lagrangian relaxation, the solution provided can still be improved using two other specific methods for both timing and power optimization.

In this work we apply two straightforward greedy methods to do local optimization. They are called Timing Recovery (TR) and Power Reduction (PR).

The first method fixes the timing violations left by LR. It is executed only if the Total Negative Slack (TNS) is higher than a threshold ϵ. The implementation submitted to both contests uses $\epsilon =$ 1e-6 ps, i.e., Timing Recovery is executed when the solution has any path with negative slack.

Algorithm 7 shows the pseudo code for the Timing Recovery algorithm. The nets n are sorted in decreasing order of the number of critical paths passing through them. Here, all paths reaching the timing endpoints with negative slacks are considered critical. The algorithm tries to decrease the delay of the gate driving n by changing the current gate-version to the next larger gate size with the same V_t.

In this method, V_t decrease is not applied since it would lead to a high leakage power increase for this particular library. At the same time, iteratively increasing gate sizes is efficient enough to solve small timing violations left by the LR while not increasing leakage power too much.

A gate g is considered *upsizable* if changing the original gate version to the next bigger size does not generate electrical violations. Moreover, the TNS with this change

Algorithm 7: Timing Recovery

```
1  $g \leftarrow$ most critical gate
2  $previousTNS \leftarrow TNS$
3  while $g$ do
4      if $g$ is upsizable then
5          upsize g
6          run incremental STA
7          if $TNS < previousTNS$ and no load/slew violations generated then
8              $previousTNS \leftarrow TNS$
9              re-sort gates
10         else
11             undo
12         end
13     end
14     $g \leftarrow$ next critical gate
15 end
```

must be smaller than the TNS of the previous solution.

The second method searches for local gate changes that optimize power without creating timing violations. The greedy Power Reduction algorithm is presented in Algorithm 8. For each gate g it tries to increase the V_t and/or to downsize the gate g.

V_t can be increased if the gate with higher V_t does not generate any electrical violation and the TNS is smaller than or equal to the previous TNS. The same is valid for downsizing a gate.

5.4 Empirical Validation

We evaluate our academic discrete gate sizing approach using the ISPD 2012 and 2013 Discrete Gate Sizing Contest infrastructures and benchmark suites.

The main difference between ISPD 2012 and 2013 contests is the interconnection modeling. Therefore, the overall Lagrangian relaxation-based flow applied to both infrastructures is the same.

The proposed approach is fully implemented in C++ without any third-party libraries.

All the results presented in this section are validated using Synopsys PrimeTime® to check the design constraints.

Algorithm 8: Power Reduction

```
repeat
    $changedCounter \leftarrow 0$
    for each gate $g$ of the circuit in topological order do
        if $V_t$ of $g$ is increasable then
            increase $V_t$ of $g$
            update timing (STA)
            if TNS $\geq$ 0 and no electrical violations generated then
                $changedCounter++$
            else
                undo
            end
        end
    end
    for each gate $g$ of the circuit in topological order do
        if $g$ is downsizable then
            downsize $g$
            update timing (STA)
            if TNS $\geq$ 0 and no load/slew violations generated then
                $changedCounter++$
            else
                undo
            end
        end
    end
until $changedCounter = 0$;
```

5.4.1 ISPD 2012 Contest

In this section we evaluate our flow using the infrastructure and benchmarks from the ISPD 2012 Discrete Gate Sizing Contest. The number of combinational gates in those circuits ranges from 23K to 861K combinational gates.

For the 2012 contest, the interconnections are modeled as simple lumped capacitances. Therefore, the RC interconnection model presented in Section 5.3.4 is *not* used.

Since the proposed lambda-delay sensitivities technique is compatible with the lumped capacitance model, it is applied in the experimental results presented in this section.

The final leakage power results in Watts (W) for each benchmark-constraint combination are presented in Table 5.7.

Our results for these benchmarks are compared with recent publications from Hu et al. (2012) and Li et al. (2012), that are also based on the ISPD 2012 Contest Bench-

Table 5.7: Leakage power (W) for ISPD 2012 benchmarks and number of combinational cells for all circuits.

Benchmark	**Comb. Cells**	**Leakage Power (W)** HU et al. 2012	LI et al. 2012	Ours	**Power Difference** Compared to HU et al.	Compared to LI et al.
DMA_slow	23K	0.145	0.153	0.132	-8.73%	-13.50%
DMA_fast		0.299	0.281	0.238	-20.29%	-15.19%
pci_bridge32_slow	30K	0.111	0.111	0.096	-13.31%	-13.31%
pci_bridge32_fast		0.183	0.167	0.136	-25.51%	-18.37%
des_perf_slow	102K	0.614	0.671	0.570	-7.14%	-15.03%
des_perf_fast		1.842	1.930	1.395	-24.27%	-27.73%
vga_lcd_slow	148K	0.351	0.375	0.328	-6.61%	-12.59%
vga_lcd_fast		0.471	0.460	0.413	-12.22%	-10.12%
b19_slow	213K	0.583	0.604	0.564	-3.28%	-6.64%
b19_fast		0.771	0.784	0.717	-7.06%	-8.61%
leon3mp_slow	540K	1.341	1.400	1.334	-0.53%	-4.72%
leon3mp_fast		1.487	1.640	1.443	-2.99%	-12.04%
netcard_slow	861K	1.770	1.780	1.763	-0.41%	-0.97%
netcard_fast		1.861	2.180	1.841	-1.07%	-15.55%
Avg.		0.845	0.895	0.784	-9.53%	-12.45%

mark Suite. The methodology proposed in this work is able to find the best solution among all algorithms, i.e., the solution with smallest leakage power and no constraints violations.Leakage power reduction of up to 27.73% is obtained with the proposed flow. Compared to Hu et al. (2012), our solution reduces leakage power in 9.53% on average and 12.45% on average when compared to (LI et al., 2012).

Considering the smaller circuits (*DMA*, *pci_bridge32*, *des_perf* and *vga_lcd*), our approach reduces leakage power by 14.76%, on average, compared to (HU et al., 2012) and 15.73% compared to (LI et al., 2012). As Hu et al. (2012) stated, the timing constraints for larger circuits (except for *netcard*) are tighter than for the smaller ones, and thus it is more difficult to reduce leakage power keeping a violation-free circuit in the former case.

Table 5.8 shows the runtimes to obtain the results in Table 5.7. Our solution is 19X faster than Hu et al. (2012) and 1.18X faster than Li et al. (2012) considering the total runtime for all benchmarks. Li et al. (2012) run the experiments on a Linux workstation with six 2666 MHz two-socket cores and 72 GB memory, using multi-threading.

Table 5.8: Runtime (minutes) for ISPD 2012 benchmarks and number of combinational cells for all circuits. Runtimes are taken from the corresponding papers.

Benchmark	Comb. Cells	Runtime (min) HU et al. 2012	Runtime (min) LI et al. 2012	Runtime (min) Ours	Speedup (X) Compared to HU et al.	Speedup (X) Compared to LI et al.
DMA_slow	23K	9.90	0.60	0.79	12.53	0.76
DMA_fast		13.90	0.60	0.92	15.11	0.65
pci_bridge32_slow	30K	10.20	1.20	0.87	11.72	1.38
pci_bridge32_fast		13.00	1.20	0.92	14.13	1.30
des_perf_slow	102K	70.10	6.00	25.31	2.77	0.24
des_perf_fast		82.70	6.60	16.37	5.05	0.40
vga_lcd_slow	148K	87.50	7.80	5.67	15.43	1.38
vga_lcd_fast		45.60	10.20	8.37	5.45	1.22
b19_slow	213K	213.90	10.20	9.15	23.38	1.11
b19_fast		206.50	12.00	11.75	17.57	1.02
leon3mp_slow	540K	1274.00	43.80	38.98	32.68	1.12
leon3mp_fast		1323.20	54.60	46.62	28.38	1.17
netcard_slow	861K	299.90	48.00	34.39	8.72	1.40
netcard_fast		1096.90	88.80	47.41	23.14	1.87
Sum (h)		79.12	4.86	4.13	19.18	1.18

Compared to our work, Li et al. (2012) use a slower machine, but uses a multi-threaded implementation. Hu et al. (2012) performed the experiments on a 3.2GHz Intel Xeon E31230 Linux workstation with 8GB of memory. The ISPD 2012 and 2013 Contests results were obtained on a Linux system with 2.93GHz CPU and 48GB memory.

As we can observe, the methods described in this work presents the best results for power and runtime compared with the state-of-the-art works. Considering the efficiency of this flow, a set of changes were made in this approach to support the ISPD 2013 Discrete Gate Sizing Contest infrastructure. The changes and results for the benchmarks of the ISPD 2013 Contest are presented next.

5.4.2 ISPD 2013 Contest

In this subsection we evaluate our flow using the infrastructure and benchmarks from ISPD 2013 Discrete Gate Sizing Contest. The number of combinational gates in those circuits ranges from 510 to 884K gates as shown in Table 5.9.

In the 2013 contest, interconnections are modeled as RC trees for which the timing

Table 5.9: Leakage power (W), runtime (min) and clock period (ps) on ISPD 2013 benchmarks comparing the contest results and our new results using accurate timing information in Timing Recovery and Power Reduction algorithms. Power results are truncated.

Benchmark	**# of Comb. Gates**	**Clock Period (ps)**	**Leakage Power (W)** Best ISPD'13	Ours New	Ours Best[a]	**Power Saved[b]**	**Runtime (min)** Best ISPD'13	Ours New
usb_phy_slow	510	450	0.00107	0.00107	0.00106	0.05%	0.6	0.5
usb_phy_fast		300	0.00160	0.00155	0.00153	3.36%	0.6	0.4
pci_bridge32_slow	28K	1000	0.05789	0.05696	0.05694	1.61%	14.3	10.5
pci_bridge32_fast		750	0.09651	0.08543	0.08503	11.47%	87.0	22.6
fft_slow	31K	1800	0.09034	0.08660	0.08654	4.14%	36.6	25.7
fft_fast		1400	0.22620	0.19430	0.19390	14.10%	52.2	40.4
cordic_slow	42K	3000	0.32379	0.27051	0.26566	16.45%	94.7	69.0
cordic_fast		2626	1.43057	1.00099	0.98017	30.03%	94.8	117.1
des_perf_slow	104K	1300	0.35300	0.33042	0.32728	6.40%	96.1	132.3
des_perf_fast		1140	0.79399	0.64882	0.64449	18.18%	280.9	347.9
edit_dist_slow	121K	3600	0.44740	0.42549	0.41603	4.90%	116.2	123.9
edit_dist_fast		3000	0.59632	0.53978	0.53547	9.48%	185.5	353.0
matrix_mult_slow	153K	2800	0.46973	0.44427	0.44291	5.42%	243.4	226.1
matrix_mult_fast		2200	2.13007	1.61093	1.54156	24.37%	416.5	396.0
netcard_slow	884K	2400	5.24566	5.15523	5.15483	1.72%	549.4	483.6
netcard_fast		2000	5.31783	5.20015	5.18158	2.21%	613.3	400.9
Avg. (all)		1860				**9.62%**	180.1	171.9
Avg. (only $fast$)		1677				**14.15%**	216.4	209.8

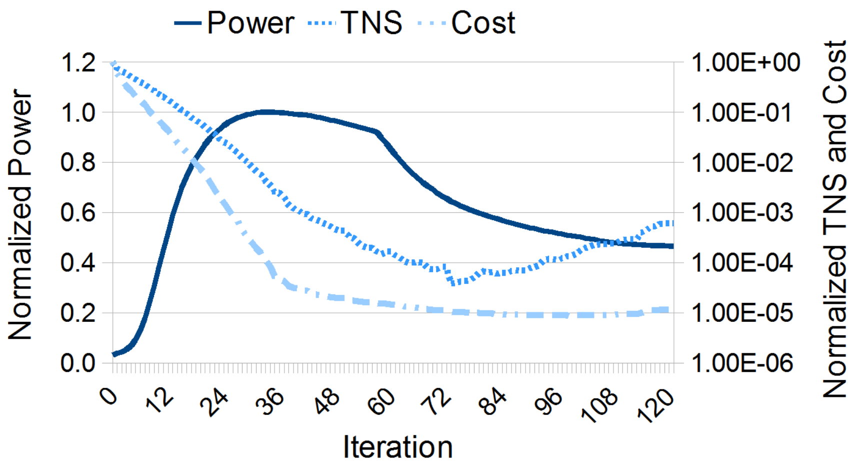
[a] Our solution without the runtime limit set in the ISPD 2013 Contest.
[b] Our solution with the runtime limit ("Ours New") compared to ISPD 2013 Contest.

is computed using the algorithm presented in Section 5.3.4. As the lambda-delay sensitivities technique is not compatible with the RC model, it is *not* applied in the experimental results presented in this subsection. Moreover, the flow includes a timing validation step which performs timing recovery using accurate timing and effective wire capacitance values from Synopsys PrimeTime® to ensure a violation-free solution.

