Novel Discrete Optimization Techniques with Applications to Complex Physical Synthesis Problems in EDA

BY

HUAN REN
B.S., Zhejiang University, Hangzhou, China, 2005
M.S. University of Illinois at Chicago, Chicago, IL, 2011

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2012

Chicago, Illinois

Defense Committee:

Shantanu Dutt, Chair and Advisor
Wenjing Rao
Kaijie Wu
John Lillis, Computer Science
Prashant Banerjee, Mechanical and Industrial Engineering

ACKNOWLEDGMENT

I would like to thank my advisor Prof. Shantanu Dutt for all the discussions, guidance and ideas on my Ph.D research during the past six and a half years (the last one year of which I was working full time, first at Intel, and then more recently at Synopsys; my interactions with Prof. Dutt continued during this time). This research and thesis could not possibly have been completed without his tremendous help. I would also like to extend my gratitude to my dissertation committee, Prof. Wenjing Rao, Prof. Kaijie Wu, Prof. John Lillis and Prof. Prashant Banerjee for valuable suggestions on shaping my thesis.

This thesis is dedicated to my parents. My family has always been the source of my strength and inspiration.

Thanks to all my former and current colleagues from the ECE department and DART Lab for their support within academics and beyond it.

HR

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

TABLE **PAGE**

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

SUMMARY

VLSI circuit designs are very challenging optimization problems in the electronic design automation (EDA) domain. There are usually 100K's to tens of millions of components in such circuits, and a wide range of metrics that need to be considered. A post place-and-route (P&R) phase called "physical synthesis" (PS) is a crucial stage where effective optimization of VLSI circuits can be performed using accurate interconnect metrics. Many design transforms have been developed to perform different types of optimizations in PS. These include cell replacement, cell sizing, cell replication, buffer insertion, supply voltage assignment and threshold voltage assignment. We have developed a novel and efficient method called "discretized network flow (DNF)" for the simultaneous application of multiple transforms on the entire circuit with tractable runtimes. This enables us to achieve an average of 10% and 16% improvement for delay and power, respectively, compared to the state-of-the-art academic or industry tools that apply the transforms sequentially. Application of DNF to the timing yield PS problem resulted in a relative yield improvement of about 16% over a state-of-the-art academic method.

DNF was also applied successfully to solve 0/1 integer linear programming problems, achieving an average speedup of 19X over the state-of-the-art academic tool SCIP with a similar optimality gap. Besides DNF, we also developed a dynamic programming method using the novel concept of weak domination for solving 0/1 integer non-linear programming problems with a guaranteed optimality gap. Compared to a state-of-the-art academic tool Bonmin with the same optimality gap, our method achieves a speedup of about 2X.

CHAPTER 1

INTRODUCTION

1.1 Physical Synthesis

As the sizes of cells in an integrated circuit keep shrinking and the number of cells keeps increasing, the length of physical interconnects between cells on a chip are becoming much larger than the dimensions of cells. Therefore, interconnects are taking more and more important roles in determining many critical metrics for integrated circuits. As a result, a design that satisfies various constraints after logic synthesis will not necessarily meet these constraints after place-and-route due to wire delay and wire power. To handle this problem, *physical synthesis* has emerged as a necessary tool for design closure. Physical synthesis begins with a placed or routed netlist, in which more accurate interconnect info is available than in the logic synthesis stage. It tries various optimization methods to the netlist, and the goal is to satisfy constraints on a set of important metrics, and optimize another set of metrics. It uses the optimization methods that are used in the logical synthesis stage as well as methods that are only applicable in place-and-route stage like changing cell position. One main difference between physical synthesis and logical synthesis is that physical synthesis will take available interconnect information into account when performing optimization.

In current industry place-and-route flow, to ensure design closure of complex circuits, physical synthesis is usually performed multiple times, one after each major step that changes the

physical interconnects. Typical times that a physical synthesis is performed are after placement, after clock tree synthesis, and after routing. Each time, the physical synthesis is performed to eliminate the constraint violations produced in the previous stage.

In the next section, we will first go through the important design metrics that are usually considered in physical synthesis.

1.2 Design Metrics

Modern integrated circuit is a very complex system. In order for it to function as expected, many different metrics need to be considered. We will provide the definitions and calculation methods for these metrics in this section.

1.2.1 Delay

The delay of a path in a circuit can be divided into two parts: the cell internal delay, and the net delay. The cell internal delay is usually given in the library. For post-route design, the net delay can be calculated using the Elmore delay model [2], or more accurately the Arnoldi [3] delay model. In post-placement but pre-route design, the capacitance, resistance and length of a net is usually estimated using the net's bounding box, and usually simple lumped capacitance and resistance model is used in delay calculation.

We have proposed a novel post-placement pre-route delay model called the γ *net delay model* in [4] that has better correlation to the post-route delay. In this mode, for an unrouted net, we assume the net routing pattern with a single trunk to the furthest sink with branches off this trunk to the other sinks, as shown in Figure 1. For a net n_j with driver u_d, and $k-1 \geq 1$ sinks (k is the total number of pins in n_j), let R_d be the driving resistance of u_d, C_g the load

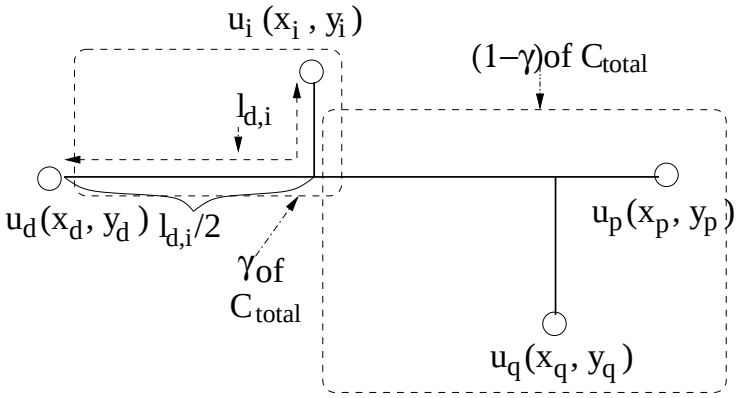

Figure 1. The γ net delay model for post-placement unrouted nets; C_{total} is the total (net and load) capacitance seen by the driver.

capacitance of a sink pin[1], r (c) the unit wire resistance (capacitance), $L(n_j)$ the total WL of n_j, and $l_{d,i}$ the interconnect length connecting driver u_d to sink u_i; see Figure 1. Referring to this figure and considering a sink u_i in n_j, the delay $D(u_i, n_j)$ to it (using the Elmore delay model) from the driver u_d, consists of three parts:

$$D_1(n_j) = R_d(c \cdot L(n_j) + (k-1)C_g) \tag{1.1}$$

$$D_2(u_i, n_j) = \frac{rc}{2} \cdot l_{d,i}^2 + r \cdot l_{d,i} C_g \tag{1.2}$$

$$D_3(u_i, n_j) = r \cdot (l_{d,i}/2)((1 - \gamma + \gamma/2)(c \cdot L(n_j) + (k-2)C_g) \tag{1.3}$$

---

[1]For simplicity of exposition, we assume uniform loads for all sink pins, though clearly our net-delay model also applies to non-uniform loads.

Circuit	mac32	matrix	vp2	mac64	%error
routed delay (ns)	3.4	3.8	4.3	6.7	0
γ delay model (ns)	3.4	4.3	4.5	7.0	5.6

TABLE I

THE CRITICAL PATH DELAY CALCULATED USING THE γ NET DELAY MODEL.

$$D(u_i, n_j) = D_1(n_j) + D_2(u_i, n_j) + D_3(u_i, n_j) \tag{1.4}$$

where $\gamma \leq 1$. Note that the $D_1(n_j)$ delay component is common to all sinks of n_j, which is due to the driving resistance of the driver and total capacitance load of wire and sinks. $D_2(u_i, n_j)$ is the wire RC delay from driver to a sink u_i. The idea behind the 3rd delay component $D_3(u_i, n_j)$ is that without an exact route, we estimate that if u_i lies in the initial γ fraction of the HPBB of n_j starting from the driver position, then, on the average, half of the interconnect length $l_{d,i}$ lies on the main trunk of the estimated route, and it "sees" the entire wire and sink capacitance of the rest of the $(1 - \gamma)$ fraction of the net. Furthermore, this main trunk part of the (u_d, u_i) interconnect can also see incremental portions of the γ fraction of the net capacitance, i.e., from the branch point to u_i, which ultimately results in this interconnect seeing a $\gamma/2$ fraction of the total (load + net) capacitance C_{total}.

The comparison between the delay estimation using our γ-delay model, and the actual routed delay is provided in Table I. The delay estimation error is only 5.6%, which shows the

accuracy of our γ net delay model. Furthermore, our delay model shows 100% fidelity with routed delay.

1.2.2 Power

The power consumption of a circuit mainly consists of dynamic power consumption and leakage power consumption. The average dynamic power $P_d(n_j)$ consumed by a net n_j with a driving gate g_d and a set of sink gates g_i is given by:

$$P_d(n_j) = 0.5f \cdot p_{sw}(n_j)(\sum_{g_i \in ip(n_j)} C(g_i)V_{dd}^2(g_d) + V_{dd}^2(g_d)C(n_j)) \tag{1.5}$$

where f is the clock frequency, $p_{sw}(n_j)$ is the switching probability of the net, $ip(n_j)$ is the set of sink gates in n_j, $C(g_i)$ is the input capacitance of sink gate g_i, $V_{dd}(g_d)$ is the supply voltage V_{dd} of drive gate g_d and $C(n_j)$ is the total interconnect capacitance of n_j. The leakage power consumption $P_l(g_i)$ of a gate g_i can be given as [5]:

$$P_l(g_i) = V_{dd}(g_i)WI_s \cdot e^{-V_{th}(g_i)/V_o} \tag{1.6}$$

where W is the transistor width and is proportional to the gate size, I_s is the zero-threshold leakage current and a constant, $V_{th}(g_i)$ is the threshold voltage V_{th} of g_i, and V_o is the constant subthreshold slope. Hence, $P_l(g_i)$ is linearly proportional to the gate size and V_{dd}, and inversely exponentially proportional to V_{th}.

It should be noted that when a low-V_{dd} gate g_d drives a high-V_{dd} gate g_i, a *level shifter/converter* needs to be inserted between them to avoid deterioration of the noise margin. Adding a level

shifter increases power consumption in three ways: 1) level shifters consume power, 2) the charging voltage for the interconnect between the level shifter and the sink cell g_i is increased, and 3) the charging voltage for the sink cell is increased; the last two increases are due to the charging voltage changing from $V_{dd}(g_d)$ to $V_{dd}(g_i)$). The second effect can be eliminated by putting the level shifter close to the sink cell so that the interconnect length between the level shifter and the sink cell is almost 0. Let the P_s be the power consumption of a level shifter given in the library. To account for the first and third effects, the power term for charging a sink gate g_i ($C(g_i)V_{dd}^2(g_d)$) in Equation 1.5 should be changed to the following formulation denoted by $P_c(g_d, g_i)$:

$$P_c(g_d, g_i) = \begin{cases} C(g_i)V_{dd}^2(g_d) & \text{if } V_{dd}(g_d) \geq V_{dd}(g_i) \\ P_s + C(g_i)V_{dd}^2(g_i) & \text{if } V_{dd}(g_d) < V_{dd}(g_i) \end{cases} \tag{1.7}$$

1.2.3 Yield

With the shrinking feature sizes of integrated circuits, it is becoming increasingly difficult to control critical device parameters during fabrication. Growing process variability has been observed in gate lengths, oxide thicknesses and doping both across dies and within the same die. Besides such process variations, temperature and supply voltage variability also causes significant performance and power variations to arise between chips with the same design. The *yield* of a design is the percentage of produced chips with the same design that can satisfy all requirements. It is important to develop design optimization techniques in logic synthesis

and physical synthesis that optimize critical metrics (e.g., timing and power) that take these process/voltage/temperature based circuit parameter variations into account so that the yield of resulting chips (the percentage of chips that meet the desired metric goals) is maximized.

To show the delay calculation when variation is considered, let us take a simple lumped delay model. The delay $d_o(g)$ at a gate g can be modeled as:

$$d_o(g) = d(g) + r(g) \times \sum fanoutsc(g_{fo}) \tag{1.8}$$

where $d(g)$ is the intrinsic gate delay of g, $r(g)$ is the driving resistance of g, and $c(g_{fo})$ is the input capacitance of a fanout gate g_{fo}. When random variations in these parameters are considered, the corresponding gate delay with variability $d_v(g)$ can be written as:

$$d_v(g) = d_o(g) + \Delta d(g) + \Delta r(g) \times \sum fanoutsc(g_{fo}) + r(g) \times \sum fanouts\Delta c(g_{fo}) \tag{1.9}$$

where $\Delta d(g)$, $\Delta r(g)$ and $\Delta c(g_{fo})$ are the random variations on parameters $d(g)$, $r(g)$ and $c(g_{fo})$, respectively. In this thesis, we assume all random variations have a Gaussian distribution. Hence, the delay at a gate also assumes a Gaussian distribution, since according to the Central Limit Theorem [6], the sum of Gaussian distributions is also a Gaussian distribution.

The delay of a design is the max of delays of all paths in the design, and, obviously, is also a random value when the variation is considered. The technique for obtain the probability distribution function (PDF) of the design delay is called the *statistical timing analysis* (SSTA).

In this thesis we are using the SSTA method proposed in [7]. This method approximates the distribution of the maximum of several Gaussian random variables as another Gaussian distribution based on the work in [8]. The detail of the approximation is given below:

Let A and B be two Gaussian variables, and (μ_A, σ_A) and (μ_B, σ_B) be their (mean, standard deviation). Also, let us define the following variables:

$$\begin{aligned} \phi(x) &= \frac{1}{\sqrt{2}} e^{-x^2/2} \\ \Phi(x) &= \int_{-\inf}^{y} \phi(x) dx \\ \theta &= |\sigma_A - \sigma_B| \\ \alpha &= (\mu_A - \mu_B)/\theta \end{aligned} \tag{1.10}$$

The mean and standard deviation for the approximating Gaussian distribution of $\max\{A, B\}$ are:

$$\begin{aligned} \mu(\max\{A,B\}) &= \mu_A \Phi(\alpha) + \mu_B \Phi(-\alpha) + \theta\phi(\alpha) \\ \sigma(\max\{A,B\}) &= (\mu_A^2 + \sigma_A^2)\Phi(\alpha) + (\mu_B^2 + \sigma_B^2)\Phi(-\alpha) \\ &\quad +(\mu_A + \mu_B)\theta\phi(\alpha) - \mu^2 \end{aligned} \tag{1.11}$$

Hence, with the above approximation the delay of a circuit also becomes a Gaussian random variable, whose μ and σ depend on gate parameters. Given a delay constraint D_c the following yield objective function is used this thesis:

$$Yd = (D_c - \mu)/\sigma \tag{1.12}$$

It is proved in [7] that the actual yield percentage is a monotonically increasing function of Yd. Hence, optimizing Yd accurately optimizes the final yield.

1.2.4 Area

The area of a design can be measured by either the total area of cells in the design or the layout area of the design. The latter one also includes the area of whitespaces in the design that are reserved to alleviate congestion. It is an important metric, since the chip cost is usually proportional to its area.

1.3 Physical Synthesis Transforms

Physical synthesis is usually done through trying various changes on a target design. Through out this thesis, we will call these changes *transforms*. The typical type of transforms include: incremental placement, cell sizing, multiple threshold voltages (V_{th}) assignment, multiple supply voltages (V_{dd}) assignment, two types of buffer insertion and cell replication.

Incremental Placement This method can reduce the critical and near critical path lengths by changing the position of cells on these paths. It can also reduce the dynamic power by reducing capacitive loads of high switching frequency wires. The advantage of incremental placement

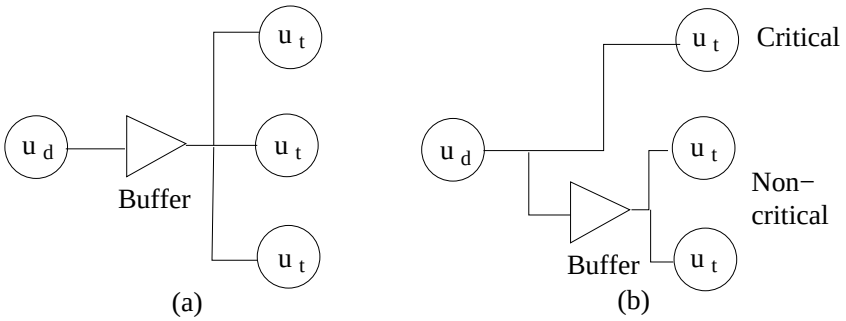

Figure 2. Two types of buffers. (a) Type 1 buffer for improving drive capability. (b) Type 2 buffer for isolating non-critical sinks.

is that by only focusing on cells connected to critical, near-critical paths or on high switching frequency wires, the run time is greatly reduced compared to performing a new placement, and the optimization is more controllable.

Cell Sizing. The size of a cell controls its input capacitance and driving resistance. Changing input capacitances and driving resistances of cells are very effective in improving timing and power. If we are able to implement cells of any size, then the problem of choosing optimal cell sizes for timing can be solved by a convex programming approach. However, since in real standard cell designs, the available cell sizes in a library are limited, the problem is actually a discrete optimization problem, which can be solved by either a fastest descent method or dynamic programming.

Driver Buffer Insertion. Optimal buffer insertion has to take routing information into consideration. However, it is also useful to estimate the effect of buffers at the placement stage so that proper white-spaces can be allocated for adding buffers, and we can know earlier whether

the design can meet timing requirements. There are two types of buffer insertions as shown in Figure 2. Type 1 buffer shown in Figure 2(a) is used for improving the drive capability of the driving cell u_d of a net, and thus is called *driver buffer* in this thesis.

Isolating Buffer Insertion The second type of buffers are shown in Figure 2 (b). Its purpose is to isolating non-critical sinks from critical ones.

Cell Replication. In cell replication, the driving cell of a net is replaced by two identical replicas, and the fanouts are partitioned into two groups for each replica. Its effect is similar to doubling the driving cell size, but it can also achieve the purpose of separating the non-critical fanouts of a net from its critical ones by connecting them to different replicas. This makes it more effective in improving delay than up-sizing driving cells when a net has large non-critical load.

V_{th} Assignment. Reducing the threshold voltage of a cell is very effective in reducing its leakage power, as the leakage power is exponentially proportional(Equation 1.6) to the threshold voltage. However, a cell with a high threshold voltage also suffers from the increase in driver resistance and internal delay. Hence, in most current cell libraries, a set of different V_{th} implementations is provided for each cell with certain type and size. Appropriate selection of cells' V_{th}'s can help satisfy the delay constraint with the least leakage power.

V_{dd} Assignment. Reducing the supply voltage of a cell reduces both its leakage power and switching power efficiently. Since these powers are quadratically and linearly proportional to the supply voltage(Equation 1.5,Equation 1.6). However, reducing the supply voltage also increases the cells driver resistance. Furthermore, using multiple V_{dd}'s brings in the need for level shifters

and the issue of forming voltage island (we will explain this in Chapter 4). This also causes increase in delay, power and area. Hence, smart choices need to be made to satisfy the delay, power and area requirements at the same time.

1.4 Previous Work in Physical Synthesis

1.4.1 Methods for Applying A Single Transform

Physical synthesis has been a critical research area in recent years. Many methods have been proposed for efficiently using a single transform.

For cell sizing, it is used to optimize many different metrics like delay, power, area and yield. There are mainly three different types of sizing algorithms: mathematical programming based methods, e.g., [9], dynamic programming based methods, e.g., [10], and sensitivity based techniques, e.g., [11]. The latter two are more suitable for discrete cell sizing. In a recent work, Hu et. al. [10] proposed a dynamic programming method for discrete cell sizing. Their goal is to optimize area under timing constraint. In order to reduce complexity, partial solutions generated go through a similarity check. For each set of similar partial solutions, only a representative partial solution is kept, and others are eliminated. Agarwal et. al. [12] proposed a sensitivity based cell sizing method for yield optimization. The method adjusts cell sizes iteratively based on the cells' yield to area change ratios (sensitivities), and in each iteration the size of the cell with the largest sensitivity is modified. The key issue in the methods is how to determine sensitivity for each cell efficiently. [12] proves that the perturbation caused by a cell size change in the delay CDF (cumulative distribution function) will keep decreasing as it propagates towards the primary outputs. Thus, an upper bound on the sensitivity for a cell size change can be

estimated without propagating its corresponding delay perturbation to the outputs. Then, the upper bound is used to efficiently eliminate cells that cannot be the most sensitive cell.

For cell replication, [13] uses it to optimize the circuit delay. It proposed a method of dividing fanouts between the original cells and replicas when the cell is duplicated. The division is based on fanouts' criticalities. It will try to put as many critical fanouts to a separate driver as possible until the fanout load is too large and the delay is becoming worse.

For multiple V_{dd} assignment, it is usually used for power optimization. The two classic methods for V_{dd} assignment are: CVS [14], which does not allow low V_{dd} cells to drive high V_{dd} cells, and ECVS, which adds level shifters when low V_{dd} cells drive high V_{dd} cells. Wu et. al. [15] proposed a slack allocation based method for dual V_{dd} assignment after placement, in which slacks are assigned to adjacent cells in the layout that are physically distant from critical paths with high V_{dd} cells in order to facilitate formation of low V_{dd} islands among these adjacent cells.

For incremental placement, [1] proposed an incremental placement method for improving delay. The method uses mathematic programming to find the best position for cells on the critical path. Then these positions are refined in a detailed placement stage to ensure that there is no cell overlap and that no cell falls between rows.

The two types of buffer insertion is usually used to improving delay. van Ginneken [16] proposed a classical buffer insertion algorithm. It uses dynamic programming which finds the optimal solution for a given Steiner tree. The assumption of the algorithm is that there is only

a single kind of buffer cells. Lillis et al. [17] extends the algorithm to trade off solution quality with buffering resources and use a buffer library with inverters and repeaters

There are not many works that address V_{th} assignment as a single transform for power minimization, though a few [18–20] use this transform together with other transforms. For cell replication, [13] proposed a method of dividing fanouts between the original cells and replicas based on their criticality, so that the top critical fanouts use a separate driver.

1.4.2 Methods for Applying Multiple Transforms

In order to achieve better physical synthesis quality, it is desirable that all these transforms be applied simultaneously rather than sequentially one after the other. Most combined algorithms simply apply them in sequential order. However, it has been proven in [21] that applying these methods sequentially will produce fairly non-optimal solutions. The simultaneous approach allows for a more globally optimal way of determining the transforms to be used across all cells (the transforms for each cell interact in determining power consumption, delay and other metrics), while the sequential approach, by definition, can at best obtain locally optimal solutions.

Considering multiple transforms simultaneously can, however, significantly increase the problem complexity. Hence, there have been few efforts along this approach, and most of those that consider multiple transforms simultaneously do so for only two transforms. Dhillon [18] and Liu [20] proposed two Lagrange relaxation based non-linear programming methods for power optimization under delay constraint under timing constraint. Multiple V_{dd}'s and V_{th}'s were applied in [18], and multiple V_{th}'s and cell sizing were used in [20]. Gao. et. al. [19] also

employed multiple V_{th}'s and cell sizing for the same problem. In this technique, linear programming is used, and the non-linear timing and power functions are approximated by piecewise linear functions. To reduce time complexity, both [18] and [19] relax the discrete voltage level and cell size constraints initially when solving the mathematical programming model of the problems. After that, [18] clusters the continuous voltage solutions to several discrete values, and [19] rounds the continuous solution to the nearest available discrete options provided by the library. On the other hand, [20] solves the discrete optimization problem using dynamic programming under the relaxation that different sizes and V_{th}'s can be selected for the same cell to optimize delays at its different fanouts. Such conflicts are then solved heuristically. In [22], the placement transform is applied, to some degree, simultaneously with other synthesis transforms to improve circuit delay. This is done by incorporating synthesis transforms into different intermediate partition levels of a partition based placement process according to the amount of their potential perturbation to the placement.

To the best of our knowledge, there are only two works that consider more than two transforms. They are both targeting power optimization. The three transforms employed by them are cell sizing, dual V_{dd}'s, and dual V_{th}'s. Among them, [23] by Srivastava et al. essentially applies these transforms sequentially rather than simultaneously. On the other hand Chinnery et al. [24] formulated the problem as an ILP problem, with a 0-1 variable indicating whether to switch a cell in the initial design to a possibly good alternative of different size, V_{dd} and/or V_{th}. The alternative is chosen based on local incremental improvement, and the ILP is solved with

simple rounding-like heuristic. In the industry side, even the state-of-the-art industry tool can only handle these transforms one-by-one in certain sequential order

1.5 Contributions

In this thesis, we propose a novel discrete optimization algorithm called the *discretized network flow* (DNF) for solving physical synthesis problems. DNF is developed from the traditional min-cost network flow algorithm. The min-cost network flow problem is a very interesting problem. Though it is essentially a linear programming problem, it can be solved much faster than general linear programming problems [25]. We added novel discretization techniques to this continuous algorithm in order to satisfy various discrete constraints posed in the physical synthesis problems.

The DNF method has the following advantages as far as physical synthesis problems are concerned:

- It is a very general discrete optimization algorithm, and can be applied to solve the physical synthesis problem with almost any known set of transforms.
- It is particularly suitable for solving *option selection problems*. In an option selection problem, for each variable, a set of options a provided. The task is to chose one option for each variable from the set that optimize a given objective function while satisfy another set of constraint functions. Many EDA problems like physical synthesis, high level synthesis and floor planning can be formulated as option selection problems.
- It can handle a wide range of circuit design metrics. The range includes linear metrics like area, non-linear metrics like delay and power, and even metrics without closed form

expression like voltage island number (which we will talk about in Chapter 4). DNF can take these metrics as either an objective function or one of the constraints with upper or lower bounds.

- Most importantly, it inherits the runtime efficiency of the min-cost network flow problem. This enables DNF to tackle the very complex problem of simultaneous application of multiple transforms. We have successfully applied DNF to solve physical synthesis problem using up to five transforms (e.g. for timing optimization, we have used cell replication, two types of buffer insertion cell sizing and incremental placement). Note that previous works use at most three transforms.

Besides physical synthesis problems, DNF can also be applied in many other areas like 0/1 integer linear/non-linear programming, the bioinformatics problem of selecting gene markers for certain disease [26], traveling salesman problems [27] and Boolean satisfiability problems [28]. DNF has achieved high quality results for these problems, and its runtime is faster than the traditional branch-and-bound and branch-and-cut methods. For example, when applied to integer linear programming problem, we saw 1.3X to 86X speed compared to the state-of-the-art solvers with similar solution qualities (we discuss the results further in Chapter 5 and 6).

The rest part of thesis is organized as follows. In Chapter 2, we discuss the basics of the min-cost network flow, and how it can be used to solve the timing-driven incremental placement problem. Timing-driven incremental placement is a typical problem in physical synthesis which considers only the incremental placement transform. In Chapter 3, we gives a more detailed explanation of DNF. The two major issues we discuss there are: 1) how to model a general

physical synthesis problems as a network flow problem, and 2) how to satisfy various discrete requirements in physical synthesis in the continuous network flow model. In Chapter 4, we apply DNF to three different physical synthesis problems: timing optimization, power optimization and yield optimization, in which multiple transforms are considered simultaneously. In Chapter 5, we illustrate the application of DNF to solving the general 0/1 integer linear programming (ILP) problems, and 0/1 integer non-linear programming (INLP) problems. In Chapter 6, we propose an alternative way, dynamic programming using the novel concept of weak domination, to solve 0/1 integer non-linear programming problems. The advantage of this new method is that it can give a guaranteed near optimality bound.