The tool described is this section achieved the first place in the ISPD 2013 Contest. The results presented at ISPD 2013 Contest were further improved, reducing the leakage power and producing solutions with no violations for all benchmarks.

The new results compared with the best results obtained at the ISPD 2013 Contest including ours are presented in Table 5.9. The timing results are also validated using Synopsys PrimeTime®. For the three circuits without power and runtime values at ISPD 2013 Contest ($cordic\_fast$, des_perf_fast, $matrix\_mult\_fast$) we are considering

Figure 5.7: Leakage power, TNS and solution cost along iterations for $cordic\_fast$.



Source: from author (2016).

our solution that was submitted to ISPD 2013 Contest.

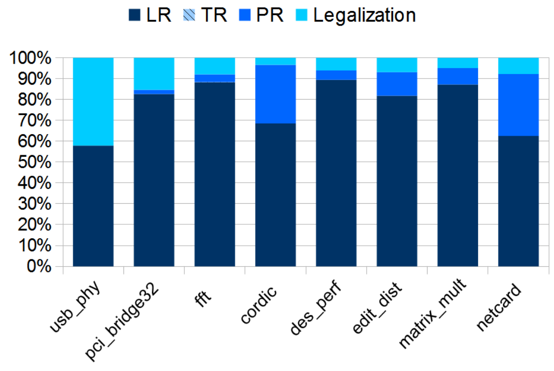
The new results provide on average 9.62% power reduction compared to the best Contest results with up to 30% power reduction. The runtime is on average 1.28X faster than the best Contest results. Considering only the fast circuits, leakage power is reduced by 14.15% and the runtime is 1.41X faster than the best Contest results.

Table 5.9 shows the results of using accurate timing analysis in the Timing Recovery and Power Reduction algorithms. It can be observed how the accurate timing helps achieving improved results. This also indicates that relying on our simplified internal STA might be preventing the Lagrangian Relaxation optimization algorithm from achieving even better results earlier. Table 5.9 also reports the best results achieved with this flow without considering the runtime limit.

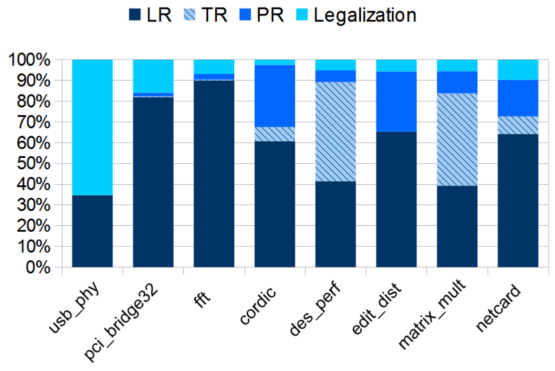
The leakage power and TNS behavior along iterations is presented in Figure 5.7 with the cost metric that chooses the best solution (line 9 in Algorithm 1) to be stored. The trend break observed in iteration 57 is due to the change of k (Algorithm 2). Setting k to a value close or smaller than one helps the convergence of TNS to a near zero value. With a k greater than one lambdas will decrease faster than increase, helping leakage power to reduce faster, as observed in the chart. Considering this, k is set to one in the beginning of LR and set to 4 when TNS is considered small (iteration 57 in this case) and is kept with this value until the final iterations. At this point k is set again to a value equal or smaller than one in order to reduce the remaining timing violations at the end of LR.

In Figure 5.8 the runtime breakdown of each algorithm for all benchmarks is presented. One can notice that the Timing Recovery algorithm only requires a significant runtime for some of the $fast$ benchmarks. This behavior is expected since circuits with

Figure 5.8: Runtime breakdown for (a) $slow$ and (b) $fast$ corners.



(a)



(b)

Source: from author (2016).

$slow$ constraints do not require hard timing legalization after the Lagrangian Relaxation optimization.

The Power Reduction (PR) algorithm shows almost the same runtime proportion for both constraints in the same circuits, varying for the different benchmarks.

A close look at the PR runtime shows which circuits need more local optimization after LR but it can also show how the incremental timing analysis used in PR is slower than the local timing update performed in LR. To better understand when each case occurs, Table 5.10 shows the total leakage power after each step of the proposed flow.

Comparing Figure 5.8 and Table 5.10 it is clear that the PR step for the $netcard$ and $edit\_dist$ benchmarks does not improve leakage power significantly despite requiring a significant runtime. So, it is possible to conclude that this large runtime is due to slow incremental STA caused by the characteristics of those circuits and their number of gates. On the other hand, for the $cordic$ and $matrix\_mult$ benchmarks the significant runtime reflects directly into power savings. That represents the case when the PR step needs more iterations to exhaust all possible local optimizations left by the LR.

The runtime breakdown for the LR algorithm alone shows that 97% of total runtime is spent performing STA, being 87% calculating wire timing, 3% calculating gate timing and only less than 2% performing Lagrangian Relaxation. This runtime breakdown shows that LR is fast to execute and the runtime is dominated by timing analysis even with LR relying only in local timing updates.

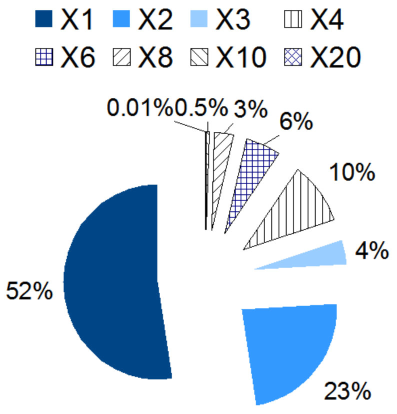
Finally, Figure 5.9 shows the gate usage for the $cordic$ benchmark. The difference between the gate usage for the two clock constraints shows how the sizing tool chooses faster gates to get timing closure with a tighter clock period. Not only bigger gates but

Table 5.10: Total leakage power after each step.

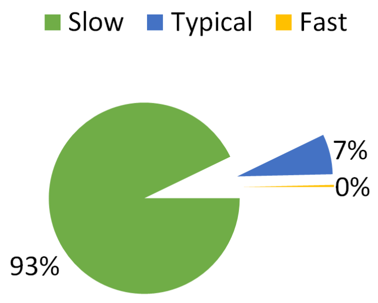
Benchmark	**Leakage Power - After:**				
	LR	TR		PR	
	W	*W*	diff.	*W*	diff.
usb_phy_slow	0.0010735	0.0010735	–	0.0010735	–
usb_phy_fast	0.0015450	0.0015830	**+2.46%**	0.0015770	**-0.38%**
pci_bridge32_slow	0.0571730	0.0571730	**+0.26%**	0.0571580	**-0.29%**
pci_bridge32_fast	0.0880710	0.0895070	**+1.63%**	0.0879155	**-1.78%**
fft_slow	0.0872705	0.0873675	**+0.11%**	0.0871495	**-0.25%**
fft_fast	0.2039270	0.2047320	**+0.39%**	0.2014390	**-1.61%**
cordic_slow	0.3091530	0.3091530	–	0.2810640	**-9.09%**
cordic_fast	1.6646800	1.6693800	**+0.28%**	1.1387300	**-31.79%**
des_perf_slow	0.3389120	0.3389120	–	0.3387110	**-0.06%**
des_perf_fast	0.7499550	0.7764230	**+3.53%**	0.7625440	**-1.79%**
edit_dist_slow	0.4293730	0.4297370	**+0.08%**	0.4293020	**-0.10%**
edit_dist_fast	0.5726620	0.5729100	**+0.04%**	0.5670630	**-1.02%**
matrix_mult_slow	0.4630130	0.4630130	–	0.4611650	**-0.40%**
matrix_mult_fast	2.0324600	2.0392700	**+0.34%**	1.6879700	**-17.23%**
netcard_slow	5.1169900	5.1170700	**+0.00%**	5.1146100	**-0.05%**
netcard_fast	5.1483100	5.1679200	**+0.38%**	5.1540800	**-0.27%**

also faster $V_{th}s$ are chosen to provide the best trade-off between timing and total leakage power. Also, the number of gates of each size/V_{th} shows how much effort is needed to get timing closure and how much leakage power the sizing tool must sacrifice to meet the desired clock constraint.

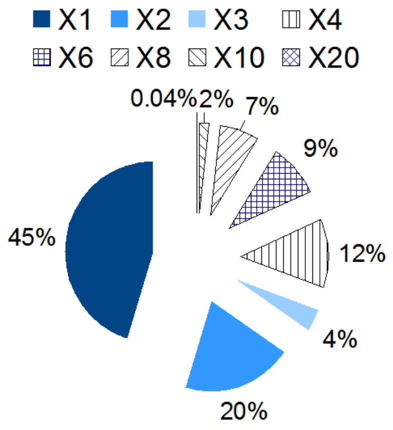
Figure 5.9: Gate usage by sizes and V_{th} for the $cordic$ benchmark. (a) and (b) for $cordic\_slow$, (c) and (d) for $cordic\_fast$.



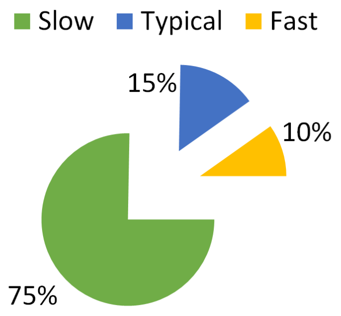
(a)



(b)



(c)



(d)

Source: from author (2016).

6 INDUSTRIAL DESIGN FLOW APPLICATION

In this chapter we detail the algorithms and the proposed modifications necessary to enable the cell selection flow to work in an industrial design flow, specifically in the post global route stage. In such a flow, different objectives must be optimized while other quality metrics are treated as constraints to the problem.

Different from the aforementioned contests, the initial solution provided to this flow (i.e. the solution after place and route) is considered to have a good quality. Therefore, that solution can be used as a reference to the cell selection optimization and also to provide a guidance to the Lagrangian relaxation-based algorithm.

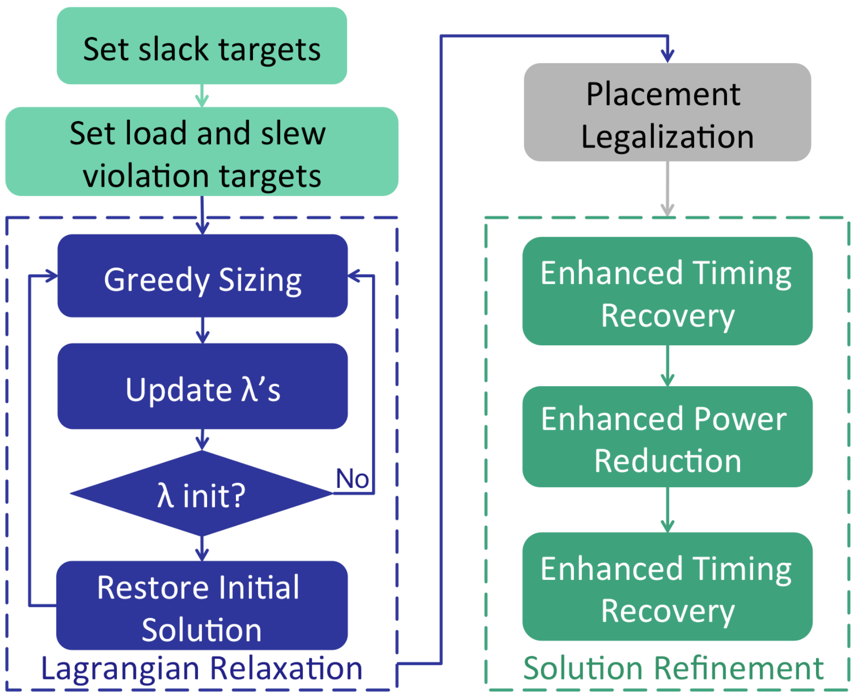
The first benefit of having a good initial solution is the electrical violations. They have already been resolved to the fullest extent practical. Since the input designs may not be closed, it is expected that electrical violations exist and are not solvable at this stage. To take that into account, the cell selection algorithm used in the Lagrangian relaxation iterations should see those violations as the maximum violation allowed for those specific nets. A decrease in the number of violations is acceptable but it is not a direct objective of the algorithm.

The second insight taken from the initial solution is the current timing information. In a pure power/area optimization stage of an industrial flow, the input timing quality must be considered as a hard constraint in the Lagrangian relaxation algorithm, i.e. timing results may not be degraded.

Another positive effect of the initial solution in our cell selection framework is a consequence of the greedy sizing method used in LR. Since the greedy algorithm relies on the cell options currently assigned to the gates, the first LR iteration will benefit from the fact that those gates already represent good choices of sizing and V_t. This also helps the algorithm to keep equal or less electrical violations than the initial solution, avoiding disruptions in the solution quality.

As mentioned before, one of the major challenges of gate sizing algorithms is the use of complex timing models needed in modern technologies and designs. The complexity of those models, and the existence of different clock domains, clock gating, accurate wire models for interconnect delay and slew propagation, make the sign-off timing engines too excessive in runtime to be used throughout the proposed optimization flow. The use of simpler timing models and/or timing estimation techniques is required to enable the use of the proposed flow in the chosen target environment.

Figure 6.1: The proposed cell selection flow.



Source: from author (2016).

The placement of gates is also important in a late optimization stage of an industrial design flow. A big change in placement will directly affect routing and, as a consequence, timing. Testing every placement change, even in a greedy way (i.e. the change caused only by the cell option being tested in the greedy cell sizing algorithm), would be prohibitive in runtime. Thus, the cell selection algorithm needs to take into account the area of the gates, also preventing overlapping and excessive (or any) overall area increase. However, limiting the maximum footprint of a cell to be the current cell area or less will degrade the final solution and also prevent further optimization.

The proposed flow incorporates the area into the objective minimization function. By doing so, the optimization objective will balance area and power accordingly. This will require scaling factors to set the correct proportion between the power, area and timing units. Also placement legalization must be performed to ensure an overlap-free design after cell selection optimization.

The new proposed flow is shown in Figure 6.1 and the details of each step are discussed in the next sections.

The two initial steps set the timing and electrical violation targets for each pin in the design. The sizing algorithm will consider the existing electrical violation as the violation limit for each pin, not allowing increase in electrical violations. The third step implements the iterative LR-based cell selection algorithm described in Section 6.1. Sections 6.1.1 to 6.1.3 discuss the initial Lagrange multiplier estimation method, the new multiplier update and the *ranking* algorithm, respectively. After that, a placement legalization is performed to fix any placement overlaps created.

In the *Solution Refinement* step, *Enhanced Timing Recovery* will work to improve solutions with slack degradation. Next, *Enhanced Power Reduction* will further reduce leakage power and area by a greedy method. The last step is a second run of *Enhanced Timing Recovery*. Both refinement methods are based on the algorithms already presented in Section 5.3 with proposed changes as discussed below.

Another placement related feature introduced in our flow is the legalization test during the post-LR solution refinement. This legalization test ensures that the new solution will have the desired effect after the final placement legalization.

In the late optimization problem, designs may have timing violations that cannot be solved by applying only cell selection algorithms. As a consequence of that, the typical LR formulation would lead to an increase in power/area in an attempt to fix all timing violations. This is not the desired effect of such an optimization process in an industrial IC design flow.

The main goal is to keep the same quality of results for timing and electrical violations (or improve them) and improve power and/or area of the design. To accomplish that, a change in the typical LR formulation (presented in Section 5.3) is required.

6.1 The New Lagrangian Relaxation Formulation

The cell selection optimization problem here, the Primal Problem (PP), can be formulated as:

$$\begin{aligned} &\textbf{PP:} \\ &\textbf{minimize} \quad \beta \times power + \theta \times area \\ &\textbf{subject to} \quad a_i + d_{i \to j} \le a_j \quad , \forall \text{ timing arc } i \to j \\ &\qquad\qquad\quad\; a_o \le T \qquad\qquad\;\; , \forall \text{ timing endpoint } o \end{aligned} \tag{6.1}$$

where β and θ are the scaling factors for each optimization objective in order to scale the different units. Both scaling factors are calculated based on the input standard cell library as follows. They reflect the average power/area change between cell options in the library.

$$\beta = \frac{N_c}{P_l(c_n) - P_l(c_0)}, \quad \theta = \frac{N_c}{A(c_n) - A(c_0)} \tag{6.2}$$

where c_n and c_0 are the largest (lowest V_t) and smallest (higher V_t) cell in the library. The θ calculation involves only a single V_t level. Applying the scaling factors makes the optimization free of power and area units. Such units may change between standard cell libraries and technologies.

The library-based parameters may also be replaced by design-dependent parameters like average power and average area for all cells in the design. We have found by experimentation that both methods lead to similar results.

Applying the Lagrangian relaxation method we obtain the LR Sub-problem (LRS) in (6.3).

$$\begin{aligned} &LRS\textbf{:} \\ &\textbf{minimize} \quad \beta \times power + \theta \times area + \\ &\qquad\qquad\quad \sum \lambda_{i \to j}(a_i + d_{i \to j} - a_j) + \\ &\qquad\qquad\quad \sum \lambda_o(a_o - T) \end{aligned} \tag{6.3}$$

Further simplification can be achieved by applying Karush-Kuhn-Tucker (KKT) conditions to optimality ($\lambda \in \Omega_\lambda$) (CHEN; CHU; WONG, 1999). Then, (6.3) is simplified resulting in the form in (6.4).

$$\begin{aligned} &LRS\ (\lambda \in \Omega_\lambda)\textbf{:} \\ &\textbf{minimize} \quad \beta \times power + \theta \times area + \alpha \times \sum \lambda_{i \to j} d_{i \to j} \end{aligned} \tag{6.4}$$

Here, the α scaling factor is introduced to normalize the timing unit. It is defined as follows:

$$\alpha = \frac{N_c}{D(c_n) - D(c_0)} \tag{6.5}$$

where delays $D(c_n)$ and $D(c_0)$ are calculated based on the standard cell library with the same reference output load.

We apply a method similar to (FLACH et al., 2014) to solve the LRS. Algorithm 9 shows the pseudo-code for the LRS solver. The sum $\sum \lambda_{i \to j} d_{i \to j}$ is referred as *lambda-delay* in the algorithms. *lambda-delay(c)* is the *lambda-delay* for all timing arcs connected to the input and output pins of the gate c. *optio*N_c represents the library option currently

Algorithm 9: SolveLRS

```
1  foreach gate $c \in Design$ do
2      $best\_option \leftarrow optioN_c$
3      $best\_cost \leftarrow \alpha \times lambda\text{-}delay(c) + \beta \times (P_l(c) + P_d(c)) + \theta \times A(c)$
4      foreach gate option $g \in F(c)$ do
5          $optioN_c \leftarrow g$
6          if electrical violations bigger than initial then
7              go to the next option
8          end
9          local timing update
10         if $new\_slack < \gamma * original\_slack$ then
11             go to the next option
12         end
13         $cost \leftarrow \alpha \times lambda\text{-}delay(c) + \beta \times (P_l(c) + P_d(c)) + \theta \times A(c)$
14         if $cost < best\_cost$ then
15             $best\_option \leftarrow g$
16             $best\_cost \leftarrow cost$
17         end
18     end
19     $optioN_c \leftarrow best\_option$
20     local timing update
21 end
```

assigned to c. $P_l(c)$ and $P_d(c)$ represent the leakage and the dynamic power for cell c, respectively.

Then, the new problem is to find the optimal set of lambdas that solve the PP. Thus, LDP is simply the maximization of LRS where λ is the variable, as presented in Chapter 5.

Algorithm 10 shows the new proposed method to solve the LDP problem. The first loop (lines 3-9) performs initial lambda estimation (details in Section 6.1.1). Second loop (lines 10-18) is the main LR flow that is limited by a maximum number of iterations or by convergence metrics.

The overall solution quality is measured by a score function. The score function penalizes timing degradation exponentially, as follows:

$$score = -\left(\Delta Power + \Delta Area + 2^{-\Delta TV} - 1\right) \tag{6.6}$$

where ΔTV represents the percentage of change in timing violation. All Δs are calculated with respect to input solution. Positive scores represent improved solutions while negative scores show solution degradation.

Algorithm 10: SolveLDP

```
store initial solution
set initial multipliers
**repeat**
    **SolveLRS**
    update timing
    **UpdateLagrangeMultipliers**
    restore initial solution
    update timing
**until** *iteration limit*;
**repeat**
    **SolveLRS**
    update timing
    **UpdateLagrangeMultipliers**
    **if** $new\_score > best\_score$ **then**
        store solution
        $best\_score \leftarrow new\_score$
    **end**
**until** *converged or iteration limit*;
restore best solution found
```

6.1.1 Initializing the Lagrange Multipliers

A well known issue in Lagrangian relaxation-based methods is how to define the initial values for the Lagrange multipliers. The initial set of multipliers plays a significant role in convergence and final quality of results, as shown in (TENNAKOON; SECHEN, 2005). However, finding a set of multipliers for a given input set of gate sizes and threshold voltages with the respective delays is a problem with a similar difficulty to the cell selection problem. Moreover, considering the KKT conditions, it may be impossible to find such a set of multipliers.

We propose a simple and straightforward method to overcome the lack of a good set of initial multipliers. The method consists of a few LR iterations where we solve the LRS problem, update the multipliers and restore the initial solution. The new set of multipliers represents a LRS solution with delays closer to the delays present in the input solution. This estimation method provides a set of initial multipliers to the main LR flow simulating an incremental approach and avoiding quality disruption.

In order to avoid excessive runtime due to sign-off timer calls, only the *ranking* method (Section 6.1.3) is used to solve the LRS, i.e., $F(c)$ has only one option that is always chosen in $SolveLRS$, with no need to update timing.

Algorithm 11: UpdateLagrangeMultipliers

```
1 $\rho_{inc} = \rho_{init} \times (1 + it)$
2 $\rho_{dec} = \rho_{init} \times (15 + it)$
3 foreach timing arc $i \to j$ do
4     $\lambda_i \leftarrow \lambda_i \times \begin{cases} \left(1 - \frac{S_{curr} - S_{init}}{\Delta WNS \times \rho_{inc}}\right)^{k_{inc}} & a_j \geq q_j - S_{init} \\ \left(1 + \frac{S_{curr} - S_{init}}{T \times \rho_{dec}}\right)^{k_{dec}} & a_j < q_j - S_{init} \end{cases}$
5 end
6 KKT projection ($\lambda \in \Omega_\lambda$)
```

6.1.2 Lagrange Multiplier Update

In the late optimization problem in industrial flows, designs may have timing violations that are expected to not be solved by any means. As a consequence, a normal LR formulation would lead to an increase in power/area in an attempt to fix all timing violations, which is not the desired effect of such optimization process. The main goal must be to keep the same timing quality of results (or improve it) and to improve power and area of the design. To accomplish that, a change in the LR formulation is required.

Reference (REIMANN; SZE; REIS, 2015) presents a new lambda update method that enables LR-based algorithms to handle designs with negative slacks. New lambda values are calculated to target the initial timing arc slack S_{init}. In this work we apply a similar but more aggressive method for the Lagrange multiplier update. Algorithm 11 shows the new update approach.

In order to achieve fast convergence and better timing quality of results, the timing reference for the multiplier increase (T) is replaced by the worst slack degradation in the current solution with respect to the initial state – called ΔWNS. Also, two step factors (ρ) are included to smooth convergence. These factors are similar to step sizes in the subgradient method. Without step factors, the solution quality has large variations along iterations (due to the discreteness of the problem) that lead to a slower convergence and degradation in quality of results.

Exponents k_{inc} and k_{dec} are defined as $k_{dec} = -k$ and $k_{inc} = {}^{1}\!/\!_{k}$. The variable k will determine the convergence priority: the constraints or the optimization objective. For $k < 1$, multipliers will increase faster, resulting in faster delay reduction in the next $SolveLRS$ run. On the other hand, $k > 1$ results in faster multiplier decrease, allowing more power/area savings while increasing delay in non-violating paths. The ranges for

Algorithm 12: OptionRanking

```
1 foreach gate option $g \in G(c)$ do
2     estimate local delays for $g$
3     $cost \leftarrow \alpha \times$ lambda-delay$(c) + \beta \times P_l(c) + \theta \times A(c)$
              $+ \zeta \times C(c)$
4     insert $g$ in $cost$-ordered vector $F(c)$
5 end
```

k are empirically defined according to the quality of previous LR iteration, as shown in (6.7). Different choices for k do not significantly affect the final quality of results, but allow faster convergence, reducing the number of iterations required.

$$k = \begin{cases} 5 & \text{if } \lambda \text{ initialization} \\ 0.2 \leq k < 1 & \text{if } TNS \text{ degraded} \\ 1.5 \leq k \leq 4 & \text{if } new\_score > score \\ 1 & \text{otherwise} \end{cases} \tag{6.7}$$

S_{curr} is the current worst slack obtained from the sign-off timer. Such a method makes the presence of multiple clock domains transparent to the LR engine, as derived by Ozdal, Burns and Hu (2012).