CHAPTER 2

MIN-COST NETWORK FLOW BASED PLACEMENT ALGORITHMS

In this chapter, we will illustrate how to use the min-cost network flow to solve the timing driven incremental placement problem.

2.1 Basics of Min-Cost Network Flow

The min-cost network flow problem is a classical network flow problem, which tries to send a given amount of flow through a network graph with the minimum cost. In a network graph, each directed arc has an associated unit flow cost and capacity. Flow on an arc cannot exceed its capacity or become negative. The flow cost of each arc is the flow amount on the arc times the unit flow cost of the arc. An example of the network flow graph and a flow in it is shown in Figure 3

Min-cost flow has found a wide range of applications from traffic system design in the macro scale to the signal routing in a NOC (network-on-chip) system. In practice, the commonly used method for solving min-cost network flow problems is the network flow Simplex method. It is not polynomial bounded in time, but gives the very good average run-time result [25].

The network flow Simplex method is an iterative improvement approach. It starts with an initial flow with the required amount which is not optimal as shown by the example in Figure 3(a). Then it will try to find cycles with negative cost in the graph and augment flows in them as shown in Figure 3(b). After each augmentation, the cost of the flow will be improved.

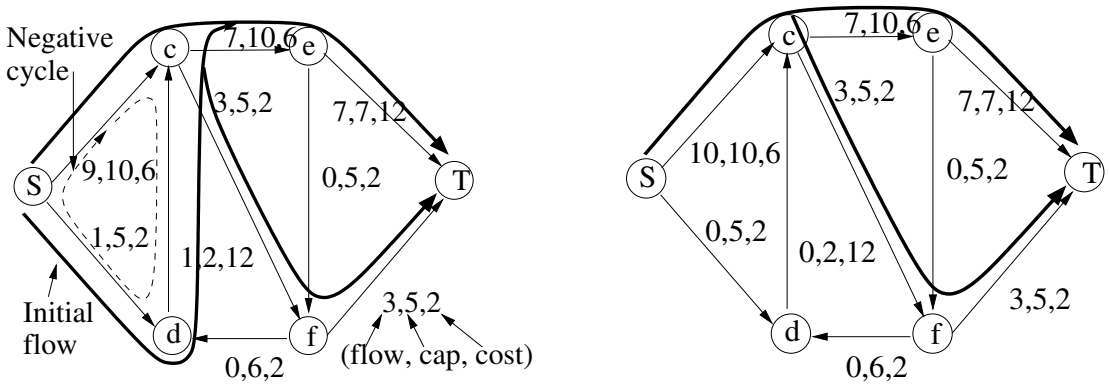

Figure 3. (a) An initial flow of 10 units from source (S) to sink (T) is shown by curved dark lines. Arc labels are arranged as $(flow, capacity, cost)$. (b) The new flow after flow augmentation in the negative cycle in [a].

It terminates when there is no cycle with negative cost, or no more flow can be augmented in any of such cycles.

2.2 Timing Driven Incremental Placement

To illustrate how the min-cost network flow method can be applied in physical synthesis. We will first look at the increment placement transform. The problem we tackle here is to use the incremental placement to optimize the delay of a circuit. More details about this work can be found in [4].

The TD incremental placement problem can be stated as:

Input: A placed circuit $\mathcal{PC}$, a set $moveC$ of new unplaced cells (the scenario of modified cells is handled by deleting them from $\mathcal{PC}$ and adding them to $moveC$).

Output: A completely placed circuit $\mathcal{PC}'$ in which: (1) there are minimal changes made to the existing placement $\mathcal{PC}$ in terms of both movements of cells in $\mathcal{PC}$ and deterioration of placement

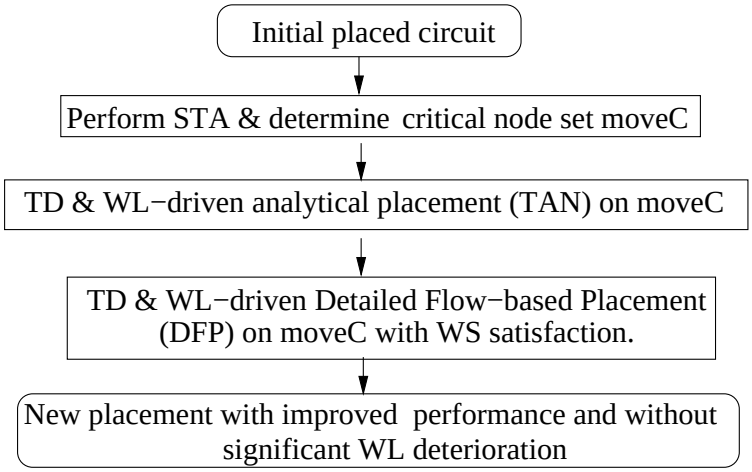

Figure 4. The flow of our TD incremental placer FlowPlace. The detailed content of each timing-driven step including WL consideration will be introduced in different sections.

metrics like total wire length (WL), (2) the critical path delay in $\mathcal{PC}'$ is significantly improved compared to the one in $\mathcal{PC}$, and (3) the given WS constraint of $\epsilon\%$ is satisfied—this means that the maximum row width $\leq (1 + \epsilon/100) \times l_{avg}$, where l_{avg} is the average total cell width in a row and is defined as l_{avg} = (total cell width in the circuit)/(the number r of rows).

Figure 4 shows the flow of our Timing-driven (TD) incremental placer. We start from a placed circuit and identify all critical and near-critical paths using static timing analysis (STA). Let this set of paths be $\mathcal{P}$ (we take paths with delay larger than 0.9 of the circuit delay as near-critical paths in our experiments). After $\mathcal{P}$ is identified, we remove all cells in all nets that lie in $\mathcal{P}$ from the layout. The removed cells form the cell set $moveC$ that will be replaced by our TD incremental placer with the goal of reducing the critical path delay. As in standard placement

flows, our incremental placement method also consists of two stages, global and detailed. In the global stage, our TD analytical placer TAN is used in which *moveC* constitutes the set of moveable cells. TAN is essentially a sophisticated quadratic solver. It can find positions for cells in *moveC* that minimize the length of the critical paths.

However, the positions found by TAN will generally be an illegal placement for cells in *moveC*. The continuous solver of TAN cannot handle the discrete requirement on cell positions: 1) the position of a cell can not fall between rows, and 2) cells cannot overlap with each other. This is where our network flow based detailed placer kicks into the picture. It is able to find legal positions for cells moved in TAN, which minimizes the movement distance from the initial position provided by TAN. In the following sections, we will explain in detail how our timing-driven (TD) detailed flow-based placer (DFP) handles this discrete optimization problem

2.2.1 Network Structure

Once a cell is moved by TAN to between rows, it needs to be moved upward or downward to adjacent rows. Once a cell is moved by TAN to a place with existing cells, to accommodate for the new incoming cell, existing cells need to be moved minimally. In a TD detailed placement, all cell movements are done based on the timing-driven costs. The timing-driven cost of moving a cell u is: a) proportional to the delay sensitivity, which is the delay change of the most critical nets with u per unit displacements of u, and b) inversely proportional to the allocated slack of the most critical net that includes u; further details are in Sec. 2.2.3.

Figure 5(a) shows the network flow graph used in DFP with arc costs and capacities. There is an arc from the source S to each new cell v (e.g. cells A_1 and A_2 in Figure 5(a)) that is

moved by TAN. The capacity of the arc is equal to the width $w(v)$ of v. For each such v, there are also two "vertical" arcs from it directed towards cells in rows immediately above and below it (there are more details to these "conceptual" arcs shown in Figure 5(c)); the capacity of each vertical arc is also $w(v)$. A total flow of $f = \sum_{v \in moveC} w(v)$ emanates from S. This flow will go through each new cell and one of its vertical arcs indicating it is moved in the corresponding vertical direction to an adjacent row.

From each *row cell* (i.e., existing cell such as cell C_{11} and C_{12} in Figure 5(a)), there are four arcs, one in each of the 2D directions (this is easily extended to 6 arcs, one in each of the 3D dimensions in case of 3D VLSICs). The vertical arcs from u go to cells in adjacent rows and model possible movement of u in the respective vertical directions; the capacity of these arcs is $w(u)$, since only u can move along these arcs. The horizontal arcs from u model possible horizontal movement of u within its row, and are potentially of capacity equal to the width of the row from u to the corresponding end of the row, since u could be moved up to either end of the row. However, since arc cost estimates become more inaccurate for large displacements, a capacity equal to the maximum of the widths of the cell in adjacent rows and new cells that have vertical arcs into u is imposed on the horizontal arcs. This allows enough horizontal flow through u that causes its required movement to remove overlaps with cells vertically moved to its position (via vertical flows into u). There can be intermediate white space within rows and these are also modeled as nodes with incoming horizontal and vertical arcs, but only one outgoing arc to the total WS node W_i of the row; the arc's cost is zero and capacity equal to amount of the intermediate white space. When there is incoming flow to a row cell from new

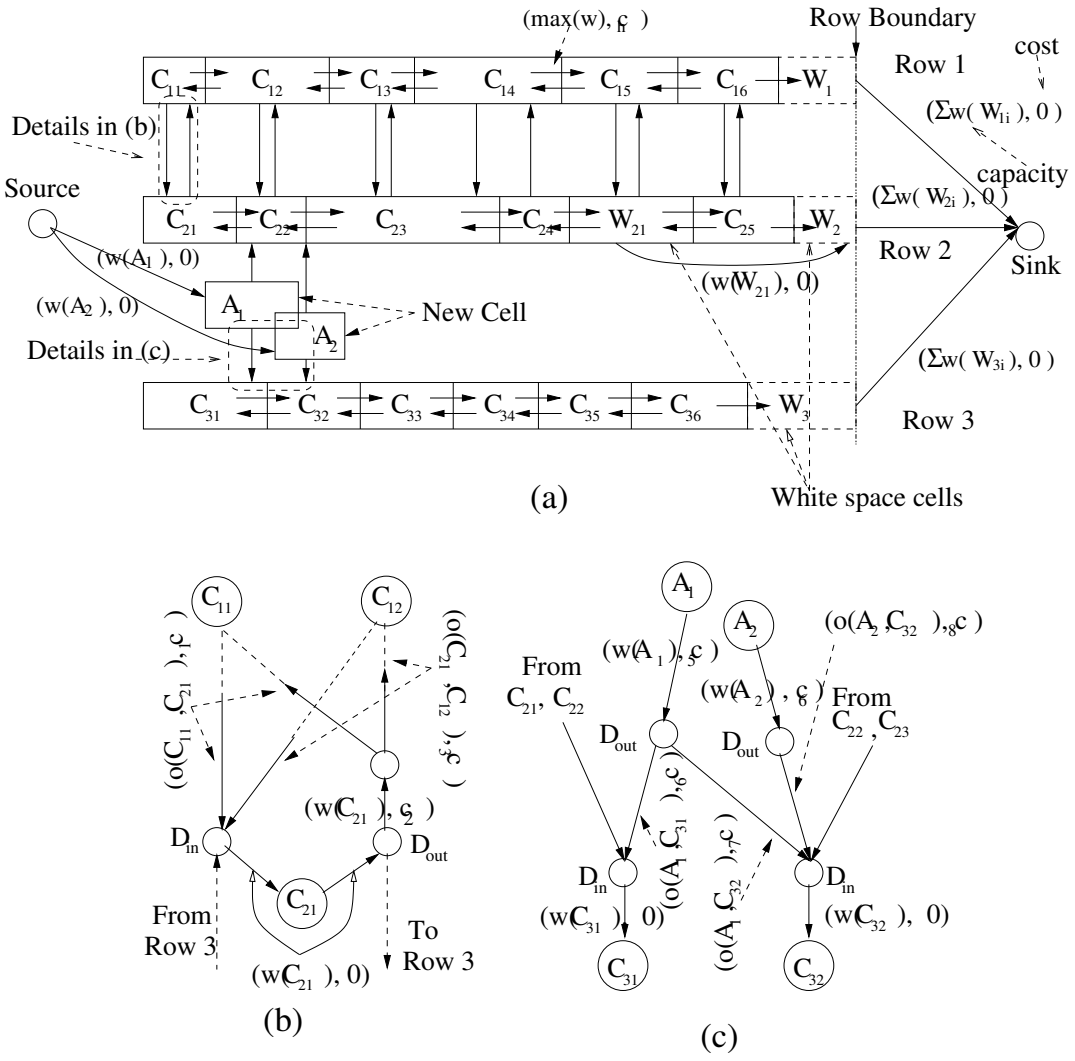

Figure 5. (a) The high-level network flow graph for placing cells A_1, A_2 in legal positions; $w(u)$ is the width of a cell u. (b) Details of flow graph structure for vertical flows between cell pairs (C_{11}, C_{21}) and (C_{12}, C_{21}); (c) Similar details of the flow graph structure for flows from the new cells into vertically adjacent row cells.

cells or row cells adjacent to it, the flow will go through one of its outgoing arcs indicating movement of the cell in the corresponding direction.

Finally, the total white space $w(W_i)$ of row i (R_i) = (max row size constraint) - ($\sum$ [cell widths in it]) is also modeled by a node W_i at the right end of the row with an incoming horizontal arc from the rightmost cell and an outgoing arc to T of zero cost and capacity = $w(W_i)$.

2.2.2 Vertical Arc Structures

Figure 5(b-c) show the details of the simplified "vertical arcs" of Figure 5(a). The detailed structures are needed to determine the vertical flow amount from a cell, say, C_{21}, to cells in rows above and below it that it would overlap if it is moved vertically up or down, respectively, to the corresponding row. The total amount of flow emanating from C_{21} corresponding to its vertical movement is $w(C_{21})$, and enters its "outflow node" D_{out} as shown in Figure 5(b). The outgoing flow from D_{out} going to the row above C_{21}'s (the structure is similar for the flow going to the row below C_{21}) splits into as many arcs as the number of cells C_{21} would overlap when moved to the row above it. The amount of flow in the arc to cell C_{1j} in this row is equal to the overlap amount $o(C_{21}, C_{1j})$ between C_{21} and C_{1j} when C_{21} is moved up; this is appropriate, since this flow amount emanating horizontally from C_{1j} would move it by an amount $o(C_{21}, C_{1j})$ that removes the overlap (if the flow emanates vertically from C_{1j}, then irrespective of the flow amount, as we will see in Sec. 2.3.1, C_{1j} is moved to the next row in the direction of the chosen vertical arc, thus removing the overlap). Note that the sum of the $o(C_{21}, C_{1j})$'s is equal to

$w(C_{21})$. Figure 5(c) shows similar structures for outgoing vertical flows from illegally placed cells A_1, A_2 going down to row R_3.

On the flip side, flows can vertically enter cell C_{21} from all cells in rows above and below it as well as from illegally placed cells near its row that would overlap it when moved to its row. However, a total flow of only $w(C_{21})$ can be allowed to enter it, and this is ensured with vertical flows from the above-mentioned cells entering the "inflow node" D_{in} of C_{21} and from there entering C_{21} via an arc of capacity $w(C_{21})$; see Figure 5(b). Henceforth, for a more conceptual discussion relating to vertical flows, we will represent vertical movements by the simple vertical arcs of Figure 5(a); the detailed structures corresponding to these vertical arcs are not necessary for subsequent discussions.

2.2.3 Timing-Driven Cost Functions

As mentioned earlier, the TD cost of an arc e emanating from cell u should be: i) proportional to the delay sensitivity, which is the delay change of the most critical net n_k through u (i.e., the net through u with the minimum allocated slack) per unit displacements of u in the direction of e, and ii) inversely proportional to the allocated slack of n_k.

The delay sensitivity is the derivative of the delay function w.r.t. u's displacement. Let n_j (n_k) be the most critical net that has u as a sink cell (driver cell), and let v (w) be the most critical sink in n_j (n_k). The sensitivity of n_j is $\frac{d(D(v,n_j))}{d(l_{d,u})}$, where $D(v, n_j)$ is the interconnect delay on n_j from its driver d to v given in Equation 1.1-Equation 1.4, and $l_{d,u}$ is the distance between the driver d of n_j and u. Similarly, the sensitivity of n_k is $\frac{d(D(w,n_k))}{d(l_{u,w})}$.

Let $\Delta l_{d,u}$ ($\Delta l_{u,w}$) be the change of $l_{d,u}$ ($l_{u,w}$) when u is moved by an unit length in e's direction. The sensitivity-based the timing-driven cost of e (i.e., its unit-flow cost) is:

$$cost_t(e) = \frac{d(D(v, n_j))}{d(l_{d,u})} \cdot \frac{\Delta l_{d,u}}{S_a(n_j)^\kappa} + \frac{d(D(w, n_k))}{d(l_{u,w})} \cdot \frac{\Delta l_{u,w}}{S_a(n_k)^\kappa} \tag{2.1}$$

$S_a(n_j)$ is the allocated slack of the most critical path that passes through net n_j. With $S_a(n_j)$ at the denominator, delay increase in a more critical net will incur a larger cost than delay increase in the non-critical net. The exponent κ is used to magnify or shrink cost differences among arcs emanating from cells connected to critical and non-critical nets; $\kappa = 2$ gives us the best overall results. The sensitivity based cost has high accuracy when cells are moved by not-very-large displacements from known positions, as is the case in incremental detailed placement.

2.3 Flow Discretization for Legal Incremental Placement Solution

As mentioned earlier, the core incremental detailed placement problem is a discrete optimization problem (DOP) with discrete constraints, and thus certain illegalities are introduced in it by using a continuous optimization method like network flow to solve it. We discuss two main illegality issues and the in-processing discretization techniques that we have developed to deal with them, i.e., techniques that work simultaneously with the network-flow algorithm. The discretization techniques may require multiple iterations of the min-cost network flow process to reach a final valid solution.

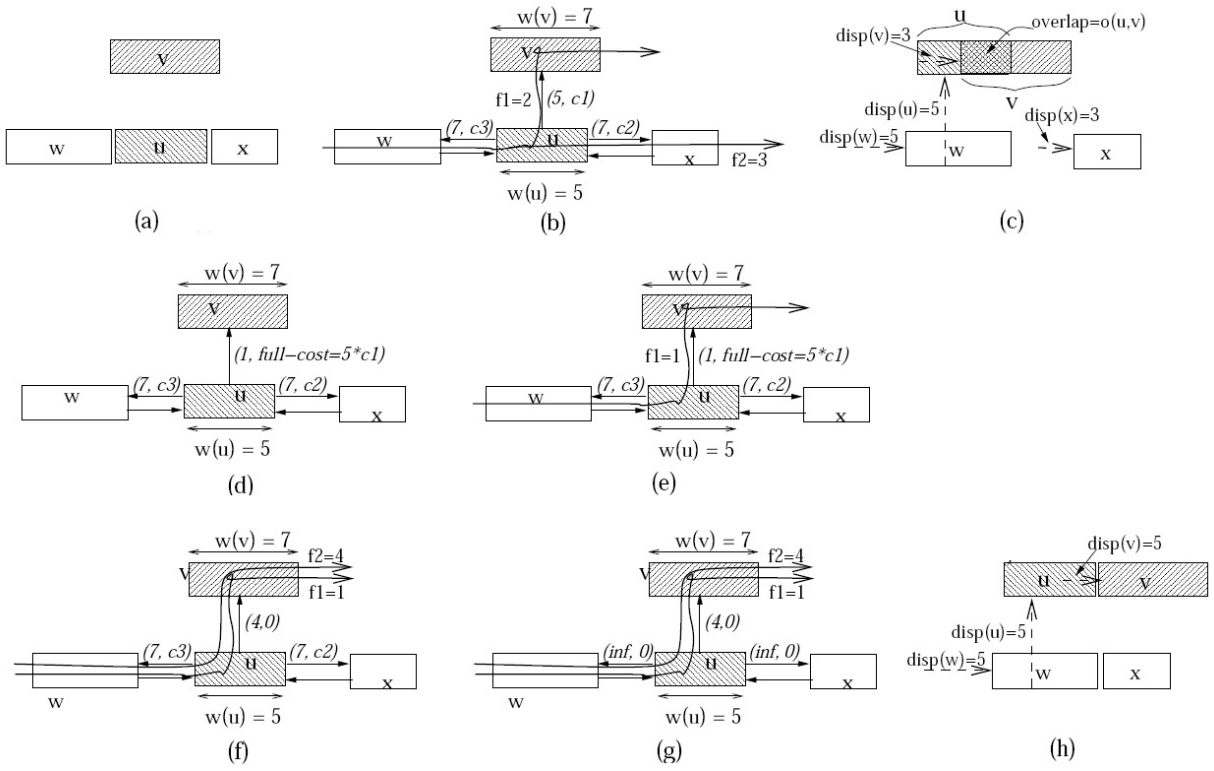

Figure 6. (a) Initial placement. (b) "Regular" cost and capacity of vertical arc (u, v) and two flows through cell u. (c) The physical translation of this flow leading to inaccurate incremental placement of affected cells; $disp(v)$ is the displacement distance of v. (d)-(h) New cost, capacity structure of arc (u, v) with dynamic update, resulting in a flow more closely mimicking the corresponding physical movement of cells, and the final accurate incremental placement of affected cells in (h). The dashed arrows in (c) and (h) represent displacements of cells at the end of the arrows.

2.3.1 Discrete Flow Requirement in Vertical Arcs

Figure 6(b) shows a vertical arc (u, v) from cell u to v of capacity $w(u) = 5$ and unit-flow cost of c_1. This arc is used to model the possible movement of u to the row immediately above it (and thus to the position of v). The physical interpretation of any flow along (u, v) has to be that u is moved to v's location, since any position in between its current position and that of v's is illegal. Thus the exact requirement of the flow amount through (u, v) should be either 0 (no movement of u) or $w(u) = 5$. Furthermore, any flow of $x < w(u)$ through (u, v) will also incur an inaccurate lower cost of $x \cdot c_1$ rather than the "full cost" of $w(u) \cdot c_1$, incurred in actually moving u to $v's$ position. The resulting inaccuracies in cell movements implied by these flows is shown in Figure 6(c).

We rectify these inaccuracies, by initially having a capacity of 1 and cost = $w(u) \cdot c_1$ (the full cost) for (u, v) as illustrated in Figure 6(d). When a flow of 1 passes through (u, v) correctly incurring the full cost of (u, v), we update (u, v)'s capacity to $w(u) - 1$ and cost to 0, thus correctly allowing an additional flow of $w(u) - 1$ to pass through it at no cost. Note that any flow entering u can exit from either the two horizontal arcs or the two vertical arcs including (u, v) that emanate from u. Note also that even with a flow of 1 through (u, v), in the physical interpretation we will move u to v's position, and thus v will be shifted to its left or right by a distance of $w(u)$ to remove its overlap with u. The resulting costs of these movements in the incremental placement of the cells affected by the flow of 1 through (u, v) will thus be incurred, irrespective of whether or not there is any more flow on (u, v). Hence for the rest of the flow coming into u, if any, we encourage $w(u) - 1$ of it to go through (u, v) by the following

mechanism: (i) change arc (u, v)'s cost to 0 (this is correct since the entire cost of arc (u, v) corresponding to a full flow has already been incurred); and (ii) maintain positive costs for the two horizontal arcs from u (see Figure 6(e-f)), as well as for the other vertical arc out of u. Only after a flow of $w(u) - 1$ passes through (u, v), do we make the cost of the horizontal arcs 0 (since u is no longer in this row) and their capacity ∞; see Figure 6(g). Figure 6(h) shows the correct cell movements implied by the resulting flow of Figure 6(g).

As a final point, we note that whenever an arc e's cost and capacity are updated, appropriate updates are made to various entities so that the correct list of negative cycles are available for cost reduction in the current max-flow.

2.3.2 Split Flows

Since a flow on a horizontal or vertical arc out of a cell u represents movement of u in the direction of the arc, as far as the incremental placement problem is concerned, a flow into u can come out of at most one outgoing arc from u. A 0/1 integer linear programming (ILP) formulation is needed for satisfying such a constraint (e.g., a binary variable is needed for each arc e that indicates if there is flow on e, and for each node, the sum of the binary variables of its outgoing arcs needs to be ≤ 1). However, 0/1 ILP is NP-hard, and such a formulation of the incremental placement problem would be intractable. The continuous optimization solution we obtain to this problem via the network flow model is in P and much faster. Of course, it has no restriction on how many outgoing arcs from a node can have flows, resulting in what we term *split flows* when more than one output arc from a node has positive flows; see Figure 7(a).

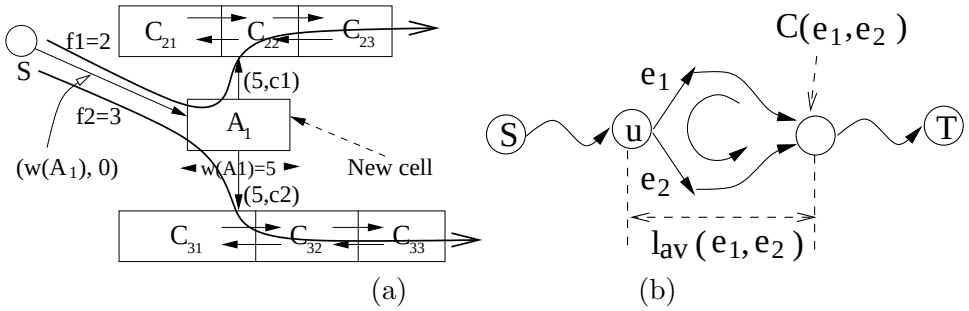

Figure 7. (a) Split flow through new cell $A1$. (b) Diverting flow from e_1 to e_2 in a split flow situation.

If after one iteration of network flow there are split flows from node u, then u will not be moved in this network flow iteration (as mentioned earlier, it can take multiple iterations to obtain a legalized placement), and in subsequent iterations we allow only one branch of the split flows from u and forbid all the other possible outgoing flows by changing the capacity of all outgoing arcs from u to 0 except the arc carrying the allowed flow. We have used two alternative heuristics to choose the allowed branch:

- *Max-flow heuristic*: For each cell u with outgoing split flows, choose the branch flow with the largest flow amount to be the allowed branch.
- *Min-cost heuristic*: For each cell u with outgoing split flows, follow the path of each branch flow up to a length of l, and choose the branch flow that has the min-cost path. Note that due to run time consideration we cannot always follow each branch flow to the sink, we thus set the limit l on the length of paths we follow. Setting l to be 100 arcs doubles the runtime compared to using the max-flow heuristic.

Ckt	**Max-flow**	**Min-cost (with exploration up to a DAG of length L)**						**Random selection**
		L = 3	L = 4	L = 5	L = 20	L = 50	L = 100	
	%Δ T	%Δ T	%Δ T	%Δ T	%Δ T	%Δ T	%Δ T	%Δ T
td-ibm01	15.0	12.7	12.8	14.2	14.7	14.9	14.9	12.3
td-ibm02	24.2	20.1	21.5	21.8	22.5	23.2	23.7	19.3
td-ibm09	13.3	9.9	10.1	10.2	10.5	10.6	10.5	6.0
td-ibm10	14.4	11.5	11.8	11.8	12.1	12.6	12.6	9.9
td-ibm17	27.1	26.0	26.0	26.1	26.2	26.2	26.2	21.8
td-ibm18	33.4	31.8	32.5	32.7	32.7	32.7	32.7	27.9
avg.	21.2	18.7	19.0	19.4	19.8	20.0	20.0	14.1

TABLE II

PERCENTAGE TIMING IMPROVEMENTS OF MAX-FLOW HEURISTIC, MIN-COST HEURISTIC WITH DIFFERENT PATH LENGTH LIMITS AND THE RANDOM SELECTION APPROACH FOR SPLIT FLOW PREVENTION.

The following theoretical result makes a simplifying assumption about the network graph for the purpose of analytically gleaning the general superiority of the max-flow heuristic.