6.1.3 Filtering Gates Options

The use of a high accuracy timer to evaluate all cell options is not feasible in terms of runtime. A less accurate timing model can be used instead. However, the less accurate timer may not only evaluate (rank) the options differently but also make any electrical violation control ineffective.

Considering these limitations, we propose the use of a less accurate timing model to rank all gate options using the same cost function of the $SolveLRS$ algorithm. The less accurate timer ignores the delay and slew rate changes in the interconnect parasitics. It also ignores effective capacitance calculations. Timing arc delays and output slews are calculated using the lookup tables in the standard cell library. After that, the sign-off timer only evaluates the top t ranked gate options, skipping several high accuracy timing updates to reduce runtime.

The *ranking* method is presented in Algorithm 12. Since there is no timing update when ranking the options, there is no accurate dynamic power calculation available, as updated input slews would be needed. Dynamic power is estimated by using the input pin capacitance of the new gate option. The input pin capacitance must also be properly scaled to become unitless. The input pin capacitance scaling factor ζ is calculated as follows.

$$\zeta = \frac{N_c}{C(c_n) - C(c_0)} \tag{6.8}$$

Similar to the other scale factors, ζ can also be calculated using design statistics like average input pin capacitance.

6.2 Solution Refinement

The fast and highly constrained convergence adopted during LR iterations generates solutions that still show space for local optimizations.

In some cases, timing must be improved to better resemble the timing quality of the initial state. The algorithm used for that end is described next. Also, power can be locally optimized by a greedy algorithm as detailed hereafter.

6.2.1 Enhanced Timing Recovery

The Timing Recovery algorithm is here adapted to handle placement legalization for each gate size/V_{th} change. Also, a new ordering for the search is set. Instead of ordering the gates as in the original work, we order the gates by their worst slack, processing the most critical gates first.

For each new gate option under test, the algorithm checks if the gate needs to be moved from its current location to fit the new size option. In the cases where the gate has its footprint increased and there is no free space to fit it, a new placement location with available space is found. Thereby, the timer takes into account the routing parasitics changes to update timing.

The second run takes advantage of capacitance and area reduction obtained in the *Enhanced Power Reduction* step. Only paths violating the initial worst slack are searched in both runs.

6.2.2 Enhanced Power Reduction

The Power Reduction algorithm can further improve the solution after LR, reducing power and area of cells. Since it only performs downsizing and V_{th} increase, placement legalization is usually not required. Nevertheless, the algorithm is also placement-aware and can find new cell placement locations if necessary.

The algorithm is here adapted to prioritize downsizing in non-critical gates in the fanout of critical paths, reducing the output load of critical gates and allowing more improvements in the next run of *Enhanced Timing Recovery*.

We also improve the algorithm to prevent True Total Negative Slack ($TTNS$) degradation. The original implementation would take advantage of worst slack propagation, powering down gates in paths dominated by a side negative slack. Since $TTNS$ calculation includes the slacks of side paths, degradations occur when any slack gets more negative. Our enhanced algorithm checks for slack degradation in the output pin of all changed gates and rejects solutions with slack degradation.

6.3 Empirical Validation

We evaluate the proposed method in the post-global routing optimization stage of an industrial design flow with fourteen high-performance industrial microprocessor blocks with 5GHz clock frequency ($174ps$ clock period) using a 22nm standard cell library with 2 or 3 threshold voltage levels available, according to the specific design.

First we present the results that do not consider TTNS in the Enhanced Power Reduction stage. Those results present more leakage power reduction but, as expected, degrade TTNS. Such an approach may be useful when designers will not apply others techniques to fix timing and want to find the best power-TNS trade-off.

6.3.1 Preliminary results

As mentioned in the previous section, the slack targets are set based on the converged timing slacks[1] and the new Lagrange update method multiplier presented in the

[1] Here, converged timing slacks refer to the end of "optimization", "clock insertion and optimization" or "routing optimization" stages.

Table 6.1: Characteristics for 14 high performance microprocessor designs.

	Sizeable	Worst	TNS	Power (mW)		
Design	Gates	Slack (ps)		Leakage	Dynamic	Total
ibm2014uP_01	88521	-72.57	-138492	79.08	13.54	92.62
ibm2014uP_02	7346	-142.43	-3004	1.07	1.28	2.35
ibm2014uP_03	7940	8.76	-18	2.73	51.37	54.10
ibm2014uP_04	5848	-8.38	-552	1.59	1.35	2.94
ibm2014uP_05	10448	-87.06	-42914	18.92	45.21	64.13
ibm2014uP_06	60768	-142.67	-37685	37.40	112.03	149.43
ibm2014uP_07	63724	-41.50	-52367	60.51	12.60	73.11
ibm2014uP_08	14372	-72.89	-38053	16.29	68.06	84.35
ibm2014uP_09	15270	-30.74	-13234	14.41	33.03	47.44
ibm2014uP_10	117011	-42.43	-65276	85.01	304.35	389.36
ibm2014uP_11	20929	-164.94	-188713	35.97	21.65	57.62
ibm2014uP_12	14529	-421.29	-363751	4.26	20.47	24.73
ibm2014uP_13	16481	-46.36	-27261	19.90	61.22	81.12
ibm2014uP_14	5595	-61.86	-6525	8.18	9.68	17.86

previous section is used in our implementation. We successfully applied the LR-based power optimization algorithm on the designs shown in Table 6.1.

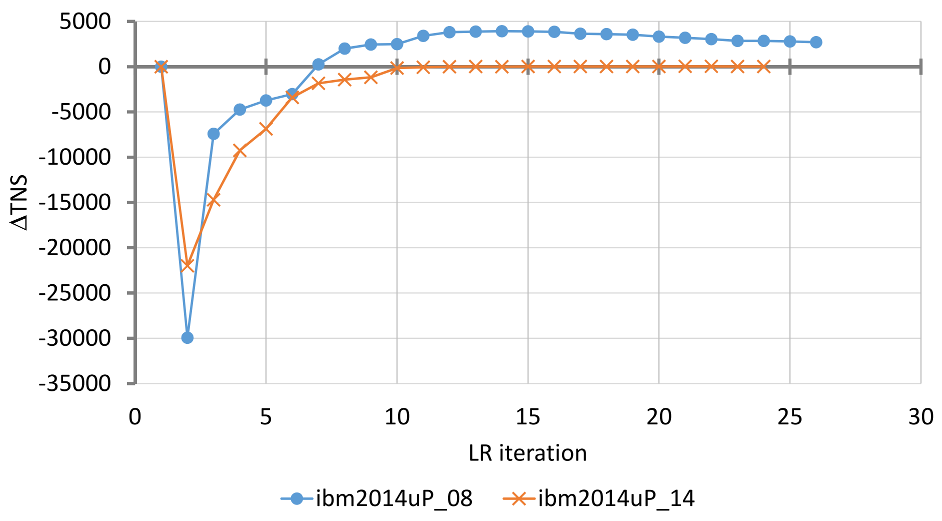
First, we would like to understand how timing quality changes in the first several LR iterations of the algorithm. Figure 6.2 shows the LR convergence for two different designs. TNS degrades drastically for first several iterations, because the multipliers (λs) are not properly set to resemble timing quality of the input solution (we initialize all multipliers with the same constant value). At the same time, these iterations are the price to pay in order to find out which timing arcs are more timing-critical and which are not. After that, further iterations can mitigate timing degradation producing a solution with similar (ibm2014uP_14) or better (ibm2014uP_08) timing quality. The designs shown in Figure 6.2 converge pretty quickly (at around the 15^{th} iteration) and the cost of "λ-tuning" is one third of the total CPU time, which is significant considering the LR-based cell selection algorithm usually takes hours of runtime for our medium-sized designs. It is good to remember that some oscillation in timing quality is expected during the LR optimization process, but excessive timing degradation is not desired in a method that is supposed to be incremental.

As stated before, using a formulation that assumes that all timing constraints can be respected in the design is not suitable for designs with known negative slack paths. Moreover, balancing those negative slacks may also degrade the timing quality of results. This situation is similar to the ISPD contest formulation, where all designs are considered to have no timing violations, and non-critical paths can have their timing relaxed as

Table 6.2: Results assuming timing closure can be achieved. Only LR step in the flow is executed.

Design	Worst Slack (ps)		TNS (ps)			Leakage Power (mW)			Dynamic Power (mW)			CPU (s)
	pre-LR	post-LR	pre-LR	post-LR	diff	pre-LR	post-LR	diff	pre-LR	post-LR	diff	
ibm2014uP_01	-75.2	-75.1	-138639	-138234	405	80.38	87.88	9.3%	13.46	13.47	0.1%	27486
ibm2014uP_02	-135.1	-135.1	-2973	-3479	-506	1.07	1.18	10.1%	1.30	1.27	−2.1%	4191
ibm2014uP_03	8.9	4.6	-16	-22	-6	2.73	3.16	15.6%	51.24	50.01	−2.4%	2345
ibm2014uP_04	-8.4	-7.7	-552	-529	23	1.59	1.59	0.2%	1.37	1.36	−0.3%	749
ibm2014uP_05	-82.1	-85.1	-43211	-45913	-2702	19.14	19.61	2.4%	45.04	45.12	0.2%	1714
ibm2014uP_06	-133.4	-148.6	-30643	-45586	-14943	38.14	38.31	0.4%	112.01	112.00	0.0%	13312
ibm2014uP_07	-38.4	-39.9	-52648	-56713	-4065	60.97	62.76	2.9%	12.60	12.60	0.0%	23021
ibm2014uP_08	-78.0	-78.0	-37504	-35653	1851	16.56	17.50	5.7%	68.68	69.16	0.7%	3900
ibm2014uP_09	-33.8	-73.3	-14110	-16018	-1908	14.77	15.79	6.9%	33.29	33.56	0.8%	4831
ibm2014uP_10	-34.6	-62.5	-70314	-65161	5153	86.11	90.23	4.8%	303.92	303.71	−0.1%	51932
ibm2014uP_11	-162.8	-162.9	-189618	-186607	3011	36.10	40.14	11.2%	21.59	21.62	0.1%	4938
ibm2014uP_12	-421.4	-421.1	-354649	-337070	17579	4.28	5.36	25.2%	20.37	20.34	−0.2%	4530
ibm2014uP_13	-49.4	-52.2	-26363	-25379	984	20.27	20.98	3.5%	60.93	60.88	−0.1%	3865
ibm2014uP_14	-61.9	-65.1	-6526	-6463	63	8.18	8.19	0.1%	9.68	9.68	0.0%	1027

Figure 6.2: TNS change along LR iterations.



Source: from author (2016).

long as they still respect the defined clock arrival time. Table 6.2 shows the Lagrangian relaxation result for 10 designs when all timing endpoints are set to target zero slack. Both of the aforementioned cases of failure are present in the results. For example, design ibm2014uP_12 shows a small timing improvement while leakage power degrades by 25%. On the other hand, design ibm2014uP_06 shows considerable timing degradation while leakage power only improves by 0.4%. None of those two cases are acceptable in an industrial design flow.

A common approach is to set the slack target for all paths as the worst slack in the design, as mentioned before.The experiment shows the importance of setting a different timing target for each timing endpoint and not a constant target for all paths. This ap-

Table 6.3: Results considering the worst slack as the target for all timing paths. Only LR step in the flow is executed.