Claim 1 *Suppose that in a network at any stage of the flow-augmentation process (passing of some of the remaining flow through the n/w), the average unsaturated arc-cost per unit flow is monotonically non-decreasing w.r.t. flow*[1]*, i.e.,* $d(av\_cost)/df \geq 0$ *and is the same (or very similar) across the network, where av_cost is the average cost among all currently unfilled arcs,*

[1] *This is a reasonable assumption, since the Simplex method augments flow in negative cycles in order of decreasing magnitude of cost improvements yielded by the cycles, i.e., the flow will more or less be first pushed through lower cost arcs and later through higher cost arcs.*

and f is the current amount of flow. Then the max-flow heuristic for split-flow prevention will always give a solution cost that is smaller than or equal to that yielded by the min-cost heuristic.

Proof: We consider a split flow at any node u with the flow passing through two outgoing arcs from u, e_1, e_2, with $f(e_1) > f(e_2)$. If a flow of value 1 is diverted from e_1 to e_2 the additional cost encountered by that flow will be $[d(av\_cost)/df] \times l_{av}(e_1, e_2)$, and vice versa, where $l_{av}(e_1, e_2)$ is the average length across all nodes u of the subpath from u to the nearest common descendant node $C(e_1, e_2)$ of e_1 and e_2; see Figure 7(b). However, since there is more flow on e_1, the total cost of diverting all the flow from e_1 to e_2 will be more than the total cost of diverting all the flow from e_2 to e_1. Thus if any other heuristic, including min-cost, chooses to divert flow from e_1 to e_2, then it will result in a larger-cost split-flow prevention solution than that obtained using the max-flow heuristic. On the other hand, if another heuristic chooses to divert flow from e_2 to e_1, then its cost will be the same as that of the max-flow heuristic. $\diamondsuit$

Empirical results using the two heuristics shown in Table II support the above analytical result. The percentage timing improvement of six representative incrementally placed circuits in Table II for the TD-IBM benchmark [29] reveals that the max-flow heuristic performs consistently better than the min-cost heuristic with path length limits of 3, 4, 5, 20, 50 and 100 arcs, and has a relatively better performance in the range of 13.4-6.0%. Table II also shows results for a randomized arc selection approach for split flow prevention; it performs significantly worse than the above two heuristics. The max-flow heuristic is thus implemented in our algorithms.

Another very telling statistics that we have collected indicates the efficacy of both the network-flow approach and the discretization technique of the max-flow heuristic for split-flow

prevention for obtaining solutions to the TD incremental placement. On comparing the final flow cost after the split-prevention discretization imposed on the necessary arcs via the max-flow heuristic to the final flow cost of a solution with unrestrained splitting (and thus illegal for our incremental placement problem), we find that the discretized flow is within only 2% of the optimal (but illegal) continuous solution. These are strong empirical results for the efficacy of our discretized network flow approach to the TD incremental placement problem.

2.3.3 Satisfying White Space Constraints

It would seem that row white space constraints are automatically satisfied due to the capacities of outgoing arcs from white-space cells and the row WS node being equal to the amount of WS they represent.

However, the problem arises due to the continuous nature of flows through vertical arcs, whereas the requirement for the detailed placement problem is for discrete (2-value) flows through them as discussed in Sec. 2.3. Referring to Figure 6(e-f), assume that u is in R_i, v in R_{i-1}, and that the total WS in R_{i-1}, $w(W_{i-1})$, is 3. A total flow of $f = 2$ (note that a total flow of 5 depicted in Figure 6(e-f) may not be available) coming from the left of R_i into u and then to v and right into the WS node of R_{i-1}, and then to the sink T will be allowed. However, if that is the only flow on arc (u, v), then the problem comes in the translation of this flow into cell movements[1]—when u is actually moved up to R_{i-1}, since $w(u) = 5$, there will actually be

[1]Note that the overlap of u and v shown in Figure 6(c) is not the issue here, as given enough WS in R_{i-1}, v and subsequent cells to its right can be moved to the right to remove all overlaps without violating the WS constraint in R_{i-1}.

a WS violation in R_{i-1} of $w(u) - w(W_{i-1}) = 2$. In other words, the total flow into a row may not always equal the total actual cell area moved into the row in the translation stage. In our detailed placement technique, we dynamically track the change in the WS amount of each row for every flow augmentation (in some cycle C) in order to detect WS violations.

In the rest of this section, we explain the network flow structure used for removing WS constraint violations. Since the WS constraint violations is also due to the root cause of flow discretization, purely network flow based method cannot completely solve the issue. In this section, besides the arc cost and the flow cost/amount based approach to discretize continuous flow solution, we propose another way of handling discretization: the policy based method. The policy is an extra constraint that we will impose each time we augment flow in the network flow simplex method. It will help to ensure the flow will finally converge to one that satisfies the discrete requirement. For WS constraint satisfaction, we propose two flow control policies, a *violation control policy* and a *thrashing control policy* to guide the WS constraint satisfaction process. These two polices allow limited but necessary temporary WS violations while proceeding towards a legal placement. With these two policies, we can guarantee the termination of our WS constraint satisfaction process, while providing a large freedom for cell movement in the process. The pseudo code for applying these two policies is given in Figure 8.

We first introduce the *violation correction structure* to remove a violation in a row. This structure is shown in Figure 9. An arc is added between the source S and the rightmost cell of each violated row R_i with violation $viol_i$. The capacity of the arc is $viol_i$, and an extra flow of this amount, termed a *violation correction flow*, emanates from S in order to push the violated

Algorithm Violation_Policy_Check

- **For** each flow augmentation in cycle C in the network flow Simplex method.
  Begin
- Identify the set of rows V_R that becomes WS-violated.
- **For** each row $R \in V_R$
  Begin
- Check violation control policy on R;
- Check thrashing control policy on R;
- **If** any of the two policies is violated
- Reverse flow augmentation in C; then forbid flow augmentation in C by putting C into a forbidden list until R is not violated.

End
End

Figure 8. Pseudo code of the violation policy check that is performed for each flow augmentation in a cycle in each network flow iteration.

amount of cell size away from the row. As a result, the detailed flow graph will have a structure that is a combination of that shown in Figure 5(a) for non-violating rows and Figure 9 for violating rows.

In order to provide a richer solution space, we allow some intermediate WS violations so that we can arrive at good WS-satisfying solutions. Intermediate WS violations are allowed in a controlled manner using a *violation control policy*, described below, that guarantees that our algorithm will terminate (see Theorem 1 given later). Under this violation control policy, we fix WS violations through multiple iterations of the network flow process. WS violation can be caused by legalizing cells in *moveC* or by flow discretizations, explained in the previous section,

and is allowed or disallowed as per our violation control policy. If after an iteration, there are rows with WS violation, we will attach a violation correction structure to these rows in the network flow graph of the next iteration.

2.3.4 Violation and Thrashing Control Policies

Our violation control policy applies within each network flow iteration (i.e., violation parameters of the policy are reset at the beginning of each iteration), and is given as follows.

- *A row that has a violation at the beginning of an iteration will not be allowed to have its violation increase.*
- *For all other rows R_i, we allow a white space violation up to the maximum cell size among R_i's adjacent rows R_{i+1} and R_{i-1}, and illegally placed cells adjacent to R_i; we denote this amount by $max\_viol(R_i)$.*
- *To compensate for the relaxed constraint, we make sure that once the white space constraint of R_i is violated by flow from a certain direction (above or below R_i), the amount of violation due to flows from that direction cannot increase until the row becomes violation-free again during subsequent flow augmentations.*

One problem of allowing temporary WS violation, even in a limited and controlled way as stated in our violation control policy, is that it may cause infinite *thrashing* in the network flow process, in which some cells may be moved between two rows repeatedly during consecutive network process iterations for violation removal. We define a *forbidden cell* for row R_i to be a

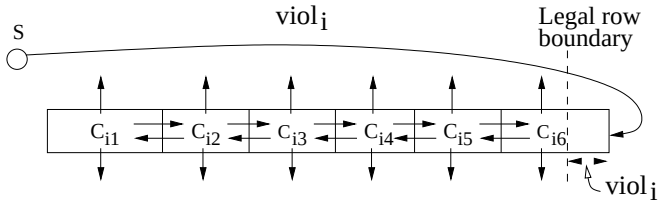

Figure 9. The violation correction structure; S is the source of the network graph.

cell that when moved into a non-violated R_i causes it to become violated. Our inter-iteration thrashing control policy to cope with this problem is:

A forbidden cell for R_i is not allowed to move back to R_i after it is moved out.

This policy (unlike the violation control policy) is an inter-iteration policy, i.e., we retain the cell and row states from one iteration to the next. A flow is allowed only if it passes both the violation control policy and the thrashing control policy. Our method terminates either if in the current network flow iteration there is no allowed flow (this is an *unsuccessful termination*), or if all illegally-placed cells are legalized and if all WS violations are eliminated (a *successful termination*). [30] provides a detailed proof that our WS satisfaction process have a very high probability to satisfy the WS constraint for designs with large or normal sizes.

Theorem 1 *[30] In a circuit with r rows and with each row having at least 128 cells, the probability that our detailed placement techniques can successfully achieve WS satisfaction is at least* $(0.9998)^r$.

In our experiments our method terminates within 7-8 iterations for each of the 27 circuits we have used, and successfully satisfies all WS constraints. Further, while our method is not guaranteed to find a WS-satisfying solution if one exists, it will, however, find such a solution with very high probability as we prove in [30].

2.4 Experimental Results

We used three benchmark suites in our experiments: 1) TD-IBM benchmark suite [29] with circuit size ranging from 13k-210k cells, 2) Faraday benchmarks from [31] with circuit size ranging from 12k-33k cells and 3) The TD-Dragon suite of [32] with circuit size ranging from 3k-26k cells. The third set of benchmarks has complete cell and timing information. The first two sets have no cell delay information. Hence, cell delay is set to be zero for the first two sets. All benchmarks are initially placed by Dragon 2.23 (the TD-Dragon benchmarks are also placed by TD-dragon which can only place this suite); see [29] for more information of the benchmarks like delay, total WL and initial placement run time. We then identify paths with delays of at least 90% of the max-path delay as *critical paths*. The γ value (Equation 1.3) is set to 1. Except for TD-Dragon benchmarks which are run with whitespace (WS) of 10% (since these circuits are relatively small), all other benchmarks are allocated a 3% WS. We use typical values for R_d, C_g, r and c for 0.18 μm technology[1]. The unit length for the TD-IBM benchmarks was taken as 0.1125 μm, and that for the Faraday ones as 0.0005 μm. Table III shows various characteristics of the

[1]These are $r = 7.6 \times 10^{-2}\, ohm/\mu$m, $c = 118 \times 10^{-18} f/\mu$m, $C_g = 10^{-15} f$, $c = 36.4 \times 10^{-18} f/\mu m$, $R_d = 1440$ ohms; for TD-Dragon benchmarks, R_d and C_g are derived from their timing library files and are similar to the above values.

initially placed benchmarks. Results were obtained on Pentium IV machines with 1GB of main memory. In all tables, positive Δ, i.e., change, numbers indicate improvement, and negative Δ numbers indicate deterioration.

Table IV shows that if the initial placement is done by a state-of-the-art WL-driven placer (Dragon 2.23 in our experiments), our purely timing-driven method can achieve up to 33% and an average of 17% delay improvement. The TD global placement (TAN) results, also given in the table, show an average timing improvement of about 22%. Thus there is an absolute deterioration of only 5% after detailed placement underscoring the efficacy of our network flow based detailed placer. The table also shows the results when we are not only using the TD cost, but use a wire length based cost also for arc cost. Compared to purely TD method, the WL increase is reduced from 9.0% to 5.8% (a 35% relative reduction), while the delay improvement is reduced by only 1.6% from 17.3% to 15.7% (a 9% relative reduction).

Table V and Table VI show that starting with circuits placed by a TD placer (TD-Dragon), we can still improve results appreciably. Since the TD-Dragon benchmark circuits are relatively small, we use a larger WS constraint of 10% here. With purely TD cost, we obtain up to 10% and an average of 6.5% timing improvement over TD-Dragon results, while the average WL increase is 6.1% (see Table V). However, as shown in the same table, the WL increase can be reduced to 5.3% with the combined cost (an average relative reduction of 13.1% compared to the purely TD cost), and the corresponding average timing improvement is about 6.3% (an average relative reduction of only 3% compared to the purely TD cost).

Ckt	**# cells**	**# nets**	**crit. len.**	**avg. len.**	**Init. pl runtime (secs)**	**delay (ns)**	**HP BB ($10^5\mu$ m)**
td-ibm01	12506	13636	134	129	402	2.2	1.8
td-ibm02	19342	19325	147	140	864	1.8	4.2
td-ibm03	22853	27118	113	102	1283	1.2	5.5
td-ibm04	27220	31679	121	119	1501	1.7	6.7
td-ibm05	28146	27490	22	45	1594	1.1	10.1
td-ibm06	32332	34654	113	110	1800	2.6	5.6
td-ibm07	45639	47786	106	107	2216	2.3	9.3
td-ibm08	51023	50227	14	49	5973	1.1	9.7
td-ibm09	53110	60606	125	128	4032	3.0	11.0
td-ibm10	68685	74179	178	171	4578	2.2	18.1
td-ibm11	70152	81402	131	128	4415	2.0	16.2
td-ibm12	70439	76313	183	182	4850	5.4	23.5
td-ibm13	83709	99106	134	131	5189	1.8	19.6
td-ibm14	147088	152138	211	205	7432	3.0	38.1
td-ibm15	161187	186218	202	198	7629	3.8	50.0
td-ibm16	182980	189259	192	185	7714	3.8	53.1
td-ibm17	184852	188503	220	205	8259	4.5	79.1
td-ibm18	210341	201640	74	79	9454	2.0	50.2
Avg.	81750	86739	134	134	4232	2.5	22.9
TDmatrix	3083	3200	73	67	254	4.3	1.0
TDmac32	8902	9115	29	27	1634	3.4	4.0
TDmac64	25616	26017	39	37	6854	7.0	20.4
TDvp2	8714	8789	75	70	966	4.5	3.4
Avg.	11578	11780	54	50	2427	5.0	7.5
matrix	3083	3200	71	65	70	4.9	0.9
mac32	8902	9115	25	25	184	3.8	3.9
mac64	25616	26017	37	36	1364	7.7	17.8
vp2	8714	8789	69	66	161	5.1	3.2
Avg.	11578	11780	51	48	444	5.0	6.5
DMA	11734	11815	17	37	384	0.4	2.1
DSP1	26301	27590	24	57	1527	0.9	4.3
DSP2	26281	27574	25	56	1602	0.8	4.3
RISC1	32622	33186	12	58	1952	1.1	5.9
RISC2	32622	33186	12	59	1906	1.3	6.1
Avg.	25912	26070	18	53	1474	0.9	4.5

TABLE III

PLACED BENCHMARK CIRCUIT CHARACTERISTICS. "CRIT. LEN." IS THE # OF CELLS IN THE MOST CRITICAL PATH, AND "AVG. LEN." IS THE AVERAGE NUMBERS OF CELLS AMONG ALL CRITICAL PATHS. THE BENCHMARK NAMES WITH TD AS THE PREFIX WERE PLACED BY TD-DRAGON AND THE REST BY DRAGON. THE "DELAY" COLUMN GIVES THE CRITICAL PATH DELAY.

Ckt	# of cells	# mv cells	%ΔWL (TAN)	%ΔT (TAN)	%ΔWL (TD)	%ΔT (TD)	runtime (secs)	%ΔWL (Comb.)	%ΔT (Comb.)
td-ibm01	13k	330	-4.1	21.0	-10.1	15.0	112	-5.9	12.5
td-ibm02	19k	662	-3.7	30.5	-11.4	24.2	102	-5.9	20.2
td-ibm03	23k	508	-3.3	32.3	-9.0	28.5	202	-7.0	27.7
td-ibm04	27k	652	-4.0	28.1	-10.9	24.1	242	-7.1	23.0
td-ibm05	28k	647	-1.6	21.2	-6.4	17.9	245	-3.2	15.4
td-ibm06	32k	704	-4.3	25.9	-9.1	21.0	305	-5.4	19.4
td-ibm07	46k	924	-2.8	18.0	-7.4	13.1	328	-4.2	12.2
td-ibm08	51k	886	-1.8	29.2	-5.3	27.1	288	-3.3	26.5
td-ibm09	53k	1154	-4.2	17.4	-10.7	13.3	524	-7.0	9.9
td-ibm10	69k	1200	-3.5	20.2	-10.6	14.4	514	-8.1	13.5
td-ibm11	70k	1174	-4.1	21.2	-10.7	14.4	603	-5.8	13.1
td-ibm12	70k	1308	-2.3	19.1	-7.8	13.2	424	-6.7	13.0
td-ibm13	84k	1128	-3.1	22.1	-9.0	18.3	472	-7.2	17.6
td-ibm14	147k	1443	-2.9	19.5	-8.9	14.8	699	-3.6	12.0
td-ibm15	161k	1426	-1.5	22.6	-5.1	18.5	754	-3.1	17.1
td-ibm16	183k	1699	-1.9	24.6	-5.1	18.5	924	-2.4	17.7
td-ibm17	185k	1984	-1.0	30.7	-3.1	27.1	957	-1.3	26.9
td-ibm18	210k	2332	-1.1	36.2	-2.8	33.4	1052	-1.3	32.7
Avg.	**81k**	**1121**	**-2.7**	**24.5**	**-8.0**	**19.9**	**487**	**-5.0**	**18.3**
DMA	12k	341	-4.7	25.1	-13.6	14.4	86	-9.3	10.5
DSP1	26k	390	-4.5	24.0	-11.9	16.1	99	-8.1	15.2
DSP2	26k	396	-4.5	23.0	-9.6	15.0	144	-6.9	15.0
RISC1	33k	671	-4.2	21.8	-10.7	16.2	166	-7.1	14.7
RISC2	33k	694	-4.1	19.9	-9.9	15	153	-7.2	14.3
Avg.	**26k**	**498**	**-4.3**	**22.7**	**-11.1**	**15.4**	**130**	**-7.7**	**13.8**
matrix	3k	290	-1.8	9.7	-10.3	7.1	123	-7.0	4.8
vp2	9k	311	-1.3	10.8	-10.5	6.0	194	-7.7	4.2
mac32	26k	352	-2.0	13.7	-8.7	9.3	172	-7.7	8.7
mac64	9k	400	-2.0	13.1	-7.0	10.2	198	-5.2	9.4
Avg.	**11k**	**338**	**-1.8**	**11.8**	**-9.1**	**8.1**	**171**	**-6.9**	**6.8**
Overall Avg.		**889**	**-2.8**	**22.3**	**-9.0**	**17.3**	**374**	**-5.8**	**15.7**

TABLE IV

TIMING AND WL RESULTS OF OUR INCREMENTAL PLACER FOR A 3% WS CONSTRAINT COMPARED TO INITIAL PLACEMENTS DONE BY DRAGON 2.23; SEE Table III FOR INITIAL PLACEMENT RESULTS. THE 4'TH AND 5'TH COLUMNS ARE THE GLOBAL PLACEMENT (TAN) RESULTS FOR A PURELY TD COST. THE 6'TH AND 7'TH COLUMNS ARE THE FINAL RESULTS (AFTER DETAILED PLACEMENT) FOR A PURELY TD A COST. THE LAST TWO COLUMNS ARE THE RESULTS WHEN USING A COMBINED WIRE LENGTH AND TD COST. POSITIVE NUMBERS MEAN IMPROVEMENT, AND NEGATIVE NUMBERS MEAN DETERIORATION.

Ckt (TD)	**#of mv cells**	**10%WS (TD)**			**10%WS (Comb.)**		
		%ΔT	**%ΔWL**	**runtime (sec)**	**%ΔT**	**%ΔWL**	**runtime (sec)**
matrix	271	5.01	-8.24	129	4.77	-6.81	139
mac32	300	8.25	-2.58	139	7.86	-2.38	150
mac64	360	10.08	-3.74	201	9.89	-3.26	223
vp2	377	2.76	-9.98	178	2.48	-8.6	182
Avg.	**327**	**6.53**	**-6.14**	**162**	**6.25**	**-5.26**	**174**

TABLE V

INCREMENTAL PLACEMENT RESULTS COMPARED TO INITIAL PLACEMENT BY TD-DRAGON; SEE Table III FOR INITIAL TD PLACEMENT RESULTS. POSITIVE NUMBERS MEAN IMPROVEMENT, AND NEGATIVE NUMBERS MEAN DETERIORATION.

Ckt (TD)	**10%WS (TD)**		**10%WS (Comb.)**		**Results in [1] w/ 10% WS**	
	%ΔT	**% ΔWL**	**%ΔT**	**%ΔWL**	**%ΔT**	**%ΔWL**
matrix	0.10	-8.20	0.10	-7.63	no impr.	N/A
mac32	4.77	-2.51	3.95	-2.33	3.8	-10.5
mac64	9.01	-4.05	8.35	-3.47	7.5	-10.9
vp2	4.12	-10.51	3.39	-8.7	no impr.	N/A
Avg.	**4.5**	**-6.32**	**4.0**	**-5.52**	**2.8**	**-10.7**

TABLE VI

RESULTS USING THE SAME DELAY MODEL AS [1]. THE RUNTIME IS SIMILAR TO USING OUR DELAY MODEL. POSITIVE NUMBERS MEAN IMPROVEMENT, AND NEGATIVE NUMBERS MEAN DETERIORATION.

Ckt	WL-Driven		Timing-Driven		
	%ΔWL from NTU-GP	Runtime (sec)	%ΔT from NTU-DP	%ΔWL from NTU-DP	Runtime (sec)
td-ibm	6.5%	6594	17.9%	-6.2%	7029
Faraday	6.9%	2402	13.4%	-8.3%	2585
Avg.	**6.6%**	**5682**	**16.9%**	**-6.7%**	**6063**

TABLE VII

COMPARING DFP TO THE NTU PLACER. POSITIVE NUMBERS MEAN IMPROVEMENT, AND NEGATIVE NUMBERS MEAN DETERIORATION.

For the TD-Dragon circuits, we also used the linear-delay model of [1] to compare our results to them. The WS constraint is also set to 10% as in [1], and the results are listed in Table VI. With a purely TD cost, under this delay model our technique obtains up to 9% and an average of 4.5% timing improvement over TD-Dragon using purely TD cost, while the average WL increase is about 6.3%. The WL increase can be reduced to about 5.5% with the combined cost, and the corresponding average timing improvement is about 4.0%. For the same set of benchmarks, an average of only 2.8% timing improvement is obtained with around 10% WL increase in [1]. Thus our results are 43-60% relatively better in the timing metric than that of [1], and 41-48% relatively better than it in the WL metric.

We also compare DFP to another state-of-the-art academic placer called NTU [33] in Table VII. NTU placer consists of two parts, NTU-GP which is a global placer and NTU-DP which is a detailed placer. For WL driven detailed placement, we measure the quality by the WL reduction compared to NTU-GP. Our WL driven results show that DFP can reduce the

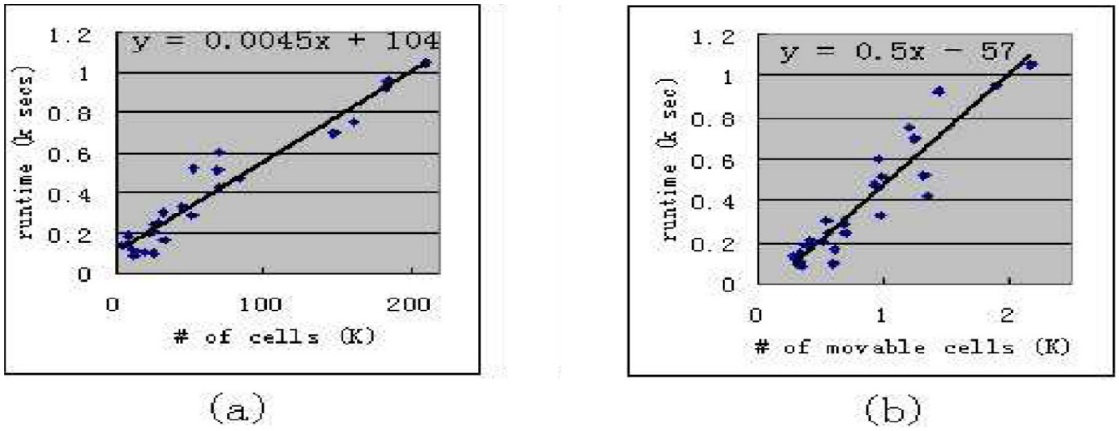

Figure 10. (a) Run time versus total number of cells. (b) Run time versus total number of movable cells

WL of NTU-GP results by 6.6%. NTU-DP obtains WL's that are about 6.8% better than those of NTU-GP; so our results are comparable to NTU-DP's. The timing driven detailed placement results are compared to the WL-driven results obtained by NTU-DP since it does not have a timing-driven mode. The results show that in timing DFP is up to about 30% and an average of 16.9% better than those of NTU-DP's, with a WL increase of 6.7% compared to NTU-DP's WL. This seems to be a good tradeoff. The above results underscore the value of our full detailed placer—having comparable WL results to the state-of-the-art NTU-DP, but, more importantly, being able to achieve high-quality timing-optimized placements with only a small deterioration in WL (compared to a high-quality WL-driven placement). To the best of our knowledge, no other state-of-the-art standard-cell placers performs TD placement, let

alone, effective TD placement as we do, on large circuits like those in the TD-IBM benchmark suite.

Our methods are also quite fast compared to a full placement; we complete incremental placement in about 10% of the initial placement time of Dragon 2.23 and in about 6% of the time of TD-Dragon. Figure 10(a)-(b) show the empirical asymptotic time complexity of our incremental placer w.r.t. both the number of cells and the number of moved cells. They show that both run time plots best match linear functions, which underscores the timing efficiency and scalability of our discretized network flow techniques.

CHAPTER 3

THE DISCRETIZED NETWORK FLOW METHOD

In the previous chapter, we explained how the min-cost network flow model can be used in incremental placement problems. In order to satisfy the discrete requirements in incremental placement, i.e., cells can not be moved between rows, and cells can only move in one direction, we proposed three different techniques to adjust the min cost flow. These techniques are: arc cost adjustment to ensure that the initial flow takes full cost and subsequent flow takes 0 cost, the min-cost/max-flow heuristic, and the violation control policy and the thrashing control policy.

In this chapter, we will explain in detail our DNF method. The DNF method is a further improvement of the network flow model we introduced in the previous chapter. It has two major changes: 1) A more general network flow structure is proposed that can be used for almost any kind of physical synthesis transform. The network flow model for DFP is very suitable for incremental placement, in which cell area can be modeled as flow amount, and movement in different directions can be modeled as arcs. However, the movement directions are not a concern for transforms other than incremental placement, and the area is not a concern for transforms like V_{dd} and V_{th} assignment. Hence, mapping network flow structures to metrics or changes that are specific to each transform will require different structures for different transforms. This will prevent us from processing multiple transforms simultaneously. 2) We proposed a single technique that can satisfy a wide range of discrete requirements in the physical synthesis

problem. The DNF method can be used to solve the physical synthesis problem with any single transform. It can also be used to tackle the problem of applying multiple transforms simultaneously.

3.1 The Option Selection Problem

In order to process different transforms with a single method, we first generalize these transforms as *option selection problems*. In general option selection problems, for each variable, a set of possible values (called options) are given. We need to choose one value for each variable so that the given objective function is optimized, and the given constraints are met. In physical synthesis problems, the scenario is similar in that for each cell a set of options is provided by each type of transforms, and we need to select and apply one option in each transform set for each cell.

The options provided by different transforms are listed below:

(1) *Cell Sizing*. The options are different sized cells provided by the library.

(2) *Driver Buffer Insertion*. The options are the different sized buffers in the library as well as not adding any driver buffer.