Design	Worst Slack (ps)		TNS (ps)		Leakage Power (mW)		Dynamic Power (mW)		CPU (s)
	pre-LR	post-LR	pre-LR	post-LR	pre-LR	post-LR	pre-LR	post-LR	
ibm2014uP_01	-73.1	-95.5	-139703	-369537	80.49	-44.4%	13.45	-0.3%	26867
ibm2014uP_02	-142.4	-142.7	-3004	-16768	1.07	-26.1%	1.29	-9.2%	3842
ibm2014uP_03	8.9	6.1	-16	-80	2.73	27.6%	51.24	-1.4%	2723
ibm2014uP_04	-8.4	-10.0	-552	-1122	1.59	0.0%	1.37	-0.3%	1970
ibm2014uP_05	-83.5	-90.9	-43124	-80768	19.13	-17.7%	45.09	-3.0%	1691
ibm2014uP_06	-142.7	-151.6	-27901	-385203	37.84	-14.1%	112.01	-0.1%	12532
ibm2014uP_07	-38.7	-48.4	-52238	-175215	61.04	-45.7%	12.60	0.0%	21889
ibm2014uP_08	-72.9	-75.8	-38008	-74927	16.56	-43.5%	68.75	-8.0%	4160
ibm2014uP_09	-33.8	-39.1	-14110	-40741	14.77	-30.1%	33.29	-2.2%	4520
ibm2014uP_10	-35.3	-44.2	-69144	-281091	85.81	-46.7%	303.93	-9.9%	45642
ibm2014uP_11	-158.7	-177.4	-191743	-256042	36.17	-46.7%	21.59	-0.2%	5089
ibm2014uP_12	-421.1	-421.1	-359714	-564317	4.26	-21.6%	20.35	-3.1%	4758
ibm2014uP_13	-51.2	-56.3	-26553	-56966	20.29	-41.5%	60.93	-2.8%	3838
ibm2014uP_14	-61.9	-61.7	-6526	-13759	8.18	-1.9%	9.68	0.0%	1080

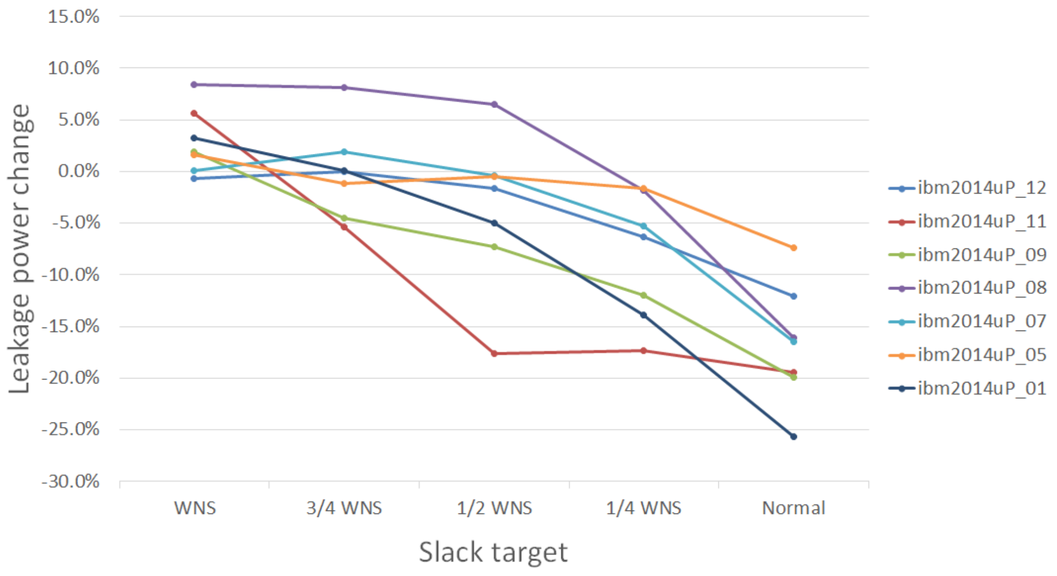
proach is similar to considering the negative slacks as impossible to solve (i.e. the best possible timing has already been achieved – or timing has converged) and to promote a clock period relaxation in order to circumvent those timing violations in critical paths. As mentioned before, this relaxation will produce a considerable timing degradation (when considering the actual clock period) in non-critical paths, achieving more power improvements.

Table 6.3 shows the results for 10 designs using this approach. As expect, timing quality degrades significantly even when WNS is kept almost the same. Such results can be seen as a lower-bound reference of the worst degradation caused by a method that ignores the input state and only balances power and timing.

In the middle of the design flow when the designers are still improving the floor plan and pin assignment, increasing the clock period makes cell selection at the end of physical synthesis trivial, and a global method (like LR) will not improve much the solution provided by the greedy algorithm. We tested this possibility by changing the slack target in the physical synthesis flow to simulate a clock period relaxation. We set the global slack target to be the worst slack, 3/4, 1/2 and 1/4 of the worst slack of a previous physical synthesis run.

Figure 6.3 shows the leakage power improvement at the end of LR iterations with the five different slack targets for the physical synthesis run. The more the clock is relaxed

Figure 6.3: Leakage power improvement at the end of LR in five different scenarios for physical synthesis slack target (target clock period).



Source: from author (2016).

(the left most case) the smaller is the leakage power improvement (leakage power may even increase). The right most results ("Normal") are the typical run, where the slack target is a positive slack value. Results show that a tighter clock period makes the standard greedy algorithm present in the industrial flow less effective, allowing more leakage power improvement in our global cell selection method.

The complete results of applying LR-based power-driven cell selection algorithm with individual slack targets at the end of our physical synthesis flow is shown in Table 6.4. "pre-GS" and "post-GS" are measurements just before and after the cell selection algorithm. It is promising to see that our method can achieve 11.7% of average leakage power saving on top of our existing physical synthesis flow, which is power-aware and has components to minimize low-V_t usage. The table also shows that the power reduction algorithm does not degrade worst slack and TNS.

However, as mentioned in before, focusing only on worst slack and TNS gives the user of a real industrial design flow a wrong perception. And we would like to also evaluate the change of timing quality of results during the run of the Lagrangian Relaxation-based method. As shown in the columns marked "$TTNS$" in Table 6.4, the degradation on timing quality of results is significant which gives a totally different conclusion comparing to "worst slack" and "TNS". Even though the cell selection method maintains worst slack and the TNS of the designs, the $TTNS$ is worsened significantly for some designs. This behavior is not desired in this real industrial design flow, since the timing

Table 6.4: Results with proposed modifications included for a set of 14 high performance microprocessor designs.

Design	Worst Slack (ps)		TNS (ps)		Leakage Power (mW)			$TTNS$ (ps)			CPU (s)
	pre-GS	post-GS	pre-GS	post-GS	pre-GS	post-GS	diff	pre-GS	post-GS	diff	
ibm2014uP_01	-72.6	-72.6	-138492	-133799	79.08	58.75	-25.7%	-882785	-956814	-74029	37068
ibm2014uP_02	-142.4	-141.3	-3004	-2480	1.07	0.99	-7.3%	-24295	-30079	-5784	3064
ibm2014uP_03	8.8	7.0	-18	-14	2.73	2.70	-1.0%	-39	-67	-27	3875
ibm2014uP_04	-8.4	-7.4	-552	-513	1.59	1.58	-0.4%	-560	-519	41	1021
ibm2014uP_05	-87.1	-86.9	-42914	-42213	18.92	17.53	-7.4%	-77630	-80474	-2845	2548
ibm2014uP_06	-142.7	-142.7	-37685	-34145	37.40	34.64	-7.4%	-64667	-66497	-1830	19120
ibm2014uP_07	-41.5	-41.8	-52367	-50054	60.51	50.57	-16.4%	-390841	-415402	-24561	26886
ibm2014uP_08	-72.9	-72.9	-38053	-33914	16.29	13.68	-16.1%	-201562	-200360	1202	5314
ibm2014uP_09	-30.7	-30.7	-13234	-12408	14.41	11.55	-19.9%	-65844	-77650	-11806	6083
ibm2014uP_10	-42.4	-42.0	-65276	-62411	85.01	72.85	-14.3%	-292779	-306952	-14173	44659
ibm2014uP_11	-164.9	-164.9	-188713	-182711	35.97	28.98	-19.4%	-1020419	-1023425	-3006	6848
ibm2014uP_12	-421.3	-421.2	-363751	-352693	4.26	3.74	-12.1%	-795556	-830203	-34648	5663
ibm2014uP_13	-46.4	-46.4	-27261	-24594	19.90	16.69	-16.1%	-137055	-143574	-6519	5656
ibm2014uP_14	-61.9	-60.9	-6525	-6430	8.18	8.12	-0.7%	-11213	-11134	78	1033
		Total:	-977845	-938379			Total:	-3965245	-4143151	-177906	

violations are expected to be fixed by other methods in different stages of the flow and less critical (but yet violating) paths should not be degraded, what would increase the effort to solve violations and undo the power improvements found. Also, other sizing algorithms applied in the flow are not allowed to degrade $TTNS$, so comparison would not be fair if our methods degrade $TTNS$.

The runs are performed on a server with 64bit-Linux with 24 Intel X5690 CPUs running at 3.47Ghz, and 142GB of memory. However, our current implementation of the LR-based algorithm only uses a single-thread. Also, despite the fact that the timing engine has multi-thread capabilities, they have insignificant impact on runtime due to the one-cell change for each timing update call. As shown in the table, the runtime can take up to 12.4 hours for the biggest design ibm2014uP_10.

6.3.2 Final results

We evaluate the proposed method in the post global routing optimization stage of an industrial design flow with a set of 14 high-performance mid-size industrial microprocessor blocks with 5GHz working frequency with a 22nm standard cell library. Considering that the input solution to our algorithm is already optimized (i.e. the best solution found by the industrial flow), all designs are challenging in terms of timing, area and power.

As proposed in (REIMANN; SZE; REIS, 2015), we also include $TTNS$ mea-

Table 6.5: Experimental results for 14 high performance microprocessor blocks.

	Worst Slack (ps)		TNS (ps)			$TTNS$ (ps)			Power		Area
Design	before	after	before	after	diff	before	after	diff	Leakage	Dynamic	
ibm22uP_01	-39.40	-39.33	-54401	-52860	1541	-392551	-414251	-21700	-16.8%	0.0%	-1.9%
ibm22uP_02	-78.34	-77.72	-144167	-132476	11691	-910468	-920704	-10236	-20.0%	-0.4%	-2.6%
ibm22uP_03	8.87	8.95	-14	-6	8	-30	-11	20	5.2%	-2.9%	-0.3%
ibm22uP_04	-82.13	-82.02	-43263	-42510	753	-76068	-76794	-726	-6.1%	-0.5%	-0.2%
ibm22uP_05	-32.74	-32.27	-14491	-13243	1248	-71828	-88072	-16244	-21.8%	0.1%	-2.7%
ibm22uP_06	-49.79	-49.75	-26647	-25587	1060	-133504	-137696	-4192	-15.5%	-0.5%	-2.0%
ibm22uP_07	-165.27	-165.23	-195168	-189647	5521	-1054130	-1075880	-21750	-18.0%	0.0%	-0.5%
ibm22uP_08	-34.20	-34.20	-70737	-68875	1862	-322544	-360042	-37498	-20.7%	-3.3%	-3.5%
ibm22uP_09	-72.89	-72.89	-37290	-34505	2785	-195777	-214362	-18585	-19.8%	-1.0%	-2.7%
ibm22uP_10	-61.86	-61.16	-6526	-6383	143	-11213	-11069	144	-0.5%	0.0%	-0.1%
ibm22uP_11	-421.88	-421.88	-359981	-348969	11012	-777205	-777115	90	-11.3%	-2.3%	-7.5%
ibm22uP_12	-135.13	-134.29	-2973	-2587	386	-22778	-26917	-4139	-8.3%	-4.0%	-6.3%
ibm22uP_13	-8.38	-7.40	-552	-515	37	-560	-521	39	-0.4%	-0.5%	-0.1%
ibm22uP_14	-142.57	-142.57	-36833	-35539	1294	-62323	-67034	-4712	-4.0%	0.0%	-0.2%
Average					2810			-9963	-11.3%	-1.1%	-2.2%
Total					39341			-139489	-16.3%	-1.8%	-2.1%

surements to show all timing changes in the design including subpaths, not only the worst timing paths that reach the timing endpoints. Table 6.5 briefly shows the results without applying the new enhancements to prevent $TTNS$ degradation in the *Enhanced Power Reduction* algorithm. Number of gates and runtimes shown in Table 6.7a. It is clear that ignoring degradation of slacks in side paths results in an unacceptable timing quality degradation. Such results show that the method is unsuitable for industrial design flows.

Nevertheless, Table 6.6 (refer to Table 6.7b for runtimes) shows results when fully applying the *Enhanced Power Reduction* method. This implementation will keep (or improve) the timing quality obtained in the Lagrangian relaxation stage, for all metrics.

Initial design information varies from Table 6.5 due to the adopted test methodology, where all results are generated running the whole physical synthesis flow. Thus, timing, power, area, number of gates are slightly different comparing both tables.

Although power improvements are smaller, the method is able to deliver considerable solution quality enhancements. It is important to remember that such improvements are over designs already power-optimized by other methods at several stages of the industrial design flow. We achieve 7.2% average (10.8% total and up to 18.2%) leakage power reduction with similar or better timing quality for all designs tested. Area used is also improved by 1.7% in average, which is remarkable in post-routing optimization stage.

Experimental data shows 10X average runtime speedup due to *ranking* method with less than 2% difference in leakage power improvement. Results are obtained using $t = 2$, i.e. only the top two ranked options are evaluated using the sign-off timer. These

Table 6.6: Experiments without $TTNS$ degradation in PR step.

	Worst Slack (ps)		TNS (ps)			$TTNS$ (ps)			Power		Area
Design	before	after	before	after	diff	before	after	diff	Leakage	Dynamic	
ibm22uP_01	-37.92	-37.92	-52108	-50126	1982	-376012	-357385	18627	-9.4%	0.0%	-0.6%
ibm22uP_02	-75.76	-75.13	-141471	-137744	3727	-906301	-897859	8442	-18.2%	-0.4%	-2.3%
ibm22uP_03	8.87	8.92	-14	-6	8	-30	-10	20	6.2%	-3.3%	-0.4%
ibm22uP_04	-82.13	-82.12	-43263	-42763	500	-76068	-75175	893	-3.6%	-0.2%	0.2%
ibm22uP_05	-33.84	-33.07	-14110	-13724	386	-72869	-71636	1232	-15.7%	0.6%	-1.8%
ibm22uP_06	-53.03	-53.16	-27185	-24680	2505	-134543	-128356	6187	-9.3%	-0.5%	-1.9%
ibm22uP_07	-165.32	-165.32	-191527	-188435	3092	-1034800	-1022700	12100	-10.2%	0.0%	1.3%
ibm22uP_08	-35.12	-36.35	-67145	-64510	2635	-305053	-290806	14247	-11.6%	-2.4%	-2.5%
ibm22uP_09	-80.36	-80.36	-39823	-36858	2965	-211734	-201734	10000	-11.4%	-0.7%	-2.3%
ibm22uP_10	-58.95	-59.05	-6419	-6425	-6	-10950	-10986	-36	-0.5%	0.0%	-0.1%
ibm22uP_11	-421.65	-421.53	-359783	-350035	9748	-787596	-764491	23105	-7.8%	-2.1%	-6.8%
ibm22uP_12	-141.00	-140.28	-3030	-2752	278	-23004	-21656	1348	-6.5%	-3.9%	-6.0%
ibm22uP_13	-8.38	-7.40	-552	-516	36	-560	-519	41	-0.3%	-0.5%	-0.1%
ibm22uP_14	-133.37	-133.37	-31798	-30242	1556	-52800	-49614	3186	-2.7%	0.0%	0.0%
Average					2101			7099	-7.2%	-1.0%	-1.7%
Total					29412			99392	-10.8%	-1.4%	-1.5%

Table 6.7: Number of gates and runtimes for Table 6.5 and Table 6.6.

(a) Results in Table 6.5.

Design	#Gates	CPU (s)
ibm22uP_01	70331	17685
ibm22uP_02	95057	24115
ibm22uP_03	8827	2540
ibm22uP_04	15777	1911
ibm22uP_05	17510	4734
ibm22uP_06	20167	3873
ibm22uP_07	24425	9454
ibm22uP_08	124670	32620
ibm22uP_09	17942	5219
ibm22uP_10	12941	785
ibm22uP_11	17480	3758
ibm22uP_12	9260	3400
ibm22uP_13	7293	808
ibm22uP_14	75148	17740

(b) Results in Table 6.6.

Design	#Gates	CPU (s)
ibm22uP_01	70333	18626
ibm22uP_02	95310	22609
ibm22uP_03	8827	3152
ibm22uP_04	15777	1931
ibm22uP_05	17561	4519
ibm22uP_06	20150	3299
ibm22uP_07	24506	6222
ibm22uP_08	124922	47285
ibm22uP_09	17793	6608
ibm22uP_10	12912	697
ibm22uP_11	17405	4128
ibm22uP_12	9334	2383
ibm22uP_13	7293	826
ibm22uP_14	74976	15579

results show the effectiveness of the proposed *ranking* method that enables the use of a sign-off timer for the cell selection algorithm.

The initial lambda tuning method is also analyzed. Table 6.8 shows power results without the initial lambda tuning iterations. Column "LR Cvg" shows when LR could converge to a better solution ("Y") or did not provide a solution better than the initial state ("N"). Comparing to Table 6.5, we see a major quality degradation mainly due to lack of convergence of the LR method. LR is not able to converge for half of the designs, where

Table 6.8: Experiments without lambda tuning iterations.

Design	Leakage Power (mW)			Dynamic Power (mW)			LR
	before	after	diff	before	after	diff	Cvg
ibm22uP_01	61.16	58.54	-4.3%	12.60	12.60	0.0%	Y
ibm22uP_02	80.64	66.99	-16.9%	13.47	13.41	-0.4%	N
ibm22uP_03	2.76	2.73	-0.8%	51.35	50.83	-1.0%	N
ibm22uP_04	19.14	18.37	-4.0%	45.26	45.22	-0.1%	N
ibm22uP_05	14.81	12.91	-12.8%	33.06	33.23	0.5%	Y
ibm22uP_06	20.30	16.66	-17.9%	61.20	60.86	-0.6%	Y
ibm22uP_07	35.31	29.38	-16.8%	21.54	21.55	0.0%	Y
ibm22uP_08	88.39	78.97	-10.7%	305.36	304.94	-0.1%	N
ibm22uP_09	16.53	13.77	-16.7%	68.29	68.21	-0.1%	N
ibm22uP_10	8.18	8.18	-0.1%	9.68	9.68	0.0%	Y
ibm22uP_11	4.25	3.89	-8.5%	20.46	19.99	-2.3%	Y
ibm22uP_12	1.07	1.04	-2.2%	1.28	1.23	-4.3%	Y
ibm22uP_13	1.59	1.59	-0.1%	1.35	1.34	-0.2%	N
ibm22uP_14	37.84	37.18	-1.7%	112.03	111.98	0.0%	N
Avg.			-8.1%			-0.6%	
Total			-10.7%			-0.2%	

all power improvement is given by PR[2]. $TTNS$ degradation for this set of results is more than 50% greater than results using initial lambda tuning. This shows the importance of the warm start with the initial lambda tuning.

[2]This implementation resembles Table 6.5, where PR implementation degrades $TTNS$.

7 CONCLUSION

The cell selection problem has been extensively explored by both industry and academia in recent years. Since optimal solutions cannot be found by a polynomial-time algorithm, many techniques have been proposed, with a variety of strengths and weaknesses.

Most academic works focus on simplified models that can no longer be applied to real-life designs. Several challenges found when applying cell selection algorithms in industrial designs are rarely addressed and are not fully exposed. Lagrangian relaxation-based algorithms are found in many publications despite the loss of its optimality for the discrete sizing and non-convex timing models. Another common approach is to model the problem as continuous optimization with later discretization, since the optimal solution can be found. However, discretization algorithms are suboptimal and usually lead to a solution far from optimal.

Recent works show the importance of using a sign-off timer during the optimization. Sign-off timing analysis is of extreme importance in the late optimization stages of industrial designs, where interconnection parasitics can already be extracted or accurately estimated. However, that leads to excessive runtime when using the full capabilities of STA, including multiple clock domains, multiple working frequencies, etc.

New algorithms and methods are needed to enable power and area optimization in late stages of industrial flows for high-performance designs. In this work we presented our state-of-the-art academic gate sizing and V_t assignment methodology that achieved the 1^{st} Place Award in the ISPD 2013 Contest. This methodology outperforms all other methods found in literature in all ISPD 2013 Contest benchmarks.

This work focused on the extension of our cell selection methodology in order to handle industrial designs in the post-placement and post-clock tree synthesis stages of an industrial design flow for high-performance microprocessors. For this application, the proposed techniques reduce leakage power by up to 18.2%, with average reduction of 10.4% without any degradation in timing quality. Also, up to 21.8% leakage power reduction is achieved in the presence of some $TTNS$ degradation. Cell area is also reduced, in average by 2.2% (with $TTNS$ degradation) and 1.7% (no $TTNS$ degradation), with up to 6.8% reduction for the latter case.

These results show the effectiveness of the proposed methodology and the improvements in this work over other works found in the literature.

REFERENCES

ABATO, R. et al. **Incremental timing analysis**. Google Patents, 1996. US Patent 5,508,937. Available from Internet: <http://www.google.com/patents/US5508937>.

AFONSO, R. et al. Power efficient standard cell library design. In: **Circuits and Systems Workshop,(DCAS), 2009 IEEE Dallas**. [S.l.: s.n.], 2009. p. 1–4.

ALDOUS, D.; VAZIRANI, U. V. ``go with the winners" algorithms. In: **35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994**. [s.n.], 1994. p. 492–501. Available from Internet: <http://dx.doi.org/10.1109/SFCS.1994.365742>.

ALPERT, C. J.; DEVGAN, A.; KASHYAP, C. A two moment rc delay metric for performance optimization. In: **Proceedings of the 2000 International Symposium on Physical Design**. New York, NY, USA: ACM, 2000. (ISPD '00), p. 69–74. Available from Internet: <http://doi.acm.org/10.1145/332357.332377>.

BAZARAA, M. S.; SHETTY, C. M. **Nonlinear Programming: Theory and Algorithms**. New York: Wiley, 1979.

BEECE, D. K. et al. Transistor sizing of custom high-performance digital circuits with parametric yield considerations. In: **47th annual ACM IEEE Design Automation Conference, DAC'2010**. Anaheim, California, USA: [s.n.], 2010. p. 781–786.

BEEFTINK, F. et al. Gate-size selection for standard cell libraries. In: **1998 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1998**. San Jose, California, USA: [s.n.], 1998. p. 545–550.

BERKELAAR, M.; BUURMAN, P.; JESS, J. Computing the entire active area /power consumption versus delay trade-off curve for gate sizing with a piecewise linear simulator. In: **Computer-Aided Design, 1994., IEEE/ACM International Conference on**. [S.l.: s.n.], 1994. p. 474–480.

BERKELAAR, M.; BUURMAN, P.; JESS, J. Computing the entire active area/power consumption versus delay tradeoff curve for gate sizing with a piecewise linear simulator. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 15, n. 11, p. 1424–1434, Nov 1996.

BERKELAAR, M.; JESS, J. Gate sizing in mos digital circuits with linear programming. In: **Design Automation Conference, 1990., EDAC. Proceedings of the European**. [S.l.: s.n.], 1990. p. 217–221.

BHASKER, J.; CHADHA, R. **Static Timing Analysis for Nanometer Designs: A Practical Approach**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009.

BORAH, M.; OWENS, R.; IRWIN, M. Transistor sizing for low power cmos circuits. **IEEE Transactions on Computer-Aided Design of Integrated circuits and Systems**, v. 15, n. 6, p. 665–671, 1996.

BORAH, M.; OWENS, R. M.; IRWIN, M. J. Transistor sizing for minimizing power consumption of cmos circuits under delay constraint. In: **1995 International Symposium on Low Power Design**. Dana Point, California - USA: [s.n.], 1995. p. 167–172.

BOYD, S. et al. Digital circuit optimization via geometric programming. **Operations Research**, v. 53, n. 6, p. 899–932, Nov.-Dec. 2005.

CELIK, M.; PILEGGI, L.; ODABASIOGLU, A. **IC Interconnect Analysis**. Springer, 2002. Available from Internet: <https://books.google.com.br/books?id=8Nu7STiANNsC>.

CHAN, P. Algorithms for library-specific sizing of combinational logic. In: **Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE**. [S.l.: s.n.], 1990. p. 353–356.

CHEN, C. P.; CHU, C. C.-N.; WONG, D. F. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. **IEEE Trans. on Computer-Aided Design**, v. 18, n. 7, p. 1014–1025, 1999.

CHEN, W.; HSIEH, C.-T.; PEDRAM, M. Gate sizing with controlled displacement. In: **Proceedings of the 1999 International Symposium on Physical Design**. New York, NY, USA: ACM, 1999. (ISPD ’99), p. 127–132. Available from Internet: <http://doi.acm.org/10.1145/299996.300038>.

CHINNERY, D.; KEUTZER, K. Linear programming for sizing, Vth and Vdd assignment. In: **ISLPED 2005**. [S.l.: s.n.], 2005. p. 149–154.

CHINNERY, D. G. **Low Power Design Automation**. Thesis (PhD) — EECS Department, University of California, Berkeley, Dec 2006. Available from Internet: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-182.html>.

CHOPRA, K. et al. Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. In: **Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on**. [S.l.: s.n.], 2005. p. 1023–1028.