(3) *Isolating Buffer Insertion*. The options are also the different sized buffers in the library as well as not adding any isolating buffer.

(4) *Cell Replication*. The options for this transform includes no replication and replications with different partition schemes of the fanouts between the two replicas. However, if we provide all possible partitions as options, the number of options will be huge, since we need to consider the possibility that some fanout gates (sinks) may also be replicated. Hence, in our method,

we pre-determine the three best timing-driven partitions based on the initial design using the algorithm in [13], and incorporate these as options. We thus have a total of four replication options for each cell: three replication options corresponding to the three chosen partitions and the no-replication option.

(5) *Incremental Placement.* There are two possible situations for incremental placement. First, multiple candidate positions for a cell can be provided by certain type of pre-processing method. Each of these positions may have its own advantage or drawbacks. Such a case can happen when there are multiple metrics that need to be considered at the same time, like delay and wire length. For this situation, each position will be an option for the cell. The second situation is that only one position is provided for a cell. In this case, we usually need to make adjustment to this position, like we do in DFP for the initial position provided by the global placer TAN. Then, the options will be the adjacent possible positions like the adjacent rows in DFP.

(6) V_{th} *Assignment.* The options are different threshold voltage values provided in the library.

(7) V_{dd} *Assignment.* The options are different supply voltage values provided in the library.

To solve the option selection problem, the network flow model in our DNF method contains two coupled substructures: one for objective function optimization, and one for satisfying various constraints. In both substructures, nodes represent options, and flow through a node indicates selection and application of the corresponding option. In the option selection problem, the discrete requirement is that only one option can be chosen for each transform for a cell. In

our DNF model, this constraint is converted to a *mutually exclusive arc (MEA)* set constraint. The MEA constraint on a set of arcs dictates that flow can only pass through at most one arc among them. *From here onwards, for easy of explanation, we will use the term "transform" to refer to a transform for a single cell.*

More details about our DNF method are provided in the following sections.

3.2 Discretized Network Flow Concepts

Given an option selection problem with a minimization objective function F, the DNF method optimizes F in two major steps: 1) It constructs a network flow graph called *the optimization graph* (OG) for F. In the OG, available options for each transform are represented by nodes. Option selection is done by flow through the graph, i.e., if an option node has flow through it, then the corresponding option is selected for the associated cell. For power optimization, the selection of an option means applying it to the corresponding cell (e.g., changing the cell to the selected size or V_{th} or assigning the selected V_{dd} to the cell). The flow cost reflects the value of F corresponding to options selected by the flow. 2) It obtains the min-cost flow in the graph that provides a valid option selection solution, i.e., one option is selected for each transform. Due to the equivalence between flow cost and F's value corresponding to the selected options, the min-cost flow selects options that minimize F.

In the rest of this section, we will describe in detail how the OG is constructed, as well as introduce the fundamental concepts of DNF. The DNF modeling for constraint satisfaction is presented in Sec. 3.3.

3.2.1 The Network Flow Graph Substructure for DNF

If a term in a function cannot be decomposed further using the add operation (i.e., decomposed into the sum of several other terms), it is called a *p-term* (for product term). Thus a p-term of a function is a term that is atomic with respect to addition. Any minimization objective function F can be expressed as the summation of linear or non-linear p-terms f_k as given below.

$$F = \sum_k f_k \tag{3.1}$$

For example, the power consumption P of a circuit (Equation 1.5, Equation 1.6) is the sum of three types of p-terms: (1) $V_{dd}^2(g_d)C(n_i)$ for charging the wire capacitance of a net n_i. (2) Gate leakage power $P_l(g_o)$ (Equation 1.6). (3) The power consumption P_c for level shifters and charging sink cells (Equation 1.7). Recall that $C(n_j)$ is the interconnect capacitance of net n_j, and $V_{dd}(g_d)$ is the supply voltage of gate g_d. Though out this section, we will use the power objective function as example in our illustration. Without loss of generality, we will consider four types of transforms: multiple V_{dd}, multiple V_{th}, cell sizing and incremental placement.

A solution S_o to an option selection problem is a set of options with one option for each transform. With a solution S_o, a unique value of the objective function F (denoted by $F(S_o)$) can be determined. We also define a partial solution s_k^j for a p-term f_k as a set of options with one option for each transform that f_k is a function of. Similarly, for an s_k^j, a unique value of f_k denoted by $f_k(s_k^j)$ can be determined. For a p-term $f_k(T_{i_1}, \ldots, T_{i_m})$ that is a function of m

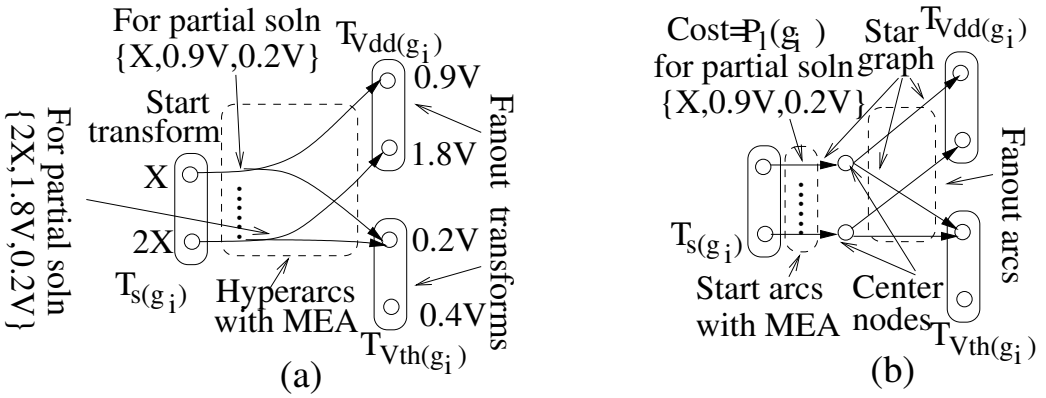

Figure 11. (a) The conceptual structure of ot-subgraphs for p-term $P_l(g_i)$. (b) Actual network flow graph implementation of the conceptual subgraph in [a], in which hyperarcs are modeled by star graphs.

transforms T_{i_1} ,...,T_{i_m}, there are totally $\Pi_{t=1}^{m}|T_{i_t}|$ possible partial solutions for f_k, where $|T_{i_t}|$ is the number of available options for transform T_{i_t}.

The OG for minimizing objective function F has a top-down structure: it consists of subgraphs, each of which corresponds to one p-term in F; these subgraphs are termed *ot-subgraphs* (for optimization p-term subgraphs). In the ot-subgraph *ot-*$G(f_k)$ for a p-term f_k, there is a node for each option of transforms that f_k is a function of. Henceforth, the term "option" will be used to refer to both a transform option and its corresponding node in ot-subgraphs. There is also a hyperarc in *ot-*$G(f_k)$ representing each possible partial solution s_k^j for f_k. The hyperarc corresponding to a partial solution s_k^j connects all the options in s_k^j. These hyperarcs are called *option selection hyperarcs.*

Take the leakage power p-term $P_l(g_i)$ of gate g_i as an example. Its formulation (see Equation 1.6) is $V_{dd}(g_i)W(g_i)I_s \cdot e^{-V_{th}(g_i)/V_o}$. $W(g_i)$ is the transistor width of g_i and is proportional to the size $s(g_i)$ of g_i. Hence, $P_l(g_i)$ is a function of $V_{dd}(g_i)$, $s(g_i)$, and $V_{th}(g_i)$. It is thus a function of three transforms on g_i: 1) V_{dd} assignment transform denoted by $T_{V_{dd}(g_i)}$, 2) cell sizing transform on g_i denoted by $T_{s(g_i)}$ and 3)V_{th} assignment transform denoted by $T_{V_{th}(g_i)}$. The corresponding ot-subgraph (for $P_l(g_i)$) is given in Figure 11(a). Two options are provided for each transform, and they are represented as nodes in the ot-subgraph. Based on the connected option nodes, the two hyperarcs shown correspond to two partial solutions (represented as {size, threshold voltage, supply voltage}): {X, 0.2 V, 0.9 V} and {2X, 0.4 V, 0.9 V}.

When constructing ot-subgraphs, we determine hyperarc directions so that the union of all ot-subgraphs (i.e., the OG) is acyclic. Specifically, if there are hyperarcs from transform T_i to transform T_j in an ot-subgraph, there should not be any hyperarc in the reverse direction (from T_j to T_i) in any other ot-subgraph. In order to ensure this, an arbitrary order R among transforms is pre-determined. In an ot-subgraph, the transform whose option nodes are the head of the hyperarcs is called the *start transform* (e.g., $T_{s(g_i)}$ in Figure 11(a)), and the other transforms whose option nodes are "tails" of the hyperarcs are called *fanout transforms* (e.g., $T_{V_{th}(g_i)}$ and $T_{V_{dd}(g_i)}$ in Figure 11(a)). The start transform is chosen to be the one that precedes the other transforms of the ot-subggraph in R. We should note here that the order among transforms does not affect the final solution, i.e., a transform occurring earlier in R does not have any priority over transforms that occur later in R. For any order R, our DNF model is able to provide the optimal solution for the original option selection problem as established in

Theorem 4 (Sec. 3.4). A simple way to understand this is that R only determines the topological order of transforms in the OG (more details of the OG are given later in this section), and in a network flow graph (e.g., the OG), even if we completely reverse the topological order of nodes, i.e., reverse all arc directions and switch the source and the sink, the min-cost flow still passes through the same set of nodes, i.e., for our problem, the same set of options is selected in the reversed OG.

A hyperarc connecting m options is implemented in the network flow graph as a *star graph* with a *start arc* from the corresponding option of the start transform, and ending in a *center node*. This node has $m-1$ branches (called *fanout arcs*) from it to the other $m-1$ options of the hyperarc (corresponding to the $m-1$ fanout transforms). The cost of a hyperarc representing a partial solution s_k^j is $f_k(s_k^j)$ and is attached to the start arc of its star graph model. Figure 11(b) is the star graph implementation of Figure 11(a) for p-term $P_l(g_i)$. Henceforth, we will use the two terms, hyperarcs and star graphs, interchangeably to refer to the connections in ot-subgraphs corresponding to partial solutions.

For our power optimization problem, the ot-subgraphs for the three types of p-terms in the power consumption function P can be constructed as follows.

1. For the leakage power p-term $P_l(g_i)$ for each gate g_i: the ot-subgraph is given in Figure 11(b).

2. For p-term $V_{dd}^2(g_d)C(n_i)$ of charging net capacitance: g_d is the driving cell of net n_i. $C(n_i)$ is the net capacitance of n_i, and is proportional to the wire length of n_i, which is in turn a function of the bin position $B(g)$'s of all gates g on n_i. Thus, $C(n_i)$ is determined by

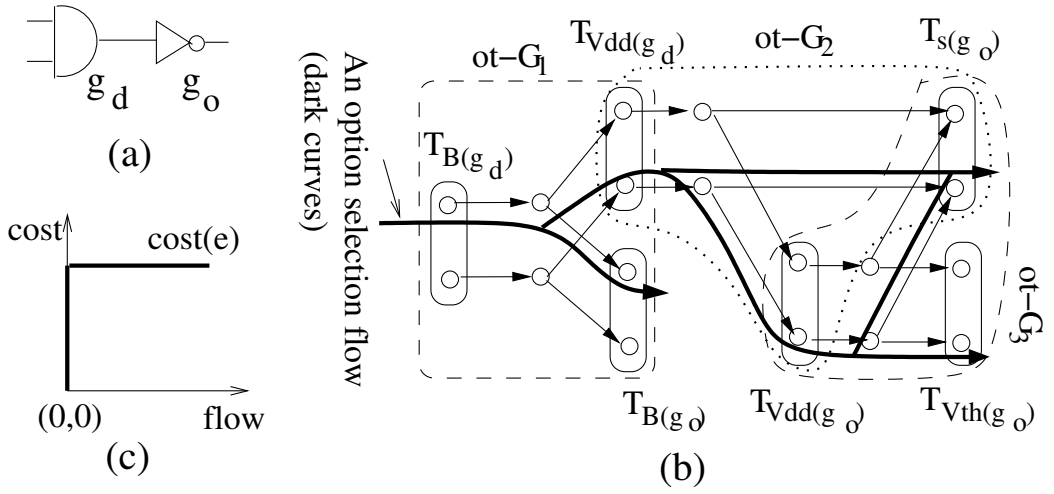

Figure 12. (a) A simple net n_i with gate g_d driving gate g_o. (b) The power OG for the net in [a]. An option selection flow is shown by dark curves. (c) The step cost function of an arc e in the OG.

re-placement transforms $T_{B(g)}$ for all gates g in n_i. Therefore, the ot-subgraph for this p-term includes the following transforms: $T_{V_{dd}(g_d)}$, $T_{B(g_d)}$ and $T_{B(g_o)}$ for all sink cells g_o on n_i. Therefore, for a net of degree k, the ot-subgraph includes one start transform and k fanout transforms, and each hyperarc (star graph) in it connects $k+1$ option nodes.

3. For p-term $P_c(g_d, g_o)$ of the level shifter power and the power for charging the input capacitance $C(g_o)$ of g_o (g_d be the driving cell). Since $C(g_o)$ is proportional to $s(g_o)$, the ot-subgraph for P_c includes three transforms: $T_{s(g_o)}, T_{V_{dd}(g_d)}$ and $T_{V_{dd}(g_o)}$. The structure of the ot-subgraph is similar to that in Figure 11(b), and the option selection hyperarcs in it form a complete tri-partite hypergraph between the three transforms.

3.2.2 The Complete Optimization Graph

The complete OG for an objective function F is formed by combining the ot-subgraphs for all p-terms in F. The combination is done by merging the common transforms between ot-subgraphs. Figure 12(b) shows an example of merging ot-subgraphs of the three types of power p-terms for a simple net given in Figure 12(a).*ot*-G_1 is for p-term $V_{dd}^2(g_d)C(n_i)$; *ot*-G_2 is for p-term P_c of the level shifter power and charging power for the input capacitance of g_o; *ot*-G_3 is for p-term $P_l(g_o)$Transforms that are not fanout transforms in any ot-subgraph (e.g., $T_{B(g_d)}$ in Figure 12(b)) should be connected to the source S in the OG, and transforms that are not the start transform of any ot-subgraph should be connected to the sink T (e.g., $T_{V_{th}(g_o)}$ in Figure 12(b)).

We note here that the flow in the OG has a discrete meaning, i.e., flow > 0 indicates selection of an option and flow $= 0$ indicates non-selection of an option; the actual flow amount does not affect the option selection results. Matching this discrete flow semantics is the cost formulation of flow f on an arc e with cost $cost(e)$, which is a step function, as opposed to a linear one in classical network flow. Specifically, the cost incurred is $cost(e)$ if $f > 0$, and 0 otherwise, as shown in Figure 12(c). The only flow amount requirement in the OG is that enough flow is sent form the source S so that at least t units of flow go through an ot-subgraph with t transforms. Since the OG is acyclic, the flow amount needed to be sent from S can be determined using a recursive relationship; more details are provided in Sec. 3.3.1.

3.2.3 Valid Flow and Flow Cost

A major difference between the DNF method and standard min cost flow is that DNF can satisfy discrete constraints imposed by the target problem. For the option selection problem, a *valid option selection flow* (termed as *valid flow* for short) in the OG should meet two discrete requirements: (1) pass through only one option node for each transform so that only one option is selected for each transform (*option exclusivity* or *OE*), and (2) pass though only one option selection hyperarc in each ot-subgraph so that only one partial solution is selected for each p-term (*partial solution exclusivity* or *PSE*). Note that these two requirements are independent. Figure 13(a) shows an flow example in *ot-*G_1 in Figure 12(b). The flow in the figure satisfies OE, but violates the PSE requirement since it passes through two hyperarcs (i.e., through two start arcs of their star graph models). On the other hand, Figure 13(b) shows an flow example through transform $T_{s(g_o)}$ in Figure 12(b). The flows in the two ot-subgraphs satisfy the PSE requirement. However, they choose different options for the common transform $T_{s(g_o)}$ between them, and hence violates the OE requirement.

The OE and PSE requirements for a valid flow can be generalized as MEA constraints. For the PSE requirement, the MEA constraint is applied on the set of start arcs in each ot-subgraph; see Figure 11. On the other hand, the OE requirement is actually a *mutually exclusive node set* constraint, which can be converted to the MEA constraint through the following structure change. We replace each option node by a pair of dummy nodes connected by a "bridge" arc as shown in Figure 13(c). Thus, a flow through an option node is equivalent to a flow through the bridge arc that replaces the node. The MEA constraint is applied to the set of bridge arcs

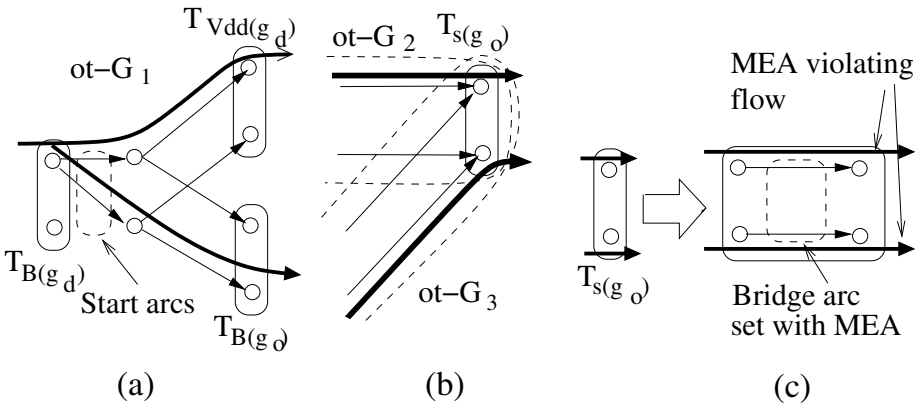

Figure 13. Flows violating PSE and OE requirements. All flows are shown by dark arcs. (a) A flow in an ot-subgraph that satisfies the option exclusivity (OE), but violates the partial solution exclusivity (PSE). (b) Flows in two ot-subgraphs ot-G_2 and ot-G_3 that select different options for the common transform $T_{s(g_o)}$. The PSE requirement is satisfied, but the OE is violated. (c) Bridge arc structures and MEA constraints are used to satisfy the OE requirement. The flow in [b] extended to the bridge arc structure of T_{i3} violates the MEA constraint on the bridge arcs, and therefore will never occur.

replacing the set of options for each transform, and a flow that satisfies this MEA also satisfies the required OE; see Figure 13(c).

The approach to ensure the satisfaction of MEA constraints is as follows. For each arc e in an MEA set, besides its original cost, termed C-cost (e.g. the p-term based cost on option selection hyperarcs in ot-subgraphs), an extra cost of C' (termed *C'-cost*) is added. The idea behind this approach is that since a valid flow passes through only one arc in each MEA set, the C'-cost it incurs in each MEA set is exactly C'. On the other hand, an invalid flow passes through at least two arcs in one or more MEA set, and thus incurs a C'-cost of at least $2C'$ in at least one MEA set. Hence, the C'-cost of an invalid flow is larger than that of a valid

flow by at least C'. Therefore, if we set C' to be larger than the C-cost difference D_C between a valid flow F_v (which can be constructed according to an arbitrary or heuristic-based option selection solution) and an invalid flow with minimum C-cost C_{min} (which can be determined using standard min-cost flow algorithm on the network graph with only C-costs and hence without any MEA constraint), then any invalid flow will incur a larger total cost (C-cost + C'-cost) than F_v. Correspondingly, the min-cost flow in the network flow graph with both C- and C'-cost will always be a valid flow. Further, it will be the valid flow with minimum C-cost, since the C'-cost incurred by any valid flow is a constant mC', where m is the number of MEA sets in the OG. In our implementation, C' is set to be $D_C + 1$. Hence, we have the following theorem.

Theorem 2 *By setting the C'-cost of each arc in MEA sets to be larger than the C-cost difference of a valid flow and a min-cost invalid flow, the min cost flow in the network flow graph with C- and C'-costs is a valid flow with minimum C-cost.* $\diamondsuit$

Proof: We can prove by contradiction. Assume the min-cost flow we select is a invalid flow with C-cost C_{inv}. Obviously, $C_{inv} > C_{min}$. Also from, previous analysis, for the C'-cost C'_{inv} of this invalid flow, we have $C'_{inv} >= C'(F_v) + C'$. Therefore, the total cost of the invalid flow $C_{inv} + C'_{inv} > C_{min} + C' > C(F_v) + C'(F_v)$. This contradicts the assumption that the invalid flow is a min-cost flow.$\diamondsuit$

For the C-cost of a valid flow in the OG, we have the following lemma.

Lemma 1 *The cost of a valid flow in the OG for an objective function F selecting solution S_o is equal to $F(S_o)$.*

Proof: Assume the valid flow selects a partial solution s_k^j for each p-term f_k in its ot-subgraph *ot-G*(f_k). Clearly, $S_o = \cup_{f_k \in F} s_k^j$. Since a valid flow passes through only the option selection hyperarc representing s_k^j in each *ot-G*(f_k), the total incurred cost is $\sum_{f_k \in F} f_k(s_k^j)$, which is exactly $F(S_o)$. $\diamondsuit$

Theorem 2 and Lemma 1 indicate that a min-cost valid flow in the OG selects the options that minimize the objective function. Since the step cost function we use for arc costs is a concave function, standard min-cost flow algorithms cannot be used for our network flow graph. Instead, we use the concave min-cost algorithm of [34] that obtains a near optimal (i.e., near min-cost) flow solution.

3.3 Network Flow Graph for Constraint Satisfaction

In this section, we present network flow structures called *constraint satisfaction graphs* (CGs) for satisfying multiple $\leq$ type linear and non-linear constraints in option selection problems, and use such structures to satisfy the cell area (i.e., bin capacity) and timing constraints when optimizing power[1]. The main idea for constraint satisfaction is to use flow amounts in CGs to represent the constraint function values, use arcs with capacities equal to the given upper bounds to limit the flow amounts to be $\leq$ these capacities, and thereby satisfy the given constraints.

---

[1] $\geq$ type constraints can be converted to $\leq$ by negation.

The CG is an extended version of the OG described in Sec. 3.2. For a constraint $H_i \leq b_i$, where b_i is a constant, the constraint function H_i, just like the optimization function F (see Equation 3.1) can always be expressed as a sum of p-terms $h_{i,k}$. Similar to the OG, the CG is built hierarchically by combining constraint p-term subgraphs (ct-subgraphs) for each $h_{i,k}$. As an example, we construct below the CG for the delay constraint of the circuit with three cells g_a, g_b, g_c shown in Figure 14(a). For the (n_i, n_j) path in the circuit, where n_i, n_j are two nets, the constraint can be expressed as $D(n_i) + D(n_j) \leq \tau$, where $D(n_i)$ is the delay on net n_i, and τ is the given delay upper bound.

Like ot-subgraphs, each ct-subgraph includes nodes that represent options, and hyperarcs that represent partial solutions. A valid flow should pass through only one hyperarc in each ct-subgraph (the PSE requirement), and one option node for each transform (the OE requirement). An example ct-subgraph for the delay p-term $R_d(g_b)C(n_j)$ for charging the capacitance of net n_j in the circuit of Figure 14(a) is given in Figure 14(b); $R_d(g_b)$ is the driving resistance of driver cell g_b. For simplicity, we only consider sizing and incremental placement transforms here, and for incremental placement options we are consider the bin based option $B(g_b)$ for a cell g_d. Bin based placement option is usually used in the global placement stage where we do not need to put a cell in a specific position, and only put cells in certain bins that are divided from the layout area is sufficient. The above p-term is a function of $s(g_b)$ (determines the driving resistance $R_d(g_b)$ of g_b), and the bin placement option $B(g_b)$ and $B(g_c)$ of g_b and g_c (these determine the net length and thus capacitance $C(n_j)$ of n_j). Hence, the ct-subgraph includes three transforms and the complete tri-partite connection of hyperarcs among them. A

structural difference between ct-subgraphs and ot-subgraphs is that an extra *branch arc* and an extra *complementary arc* is attached to each option selection hyperarc; these arcs emanate from the center node in the star graph model of the hyperarc.

The function of branch arcs is to output a flow of amount equal to the p-term value with the selected partial solution. Specifically, if a valid flow passes through the hyperarc corresponding to a partial solution $s_{i,k}^j$ in the ct-subgraph *ct-G*$(h_{i,k})$ for a p-term $h_{i,k}$, the branch arc attached to the hyperarc should shunt out a flow of amount $h_{i,k}(s_{i,k}^j)$ from *ct-G*$(h_{i,k})$. This is accomplished by having the capacity of the branch arc $= h_{i,k}(s_{i,k}^j)$, and pushing a flow of at least that amount into *ct-G*$(h_{i,k})$; details on these issues are given in Sec. 3.3.1. An example of an option selection flow in the ct-subgraph for p-term $R_d(g_b)C(n_j)$ is given in Figure 14(c). When flow passes through the star graph structure in Figure 14(c) corresponding to size X and bin B_2 for g_b, and bin B_1 for g_c, the value of the p-term, and thus the branch flow amount, is $R_d(g_b(X)) \cdot c \cdot l(B_1, B_2)$, where $R_d(g_b(X))$ is the driving resistance of g_b for size X, $l(B_1, B_2)$ is the distance between bins B_1 and B_2, and thus wire length of net n_j connecting g_b and g_c, and c is the unit distance wire capacitance.

Branch arcs from all ct-subgraphs for a constraint $H_i \leq b_i$ are directed towards a *constraining node*, and the flows on branch arcs (called *branch flows*) are sent into a *constraining arc* of capacity of the given upper bound b_i of the constraint. An example of the constraining node and the constraining arc for the delay constraint of circuit in Figure 14(a) is shown in Figure 15. For a valid flow in the CG selecting a solution $S_o = \cup_{h_{i,k} \in H_i} \{s_{i,k}^j\}$ for constraint $H_i \leq b_i$, the total amount of branch flows $= \sum_{h_{i,k} \in H_i} h_{i,k}(s_{i,k}^j)$, which is exactly the value of

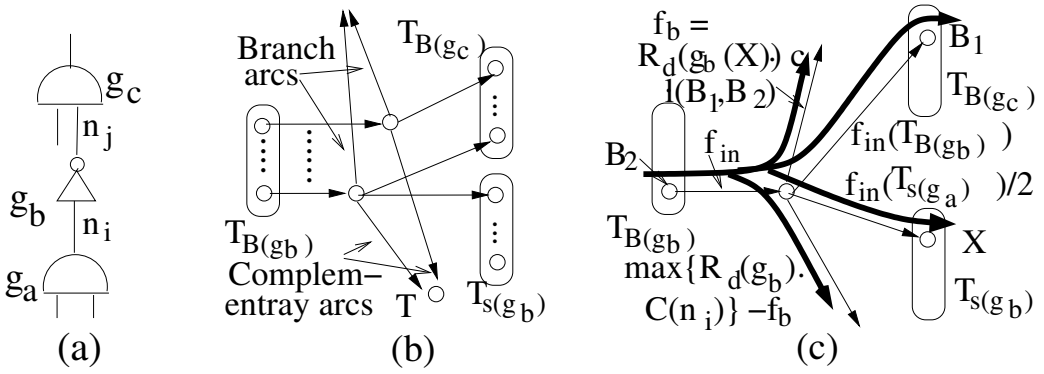

Figure 14. (a) An example circuit. (b) The ct-subgraph for delay p-term $R_d(g_b)C(n_j)$. (c) The flow distribution in the ct-subgraph shown in [b].

H_i (denoted by $H_i(S_o)$) with the selected solution S_o. The capacity of the constraining arc is b_i, and this limits the flow amount through it to be $\leq b_i$. Thus the options selected by a valid flow in the CG always constitute a feasible solution, i.e., satisfying the constraint $H_i \leq b_i$. On the other hand, if a flow selects a solution that violates the constraint, it will be an invalid flow that violates OE and/or PSE requirements as we will demonstrate in the next subsection.