CHOU, H.; WANG, Y.-H.; CHEN, C. C.-P. Fast and effective gate sizing with multiple-Vt assignment using generalized Lagrangian relaxation. In: **ASPDAC 2005**. [S.l.: s.n.], 2005. p. 381–386.

CIRIT, M. A. Transistor sizing in cmos circuits. In: **24th ACM/IEEE Design Automation Conference**. Miami Beach, Florida - USA: [s.n.], 1987. p. pp. 121–124.

COUDERT, O. Gate sizing: a general purpose optimization approach. In: **European Design and Test Conference, 1996. ED TC 96. Proceedings**. [S.l.: s.n.], 1996. p. 214–218.

COUDERT, O. Gate sizing for constrained delay/power/area optimization. **Very Large Scale Integration (VLSI) Systems, IEEE Transactions on**, v. 5, n. 4, p. 465–472, Dec 1997.

COUDERT, O.; HADDAD, R. Integrated resynthesis for low power. In: **Proceedings of the 1996 International Symposium on Low Power Electronics and Design**. Piscataway, NJ, USA: IEEE Press, 1996. (ISLPED '96), p. 169–174. Available from Internet: <http://dl.acm.org/citation.cfm?id=252493.252596>.

COUDERT, O.; HADDAD, R.; MANNE, S. New algorithms for gate sizing: a comparative study. In: **Design Automation Conference Proceedings 1996, 33rd**. [S.l.: s.n.], 1996. p. 734–739.

DAI, Z.-J.; ASADA, K. Mosiz: a two-step transistor sizing algorithm based on optimal timing assignment method for multi-stage complex gates. In: **Custom Integrated Circuits Conference, 1989., Proceedings of the IEEE 1989**. [S.l.: s.n.], 1989. p. 17.3/1–17.3/4.

DARTU, F.; MENEZES, N.; PILEGGI, L. Performance computation for precharacterized cmos gates with rc loads. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 15, n. 5, p. 544–553, May 1996.

DARTU, F. et al. A gate-delay model for high-speed cmos circuits. In: **Design Automation, 1994. 31st Conference on**. [S.l.: s.n.], 1994. p. 576–580.

DETJENS, E. et al. Technology mapping in mis. In: **Proc. of the Intl. Conf. of Computer Aided Design**. [S.l.: s.n.], 1987. p. 116–119.

DUTT, S.; REN, H. Timing yield optimization via discrete gate sizing using globally-informed delay pdfs. In: **Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on**. [S.l.: s.n.], 2010. p. 570–577.

ELFADEL, I. M.; LING, D. D. Zeros and Passivity of Arnoldi-Reduced-Order Models for Interconnect Networks. In: **DAC**. [s.n.], 1997. p. 28–33. Available from Internet: <http://doi.acm.org/10.1145/266021.266030>.

ELMORE, W. The transient analysis of damped linear networks with particular regard to wideband amplifiers. **J. Applied Physics**, v. 19, 1948.

FISHBURN, J. P.; DUNLOP, A. E. Tilos: A posynomial programming approach to transistor sizing. In: **Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design**. [S.l.: s.n.], 1985. p. 326–328.

FLACH, G. et al. Simultaneous gate sizing and vth assignment using lagrangian relaxation and delay sensitivities. In: **2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)**. [s.n.], 2013. p. 84–89. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6654627>.

FLACH, G. et al. Effective Method for Simultaneous Gate Sizing and V_{th} Assignment Using Lagrangian Relaxation. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 33, n. 4, p. 546–557, April 2014. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6774518>.

FLACH, G. A. **Discrete gate sizing and timing-driven detailed placement for the design of digital circuits**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, 2015. Available from Internet: <http://hdl.handle.net/10183/134330>.

GHIASI, S. et al. A unified theory of timing budget management. In: **Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on**. [S.l.: s.n.], 2004. p. 653–659.

GLASSER, L. A.; HOYTE, L. P. Delay and power optimization in vlsi circuits. In: **Proceedings of the 21st Design Automation Conference**. Piscataway, NJ, USA: IEEE Press, 1984. (DAC '84), p. 529–535. Available from Internet: <http://dl.acm.org/citation.cfm?id=800033.800849>.

GUNTZEL, J. L. A. **Functional Timing Analysis of VLSI Circuits Containing Complex Gates**. Thesis (PhD) — Universidade Federal do Rio Grande do Sul, RS-Brazil, 2000.

GUPTA, R. et al. The elmore delay as bound for rc trees with generalized input signals. In: **DAC '95: Proceedings of the 32nd annual ACM/IEEE Design Automation Conference**. New York, NY, USA: ACM, 1995. p. 364–369.

HARRIS, D. et al. The fanout-of-4 inverter delay metric. **Unpublished Manuscript. http://odin.ac.hmc.edu/harris/research**, 1997.

HASHIMOTO, M.; ONODERA, H.; TAMARU, K. A power optimization method considering glitch reduction by gate sizing. In: **Proceedings of the 1998 International Symposium on Low Power Electronics and Design**. New York, NY, USA: ACM, 1998. (ISLPED '98), p. 221–226. Available from Internet: <http://doi.acm.org/10.1145/280756.280907>.

HEDLUND, K. Aesop: A tool for automated transistor sizing. In: **Design Automation, 1987. 24th Conference on**. [S.l.: s.n.], 1987. p. 114–120.

HELD, S. Gate sizing for large cell-based designs. In: **Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.** [S.l.: s.n.], 2009. p. 827–832.

HITCHCOCK, R.; SMITH, G. L.; CHENG, D. D. Timing analysis of computer hardware. **IBM Journal of Research and Development**, v. 26, n. 1, p. 100–105, Jan 1982.

HITCHCOCK R.B., S. Timing verification and the timing analysis program. In: **Design Automation, 1982. 19th Conference on**. [S.l.: s.n.], 1982. p. 594–604.

HOPPE, B. et al. Optimization of high-speed cmos logic circuits with analytical models for signal delay, chip area and dynamic power dissipation. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, USA, v. 9, n. 3, p. 236–246, 1990.

HU, J. et al. Sensitivity-guided metaheuristics for accurate discrete gate sizing. In: **Proceedings of the International Conference on Computer-Aided Design**. New York, NY, USA: ACM, 2012. (ICCAD '12), p. 233–239. Available from Internet: <http://doi.acm.org/10.1145/2429384.2429428>.

HU, S.; KETKAR, M.; HU, J. Gate sizing for cell library-based designs. In: **Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE**. [S.l.: s.n.], 2007. p. 847–852.

HU, S.; KETKAR, M.; HU, J. Gate sizing for cell-library-based designs. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 28, n. 6, p. 818–825, June 2009.

HUANG, Y.-L.; HU, J.; SHI, W. Lagrangian relaxation for gate implementation selection. In: **Proceedings of the 2011 International Symposium on Physical Design**. New York, NY, USA: ACM, 2011. (ISPD '11), p. 167–174. Available from Internet: <http://doi.acm.org/10.1145/1960397.1960436>.

KAHNG, A. et al. High-performance gate sizing with a signoff timer. In: **Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on**. [S.l.: s.n.], 2013. p. 450–457.

KAO, W.; FATHI, N.; LEE, C.-H. Algorithms for automatic transistor sizing in cmos digital circuits. In: **Design Automation, 1985. 22nd Conference on**. [S.l.: s.n.], 1985. p. 781–784.

KAO, W.; MOVAHED-EZAZI, M.; SABIERS, M. Aries: A workstation based, schematic driven system for circuit design. In: **Design Automation, 1984. 21st Conference on**. [S.l.: s.n.], 1984. p. 301–307.

KASAMSETTY, K.; KETKAR, M.; SAPATNEKAR, S. A new class of convex functions for delay modeling and its application to the transistor sizing problem [cmos gates]. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 19, n. 7, p. 779–788, Jul 2000.

KASAMSETTY, M. K. K.; SAPATNEKAR, S. S. A new class of convex functions for delay modeling and their application to the transistor sizing problem. **IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems**, v. 19, n. 7, p. 779–788, 2000.

KASHYAP, C. V. et al. Peri: A technique for extending delay and slew metrics to ramp inputs. In: **Proceedings of the 8th ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems**. New York, NY, USA: ACM, 2002. (TAU '02), p. 57–62. Available from Internet: <http://doi.acm.org/10.1145/589411.589424>.

KIRKPATRICK, T.; CLARK, N. Pert as an aid to logic design. **IBM Journal of Research and Development**, v. 10, n. 2, p. 135–141, March 1966.

KURSUN, E.; GHIASI, S.; SARRAFZADEH, M. Transistor level budgeting for power optimization. In: **Quality Electronic Design, 2004. Proceedings. 5th International Symposium on**. [S.l.: s.n.], 2004. p. 116–121.

LEE, J.; GUPTA, P. Discrete circuit optimization. **Foundations and Trends® in Electronic Design Automation**, v. 6, n. 1, p. 1–120, 2012. Available from Internet: <http://www.nowpublishers.com/article/Details/EDA-019>.

LI, L. et al. An efficient algorithm for library-based cell-type selection in high-performance low-power designs. In: **Proceedings of the International Conference on Computer-Aided Design**. New York, NY, USA: ACM, 2012. (ICCAD '12), p. 226–232. Available from Internet: <http://doi.acm.org/10.1145/2429384.2429427>.

LI, W. Strongly np-hard discrete gate sizing problems. In: **Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on**. [S.l.: s.n.], 1993. p. 468–471.

LI, W. et al. On the circuit implementation problem [combinatorial logic circuits]. In: **Design Automation Conference, 1992. Proceedings., 29th ACM/IEEE**. [S.l.: s.n.], 1992. p. 478–483.

LI, W.-N. et al. On the circuit implementation problem. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 12, n. 8, p. 1147–1156, Aug 1993.

LIAO, C.; HU, S. Approximation scheme for restricted discrete gate sizing targeting delay minimization. **Journal of Combinatorial Optimization**, Springer US, v. 21, n. 4, p. 497–510, 2011. Available from Internet: <http://dx.doi.org/10.1007/s10878-009-9267-0>.

LIN, S.; MAREK-SADOWSKA, M.; KUH, E. Delay and area optimization in standard-cell design. In: **Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE**. [S.l.: s.n.], 1990. p. 349–352.

LIU, Y.; HU, J. A new algorithm for simultaneous gate sizing and threshold voltage assignment. In: **Proceedings of the 2009 International Symposium on Physical Design**. New York, NY, USA: ACM, 2009. (ISPD '09), p. 27–34. Available from Internet: <http://doi.acm.org/10.1145/1514932.1514940>.

LIU, Y.; HU, J. A new algorithm for simultaneous gate sizing and threshold voltage assignment. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 29, n. 2, p. 223–234, Feb 2010.

LIU, Y.; HU, J. Gpu-based parallelization for fast circuit optimization. **ACM Trans. Des. Autom. Electron. Syst.**, ACM, New York, NY, USA, v. 16, n. 3, p. 24:1–24:14, jun. 2011. Available from Internet: <http://doi.acm.org/10.1145/1970353.1970357>.

LORENA, L. A.; SENNE, E. L. F. Improving traditional subgradient scheme for lagrangean relaxation: an application to location problems. **International Journal of Mathematical Algorithms**, v. 1, p. 133–151, 1999.

MA, Q.; YOUNG, E. Network flow-based power optimization under timing constraints in msv-driven floorplanning. In: **Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on**. [S.l.: s.n.], 2008. p. 1–8.

MARPLE, D. Transistor size optimization in the tailor layout system. In: **Design Automation, 1989. 26th Conference on**. [S.l.: s.n.], 1989. p. 43–48.

MARPLE, D. P.; GAMAL., A. E. Optimal selection of transistor sizes in digital vlsi circuits. In: **Advanced Research in VLSI, Proceedings of the 1987 Stanford Conference**. [S.l.]: MIT Press, 1987. p. 151–172.

MARRANGHELLO, F. S. et al. Transistor sizing analysis of regular fabrics. In: **1st Exploiting Regularity in the Design of IPs, Architectures and Platforms Workshop, (ERDIAP 2011)**. Como, Italy: [s.n.], 2011. p. 235–242.

MATSON, M.; GLASSER, L. Macromodeling and optimization of digital mos vlsi circuits. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 5, n. 4, p. 659–678, October 1986.

MCNALL, K.; CASAVANT, A. Automatic operator configuration in the synthesis of pipelined architectures. In: **Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE**. [S.l.: s.n.], 1990. p. 174–179.

MENEZES, N.; BALDICK, R.; PILEGGI, L. A sequential quadratic programming approach to concurrent gate and wire sizing. In: **Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on**. [S.l.: s.n.], 1995. p. 144–151.