Finally, as explained for ot-subgraphs ((Sec. 3.2.2), combining ct-subgraphs is also done by merging common transforms among them. Figure 15 illustrates the CG for delay constraint $D(n_i) + D(n_j) \leq \tau$ of circuit in Figure 14(a). For clarity of exposition, here we consider only: 1) two types of delay p-terms: for charging the net capacitance, $R_d(g_a)C(n_i)$ and $R_d(g_b)C(n_j)$), and for charging the sink cell input capacitance, $R_d(g_a)C(g_b)$ and $R_d(g_b)C(g_c)$); 2) two transforms: re-placement and cell sizing.

3.3.1 Flow Amount and Arc Capacity in Ct-subgraphs

In order to ensure the correct functioning of each ct-subgraph—outputting a flow amount on a branch arc equal to the p-term value corresponding to the selected transform options—the incoming flow amount and arc capacities in each ct-subgraph must be determined accordingly. This is quite different from ot-subgraphs where we only care about the presence or absence of a flow and the arc costs, while the flow amount does not matter. Conversely, in ct-subgraphs, cost is irrelevant, and thus costs of arcs in ct-subgraphs are all 0.

The capacity of the branch arc attached to a hyperarc corresponding to $s^r_{i,k}$ in the ct-subgraph *ct-G*$(h_{i,k})$ is $h_{i,k}(s^r_{i,k})$, i.e., the desired branch flow amount as explained in the previous sub-section. Further, as we can see in Figure 14(c), the incoming flow to a ct-subgraph is distributed on the outgoing arcs in a star graph structure, i.e., the branch arc, the complementary arc and the fanout arcs. Hence, in order to ensure a full flow on the branch arc when $s^r_{i,k}$ is selected, the incoming flow amount to *ct-G*$(h_{i,k})$ should be equal to the total capacities of the branch arc, the complementary arc and the fanout arcs in the star graph structure for $s^r_{i,k}$.

Complementary arcs are arcs directed to the sink for balancing the branch flow amount difference from different hyperarcs in a ct-subgraph so that irrespective of which option selection hyperarc is chosen, the needed incoming flow amount for a ct-subgraph is fixed. In *ct-G*$(h_{i,k})$, if a valid flow passing through a hyperarc corresponds to a partial solution $s^r_{i,k}$, the flow amount sent on the complementary arc attached to the hyperarc should be $h^{max}_{i,k} - h_{i,k}(s^r_{i,k})$, where $h^{max}_{i,k}$ is the maximum possible value of p-term $h_{i,k}$ over all possible partial solutions $s^r_{i,k}$. Thus, the

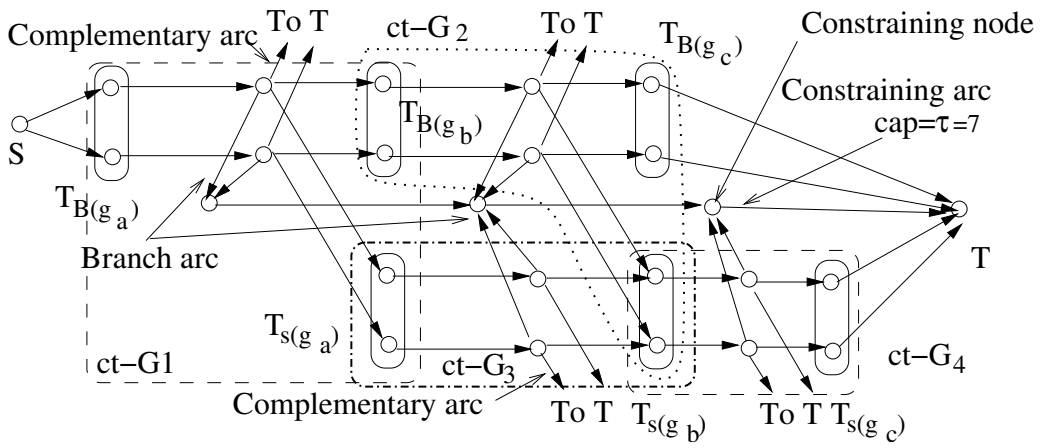

Figure 15. The delay CG of the circuit in Figure 14(a) for the constraint $D(n_i) + D(n_j) \leq \tau$. *ct-G*$_1$ is for p-term $R_d(g_a)C(n_i)$, *ct-G*$_2$ is for p-term $R_d(g_b)C(n_j)$ (*ct-G*$_2$ is shown in detail in Figure 14(b)), *ct-G*$_3$ is for p-term $R_d(g_a)C(g_b)$, and *ct-G*$_4$ is for p-term $R_d(g_b)C(g_c)$.

total flow amount on the pair of branch and complementary arcs of the chosen hyperarc (i.e., chosen partial solution) is always $h_{i,k}^{max}$, irrespective of which partial solution is chosen.

Let $f_{in}(h_{i,k})$ denote the needed incoming flow for ct-subgraph *ct-G*$(h_{i,k})$, $f_{in}(T_j)$ denote the incoming flow amount needed for a transform T_j, and $d_{in}(T_j)$ be the number of ct-subgraphs that have T_j as a fanout transform. $f_{in}(h_{i,k})$ can be expressed as follows.

$$f_{in}(h_{i,k}) = h_{i,k}^{max}(s_{i,k}) + \sum_{T_j \in fanout(h_{i,k})} f_{in}(T_j)/d_{in}(T_j) \tag{3.2}$$

where $fanout(h_{i,k})$ is the set of fanout transforms of *ct-G*$(h_{i,k})$. $f_{in}(T_j)/d_{in}(T_j)$ is the amount of flow that needs to be sent to a fanout transform T_j in a ct-subgraph. For example, in the CG in

Figure 15, $T_{s(g_b)}$ is a fanout transform of two ct-subgraphs, and $T_{B(g_c)}$ is the fanout transform of one ct-subgraph.

Furthermore, the flow $f_{in}(T_j)$ sent to a transform T_j (e.g., $T_{B(g_b)}$ in Figure 15) is the incoming flow to the set (denoted by $ct_{out}(T_j)$) of ct-subgraphs that have T_j as the start transform (e.g., *ot*-G_4 in Figure 15). Hence, we have the following equation:

$$f_{in}(T_j) = \sum_{ct\text{-}G(h_{i,k}) \in ct_{out}(T_j)} f_{in}(h_{i,k}) \tag{3.3}$$

Equation 3.2 and Equation 3.3 constitute the recursive relationship for determining $f_{in}(T_j)$ and $f_{in}(h_{i,k})$ for each transform and ct-subgraph. In the boundary case where a transform T_j is directly connected to the sink node T, $f_{in}(T_j)$ becomes 1 since we still need a unit flow to perform option selection in each of these fanout transforms. With Equation 3.2 and Equation 3.3, we can determine each $f_{in}(h_{i,k})$ and $f_{in}(T_j)$. A transform T_s that is not a fanout transform in any ct-subgraph is connected to the source S. Then, the flow amount that needs to be sent from S is $\sum_{\{T_s\}} f_{in}(T_s)$.

After $f_{in}(h_{i,k})$ for each ct-subgraph *ct*-$G(h_{i,k})$ and $f_{in}(T_j)$ for each transform T_j are determined, the capacities of arcs in the star graph structures in *ct*-$G(h_{i,k})$ can be set accordingly. The same settings are used for all star graph structures in *ct*-$G(h_{i,k})$: the capacity of start arcs is $f_{in}(h_{i,k})$, and the capacity of fanout arcs to transform T_j is $f_{in}(T_j)/d_{in}(T_j)$.

The recursive relationship in Equation 3.2 and Equation 3.3 can also be used to determine the input flow amount $f_{in}(p_k)$ for each ot-subgraph *ot*-$G(p_k)$ (p_k is a p-term in the objective function) and $f_{in}(T_j)$ for each transform T_j in an OG. However, since there are no branch or

complementary arcs in an OG, the first term in Equation 3.2 becomes 0. The capacity of start arcs in *ot-G*(p_k) is $f_{in}(p_k)$, and the capacity of fanout arcs to a transform T_j is $f_{in}(T_j)/d_{in}(T_j)$.

Lemma 2 *In a CG, if the flow amount sent from S is* $\sum_{\{T_s\}} f_{in}(T_s)$*, then in any valid flow, the incoming flow to each ct-subgraph ct-*$G(h_{i,k})$ *has the correct amount* $f_{in}(h_{i,k})$ *determined by the recursive relationship in Equation 3.2 and Equation 3.3.*

Proof Outline: Besides branch arcs and complementary arcs, since the CG is constructed in the same way as the OG, it is also an acyclic graph. The predetermined transform order R is a topological order of transforms in the CG. We can then prove by induction on the rank of transforms T_j in R that ct-subgraphs in $ct_{out}(T_j)$ have the correct incoming flow amount as given Equation 3.2. A detailed proof is given in [35]. $\diamondsuit$

Theorem 3 *In a CG for a feasible constraint* $H_i \leq b_i$*, a valid flow always selects a solution that satisfies the constraint.*

Proof: We prove this by contradiction. Assume a valid flow F_v selects a solution V_s that violates the constraint. Then, the total branch flow amount $H_i(V_s)$ is larger than the constraining arc capacity b_i. Thus, there exits at least one ct-subgraph *ct-*$G(h_{i,k})$ in which part of the branch flow is blocked due to the capacity limit of the constraining arc as shown in Figure 16. Assume that in *ct-*$G(h_{i,k})$, F_v passes through a hyperarc corresponding to a partial solution $s^j_{i,k}$. The branch flow amount f_b is less than the branch arc capacity $h_{i,k}(s^j_{i,k})$, and the constraining arc is already full.

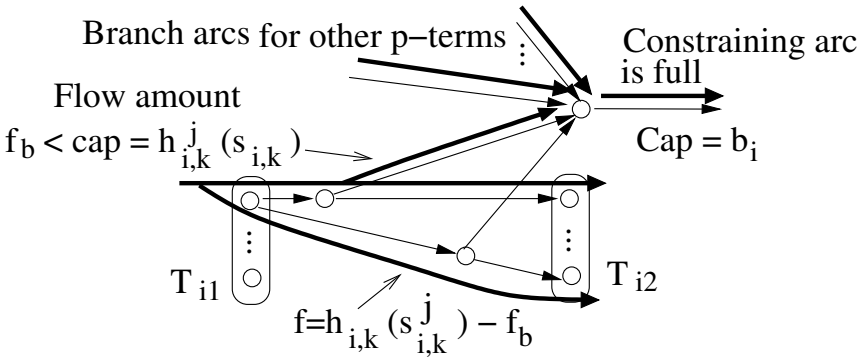

Figure 16. A resulting invalid flow in a ct-subgraph when the selected options violates some constraint $H_i \leq b_i$.

The non-full flow on the branch arc makes the total outgoing flow amount on the hyperarc for $s^j_{i,k}$ less than $f_{in}(h_{i,k})$, while according to Lemma 2 the incoming flow amount to the ct-subgraph is still $f_{in}(h_{i,k})$. This forces a flow amount of $h_{i,k}(s^j_{i,k}) - f_b$ to be routed through other hyperarc(s) in *ct-G*$(h_{i,k})$, which obviously makes F_v an invalid flow, since the PSE requirement is violated. Therefore, we reach a contradiction with our valid flow assumption. $\diamondsuit$

The above theorem also leads to the following corollary.

Corollary 1 *If there is no valid flow possible in the CG, then the constraint is infeasible.* $\diamondsuit$

The infeasibility condition can thus be detected if the min-cost flow in the CG with C' costs is an invalid flow (violates OE and/or PSE requirements).

In the following part, we will illustrate the CG graph for two common metrics: bin capacity and delay. We will illustrate how to build CG or OG graphs for more complex special metrics in later chapters when discussing the corresponding problems.

3.3.2 Bin Capacity Constraints

Usually, in incremental placement, in order to avoid overlapping or make sure that the overlap amount is acceptable, we will divide the whole layout area of a chip into bins, and constraining that the total area of cells in a bin cannot exceeds the bin area. The bin capacity constraints can be formulated as follows. Let $to(B_i)$ be the set of cells that can be moved into a bin B_i (including the current cells of B_i), $\{g_d\}$ be the set of gates that drive a gate g, a_l be the area of a level shifter. We also define two binary variables: 1) $b(g, B_i)$, which is 1 if we choose to put a cell g in B_i, and 0 otherwise; 2) $e(g_d, g)$, which is 1 if a level shifter exists (is needed) in the interconnect from g_d to g, and 0 otherwise. To satisfy the capacity constraint of bin B_i, we must ensure that:

$$\sum_{g \in to(B_i)} [s(g)b(g, B_i) + \sum_{\forall g_d} a_l \cdot e(g_d, g)b(g, B_i)] < area(B_i)$$

where $area(B_i)$ is the area of B_i. The second p-term in the above formulation is for level shifter area, which should be put in the same bin as g. $b(g, B_i)$ is a function of the position $B(g)$ for g, which is determined by the re-placement transform $T_{B(g)}$. Thus, each ct-subgraph for the first p-term $s(g)b(g, B_i)$ is a complete bipartite graph between two transforms $T_{s(g)}$ and $T_{B(g)}$ with branch and complementary arcs. The capacity of the branch arc attached to a hyperarc connecting a sizing option $O^k_{s(g)}$ is the cell area corresponding to the option. $e(g_d, g)$ is determined by V_{dd}'s of g_d and g. Hence, the corresponding ct-subgraph includes $T_{V_{dd}(g_d)}$, $T_{V_{dd}(g)}$ and $T_{B(g)}$. The branch arc capacity in the ct-subgraph is always a_l.

P-term type	**Transforms in the ct-subgraph**
$R_d(n_k)cL(n_k)$	$T_{s(g_i)}$, $T_{V_{dd}(g_i)}$, $T_{V_{th}(g_i)}$ and $T_{B(g)}$ for $g \in n_k$
$R_d(n_k)C(g_k)$	$T_{s(g_i)}$, $T_{V_{dd}(g_i)}$, $T_{V_{th}(g_i)}$ and $T_{s(g_k)}$
$rc \cdot l_{i,j}^2$	$T_{B(g_i)}$ and $T_{B(g_k)}$
$rl(i,j)C(g_k)$	$T_{B(g_i)}$, $T_{B(g_k)}$ and $T_{s(g_k)}$
$rc \cdot l(i,j)L(n_k)$	$T_{B(g_i)}$, $T_{B(g_k)}$ and $T_{B(g)}$ for $g \in n_k$

TABLE VIII

THE TRANSFORMS IN CT-SUBGRAPHS FOR DIFFERENT TYPES OF P-TERMS IN THE TIMING CONSTRAINT FUNCTION.

3.3.3 Circuit Delay Constraint

To satisfy circuit delay constraint, we consider a set $\mathcal{P}$ of critical and near critical paths. One constraint is set for each path $P \in \mathcal{P}$ as follows:

$$\sum_{(g_i,g_j)\in P} D(g_i, g_j) < \tau \tag{3.4}$$

where (g_i, g_j) is an interconnect between cells g_i and g_j, $D(g_i, g_j)$ is the delay on (g_i, g_j), and τ is the given circuit delay constraint. The formulation for $D(g_i, g_j)$ is given in Equation 1.1. The various types of p-terms, and the related transforms of each p-term is listed in Table VIII. Ct-subgraphs can be constructed accordingly. A detailed example is given in Figure 14-Figure 15 and discussed in Sec. 3.3 for a subset of p-terms of $D(g_i, g_j)$ and two transforms (for simplicity of exposition).

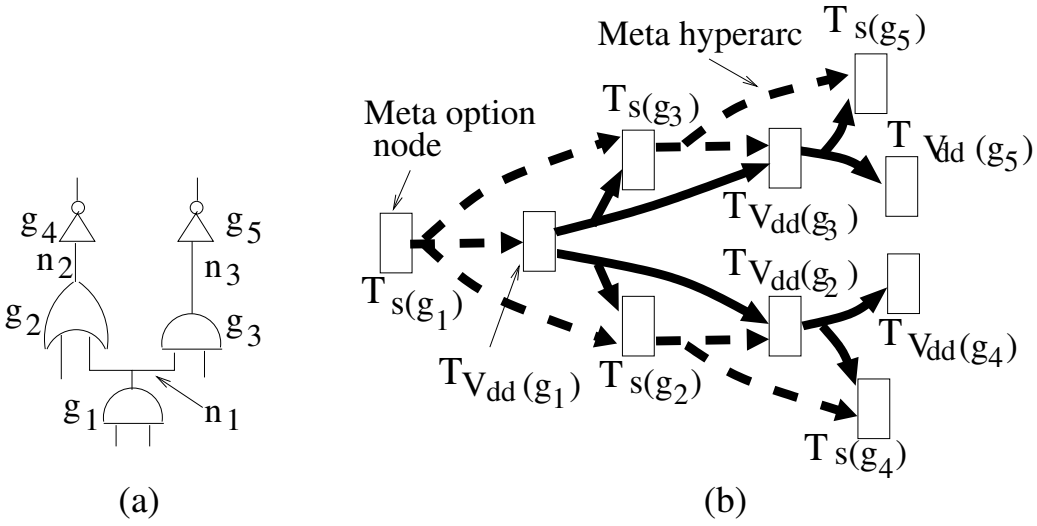

Figure 17. (a) A subcircuit with three nets and five gates. (b) A part of the COSG for the subcircuit in [a] after combining the power OG and the circuit delay CG. For clarity we only show supply voltage and cell sizing transforms. Each meta option node represents the set of option nodes for a transform; each meta-hyperarc represents the set of hyperarcs in a subgraph. The dashed meta-hyperarcs are delay ct-subgraphs for circuit delay p-terms of type $R_d(n_k)C(g_k)$ for the three nets in [a], where $R_d(n_k)$ is the driving resistance of n_k and $C(g_k)$ is the input capacitance of a sink cell in n_k. The solid meta-arcs are power ot-subgraphs for power p-terms P_c in Equation 1.7 for the three nets.

An alternative way of satisfying the circuit delay constraint when the number of critical and near critical paths is large is using slack allocation, and setting a delay upper bound for each interconnect, which is its original delay plus its allocated slack. As is well known, such a net based constraint over-constrains the basic problem, which is essentially path-based. In our implementation, we used the path based constraint (Equation 3.4).

3.4 Complete Option Selection Network Flow Model

For an option selection problem with an objective function F and multiple constraints, the complete option selection network flow graph (COSG) is formed by combining the OG for F and the CGs for all the constraints. The combination is still done by merging common transforms in these graphs. All MEA constraints in the OGs and the CGs are inherited in the complete graph, i.e., a valid flow in the COSG has to satisfy all these MEA constraints. Hence, a valid flow in the COSG consists of valid sub-flows in its constituent OG and CGs. Further, the MEA constraint on bridge arcs ensures that only one option is selected for each transform by a valid flow in the COSG, i.e., all valid sub-flows in the COSG's constituent graphs select the same set of options. Figure 17(b) illustrates part of the COSG after merging the power OG and the delay CG for the subcircuit shown in Figure 17(a).

As described in Sec. 3.2.3, we use C'-costs on MEA arcs to ensure that for a feasible problem, a min-cost flow is a valid flow. However, for simplicity of exposition, in the rest of this subsection, by the cost of valid flows we only mean the C-cost part of the flow cost (i.e., the corresponding p-term values of the optimization function).

Lemma 3 *A valid flow in the COSG selects a feasible solution of the corresponding option selection problem, if one exists.*

Proof: A valid flow in the COSG that selects a solution S_o consists of valid sub-flows in each constituent CGs that together select S_o. Based on Theorem 3, this means that S_o satisfies all the given constraints, and is a valid solution. $\diamondsuit$

The option selection problem can be solved according to the following theorem.

Theorem 4 *A min-cost valid flow in the COSG selects the optimal feasible solution for the corresponding option selection problem, if the problem is feasible.*

Proof: For a valid flow in the COSG that selects a solution S_o, based on Lemma 1, its cost in the OG part in the COSG is $F(S_o)$, where F is the objective function. Further, the arcs in CGs all have 0 cost. Hence, the min-cost valid flow in the COSG selects the feasible solution S_o that minimizes F. $\diamondsuit$

The problem of finding the min-cost valid flow is equivalent to the problem of finding the min-cost flow in the COSG with C' cost, according to Theorem 2. We solve the latter problem near optimally using the algorithm in [34]. Also, similar to Corollary 1 for CGs, an invalid flow in the COSG indicates that the option selection problem is infeasible. This condition can be easily verified by checking if the min-cost flow in the COSG with C' costs is a valid flow.

3.4.1 Timing Complexities

We analyze the worst and average time complexity of solving the power optimization problem with the four types of transforms mentioned earlier under cell area, critical path delay and voltage island constraints. Given a circuit with N cells and E nets, the maximum number m of transforms that a p-term is a function of, and the maximum number p of options of any transform, it is easy to show that: a) the total number of ot- and ct-subgraphs in the COSG is $O(N+E)$, and b) the total number of arcs in the COSG is $O(mp^m(N+E)) = O(N+E)$ as m and p are constants (and generally small ones) w.r.t. circuit size $N+E$; details are given in [35].

To solve the min-cost flow in the COSG with concave arc costs (see Figure 12(b)), we use the method proposed in [34], which solves the standard (i.e. with linear arc costs) min-cost network flow problems on the same network flow graph repeatedly with dynamically adjusted linear arc costs to approximate the concave costs. The complexity of solving a standard min-cost flow problem is $O(U \cdot A^2)$ [36], where U is the amount of flow to be sent, and A is the number of arcs. In the COSG model, the amount of flow has to be able to support all branch flow amounts, and hence is proportional to the sum of the maximum possible values of all constraint functions. This value is in turn proportional to the number of p-terms in constraint functions, since the maximum value for each p-term (e.g., a maximum net delay or a maximum cell size) is usually a small constant, and thus $U = O(N + E)$. Therefore, the complexity of solving the standard min-cost flow on the COSG is $O((N + E) \cdot (N + E)^2) = O((N + E)^3)$. Let I_t denote the number of the standard network flow iterations needed when solving the DNF problem. Then, the worst case complexity of our method is $O(I_t(N + E)^3)$. Since in ASIC circuits, the number of nets and the number of cells are of the same order, the worst case complexity can be simplified to $O(N^3 I_t)$.

We now derive the average-case complexity of DNF. As shown in Table X, the number I_t of standard network flow iterations increases much slower than the circuit size increase. Hence, the average complexity of our technique is mostly determined by the average complexity of the network simplex method. Despite the $O(A^2U)$ worst case complexity, the average complexity of the network simplex method ranges only from $O(A)$ to $O(A^2)$ [37]. This makes the average complexity of our method range from $O(N)$ to $O(N^2)$ This linear average-case complexity is

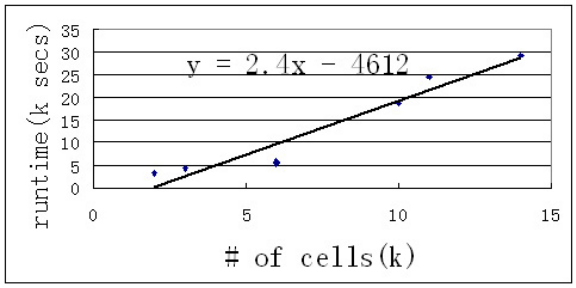

Figure 18. Runtime plot of the DNF method versus the number of cells when applying four transforms to perform power optimization.

borne out by empirical data shown in the runtime plot in Figure 18, which shows that a linear function is the best fit for the runtime data points.

CHAPTER 4

APPLYING DNF TO VARIOUS PHYSICAL SYNTHESIS PROBLEMS

In this chapter, we will illustrate how to solve various physical synthesis problems with DNF, and the corresponding results.

4.1 Solving the Timing-Driven Physical Synthesis Problem Using DNF

Given a placed circuit $\mathcal{PC}$, to optimize timing, we are consider five types of timing improvement physical synthesis transforms: cell sizing, incremental placement, cell replication and two types of buffer insertion. The objective is to produce a new placed circuit with improved critical path delay. The constraint we consider is the total area. We define $\mathcal{P}_\alpha$ to be the set of paths with a delay greater than a $1-\alpha$ ($\alpha < 1$) fraction of the most critical path delay. In our experiments, α is set to be 0.1. In order to develop an efficient approach, we only consider applying transforms on cells that are in $\mathcal{P}_\alpha$. This simplification does not limit the optimization potential of our method, since our method can be used in an incremental way. Specifically, we can iterate it several times to take more paths into consideration until a desired circuit performance under the given constraints is reached (or failing which, the best possible performance under the given constraints is obtained).

The options for cell sizing, cell replication and the two buffer insertion transforms are the same as described in Chapter 1. For incremental placement, we pre-determine the timing optimal positions of cells in $\mathcal{P}_\alpha$ with timing-driven global placement TAN introduced in Chapter 2.

We then set up two options for cell replacement for cells in $\mathcal{P}_\alpha$: remaining in the original position or being moved to the new position indicated by TAN. It should be noted that moving cells to a global timing optimal but illegal placement position does not guarantee a timing improvement considering the detailed placement cost. Hence the original position can be a better choice in some cases. In our algorithm, along with the choice of a placement option, a flow through it will be sent to the corresponding position in the DFP graph (see Sec. 2.2.1) to find a legal position, and estimate the legalization cost. Figure 19 shows an example of injection arcs from the options of placement transform $O_{pl}(u)$ for a cell u to the DFP. $pos_u(i)$ denotes the global placement position of u determined by option i for transform $O_{pl}(u)$.

We define $CS(n_j)$ to be the set of critical sinks of n_j, which is: 1) all sinks in $\mathcal{P}_\alpha$ if the net is in $\mathcal{P}_\alpha$, or 2) the sink with the minimum slack otherwise. We then have the following delay objective functions F_t as the weighted summation of critical interconnect delay:

$$F_t = \sum_{n_j} \sum_{u_i \in CS(n_j)} D(u_i, n_j)/S_a^\alpha(n_j, u_i) \tag{4.1}$$

where $D(u_i, n_j)$ is the delay of net n_j to a sink u_i, $S_a(n_j)$ is the allocated slack of n_j to u_i (slack of the max-delay path through n_j and u_i divided by the number of nets in the path), and α is the exponent of $S_a(n_j, u_i)$ used to adjust the weight difference in F_t between critical and non-critical paths. Since the weight is inversely proportional to the allocated slack, interconnects on critical paths have larger weight. Based on experimental results, α is chosen as 1. By only considering critical sinks in F_t, we reduce the effect of fairly non-critical paths when performing timing optimization.

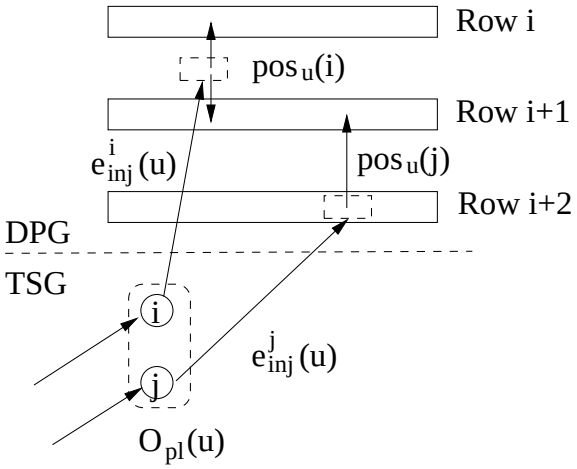

Figure 19. Placement injection arcs from different placement options of a cell u.

4.1.1 Experimental Results

We used three benchmark suites in our experiments: 1) The TD-Dragon suite of [32], 2) ISCAS'85 benchmarks and 3) TD-IBM benchmark suite from [4]. We consider five different sizes of a cell for sizing transform: $w/4$, $w/2$, w, $2w$ and $4w$ where w is the original width of the cell. Results were obtained on Pentium IV machines with 1GB of main memory.

The size of the benchmarks are shown in Table IX. The initial delay, area and WL metrics are given in Table III in Sec 2.4. We obtain results for our method DNF that considers synthesis transforms simultaneously, and for a method that applies transforms in a good sequential order. In the latter case, we apply one transform at a time to all cells and nets in $\mathcal{P}_\alpha$ in the order of decreasing ratio of timing-improvement to area-increase of each transform. This provides the maximal benefit-to-cost ratio for a sequential approach, and, as indicated by our experiments,

Ckt	# cells	# cri. cells	Our method (DNF) %ΔT	%ΔA	%ΔWL	runtime (secs)	Seq. appl. of transforms %ΔT	%ΔA	%ΔWL	runtime (secs)
td-ibm01	12K	266	28.0	-2.2	-11.3	633	20.1	-2.2	-9.8	275
td-ibm02	19K	584	29.7	-2.4	-12.6	628	24.6	-2.4	-10.5	338
td-ibm03	23K	425	44.2	-1.9	-8.8	993	37.1	-2.4	-8.9	529
td-ibm04	27K	559	27.8	-1.9	-12.4	1151	28.7	-2.1	-11.0	684
td-ibm05	28K	593	31.9	-2.5	-8.0	1134	23.0	-2.0	-8.5	649
td-ibm06	32K	572	37.7	-2.2	-11.2	1649	24.5	-2.0	-8.6	1124
td-ibm07	46K	705	33.4	-2.6	-8.6	1697	17.0	-1.9	-7.5	1134
td-ibm08	51K	694	39.0	-1.9	-7.2	1827	31.5	-1.7	-8.0	1055
td-ibm09	53K	889	34.9	-1.8	-10.2	2985	23.6	-2.0	-11.0	1722
td-ibm10	69K	912	23.2	-2.4	-8.1	3028	17.2	-1.9	-8.0	1803
td-ibm11	70K	910	31.5	-1.6	-10.5	3417	32.4	-2.7	-9.8	2154
td-ibm12	70K	916	38.4	-1.6	-10.8	2158	29.9	-1.5	-9.3	1536
td-ibm13	84K	889	36.2	-1.7	-9.4	3338	26.0	2.6	-8.6	1689
td-ibm14	147K	1185	22.4	-1.5	-7.5	4028	15.8	-1.6	-7.0	2199
td-ibm15	161K	1201	37.5	-1.4	-5.9	4087	25.8	-1.3	-5.7	3318
td-ibm16	183K	1234	38.2	-1.3	-7.0	5847	21.2	-1.1	-7.9	3542
td-ibm17	185K	1687	48.0	-1.3	-3.3	5812	32.3	-1.3	-5.5	3914
td-ibm18	210K	1779	46.0	-0.9	-3.3	5993	36.6	-0.9	-4.8	4378
Avg.		**888**	**34.8**	**-1.8**	**-8.7**	**2796**	**25.9**	**-1.7**	**-8.4**	**1780**
matrix	3.0K	201	9.5	-9.5	-10.2	1124	6.1	-9.9	-13.0	992
vp2	8.9K	218	13.7	-9.0	-10.2	1347	6.9	-9.9	-12.9	1038
mac32	25K	257	18.6	-9.2	-9.8	1397	11.1	-9.8	-11.9	1134
mac64	8.7K	314	18.4	-8.9	-7.6	1319	11.3	-10.0	-12.4	1208
Avg.		**248**	**15.1**	**-9.2**	**-9.5**	**1296**	**8.8**	**-9.9**	**-12.6**	**1093**
C432	160	50	16.8	-9.7	-11.4	334	8.5	-10.0	-14.5	129
C499	202	51	17.8	-10.0	-12.5	276	9.1	-10.0	-12.7	211
C880	383	77	19.5	-10.0	-11.2	248	9.5	-10.0	-13.1	231
C1355	544	85	15.9	-9.5	-9.6	382	10.6	-10.0	-12.0	244
C1908	880	88	24.1	-9.8	-12.1	376	14.9	-9.5	-12.3	223
C2670	1.3K	91	15.1	-9.2	-8.4	357	10.5	-9.8	-10.1	241
C3540	1.7K	124	25.8	-9.2	-11.6	501	15.6	-10.0	-13.3	338
C5315	2.3K	138	25.8	-9.3	-8.9	772	17.2	-9.5	-8.9	361
C6288	2.4K	299	25.4	-9.0	-7.5	859	16.9	-9.7	-10.0	593
C7552	3.5K	199	18.1	-9.3	-8.8	724	12.2	-9.9	-8.5	674
Avg.		**120**	**20.4**	**-9.5**	**-10.2**	**483**	**12.5**	**-9.8**	**-11.5**	**314**
Overall Avg.		**568**	**27.8**	**-5.1**	**-9.2**	**1885**	**19.6**	**-5.3**	**-9.9**	**1236**

TABLE IX

RESULTS FOR OUR METHOD DNF AND FOR A SEQUENTIAL APPLICATION OF TRANSFORMS. THE COMPARISONS ARE TO INITIAL PLACEMENTS BY DRAGON 2.23 (SEE Table III FOR INITIAL PLACEMENT RESULTS). %ΔT IS THE PERCENTAGE TIMING IMPROVEMENT, %ΔA AND %ΔWL ARE THE PERCENTAGE CHANGES OF TOTAL CELL AREA AND WL, RESPECTIVELY (A NEGATIVE VALUE INDICATES DETERIORATION).

also provides the best delay improvements compared to other orders like best delay improvement. Thus we apply transforms in the order: (1) incremental placement, (2) type 2 buffer insertion, (3) cell resizing, (4) type 1 buffer insertion, and (5) cell replication. After applying all the transforms, we perform incremental detailed placement using the DFP to get a legal solution. The algorithm for incremental placement is explained in Chapter 2, which, as mentioned earlier, is also used in DNF to perform incremental placement simultaneously with the synthesis option selection process in the TSG. The algorithm for buffer insertion is from [21], and the replication algorithm is from [13]. For cell resizing, we use a network flow technique with only cell-sizing options [38]. We compare this technique to an exhaustive enumeration sizing algorithm that gives the optimal solution. For small circuits on which we could perform exhaustive enumeration, the results produced by our resizing technique is only an absolute of 1% off the optimal value at $\frac{1}{60}$'th of its runtime [38], which establishes both the quality and efficiency of our network flow based cell sizing algorithm.

The results for DNF and sequential transform applications are shown in Table IX. For TD-IBM benchmarks, we allow an extra 3% layout area compared to the original layout so that the total cell area increase is within 3%. For TD-dragon and ISCAS'85 benchmarks, this parameter is set to be 10% since these circuits are relatively small. We obtain a timing improvement of up to 48% and an average of 27.8% while satisfying the given area constraints. Our method is consistently better than the sequential application of transforms. The average timing margin between our method and the sequential application one is an absolute of 8.2% (27.8% vs 19.6%) of the delays of the initial designs, and 40% relatively. Compared to the

delay obtained by the sequential application one, DNF improves it further by 10.2% on average ($10.2\% = 8.2\% \div (1 - 19.6\%)$).

Furthermore, if we want to perform in-place physical synthesis, i.e., we do not want to change the position of cells in order to preserve interconnect-related metrics of the initial placement such as routability, WL and net switching power, then the placement option will not be considered. The average timing improvements in this scenario for DNF and the sequential method are 18.9% and 12.6%, respectively, which implies a 50% relatively better performance by our technique. These results show that our algorithm is much better at finding global near-optimal choices than the sequential method. The WL and cell area increases are similar for both approaches. Our run time is only about 50% larger than that of the sequential approach.

4.2 Solving Power-Driven Physical Synthesis Problems Using DNF

We are solving the following problem in this section. Given a circuit $\mathcal{C}$ and an initial sizing, V_{dd}, V_{th} assignment, and placement solution for $\mathcal{C}$, we need to re-determine cell V_{dd}'s and V_{th}'s, cell sizes and their possibly new placement positions in order to optimize dynamic and leakage power consumption subject to circuit delay, cell area and voltage island shape and number constraints. The final output is a new legal placement that optimizes power and satisfies the given constraints.

Our algorithm is applied in the physical synthesis and placement stage, and produces a legal placement at the end. Assigning V_{dd} in the physical synthesis and placement stage generates another problem, that of voltage island formation. In order to facilitate an efficient power distribution network, cells with the same V_{dd} should be clustered into several rectangular regions

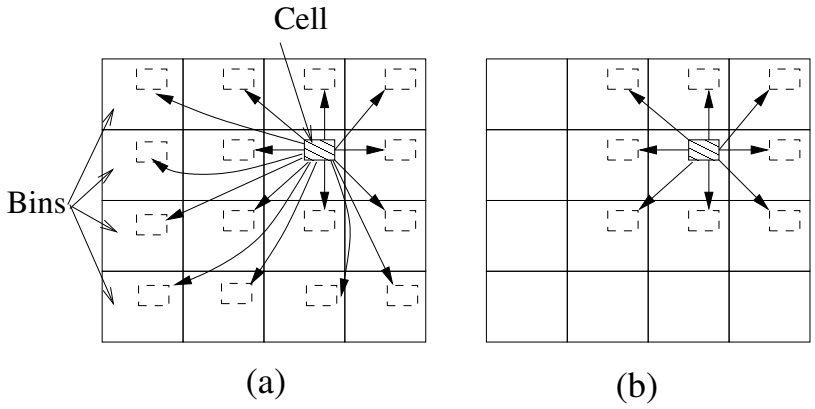

Figure 20. (a) Possible inter-bin movement options for a cell. (b) In our implementation, cell movements are limited to adjacent bins.

called "voltage islands" in the layout. Due to design and routing cost considerations, there is usually an upper bound on the total number of voltage islands [39]. Most current V_{dd} assignment methods are designed for unplaced circuits, and hence do not take voltage island formation into consideration. Wu et. al. [39] proposes a V_{dd} adjustment method after the placement stage that modifies a V_{dd} assignment solution in order to help form voltage islands and satisfy the island number constraint. Clearly, a post-processing adjustment method cannot give an optimal solution. On the other hand, our proposed method will simultaneously perform V_{dd} assignment as well as other power reduction transforms and satisfy the constraint of generating rectangular voltage islands up to a given upper bound. Besides, the voltage island number and shape constraint, we also consider the regular delay and area constraint.

In order to satisfy the voltage island constraints, cells are usually needed to be moved close to other cells with the same V_{dd}. Hence, we are providing more incremental placement options in our DNF structures as follows. The layout area is divided into bins. In the incremental placement transform, a cell in bin A in the initial placement can choose either to stay at its current position A or to move to another bin, as shown in Figure 20(a). In our implementation, we limit cell movement to only adjacent bins, i.e., only A and the eight bins around A as shown in Figure 20(b) are available options for the cell. The rationale for our choice to make incremental placement limited to adjacent bins is to prevent significant deteriorations in routability and WL from the initial placement, which is usually a WL driven one.

4.3 Satisfying Voltage Island Constraints

The voltage island constraint states that the assigned cell V_{dd}'s and their positions must be such that the cells are clustered into at most N_v^{up} iso-voltage rectangular regions (i.e., regions which contain only cells with the same V_{dd} assignment); N_v^{up} is the given voltage island number upper bound. The reason for the rectangular shape is that it is difficult to produce a high-quality power distribution network for islands with irregular shapes [40]. The reason for limiting the number of voltage island is that using more islands increases the design complexity and occupies more routing resources to connect these islands to power inputs.

In this work, instead of using a post processing V_{dd} adjustment method as in [39], we satisfy this constraint simultaneously with V_{dd} and other transform option selections. We simultaneously generate bin-level voltage islands, and thus we additionally impose an auxiliary constraint that all cells in a bin have to be assigned the same V_{dd}; we call this the *iso-bin*

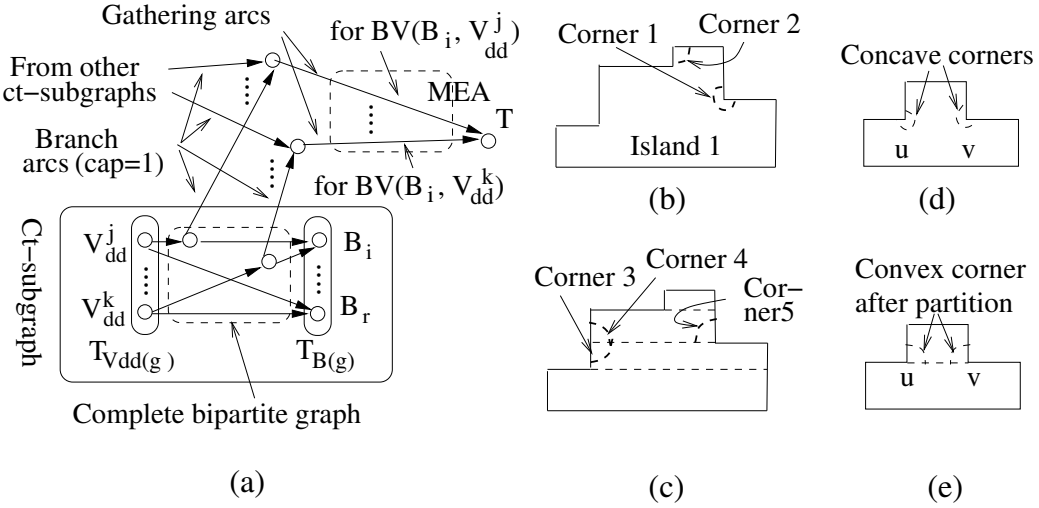

Figure 21. (a) A ct-subgraph and associated structure for satisfying the iso-bin-V_{dd} constraint. The branch arcs here have cap=1. (b) Zigzag shaped region formed by voltage boundaries. (c) Rectangular voltage islands generated in it. (d) Two concave corners at intersection points u and v in natural islands that are collinear. (e) After partitioning the two concave corners, two convex corners are generated.

V_{dd} constraint. With this extra constraint, we can generate voltage islands based on the V_{dd}'s of bins. Note that cells are moved to their final bin positions by a part of the flow, as we simultaneously determine the necessary number of bin-level voltage islands and constraint it to be $\leq N_v^{up}$.

4.3.1 Bin Based V_{dd} Assignment

The iso-bin-V_{dd} constraint is different from the area/delay constraints we dealt with previously (e.g., in Secs 3.3.2-3.3.3). For the latter, an upper bound is given on the constraint metric, while the iso-bin-V_{dd} constraint is a mutual exclusiveness constraint as formulated next.

We define a binary variable $v(g, V_{dd}^j)$, which is = 1 if a gate g is assigned a supply voltage V_{dd}^j, and = 0 otherwise. The possibility of a bin B_i using a supply voltage V_{dd}^j can be represented by the *bin voltage selection* variable $BV(B_i, V_{dd}^j)$ defined as:

$$BV(B_i, V_{dd}^j) = \sum_{g \in to(B_i)} v(g, V_{dd}^j) \cdot b(g, B_i) \tag{4.2}$$

Recall from Sec. 3.3.2 that $b(g, B_i)$ is a binary variable indicating if g is placed in B_i, and $to(B_i)$ is the set of gates that can be moved to B_i. $BV(B_i, V_{dd}^j) = 0$ when no gate moved to B_i is assigned voltage V_{dd}^j, and $BV(B_i, V_{dd}^j) > 0$ otherwise. However, for B_i to be contained in a voltage island it is necessary that all gates moved to it be assigned the same V_{dd}. This gives rise to the following mutual exclusiveness constraint: In set $\{BV(B_i, V_{dd}^j) \mid \forall\ V_{dd}^j\}$ only one $BV(B_i, V_{dd}^q)$ can be > 0.

In order to satisfy this constraint, we can use the previously described constraint satisfaction graph structure, but with a small difference. Cell-centric ct-subgraphs are used to generate the branch flows corresponding to p-terms $v(g, V_{dd}^j) \cdot b(g, B_i)$ of Equation 4.2. But in each ct-subgraph g is a constant, and V_{dd}^j and B_i vary across all possible option nodes for the $T_{V_{dd}(g)}$ and $T_{B(g)}$ transforms; see Figure 21(a). Thus each such ct-subgraph has multiple p-terms (all for gate g), and these p-terms are in Equation 4.2 of different $BV(B_i, V_{dd}^j)$ variables of B_i's to which g can be moved. Such a ct-subgraph is shown in Figure 21(a), and includes $T_{V_{dd}(g)}$ and $T_{B(g)}$ of a gate g. The branch arc attached to an option selection arc (V_{dd}^j, B_i) is responsible for generating the branch flow for p-term $v(g, V_{dd}^j) \cdot b(g, B_i)$. The flow amount on the branch arc is 1 (equal to its capacity) when (V_{dd}^j, B_i) is chosen, and 0 otherwise. This is exactly equal to

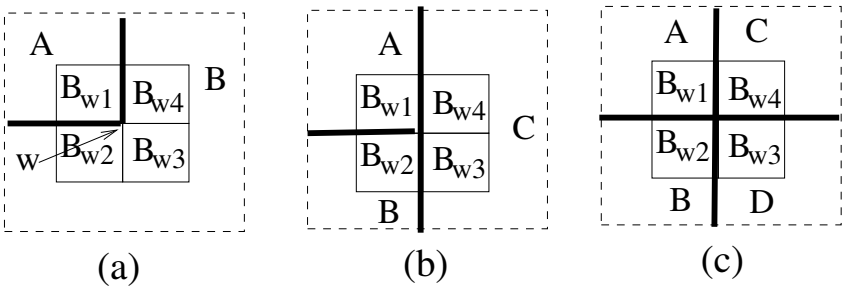

Figure 22. Three possible intersection patterns of voltage boundaries (indicated by thick lines) at a bin corner w. The four bins around w are shown.

the p-term value. The branch flow is sent to the "gathering" arc for the bin voltage selection variable $BV(B_i, V_{dd}^j)$ that the p-term belongs to.

For each $BV(B_i, V_{dd}^j)$ there is a *gathering* arc (analogous to the constraining arc of $H_i \leq b_i$ type constraints) $e(B_i, V_{dd}^j)$ that collects branch flows from p-terms across such ct-subgraphs of gates $g \in to(B_i)$; see Figure 21(a). Since there is no upper bound constraint on the value of each $BV(B_i, V_{dd}^j)$, the capacity of a gathering arc is infinite. To satisfy the aforementioned mutual exclusiveness constraint, as shown in Figure 21(a), an MEA constraint is applied to the set of gathering arcs each of B_i, i.e., to each set of arcs: $\{e(B_i, V_{dd}^j) \mid \forall\, V_{dd}^j\}$.

4.3.2 Satisfying Voltage Island Number and Shape Constraints

When a flow assigns a V_{dd} to each bin (as described in Sec. 4.3.1), *voltage boundaries* are automatically generated between adjacent bins with dissimilar V_{dd}'s. A voltage boundary is either a vertical or a horizontal bin boundary between two bins with different V_{dd}'s. Voltage boundaries formed by the V_{dd} selection flows for each bin form *natural* voltage islands that

minimize power. Natural islands can have zigzag shapes (as opposed to rectangular ones) as is the case for island 1 in Figure 21(b). To form rectangular islands, we further determine their natural partitions into such regions as shown in Figure 21(c). Therefore, the key issues are to determine in the DNF flow the number of natural partitions across all zigzag islands into rectangular ones (for naturally occurring rectangular islands there will be no partitions), relate this to the total number of all rectangular islands, and constrain this number to satisfy the given upper bound N_v^{up}. A critical determination in this is to identify where at the boundaries of zigzag regions' partitions are needed to form maximal rectangular islands. Towards this end, we define a concave (convex) corner on a natural island I_j as one whose internal angle in I_j is 270 degrees (90 degrees); see Figure 21(b). A partition is needed at each such corner, since a rectangular island cannot have concave corners. The partition is done consistently horizontally or consistently vertically from each concave corner until the opposite boundary of the zigzag region is reached. Partitioning a zigzag island at each concave corner

$$N_v = (3N_{270} + N_{90})/4 \tag{4.3}$$

where N_{270} is the number of concave corners and N_{90} is the number of convex corners in the original (non-partitioned) natural islands. The division by 4 reflects the fact that each rectangular island has 4 (unshared) convex corners.

Note that corners on natural islands can only occur when a horizontal voltage boundary intersects a vertical boundary. There are three different intersection scenarios: 1) The intersection forms a convex corner in a natural island A and a concave corner in a natural island

B as shown by Figure 22(a). 2) It forms two convex corners in two natural islands A and B as shown in Figure 22(b); 3) It forms four convex corners in four different natural islands as shown in Figure 22(c). The patterns shown in Figure 22(a-b) for scenarios 1 and 2 can be rotated in multiples of 90 degrees to yield other isomorphic patterns of the same scenarios. Based on Equation 4.3, each scenario i contributes to N_v by an a-prior known amount $\Delta N_v(i)$. We should further note that the intersection of voltage boundaries can only happen at bin corner points w, and that the different intersection patterns can be determined by the chosen V_{dd}'s of the four bins $B_{w1}, \ldots, B_{w4}$ around w as shown in Figure 22. Hence, we can construct a ct-subgraph like structure for each bin corner point w (the total number of bin corners is $O(N)$, where N is the number of cells in the circuit) that allows a flow amount equal to the ΔN_v corresponding to the intersection pattern at w, simultaneously with the V_{dd} selection of the four bins around w. This structure contains the V_{dd} options of the four bins around w, and hyperarcs connecting all possible combinations of the four V_{dd} selections; if there are r V_{dd} levels, then there are r^4 combinations for the four bins' V_{dd}'s, and thus as many hyperarcs (generally $r \in [2, 4]$, so this is not a large number). The capacity of the branch arc on a hyperarc connecting options is equal to the ΔN_v corresponding to that combination of bin V_{dd}'s connected by the hyperarc. For example, for a hyperarc that connects those V_{dd} option nodes for the 4 bins for which $V_{dd}(w_1) \neq V_{dd}(w_2) = V_{dd}(w_3) = V_{dd}(w_4)$, we obtain the scenario-1 boundary intersection ($V_{dd}(w_i)$ denotes the V_{dd} chosen for bin B_{wi}). For this case, we know from Equation 4.3 that it contributes a total count of 4 (3 due to one concave corner plus one original convex corner) to N_v ($\Delta N_v = 4$); thus a branch arc from such hyperarcs will have a

capacity of 4. Flows from all the branch arcs across all such structures of all 2×2 bin regions (the number of these regions are proportional to the number of bins and thus is $O(N)$) are gathered and sent into a constraining arc with a capacity of N_v^{up} to ensure satisfying the upper bound constraint on the number of rectangular islands.

Our aforementioned method works accurately in most cases. However, in some cases, concave corners of a natural island at its opposite boundaries can be collinear as shown in Figure 21(d). This means that the partitions they engender are the same; see Figure 21(e). We thus we double-count the number of voltage rectangles for these two concave corners, and thereby over-constrain the problem.

The following result establishes the basic goodness vis-a-vis the power metric of our method for rectangular voltage island determination and constraint satisfaction. The COSG contains the ct-subgraphs for: a) the iso-bin-V_{dd} constraint (Sec. 4.3.1), and b) the voltage island constraint discussed here.

Theorem 5 *If there are no collinear concave corner pairs across the natural voltage islands formed by our method, then the min-cost valid flow in the COSG determines a feasible optimal solution. Otherwise, assuming that the probability of collinear pairs is very small, in the general case a min-cost valid flow in the COSG provides a near-optimal solution* [1].

Proof: Follows from the previous discussion; see [35] for details. $\diamondsuit$

---

[1] *The other source of near-optimality in our technique, alluded to earlier, is not a modeling one, but due to our use of the concave near-min-cost algorithm of [34]—the concave min-cost problem is NP-hard.*

Ckt	**# cells**	**transf.**		**3 transf.**		**transf.**							
		DNF	**[19]**	**DNF**	**[23]**	**DNF**				**Seq-std**		**Seq-nf**	
		ΔP %	**ΔP %**	**ΔP %**	**ΔP %**	**ΔP %**	**t (sec)**	**# VI**	I_t	**ΔP %**	**t (sec)**	**ΔP %**	**t (sec)**
s5378	2K	14	16	25	21	30	0.9h	12	288	18	0.4h	17	0.4h
s9234	3K	17	15	22	22	24	1.2h	15	241	21	0.8h	23	0.5h
s13207	6K	17	18	23	13	26	1.5h	13	229	23	1.2h	26	0.6h
s15850	6K	11	13	27	13	21	1.6h	16	195	14	0.9h	16	0.8h
s38417	10K	12	10	18	14	29	5.2h	16	315	14	1.7h	13	1.6h
s38584	11K	19	17	30	15	25	6.8h	16	317	17	1.8h	20	1.6h
s35932	14K	17	17	29	18	34	8.1h	16	344	20	1.9h	18	1.8h
Avg.		**15**	**15**	**25**	**17**	**27**	**3.6h**	**15**		**18**	**1.1h**	**19**	**1.0h**
DMA	12K	16	13	25	17	29	3.4h	16	297	16	1.7h	16	1.1h
DSP1	26K	12	10	19	11	21	6.9h	15	308	11	3.5h	14	2.4h
DSP2	26K	12	10	18	10	21	7.0h	16	303	12	3.7h	14	2.4h
RISC1	33K	19	16	27	18	25	9.7h	15	320	18	7.1h	17	3.6h
RISC2	33K	17	15	27	17	27	9.6h	16	316	22	6.9h	24	3.7h
Avg.		**15**	**13**	**23**	**15**	**25**	**7.3h**	**16**		**16**	**4.6h**	**17**	**2.6h**
Overall Avg.		**15**	**14**	**24**	**16**	**26**	**5.1h**	**15**		**17**	**2.6h**	**18**	**1.7h**

TABLE X

RESULTS FOR OURS AND COMPETING METHODS FOR THREE DIFFERENT CONFIGURATIONS WITH DUAL V_{DD}'S USED FOR THE MULTIPLE V_{DD} TRANSFORM. THE INITIAL DESIGNS ARE OBTAINED THROUGH THE SYNTHESIS TOOL SYNOPSYS'S "DESIGN COMPILER" WITH THE HIGHEST SUPPLY VOLTAGE AND THE LOWEST THRESHOLD VOLTAGE FOR ISCAS'89 BENCHMARKS. FOR FARADAY BENCHMARKS, THE INITIAL DESIGNS ARE GIVEN WITH THE BENCHMARKS. ΔP IS THE % POWER IMPROVEMENT OVER THE INITIAL DESIGN. A POSITIVE NUMBER INDICATES IMPROVEMENT. T IS THE CPU RUNTIME. IN THE FOUR TRANSFORM CASE, WE ALSO SHOW THE NUMBER OF VOLTAGE ISLAND GENERATED BY OUR METHOD (# VI COLUMN), AND THE NUMBER OF STANDARD NETWORK FLOW PROCESSES IN SOLVING OUR DNF MODELS (I_T COLUMN). THE NUMBER OF VOLTAGE ISLANDS GENERATED BY SEQ-STD IS ALWAYS 16.

Empirical evidence on the number of rectangular voltage islands that we actually generate (see Table X-Table XI) shows that on the average we are only 4% below the upper bound of 16, and in 58% of the cases we generate exactly 16 islands (meaning that in these cases there are no collinear corner pairs). This provides empirical evidence for our assertion about the low probability of the collinear concave-corner case.

4.4 Experimental Results

We used two standard-cell benchmark suites in our experiments: 1) ISCAS'89 benchmark suite, and 2) Faraday benchmark suite from [31]; the macros in the Faraday benchmarks are removed to yield standard-cell benchmarks. We use a $45nm$ standard cell library for the ISCAS'89 benchmarks, and a $180nm$ library for Faraday benchmark. Dual V_{th}'s 0.2V and 0.4V are applied. Up to four V_{dd}'s are used: 0.9V, 1.2V, 1.5V and 1.8V. We use a bin size of ten rows by ten average cell widths for Faraday benchmarks. The upper bound on the voltage island number is 16, which is around the average number of voltage islands generated in [39]. Results were obtained on Pentium IV machines with 1GB of main memory.