MENEZES, N.; BALDICK, R.; PILEGGI, L. A sequential quadratic programming approach to concurrent gate and wire sizing. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 16, n. 8, p. 867–881, Aug 1997.

NAGEL, L. W.; PEDERSON, D. **SPICE (Simulation Program with Integrated Circuit Emphasis)**. [S.l.], 1973. Available from Internet: <http://www.eecs.berkeley.edu/Pubs/TechRpts/1973/22871.html>.

NGUYEN, D. et al. Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization [logic ic design]. In: **Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on**. [S.l.: s.n.], 2003. p. 158–163.

NIKOUBIN, T. et al. Simple exact algorithm for transistor sizing of low-power high-speed arithmetic circuits. **VLSI Design**, New York, v. 2010, 2010.

NILSSON, J.; RIEDEL, S. **Electric Circuits**. Pearson/Prentice Hall, 2005. Available from Internet: <https://books.google.com.br/books?id=1Cs9kgEACAAJ>.

O'BRIEN, P.; SAVARINO, T. Modeling the driving-point characteristic of resistive interconnect for accurate delay estimation. In: **Computer-Aided Design, 1989. ICCAD-89. Digest of Technical Papers., 1989 IEEE International Conference on**. [S.l.: s.n.], 1989. p. 512–515.

ODABASIOGLU, A.; CELIK, M.; PILEGGI, L. Prima: passive reduced-order interconnect macromodeling algorithm. In: **Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on**. [S.l.: s.n.], 1997. p. 58–65.

OZDAL, M.; BURNS, S.; HU, J. Gate sizing and device technology selection algorithms for high-performance industrial designs. In: **Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on**. [S.l.: s.n.], 2011. p. 724–731.

OZDAL, M.; BURNS, S.; HU, J. Algorithms for gate sizing and device parameter selection for high-performance designs. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 31, n. 10, p. 1558–1571, Oct 2012.

OZDAL, M. M. et al. The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite. In: **ISPD 2012**. Napa, CA, EUA: [s.n.], 2012. p. 161–164.

OZDAL, M. M. et al. An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest. In: **ISPD 2013**. The Ridge Tahoe, Stateline, NV, EUA: [s.n.], 2013. p. 168–170.

PILLAGE, L.; HUANG, X.; ROHRER, R. Awesim: Asymptotic waveform evaluation for timing analysis. In: **Design Automation, 1989. 26th Conference on**. [S.l.: s.n.], 1989. p. 634–637.

PILLAGE, L.; ROHRER, R. Asymptotic waveform evaluation for timing analysis. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 9, n. 4, p. 352–366, Apr 1990.

PINCUS, J. D.; DESPAIN, A. M. Delay reduction using simulated annealing. In: **Proceedings of the 23rd ACM/IEEE Design Automation Conference**. Piscataway, NJ, USA: IEEE Press, 1986. (DAC '86), p. 690–695. Available from Internet: <http://dl.acm.org/citation.cfm?id=318013.318141>.

POSSER, G. et al. Gate sizing using geometric programming. **Analog Integrated Circuits and Signal Processing**, v. 73, n. 3, p. 831–840, 2012.

PURI, R.; KUNG, D. S.; DRUMM, A. D. Fast and accurate wire delay estimation for physical synthesis of large ASICs. In: **Proceedings of the 12th ACM Great Lakes Symposium on VLSI**. New York, NY, USA: ACM, 2002. (GLSVLSI '02), p. 30–36. Available from Internet: <http://doi.acm.org/10.1145/505306.505314>.

QIAN, H.; ACAR, E. Timing-aware power minimization via extended timing graph methods. **ASP Journal of Low Power Electronics**, p. 318–326, 2007.

QIAN, J.; PULLELA, S.; PILLAGE, L. Modeling the "Effective capacitance" for the RC interconnect of CMOS gates. **Trans. Comp.-Aided Des. Integ. Cir. Sys.**, IEEE Press, Piscataway, NJ, USA, v. 13, n. 12, p. 1526–1535, nov. 2006. Available from Internet: <http://dx.doi.org/10.1109/43.331409>.

RABAEY, J. M.; CHANDRAKASAN, A. P.; NIKOLIC, B. **Digital Integrated Circuits**. [S.l.]: Prentice-Hall, 2002.

RAHMAN, M. et al. Design automation tools and libraries for low power digital design. In: **Circuits and Systems Workshop (DCAS), 2010 IEEE Dallas**. [S.l.: s.n.], 2010. p. 1–4.

RAHMAN, M.; SECHEN, C. Post-synthesis leakage power minimization. In: **Design, Automation Test in Europe Conference Exhibition (DATE), 2012**. [S.l.: s.n.], 2012. p. 99–104.

RAHMAN, M.; TENNAKOON, H.; SECHEN, C. Power reduction via near-optimal library-based cell-size selection. In: **Design, Automation Test in Europe Conference Exhibition (DATE), 2011**. [S.l.: s.n.], 2011. p. 1–4.

RAHMAN, M.; TENNAKOON, H.; SECHEN, C. Library-based cell-size selection using extended logical effort. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 32, n. 7, p. 1086–1099, July 2013.

RATZLAFF, C.; PILLAGE, L. Rice: rapid interconnect circuit evaluation using awe. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 13, n. 6, p. 763–776, Jun 1994.

REIMANN, T. et al. Simultaneous gate sizing and vt assignment using fanin/fanout ratio and simulated annealing. In: **2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)**. [s.n.], 2013. p. 2549–2552. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6572398>.

REIMANN, T.; SZE, C. C.; REIS, R. Challenges of cell selection algorithms in industrial high performance microprocessor designs. **Integration, the {VLSI} Journal**, v. 52, p. 347 – 354, 2016. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S0167926015001169>.

REIMANN, T.; SZE, C. C. N.; REIS, R. Gate sizing and threshold voltage assignment for high performance microprocessor designs. In: **The 20th Asia and South Pacific Design Automation Conference**. [s.n.], 2015. p. 214–219. Available from Internet: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7059007>.

REIMANN, T. J.; SZE, C. C.; REIS, R. Cell selection for high-performance designs in an industrial design flow. In: **Proceedings of the 2016 on International Symposium on Physical Design**. Santa Rosa, California, USA: ACM, 2016. (ISPD '16), p. 65–72. Available from Internet: <http://doi.acm.org/10.1145/2872334.2872358>.

REN, H.; DUTT, S. A network-flow based cell sizing algorithm. In: **The International Workshop on Logic Synthesis**. [S.l.: s.n.], 2008. p. 7–14.

REN, H.; DUTT, S. Effective power optimization under timing and voltage-island constraints via simultaneous v_{dd}, v_{th} assignments, gate sizing, and placement. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 30, n. 5, p. 746–759, May 2011.

ROY, S.; CHEN, C.-P.; CHEN, C. Convexfit: an optimal minimum-error convex fitting and smoothing algorithm with application to gate-sizing. In: **Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on**. [S.l.: s.n.], 2005. p. 196–203.

ROY, S. et al. Numerically convex forms and their application in gate sizing. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 26, n. 9, p. 1637–1647, Sept 2007.

SANTOS, C. et al. Effects of using a pin-to-pin delay model on a library-free transistor/gate sizing scheme. In: **IEEE International Midwest Symposium on Circuits and Systems, MSCAS 2005**. Covington, KY: [s.n.], 2005/a. v. 1, p. 315–318.

SANTOS, C. et al. Incremental timing optimization for automatic layout generation. In: **IEEE International Symposium on Circuits and Systems, ISCAS 2005**. Kobe, Japan: [s.n.], 2005/b. v. 4, p. 3567–3570.

SANTOS, C. et al. A transistor sizing method applied to an automatic layout generation tool. In: **16th Symposium on Integrated Circuits and Systems Design, SBCCI 2003**. São Paulo: [s.n.], 2003. p. 303–307.

SANTOS, C. L. dos. **Verificação e Otimização de Atraso durante a Síntese Física de Circuitos Integrados CMOS**. Dissertation (Master) — PPGC - UFRGS, Porto Alegre, RS, 2005.

SAPATNEKAR, S.; RAO, V.; VAIDYA, P. A convex optimization approach to transistor sizing for cmos circuits. In: **IEEE International Conference on Computer-Aided Design, ICCAD 1991**. Santa Clara, CA , USA: [s.n.], 1991. p. 482–485.

SAPATNEKAR, S. S. **Timing**. [S.l.]: Springer US, 2004.

SAPATNEKAR, S. S. et al. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. **IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems**, v. 12, n. 11, p. 1621–1634, 1993.

SHAH, S. et al. Discrete vt assignment and gate sizing using a self-snapping continuous formulation. In: **Computer-Aided Design, 2005. ICCAD-2005. IEEE/ACM International Conference on**. [S.l.: s.n.], 2005. p. 705–712.

SHARMA, A. et al. Fast Lagrangian Relaxation Based Gate Sizing Using Multi-Threading. In: **Proceedings of the IEEE/ACM International Conference on Computer-Aided Design**. Austin, TX, USA: [s.n.], 2015. (ICCAD '15), p. 426–433. Available from Internet: <http://dl.acm.org/citation.cfm?id=2840819.2840879>.

SHYU, J.-M. et al. Optimization-based transistor sizing. **Solid-State Circuits, IEEE Journal of**, v. 23, n. 2, p. 400–409, April 1988.

SILVA, M. Sparse matrix storage revisited. In: **Proceedings of the 2nd conference on Computing frontiers**. New York, NY, USA: ACM, 2005. (CF '05), p. 230–235. Available from Internet: <http://doi.acm.org/10.1145/1062261.1062299>.

SILVEIRA, L.; KAMON, M.; WHITE, J. Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-d interconnect structures. **Components, Packaging, and Manufacturing Technology, Part B: Advanced Packaging, IEEE Transactions on**, v. 19, n. 2, p. 283–288, May 1996.

SILVEIRA, L. M. et al. A coordinate-transformed arnoldi algorithm for generating guaranteed stable reduced-order models of RLC circuits. In: **ICCAD**. [s.n.], 1996. p. 288–294. Available from Internet: <http://dx.doi.org/10.1109/ICCAD.1996.569710>.

SINGH, J.; LUO, Z.-Q.; SAPATNEKAR, S. A geometric programming-based worst case gate sizing method incorporating spatial correlation. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 27, n. 2, p. 295–308, Feruary 2008.

SINGH, J. et al. Robust gate sizing by geometric programming. In: **Design Automation Conference, 2005. Proceedings. 42nd**. [S.l.: s.n.], 2005. p. 315–320.

SIRICHOTIYAKUL, S. et al. Stand-by power minimization through simultaneous threshold voltage selection and circuit sizing. In: **Design Automation Conference, 1999. Proceedings. 36th**. [S.l.: s.n.], 1999. p. 436–441.

SRIVASTAVA, A.; SYLVESTER, D. **Statistical Analysis and Optimization for VLSI: Timing and Power**. Boston, MA :: Springer US,, 2005. (Series on Integrated Circuits and Systems). Available from Internet: <http://dx.doi.org/10.1007/b137645>.

SRIVASTAVA, A.; SYLVESTER, D.; BLAAUW, D. Power minimization using simultaneous gate sizing, dual-vdd and dual-vth assignment. In: **Design Automation Conference, 2004. Proceedings. 41st**. [S.l.: s.n.], 2004. p. 783–787.

SUNDARARAJAN, V.; SAPATNEKAR, S.; PARHI, K. Fast and exact transistor sizing based on iterative relaxation. **Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on**, v. 21, n. 5, p. 568–581, May 2002.

SUTHERLAND, I.; SPROULL, B.; HARRIS, D. **Logical Effort: Designing Fast CMOS Circuits**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

SYNOPSYS. **PrimeTime® User Guide: Advanced Timing Analysis, Version V-2004.06**. 2004. Available from Internet: <http://www.synopsys.com>.

TENNAKOON, H.; SECHEN, C. Gate sizing using lagrangian relaxation combined with a fast gradient-based pre-processing step. In: **ICCAD 2002**. [S.l.: s.n.], 2002. p. 395–402.

TENNAKOON, H.; SECHEN, C. Efficient and accurate gate sizing with piecewise convex delay models. In: **42nd annual conference on Design automation**. Anaheim, California, USA: [s.n.], 2005. p. 807–812.

WANG, J.; DAS, D.; ZHOU, H. Gate sizing by lagrangian relaxation revisited. **IEEE Trans. on Computer-Aided Design**, v. 28, n. 7, p. 1071–1084, July 2009.

YOSHIDA, H.; FUJITA, M. Performance-constrained transistor sizing for different cell count minimization. **Information and Media Technologies**, v. 6, n. 1, p. 1–11, 2011.