We obtain results for our method for three different configurations: simultaneously applying two, three and four power reduction transforms. In each configuration, we also implement a state-of-the-art method for comparison. For two transforms, we implement the algorithm in [19], which applies cell sizing and dual V_{th} transforms. For three transforms, we implement the algorithm in [23], which applies cell sizing, dual V_{dd}'s and dual V_{th} transforms. In [23], no level shifter is considered, and hence a low V_{dd} gate is not allowed to drive a high V_{dd} gate. To make a fair comparison, we also impose this constraint on our method when applying these 3 transforms.

Voltage island constraints are not considered in the two and three-transform configurations, since the competing methods do not address them. Furthermore, the two competing methods in these cases can only tackle dual V_{dd}'s. Thus, in our method we also only use dual V_{dd}'s (0.9 V and 1.8 V) when comparing to them. For four transforms, since there is no prior published work, for comparison purposes, we apply each transform in the best sequential order—the decreasing order of the amount of power reduction afforded by each transform. To the best of our knowledge, this is the approach used in current industry tools that apply multiple transforms for design closure. The transform order we use is: dual V_{dd}, cell sizing, dual V_{th}. There are two sequential approaches we use when applying the transforms sequentially in the aforementioned order:

- For each individual transform, we use one of the most effective prior algorithms: [14] for dual V_{dd}, [10] for cell sizing, [10] for dual V_{th}[1], and DNF for re-placement (implemented with our DNF technique applied to a COSG with only the re-placement transform). This sequential approach is called *Seq-std* (for standard sequential method).
- Use our DNF method for each transform in which the COSG only includes the options provided by that transform. This sequential approach is called *Seq-nf* (for network-flow based sequential) method.

---

[1]Since there is no recent work on V_{th} assignment only, we modified the cell sizing algorithm in [10] to do V_{th} assignment. The V_{th} assignment problem is similar to a cell sizing problem in which the input capacitance of all sizes is the same, and only the cell delay and driving resistance change.

Benchmark	0% extra delay							5% extra delay						
	DNF			Seq-std		Seq-nf		DNF			Seq-std		Seq-nf	
	ΔP %	t (sec)	# VI	ΔP %	t (sec)	ΔP %	t (sec)	ΔP %	t (sec)	# VI	ΔP %	t (sec)	ΔP %	t (sec)
ISCAS'89	30	5.5h	15	22	1.3h	24	1.4h	35	5.1h	15	26	1.2h	27	1.5h
Faraday	31	12.9h	16	18	5.3h	19	3.7h	36	12.2h	16	20	5.2h	22	3.5h
Overall	**30**	**8.6h**	**15**	**20**	**3h**	**22**	**2.4h**	**35**	**8.0h**	**15**	**23**	**3.5h**	**25**	**2.3h**

TABLE XI

THE AVERAGE POWER IMPROVEMENTS WITH FOUR TRANSFORMS AND FOUR V_{DD}'S FOR THE MULTIPLE V_{DD} TRANSFORM. THE COMPARISONS ARE TO AN INITIAL DESIGN WITH SINGLE V_{DD} AND V_{TH} OBTAINED THROUGH "DESIGN COMPILER" FOR ISCAS'89 BENCHMARKS (SEE CAPTION OF Table X FOR DETAILS), AND GIVEN WITH THE BENCHMARKS FOR FARADAY BENCHMARKS. A POSITIVE NUMBER INDICATES IMPROVEMENT. THE NUMBER OF VOLTAGE ISLANDS GENERATED BY THE SEQ-STD IS ALWAYS 16.

We measure the power improvement from the initial WL-optimized placement and sizing solution (the latter is provided with the benchmark where it was generated with a given delay constraint using synthesis tools) along with the highest V_{dd} and the lowest V_{th} for each cell. The delay constraint imposed is either 0% or 5% more than the initial placement's delay; all relevant constraints including delay were satisfied by all methods. We first discuss results for the 0% extra delay constraint and dual V_{dd} case that are reported in Table X. Our method obtains up to 34% and an average of 26% power improvement for the 4-transform case, which compares very favorably to the 18% average power improvement of the beset sequential method *Seq-nf*. DNF's improvements are also better than the improvements of the other competing methods

across the 2, 3 an 4 transform configurations by 7%, 50% and 44%, respectively. We also note that for the two sequential methods in the four transform configuration, using our DNF technique for each transform (*Seq-nf*) obtains a better power improvement than using the prior most effective methods (*Seq-std*). Thus this shows that the basic DNF formulation provides higher quality solutions than state-of-the-art methods even for application to single transforms, and establishes the efficacy of the general DNF formulation, whether for simultaneous multiple transforms or single transforms.

We also obtained results with four V_{dd}'s and four transforms for our method as well as for the best sequential order method (Table XI). With the 0% (5%) extra delay constraint, DNF obtains up to 42% (45%) and an average of 30% (35%) power improvement, and this improvement is 36% (40%) better than that of the best sequential method (*Seq-nf*). These results also show that our simultaneous method using DNF better explores the available optimization space than either sequential method (going from the 0% to 5% extra delay relaxation, the gap between DNF method's power improvement and that of the best sequential method goes from 42% to 45%).

Finally, we compared our method to the popular industry tool Synopsys's IC Compiler (ICC) for ISCAS'89 benchmarks. In this comparison, we ran ICC in full power optimization mode (in this mode, ICC uses all the four transforms that we do, and the congestion consideration in ICC is turned off) with a timing constraint that corresponds to that of the timing obtained by ICC run in full timing optimization mode (achieved by setting a very small, and thus unachievable, timing constraint, and ICC returns the best possible timing result). The output of power-

optimizing ICC is the input to the DNF technique, and the timing constraint for DNF is the timing obtained by ICC. There are, however, some small differences between the configuration and timing model of ICC and DNF that we would like to note:

1. Unlike DNF, ICC cannot automate voltage island formation from scratch, and requires the designer to specify initial voltage islands that it may subsequently modify. To achieve this, we partition each circuit into 16 subcircuits (the same number as the voltage island number constraint for our method) using the min-cut partitioner hMetis, and assign a V_{dd} to each subcircuit using the technique of [14] .

2. In general, industry tools use more complex delay and power models than those used by academic tools including DNF. We do the following modifications in our delay/power calculation which are employed in ICC to alleviate the inconsistencies: a) Using the library's delay/power lookup table. b) Considering the effect of rise and fall times on cell delay. c) Using ICC's "virtual routing" of the initial design to estimate net initial WL. We also perform WL estimation for the cell placement transform using simple incremental routing. However, some other features in the ICC models were not possible for us to implement due to insufficient information. These include: a) considering various intra- and inter-metal layer coupling capacitances (we consider an average coupling capacitance given in the library lookup tables); and b) trying to place level-shifter cells on nets from lower V_{dd} drivers to higher V_{dd} sinks at the "boundary positions" of their voltage islands to facilitate power net routing (we put it close to the sink cell to lower the dynamic power of such nets).

The above inconsistencies, especially the one concerning detailed coupling capacitance information, result in our timing and power analysis numbers being about 5-10% below ICC's for the same designs. In Table XII, we report DNF's power improvement results under both our and ICC's delay/power analysis to show the differences between them; however, the correct improvement numbers to consider are those given by our analysis, as these are consistent with the modeling of these metrics in DNF. Under our analysis, DNF achieves up to 16% and an average of 13% power improvement compared to ICC. Under ICC's analysis, DNF's average improvement over ICC is 12%, and there is a small amount (5%) of delay constraint violation. These phenomena are understandable, since, as mentioned above, ICC's evaluation models for these metrics are a little different from those of DNF.

For the type of complex design that our method performs, and for its high efficacy in both power optimization and multi-constraint satisfaction, DNF's runtimes are reasonable, and only about 3 times that of the much less conceptually complex sequential method `Seq-nf` (and only 2 times that of `Seq-std`). The empirical complexity of DNF w.r.t. the number of cells in a circuit is linear as shown in Figure 23, which underscores the good scalability of our method.

In summary, all results clearly show the significantly greater efficacy of simultaneous application of transforms compared to their sequential application. Also, our DNF-based techniques for simultaneously applying all transforms have achieved this greater efficacy in a time-efficient and scalable manner.

Benchmark		s13207	s15850	s38417	s38584	s35932	**Avg.**
ICC Power (μW)		306	311	498	544	682	
Our rep.	**%ΔP**	16	12	15	10	12	**13**
	%ΔT	0	1	1	1	1	**1**
ICC's rep.	**%ΔP**	14	11	14	9	11	**12**
	%ΔT	-7	-4	-4	-5	-4	**-5**
ICC runtime(h)		0.2	0.25	0.34	0.41	0.5	**0.34**
DNF runtime(h)		3.5	3.0	4.4	7.8	13.4	**6.4**

TABLE XII

DNF'S POWER AND DELAY IMPROVEMENTS OVER SYNOPSYS'S IC COMPILER (ICC) FOR LARGE ISCAS'89 BENCHMARKS USING DUAL V_{DD}'S. BOTH SETS OF RESULTS ARE FOR POST DETAILED-PLACEMENT BUT PRE-ROUTING DESIGNS. NEGATIVE VALUES MEAN DETERIORATION. "OUR REP." ("ICC'S REP.") IS DNF'S IMPROVEMENT CALCULATED USING OUR (ICC'S) DELAY/POWER MODELS. THE RUN TIMES OF BOTH METHODS ARE ALSO GIVEN, THOUGH THEY ARE PROBABLY NOT COMPARABLE FOR THE FOLLOWING REASON: ICC IS EVEN FASTER THAN THE SEQUENTIAL METHODS CODED BY US BY A FACTOR OF ABOUT 5, WHICH INDICATES THAT VARIOUS EFFICIENT DATA STRUCTURES AND COMPILER OPTIMIZATIONS HAVE PROBABLY BEEN USED FOR ICC THAT HAVE NOT BEEN USED IN OUR CODES, AND WHICH CAN POTENTIALLY REDUCE THE RUNTIME OF DNF SIGNIFICANTLY.

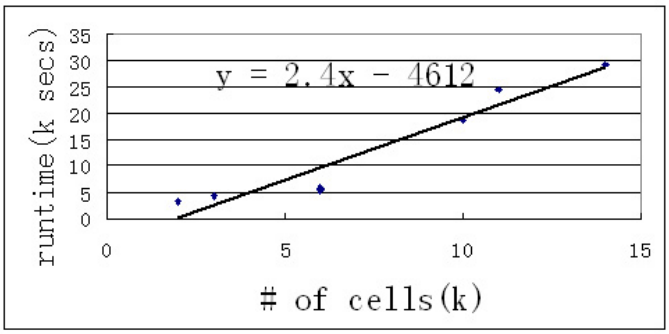

Figure 23. Runtime plot of the DNF method versus the number of cells.

4.5 Solving the Yield-Driven Physical Synthesis Problems Using DNF

For yield optimization, we only consider here the cell sizing transform. Given a placed circuit and the variation of cells in a library, we want to size the cells in the design, so that the percentage of produced chips that meet the delay requirement is maximized. Cell sizing is very efficient in improve the yield. Generally speaking, the larger the size of a cell is, the smaller its fractional or percentage variation will be. Of course, larger cells can sometimes increase the delay of critical paths due to their increased input load (in other instances they can decrease the delay due to their increased drive strength), and also increase both leakage and dynamic power. Hence, as far as timing yield is concerned it is an optimal balancing act that is needed to determine the "right" cell sizes that maximizes timing yield or meets a lower bound yield requirement.

The Gaussian distribution of circuit delay can be derived by using SSTA in which Equation 1.11 is applied to each pin in a topological order to derive each pin's arrival delay PDF. However, obviously the μ and δ of the distribution will be a very complex function of μ and δ of the variation of each cell in the circuit. We proposed two methods to simplify the objective function, the *full circuit delay PDF method* and *the critical path set delay PDF method*.

4.5.1 The Full Circuit Delay PDF Method

To simplify objective function Yd (Equation 1.12) so that it can be handled efficiently by DNF, we can take an order-k Taylor's series expression of the Yd, where $k \leq 10$. Note that for order-k Taylor's series expression, the highest order of product terms is k. An example of order 2 Taylor's series is given in Equation 4.4. In our experiments, we have tested Taylor's

series expansion of Yd for k=1,2,3. Our results show that we obtain very diminishing results for $k > 2$—the yield improved by 1% but runtime increases by a factor of 40, when going from k=2 to 3. Hence, in our implementation, we choose the second order Taylor's series as the simplified objective function.

The second order Taylor series for a function f that depends on a set of variables X is given as:

$$T(f(X), X_s) = f(X_s) + \sum_{x \in X} \frac{df(X_s)}{dx}(x - x_s) + 0.5 \sum_{x,y \in X} \frac{df(X_s)}{dxdy}(x - x_s)(y - y_s) \tag{4.4}$$

where X_s is an initial (starting) solution, and x_s (y_s) is the value for variable x (y) in the initial solution. The key issue of obtaining the second order Taylor series for a complex function is, of course, determining the various derivatives at the initial solution point. We use recursively the chain rule given below to calculate the derivatives

$$\frac{df}{dx} = \frac{df}{da} \cdot \frac{da}{dx}$$

where a is an intermediate variable. The arrival time of each gate is used as the intermediate variable in our case. Let $A_i(g)$ ($A_o(g)$) denote the max arrival time (arrival time) at the input (output) of each gate g. We define the *derivative vector* of a random variable u w.r.t. another

random variable v (denoted by $\frac{du}{dv}$) as the set of four derivatives $\{\frac{d(\mu(u))}{d(\mu(v))}, \frac{d(\mu(u))}{d(\sigma(v))}, \frac{d(\sigma(u))}{d(\mu(v))}, \frac{d(\sigma(u))}{d(\sigma(v))}\}$. Correspondingly, the chain rule for derivative vectors can be written as:

$$\frac{du}{dv} = \frac{du}{dw} \cdot \frac{dw}{dv} = \{\frac{d(\mu(u))}{d(\mu(w))} \cdot \frac{d(\mu(w))}{d(\mu(v))} + \frac{d(\mu(u))}{d(\sigma(w))} \cdot \frac{d(\sigma(w))}{d(\mu(v))}, \frac{d(\mu(u))}{d(\mu(w))} \cdot \frac{d(\mu(w))}{d(\sigma(v))} + \frac{d(\mu(u))}{d(\sigma(w))} \cdot \frac{d(\sigma(w))}{d(\sigma(v))}, \frac{d(\sigma(u))}{d(\mu(w))} \cdot \frac{d(\mu(w))}{d(\mu(v))} + \frac{d(\sigma(u))}{d(\sigma(w))} \cdot \frac{d(\sigma(w))}{d(\mu(v))}, \frac{d(\sigma(u))}{d(\mu(w))} \cdot \frac{d(\mu(w))}{d(\sigma(v))} + \frac{d(\sigma(u))}{d(\sigma(w))} \cdot \frac{d(\sigma(w))}{d(\sigma(v))}\} \tag{4.5}$$

The derivative vectors that need to be determined are $\frac{dD}{d(p(g))}$ for circuit delay D w.r.t. the parameters $p(g)$ (e.g., intrinsic delay, driving resistance and input capacitance) for each gate g. As for the intermediate variables, we also calculate $\frac{dD}{d(A_i(g))}$. We note that for each gate, we can directly calculate three "local" derivative vectors $\frac{d(A_i(g_{fo}))}{d(A_o(g))}$, $\frac{d(A_o(g))}{d(A_i(g))}$ and $\frac{d(A_o(g))}{d(p(g))}$, where g_{fo} is a fanout gate of g. Since $A_i(g_{fo})$ is the maximum $A_o(g)$ of its fanin gates g, $\frac{d(A_i(g_{fo}))}{d(A_o(g))}$ can be determined according to the max operation approximation (Equation 1.11). The rest two derivative vectors can be calculated according to the relationship that $A_o(g) = A_i(g) + d_v(g)$, where $d_v(g)$ is given in (Equation 1.9). The derivative determination for each gate is done in the topological order. The recursive calculation steps are as follows:

1. At the boundary case, i.e., the primary output gate g, since there are no fanout gates, we calculate $\frac{dD}{d(A_o(g))}$ instead of $\frac{d(A_i(g_{fo}))}{d(A_o(g))}$ for local derivatives. Then,

$$\frac{dD}{d(p(g))} = \frac{dD}{d(A_o(g))} \cdot \frac{d(A_o(g))}{d(p(g))} \tag{4.6}$$

.

$$\frac{dD}{A_i(g)} = \frac{dD}{d(A_o(g))} \cdot \frac{d(A_o(g))}{d(A_i(g))} \tag{4.7}$$

.

2. For gates g that are not output gates. Assume we have obtained $\frac{dD}{d(A_i(g_{fo}))}$ for all its fanout gates g_{fo} and local derivative vectors for g. Then,

$$\frac{dD}{d(p(g))} = \sum_{g_{fo}} \frac{dD}{d(A_i(g_{fo}))} \cdot \frac{d(A_i(g_{fo}))}{d(A_o(g))} \cdot \frac{d(A_o(g))}{d(p(g))} \tag{4.8}$$

.

$$\frac{dD}{d(p(g))} = \sum_{g_{fo}} \frac{dD}{d(A_i(g_{fo}))} \cdot \frac{d(A_i(g_{fo}))}{d(A_o(g))} \cdot \frac{d(A_o(g))}{d(A_i(g))} \tag{4.9}$$

.

Note that by determining a derivative, we mean calculating its numeric value—there is no need to record the detailed expression for each derivative. The complexity of the recursive method for determining the n partial derivatives of a function is $O(n)$. After obtaining the

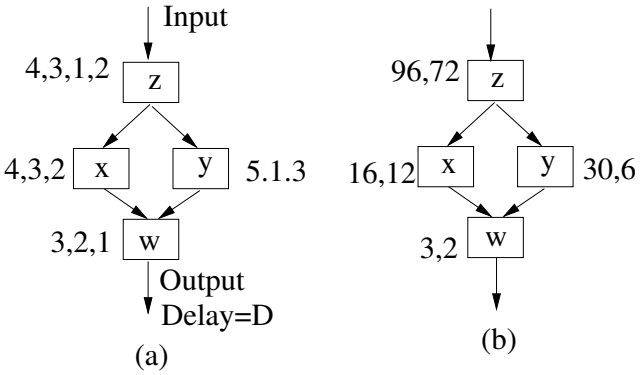

Figure 24. (a) The local derivatives of each gate g. They are shown in the order: $\frac{d(A_o(g))}{d(p(g))}$, $\frac{d(A_o(g))}{d(A_i(g))}$, $\frac{d(A_i(g_{fo}))}{d(A_o(g))}$. Note that the last element of this derivative vector is repeated for every fanout gate g_{fo} of g; thus z with 2 fanout gates has a 4-element derivative vector, while each of the other gates (each with 1 fanout gate) has a 3-element vector. (b) The determined derivatives for each gate g. They are shown in the order: $\frac{dD}{d(p(g))}$, $\frac{dD}{d(A_i(g))}$.

second order Taylor's series for Yd, we solve this simplified objective function using DNF which we describe in the Appendix.

An example for derivative determination is given in Figure 24. A simple circuit of four gates w, x, y and z is shown. For clarity of exposition, we only consider one element in the derivative vector instead of all four. Figure 24(a) lists the local derivative vectors. For each gate, the first value is $\frac{d(A_o(g))}{d(p(g))}$, the second value is $\frac{d(A_o(g))}{d(A_i(g))}$, and the rest are $\frac{d(A_i(g_{fo}))}{d(A_o(g))}$ ($\frac{dD}{d(A_o(g))}$ for the output gates), for each fanout gate g_{fo}. Figure 24(b) gives the derivative vectors $\frac{dD}{d(p(g))}$ and $\frac{dD}{d(A_i(g))}$ for each gate. For the output gate z, according to Equation 4.6 and Equation 4.7, $\frac{dD}{d(p(z))} = 1 \times 3 = 3$ and $\frac{dD}{d(A_i(g))} = 1 \times 2 = 2$. For gate x (gate y is in a similar position as x), according to Equation 4.8 and Equation 4.9, $\frac{dD}{d(p(z))} = 2 \times 2 \times 4 = 16$

and $\frac{dD}{d(A_i(g))} = 2 \times 2 \times 3 = 12$. Finally for the input gate z, according to Equation 4.8 and Equation 4.9, $\frac{dC}{d(p(z))} = 12 \times 1 \times 4 + 6 \times 2 \times 4 = 96$ and $\frac{dC}{d(A_i(g))} = 12 \times 1 \times 3 + 6 \times 2 \times 3 = 72$.

One property of $T(f(X), X_s)$ is that it only has a small error (w.r.t $f(X)$) at solution points close to X_s. Thus if $T(f(X), X_s)$ is optimized and the optimal solution X_o for $T(f(X), X_s)$ is far from X_s, then $T(f(X), X_s)$ may deviate from $f(X)$ by a large amount at X_o. Thus X_o may not be a near-optimal solution for $f(X)$. We resolve the potential optimal solution inaccuracy problem by iteratively adjusting the initial solution point. In each iteration, a new Taylor series is generated for the original objective function at the optimal solution point obtained in the last iteration. The pseudo code for the iterative algorithm *Iter_Taylor* is given in Figure 25.

The rationale behind the termination condition of *Iter_Taylor* is as follows. The idea of optimizing a function f through iteratively optimizing f's Taylor expansion has been used in continuous optimization (e.g., Newton's method), and it has been shown both theoretically and empirically that the iterative process converges well to either a global or local minima. Hence, a good termination condition in continuous optimization can simply be: error $e < \epsilon$, where ϵ is a given error upper bound. However, in discrete optimization such as the cell sizing problem we tackle here, the convergence to a particular minima is not established. Intuitively, this is mainly because generally a non-infinitesimal distance exists between any two discrete solutions. Hence, Taylor expansions at different solution points (even adjacent ones) cannot be similar enough to have the same local minima. In our experiments we have observed that when the solution is close to a local minima, in most cases the iterative process will jump around several close discrete solutions, with one of them being the local minima. Such a "thrashing" pattern can be

Algorithm `Iter_Taylor`

1. Obtain the static timing optimal gate sizing solution X_t ignoring random variations. Use X_t as the starting solution X_s.
2. Let X_g be the best solution, and e_{min} be the minimum error between $\mathrm{Yd}(X)$ and its Taylor expansion. Initially, X_g=X_t, and $e_{min} = \infty$.
3. Repeat
4. Derive the Taylor series $T(\mathrm{Yd}(X), X_s)$ for $\mathrm{Yd}(X)$ at X_s.
5. Optimize $T(\mathrm{Yd}(X), X_s)$, and get the optimal solution X_o.
6. if $\mathrm{Yd}(X_o) > \mathrm{Yd}(X_g)$ then $X_g = X_o$.
7. Compute the % error e between $T(\mathrm{Yd}(X_o), X_s)$ and $\mathrm{Yd}(X_o)$.
8. if $e < e_{min}$ then $e_{min} = e$.
9. $X_s = X_o$.
10. Until e_{min} does not decrease in r successive iterations.
11. return(X_g).

End Algorithm

Figure 25. Yield optimization using iterative Taylor's series approximation.

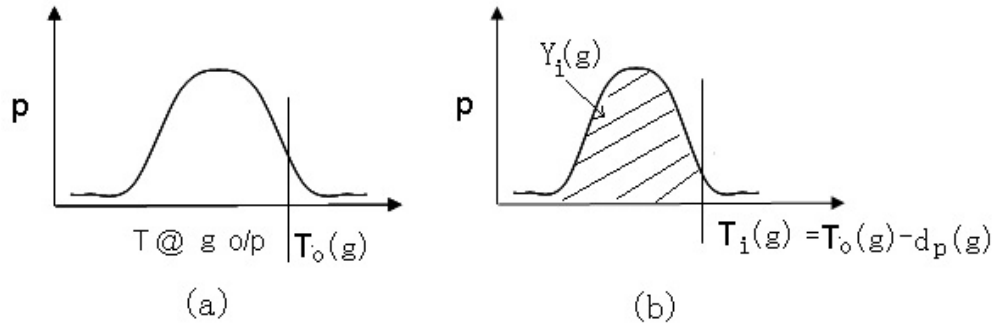

Figure 26. The process for propagating $Y_o(g)$ to $Y_i(g)$. (a) The delay PDF at the output of g. (b) The propagated $T_i(g)$ and $Y_i(g)$ at the input of g.

detected (with high probability, as our experiments suggest) by examining e_{min} in *Iter_Taylor*. Based on our experiments, we choose $r = 3$ so that in most cases no more improvement is possible by continuing iterations beyond this termination condition.

4.5.2 The Critical-Path Set Delay PDF Method

In the first method (the full circuit method—FCM), the objective function Yd is a function of the μ and σ of cell parameters for all cells in the circuit. Taking the parameters of all cells into consideration, especially for those that has little impact on the final yield not only increases the problem complexity a lot (note that the number of second order terms in the Taylor series is proportional to $O(n^2)$, where n is the number of cells considered), but also distracts the optimization focus (e.g., assign unnecessarily large cell area to cells with little impact, and not leave enough area for critical cells). This is especially true in a practical scenario where there is a cell area constraint on the yield optimization problem.

Furthermore, in order to obtain the closed form objective function, the operation for determining the max arrival time's PDF from multiple paths involves an approximation. The error due to this approximation grows with the depth of the circuit, and thus for circuits of even medium complexity, the obtained closed form expressions for the μ and σ of the circuit delay can have an appreciable error from these parameters in the actual delay PDF of the circuit.

To remedy the above drawbacks, we propose to use a path based approach in which we determine yield-critical paths, and focus on optimizing the yield of these paths only. Similar approaches have been used to optimize circuit delays in the non-statistical domain [4]. They usually involves three steps: (a) using STA to enumerate critical and near-critical paths (e.g., those paths with delays within, say, 15%, of the critical-path delay); (b) improve the delay of these critical and near-critical paths by either minimizing a weighted summation of their delays or satisfying a constraint that the delay of all these paths should be smaller than a given upper bound (the latter problem is a delay constraint one, as opposed to the delay optimization one that the former addresses); (c) repeat steps (a) and (b) to ensure convergence (e.g., no new critical or near critical path appear).

Using such a path based approach in the statistical domain can avoid the two drawbacks of the FCM method. Since the optimization is only performed for cells in critical and near critical paths, the distraction from cells that have little impact on final yield is eliminated. Further, there is no explicit max operation needed. Hence, the approximation error for max operation can be avoided. However, in the statistical delay environment, the concept of critical and near critical paths is not that obvious. For example, should the μ of the cell delay distribution be used

to determine critical paths or should the worst-case delay of $\mu+3\sigma$ be used? Neither is actually a good choice. The former is essentially useless since it does not capture the delay variability, and the latter will only identify statistically worst-case critical paths which overconstrains the problem (the probability of any path having a worst-case delays based on $\mu + 3\sigma$ is extremely small). In fact, the criticality measurement should be related to a required yield value. For example, if the required yield is 100%, then the worst case delay of $\mu + 3\sigma$ is a good choice; on the other hand, if the required yield is 50%, then the average delay μ is appropriate. Given an yield requirement Y, we propose the following general way to obtaining *yield-characterized (YC)* critical and near critical paths.

The YC critical path selection technique starts from an initial static timing optimized solution, which can be obtained using any state-of-the-art method [38]. The PDF of arrival time at the input and output of each cell is then determined using SSTA.

For a circuit with m output gates $g_1, \ldots, g_m$, the yield of the circuit Y_c can be written as[1] : $Y_c = \Pi_{k=1}^{m} Y_k$, where Y_k is the yield at the output of g_k (i.e., for g_k and its input cone). Hence, optimizing Y_c can be done by optimizing each Y_k. However, the influences on Y_c of different Y_k's are different. Let Y_c' (Y_k') denote the value of Y_c (Y_k) for the initial solution. If Y_k is improved by Δy from the initial solution, then the improvement on Y_c will be $\Delta y \frac{dY_c}{dY_k}|_{Y_c=Y_c'} = \Delta y \frac{Y_c'}{Y_k'}$. Thus, the smaller Y_k' is, the larger will be the influence of a change in Y_k on Y_c. Of course, this

---

[1]Note that we assume that the statistical delays of paths are independent. This assumption is, of course, not accurate for re-converging paths. However, as shown in [7], such inaccuracy only causes a 1% difference from Monte Carlo simulation results. On the other hand, the assumption greatly simplifies the problem.

derivative based analysis is accurate only for small changes. However, generally speaking, when the circuit yield constraint Y is given, performing optimization on an output gate g_k (as well as its input cone) whose Y'_k is less than the average yield requirement $Y^{1/m}$ at g_k's output, is more effective than performing optimization on g_k's with $Y'_k \geq Y^{1/m}$. Furthermore, we define the *yield characterized (YC) delay* $T_y(g_k)$ for g_k to be the delay on the delay PDF at the output of g_k for yield $Y^{1/m}$, and let D be the delay on the circuit delay PDF for the yield requirement Y. Then the metric $T_y(g_k) - D$ indicates how much we can improve the timing of g_k and its input cone until the improvement becomes ineffective for improving circuit yield. Therefore, since D is almost a constant w.r.t. the $T_y(g_k)$'s, $T_y(g_k)$ is a good measure of the YC criticality of g_k. Large YC criticality ($T_y(g_k)$) indicates that there is more yield optimization potential for g_k and its input cone.

The above YC criticality determination for output gates can be generalized for determining the YC criticality of fanins of a multiple fanin gate v. Let $Y_i(v)$ denote the yield requirement at the input of v, and (u, v) be a fanin to v from gate u. $(Y_i(v))^{1/k}$ is the yield limit at the output of u till which the optimization on u and its input cone is effective (called the *effective yield limit* for short), where k is the number of fanin gates of v. The YC delay $T_y(u, v)$ for (u, v) is defined as the delay on the delay PDF at the output of u for yield $(Y_i(v))^{1/k}$. Then, the YC criticality of (u, v) (i.e., the effectiveness of optimizing its input cone) is measured by $T_y(u, v)$, and the fanin with largest YC delay is the YC critical fanin to v. An example for determining the YC critical fanin is given in Figure 27. The yield requirement $Y_i(v)$ can be propagated from the circuit yield requirement Y according to the following rules.

1. Propagate the yield requirement $Y_o(g)$ at the output of g to $Y_i(g)$ as follows:

   - Determine the delay $T_o(g)$ for the yield point $Y_o(g)$ on the delay PDF at g's output. Let $d_p(g)$ be the delay that satisfies $P(d_v(g) \leq d_p(g)) = Y_o^{1/2}(g)$, where $d_v(g)$ is the delay of g including random variations as given in Equation 1.9. Then propagate the delay point $T_o(g)$ at the output to the delay point $T_i(g)$ at the input as $T_o(g) - d_p(g)$. The rationale behind this propagation will be explained shortly. An example of this propagation is given in Figure 26.
   - $Y_i(g)$ is the area on the delay PDF at the input of g with delay $\leq T_i(g)$.

2. Propagate $Y_i(g)$ to $Y_o(g_{fi})$ as follows, where g_{fi} is the fanin gate of g: If there are q fanin gates, then

$$Y_o(g_{fi}) = (Y_i(g))^{1/q} \tag{4.10}$$

   Thus, the propagated yield requirement at the output of g_{fi} is its effective yield limit.

3. If a gate g has multiple fanout gates, multiple $Y_o(g)$'s can be propagated from different fanouts according to the above rule. Then, the maximum one among them is chosen as $Y_o(g)$.

Note that if we add a zero-delay dummy cell g_d to all circuit outputs, its $Y_i(g_d)$ is the given yield requirement Y for the circuit. Starting from $Y_i(g_d)$, we can get Y_i for each gate using the

above rules. The rationale for choosing $d_p(g)$ as the delay propagation difference from $T_o(g)$ to $T_i(g)$ is to ensure that the yields corresponding to these two delay points on the delay PDF of the max-input and output of g is similar. Note that $Y_o(g) = Y_i(g) * P(d_v(g) \leq d_p(g))$. Then, since $P(d_v(g) \leq d_p(g)) = (Y_o(g))^{1/2}$, $Y_i(g) = (Y_o(g))^{1/2}$. Hence, the yield related requirements $Y_i(g)$ and $P(d_v(g) \leq d_p(g))$ are well-balanced.

The YC critical path from the input to a cell w can be determined by iteratively identifying the YC critical fanin in the input cone to w. Starting from w, we first identify the YC critical fanin (v, w) to w; then the next YC critical fanin (u, v) is identified to v, and so on. This process continues until the primary inputs of the circuit are reached, and all identified interconnects consists the YC critical path to w. The YC critical path of the circuit is then the YC critical path to g_d.

To determine the YC near-critical paths, we first define the *one-hop change* on a path P. The one-hop change on P will create a new path P' by switching a YC critical subpath in P to a gate w to a YC non-critical subpath to w as follows. First change the interconnect (v, w) in P to another interconnect (u, w) to w, and then replace the path to v in P by the YC critical path to u. Hence, P and P' will have a common path after w, and branch at w. The distance of the one-hop change is defined as $T_y(v, w) - T_y(u, w)$.

The distance of a path Q from the YC critical path is defined as the sum of distances of all one-hop changes needed to change the YC critical path to Q. We use a path's *yield characterized (YC) delay* to measure path criticalities. Given a yield constraint, we define the YC delay (denoted by $T_y(P)$) of the YC critical path P as the delay value on its delay PDF that

corresponds to the yield constraint. The YC delay $T_y(Q)$ of a near critical path Q is defined as $T_y(P)-$ (the distance of Q from P). Large YC delay indicates high criticality.

To identify YC critical and near critical paths, we can use a method similar to the Dijkstra's algorithm. The detailed steps of the method is as follows:

1. Identify the YC critical path.
2. Construct a candidate path set, which includes all paths that can be derived through a single one-hop change (called *one hop away*) from the YC critical path. Then, the next two steps are repeated until the next YC near-critical path's YC delay is less than the threshold set for near-critical paths (85% of the YC delay of the YC critical path).
3. Select the path Q from the candidate path set with the largest YC delay. Q is the next YC near-critical path.
4. Remove Q from the candidate path set. Then, add all paths that are one hop away from Q to the candidate path set.

The result below establishes the theoretical correctness of our near-critical path set determination method.

Theorem 6 *The $(i+1)$'th $(i \geq 1)$ YC near-critical path is one hop away from one of the first i YC near-critical paths, where the YC critical path is the 1'st YC near-critical path.*

Proof Outline: If an fanin (u, v) is not the *YC critical fanin* to v, we call it a *YC non-critical* fanin. In the $(i+1)$'th YC near-critical path Q, since it is not the YC critical path, we can

always identify a set of YC non-critical fanins on it. Let (u, v) be one of the YC non-critical fanin on Q that is closest to the starting cell s of Q. This means the subpath from s to u on Q is on the YC critical path through u (otherwise, at least one fanin/interconnect on this subpath is YC non-critical and is closer to s than (u, v), and we reach a contradiction to our assumption that (u, v) is the YC non-critical interconnect on Q that is closest to s). Assume (w, v) is the YC critical fanin to v. Performing a one-hop change on Q by switching from (u, v) to (w, v) generates a new path Q'. Obviously, Q' has larger YC delay than Q, and thus must be one of the first i YC near-critical paths. Further, by the definition of the aforementioned one-hop change on Q, performing a reverse one hop change on Q' by switching from (w, v) to (u, v) will generate Q. Therefore, Q is one-hop away from Q', and the theorem is true. $\diamondsuit$.

In our method for determining the set of YC near-critical paths, after determining the i'th YC near-critical path, the candidate path set includes all paths that are one-hop away from one of the first i YC near-critical paths. Therefore, the path with the largest YC delay in the candidate path set will be the $(i + 1)$'th YC near-critical path for the circuit according to Theorem 6.

To optimize the delay under a given yield constraint, we use an objective function that is the weighted summation of critical and near critical path delays at the yield requirement point ($Y^{1/m}$ where m is the number of output gates). The weight is inversely proportional to the yield-characterized delay slack of each path, which is defined as $1.1T_y(P) - T_y(Q)$. The coefficient of 1.1 is to ensure that all slacks are positive, and provides a relative measure of criticality of the set of critical and near critical paths. The aforementioned objective function

is also decomposed into quadratic terms using order-2 Taylor's expansion (just as detailed for the FCM method), and then solved using DNF.

By using the approach for the timing optimization under yield constraint problem as a subroutine, we can also solve the dual problem of optimizing yield under a given timing constraint D_c. The subroutine can be used as a tester that determines, given a yield Y, if the near-optimal delay that we have obtained for the yield point $\leq D_c$. If it is not, the we need to test another yield $Y' < Y$, otherwise we can test another yield $Y' > Y$. The choosing of the next yield goal Y' can be determined in a binary search manner, and the entire process terminates if the new Y' differs from Y by less than a predetermined amount δ. The pseudo code is given in Figure 28.

4.6 Experimental Results

The proposed statistical optimization methods were implemented and tested on ISCAS'85 and Faraday benchmarks. A 180nm cell library is used, and for each cell type, there are six different sized implementations in the library. The standard deviation (σ) of cell parameters is 21, 17, 13, 9, 5, 1% of the nominal value from the minimum sized implementation to the maximum sized one [1]. Choosing such standard deviation values is consistent with the observation that large cells should have smaller standard deviation [7]. The deterministic cell sizing is done using a network flow based method from [38], and it optimizes the worst case delay at $\mu + 3\sigma$. The yield evaluation is done using SSTA with the max approximation as given in [7]. The av-

[1] As predicted in [41], the standard deviation of variations can be over 14% of the nominal value for technique below 90nm.

erage delay obtained by SSTA for the deterministic solution is used as the timing constraint for calculating yield. Hence, the yield for the deterministic solution is always 50%. For statistical cell sizing, we use the same area constraint as the deterministic sizing (which means 0% area increase). All programs are run on a machine with a 3GHz CPU and 1GB RAM.

Table XIII shows yield improvement compared to the deterministic method for three statistical methods, sensitivity based (Sen), non-path based (FCM), and path based (PBM). For the sensitivity based method, we implemented one of the state-of-the-art algorithm proposed in [12], which provides a sensitivity based technique for solving the problem of timing optimization under a yield constraint. This algorithm is then used in our binary-search based algorithm `Yield_Opt(`D_c`)` to obtain a yield optimized design under a given timing constraint. For our PBM method, the paths that are within 85% of the YC delay of the YC critical path are determined as the set of near-critical paths.

The PBM method achieves a maximum of 43% and an average of 37% yield improvement. On average the yield improvement obtained by PBM is 6% relatively more than obtained by FCM, and 19% relatively more than Sen's. Further, PBM generates larger yield improvements compared to the other two methods for most benchmark circuits, which illustrates the consistently high quality of PBM. The table also shows that the leakage power change when optimizing yield is minimal (1-2% on average) and comparable across the three methods.

Since PBM uses global optimization, it is expected that its runtime will be larger than the sensitivity based local search method Sen. According to Table XIII, the run time of PBM is about 6 times of that of Sen. However, our method still displays a linear complexity w.r.t.

Ckt	**# cells**	**# nets**	**Sen [12]**			**FCM**			**PBM**		
			% Δ Y	**Runtime (sec)**	**% ΔP_L**	**% Δ Y**	**Runtime (sec)**	**% ΔP_L**	**% Δ Y**	**Runtime (sec)**	**% ΔP_L**
C432	160	196	18	8	4	18	44	2	20	66	6
C499	202	243	34	15	4	34	79	5	39	105	4
C880	383	443	23	29	7	26	142	7	27	184	1
C1355	544	587	35	27	-1	43	184	1	43	213	6
C1908	880	913	34	49	-5	42	510	-5	41	525	1
C2670	1.3K	1.5K	37	69	3	38	532	-4	39	380	-4
C3540	1.7K	1.7K	31	112	5	36	568	7	39	716	2
C5315	2.3K	2.5K	21	130	-2	25	811	1	24	1146	-6
C6288	2.4K	2.4K	36	110	-3	41	1005	-2	43	1295	0
C7552	3.5K	3.7K	36	130	4	40	948	5	41	1241	5
Avg.			**30**	**68**	**1**	**34**	**482**	**2**	**36**	**587**	**2**
DMA	12K	12K	33	362	5	37	1953	1	42	2411	6
DSP1	26K	28K	35	701	1	39	3438	3	39	4021	5
DSP2	26K	28K	34	710	-4	39	3216	6	38	4098	1
RISC1	33K	33K	29	751	5	34	4514	-1	35	5116	-1
RISC2	33K	33K	29	743	3	32	4503	5	34	5168	4
Avg.			**32**	**653**	**2**	**36**	**3524**	**3**	**38**	**4162**	**3**
Overall Avg.			**31**	**263**	**1**	**35**	**1496**	**2**	**37**	**1778**	**2**
Norm. Avg.			**1**	**1**	**1**	**1.13**	**6**	**2**	**1.19**	**7**	**2**

TABLE XIII

YIELD IMPROVEMENT RESULTS FOR THREE STATISTICAL METHODS OVER THE DETERMINISTIC DESIGN (WHOSE YIELD IS 50%). THE DETERMINISTIC DESIGN IS OBTAINED BY OPTIMIZING THE WORST CASE DELAY. THE $\%\Delta Y$ COLUMNS GIVE THE PERCENTAGE YIELD IMPROVEMENTS. A POSITIVE NUMBER INDICATES IMPROVEMENT. THE TOTAL CELL AREA INCREASE CONSTRAINT FOR ALL METHODS IS 0%. THE LEAKAGE POWER CHANGE FROM THE DETERMINISTIC DESIGN IS LISTED IN THE $\%\Delta P_L$ COLUMNS.

the number of cells in the circuit as shown in the plot in Figure 29. This enables PBM to be used on large scale problems. Finally, Table XIII shows that PBM's run time is a little larger than FCM's (by about 19%). This is mainly due to that fact that to solve the problem of yield optimization under a timing constraint (the "yield optimization problem"), PBM needs to iteratively solve the dual problem of timing optimization under a given yield constraint (the "timing optimization problem"); see Figure 28. Given the fact that the number of iterations is usually 6-8, the run time for each iteration, which is the run time for PBM to solve a single timing optimization problem, is much less than the run time for FCM to solve an yield optimization problem (though the complexities of the two problem are similar).

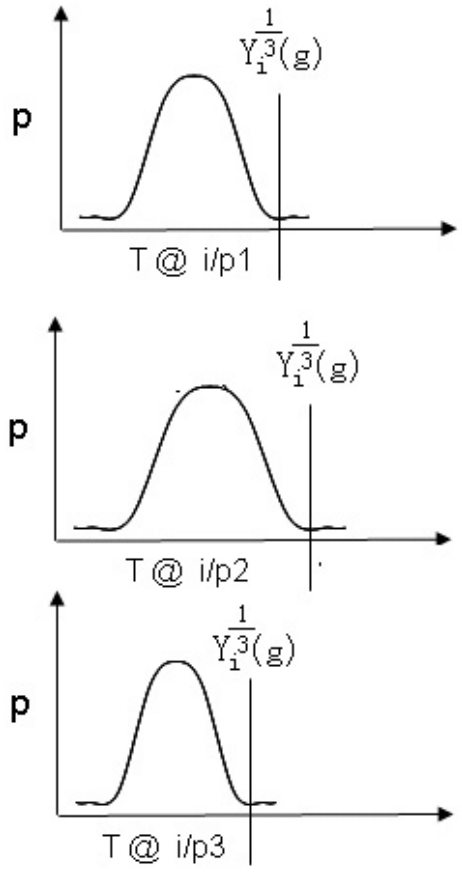

Figure 27. The criticality comparison of three fanins to gate g. The delay PDFs at the three inputs i/p1, i/p2 and i/p3 of g are shown. The middle input i/p2 has the largest delay at yield point $(Y_i(g))^{1/3}$, and hence is critical.

Algorithm `Yield_Opt`(D_c) /* D_c is the delay constraint */

1. We start with an initial yield constraint of Y_s that can be arbitrarily chosen, e.g., 50%.
2. Initially, the upper bound for the yield constraint is 100%, and the lower bound is 0%
3. Repeat
4. Obtain the best possible delay D_b and the corresponding solution
   X_b at Y_s by solving the timing optimizing under yield constraint
   problem.
5. If $D_b \leq D_c$ then update the lower bound to Y_s.
6. If $D_b > D_c$ then update the upper bound to Y_s.
7. The next Y_s is determined as (upper+lower bound)/2.
8. Until upper bound − lower bound $< \delta$
9. Return(X_b).

End Algorithm

Figure 28. Yield optimization using an iterative Taylor's series approximation and optimization approach in a binary-search framework.

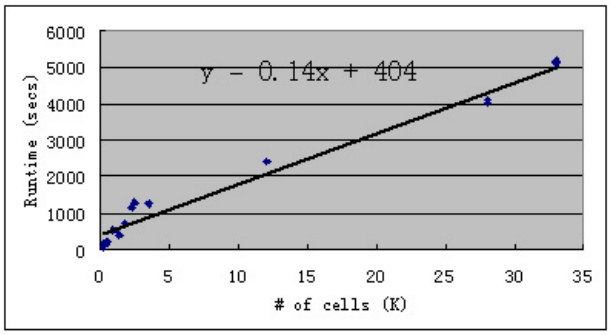

Figure 29. Runtime plot of the PBM method versus the # of cells.

CHAPTER 5

DNF'S APPLICATION TO 0/1 ILP AND INLP PROBLEMS

In this chapter, we discuss the modeling of 0/1 integer programming problems as DNF problems, and present experimental results for DNF solutions to these problems.

5.1 DNF Modeling

Due to its generality, our DNF method can also be applied to general discrete integer linear/non-linear programming (ILP/INLP) problems. In particular, it is very suitable for problems where each variable takes a binary value. For such cases, we can again view the problem as an option selection problem, with each variable having two options value 0 and value 1. The ot- and ct-subgraphs can be built for each p-term in the objective and constraint functions. For a p-term that depends on k binary variables, the corresponding subgraph will have 2^k hyperarcs/arcs.

When applied to general 0/1 ILP/INLP problems, there are two issues that we need to pay attention to. First, for the ILP problem, our optimization and constraining graphs can be greatly simplified. Figure 30 shows an example of the DNF graph for a simple ILP problem given below.

$$\text{Min.} \quad -x_1 - 3x_2 - x_3$$

$$s.t. \quad 2x_1 + 2x_2 - 4x_3 \leq 1 \tag{5.1}$$

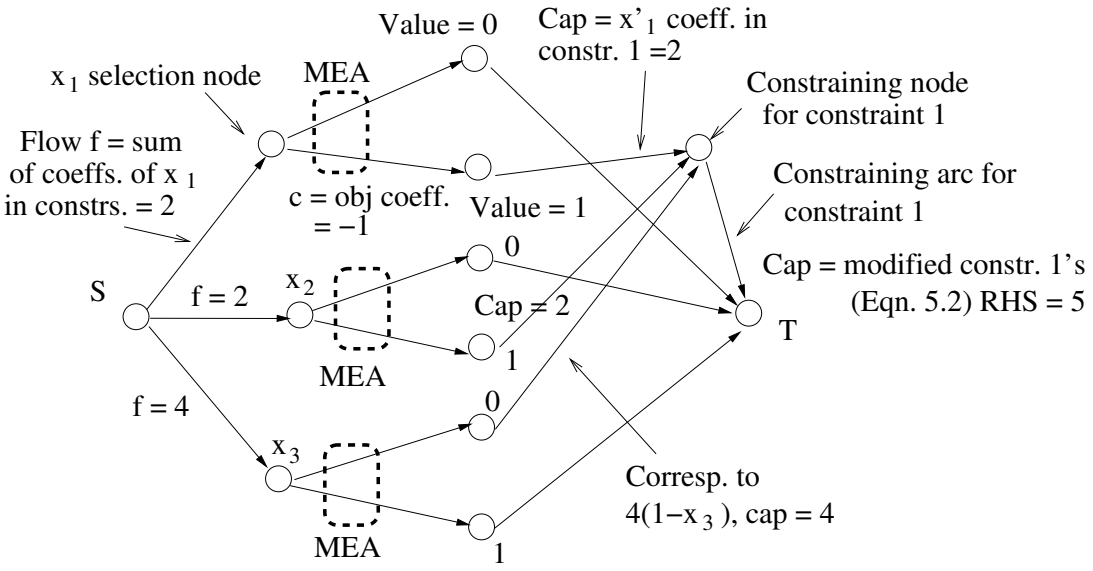

Figure 30. A DNF structure for the ILP problem given in Equation 5.1.

In the DNF graph, for each variable there is a selection node. Each selection node has two outgoing MEA arcs to the two value nodes for the corresponding variable. Flow through one of the value nodes indicating the corresponding value is chosen. The cost on the MEA arc to value 1 is equal to the coefficient of the corresponding variable in the objective function. For each constraint, there is a constraining node that gathers flow from value 1 node of each variable, and the flow amount is equal to the variable's coefficient in the constraint.

The second issue we need to note is that when a variable has a negative coefficient in a constraint, then we actually need to gather a negative flow amount from the variable's value 1 node to the constraining node. Negative flow amount is not allowed in the network flow graph. To handle this problem, we will add a constant to the terms with negative coefficients

to form a new combined term. For example, in the constraint in Equation 5.1, we will change the constraint to

$$2x_1 + 2x_2 + 4(1 - x_3) \leq 5 \tag{5.2}$$

In this way, value 0 node for x_3 will be connected to the constraining node, and sending a flow of amount 4 to the constraining node when it is selected. This indicates the combined p-term $4(1 - x_3)$'s value when $x_3 = 0$.

5.2 Experimental Results

We have tested our DNF method on various benchmarks. The integer non-linear programming (INLP) problems are obtained from: AMPL MIQP [42] and MINLP [43]. The integer linear programming problems (ILP) come from: MIPLIB [44] and Pseudo-Boolean Benchmarks [45]. The cubic benchmarks (those starting with "c-" in their names) are constructed artificially by us by multiplying each quadratic p-term in the objective function of a quadratic benchmark with another 0-1 variable randomly chosen from that benchmark.

Table XIV gives results of our DNF method and compares them to results obtained by state-of-the-art methods Couenne [46], Bonmin [47] and SCIP [48]. DNF solutions to 0/1 ILP problems are solved using the model described in Sec 5.1, and the concave min-cost network flow method described in Chapter 3. DNF modeling for 0/1 INLP problems is a combination of the model of Sec 5.1 for 0/1 ILP's, and the modeling of non-linear p-terms in ot- and ct-subgraphs discussed in Sec. 3.2-Sec. 3.3; the optimization algorithm is again the aforementioned concave min-cost network flow method.

For non-convex and convex INLPs, we use Couenne to obtain optimal solutions. Since Couenne does not provide the feature of solving problems near-optimally, we use Bonmin (which handles only convex problems) for near-optimal results that are more directly comparable to our DNF method. For ILPs, we use SCIP to solve these problems both optimally and near-optimally. The near-optimality upper bound ϵ is set to 10% for Bonmin and SCIP (in near-optimality mode). We note that the three competing solvers we use, Couenne, Bonmin and SCIP, are all the most state-of-the-art and publicly available solvers in their respective areas: non-convex, convex, and linear discrete optimization, respectively. All results, except the optimal result for protfold (see below), were obtained on Pentium IV machines with 1GB of main memory. As can be seen, the results for DNF are quite promising: (a) It obtains results that are on the average only 14-15% from optimal for the non-linear problems. Optimal results are obtained by Couenne with 5X-22X of the DNF method's runtime. Compared to the near-optimal technique Bonmin, the DNF method is 2.5X faster with a quality gap of 9% from Bonmin. (b) For ILPs, DNF produces results that are obtained 19X faster than SCIP, run in near-optimal mode, with comparable solution qualities. These are significant runtime improvements over comparable-quality state-of-the-art solvers.

Non-convex INLPs									
Bench-marks	**# of var.**	**# of constr.**	**Non-0 Coef.**	**Couenne**		**DNF method**			
				Val.	**Run-time**	**Val.**	**Opt. gap(%)**	**Run-time**	**Spdup over opt**
iran13x13	169	195	1014	3258	921s	3784	16	194s	4.7
imisc07	258	212	8620	2814	2928s	3275	16	340s	8.6
pb30235	600	50	1801	$3.38 \cdot 10^6$	4h	$3.74 \cdot 10^6$	11	648s	22.2
c-imod011	96	448	1404	$1.84 \cdot 10^{-4}$	1427s	$2.1 \cdot 10^{-4}$	14	162s	8.8
c-qap	225	31	954	$2.7 \cdot 10^5$	3042s	$3.1 \cdot 10^5$	13	242s	12.5
c-pb30235	600	50	1801	$1.4 \cdot 10^6$	5.4h	$1.6 \cdot 10^6$	16	1405s	14
Avg.							15		11.8

Convex INLPs										
Bench-marks	**# of var.**	**# of constr.**	**Non-0 Coef.**	**Bonmin**			**DNF method**			
				Val.	**Opt. gap(%)**	**Run-time**	**Val.**	**Opt. gap(%)**	**Run time**	**Spdup (10%)**
ibell3a	31	104	150	$9.2 \cdot 10^5$	4	10s	$10.02 \cdot 10^5$	14	6s	1.7
imod011	96	448	1404	$6.0 \cdot 10^{-4}$	7	187s	$6.54 \cdot 10^{-4}$	17	80s	2.3
qap	225	31	954	$4.1 \cdot 10^5$	5	508s	$4.23 \cdot 10^5$	9	144s	3.5
Avg.					5.3			14		2.5

ILPs										
Bench-marks	**# of var.**	**# of constr.**	**Non-0 Coef.**	**SCIP(10%)**			**DNF method**			
				Val.	**Opt. gap(%)**	**Run-time**	**Val.**	**Opt gap(%)**	**Run-time**	**Spdup (10%)**
frb53-24-1	1272	94227	189726	51	4	2.0h	51	4	794s	9
frb56-25-1	1400	109676	220752	51	9	2.6h	49	12	972s	11
frb59-26-1	1534	126555	254644	55	7	2.4h	52	12	1192s	7.2
protfold	1835	2111	23491	-27*	13	12h	-27	13	249s	86
air5	7195	425	52121	28104	7	265 s	26643	1	358s	0.74
p2756	2756	754	8937	3212	3	344s	3462	11	254s	1.3
Avg.					7.1			8.8		19

*Manually terminated after 12h. The best feasible solution found is reported.

TABLE XIV

THE RESULTS OF OUR DNF METHOD AND SEVERAL COMPETING METHODS FOR SOLVING VARIOUS 0/1 IP PROBLEMS. THE "OPT. GAP(%)" IS THE PERCENTAGE DIFFERENCE COMPARED TO THE OPTIMAL SOLUTION VALUE. THE "SPDUP OVER OPT" IS [THE RUN TIME FOR OPTIMAL SOLUTION]/[OUR RUN TIME], AND THE "SPDUP(10%)" IS [THE RUN TIME FOR THE NEAR OPTIMAL SOLUTION]/[OUR RUN TIME]. THE UPPER BOUND ϵ ON THE OPTIMALITY GAP FOR (GUARANTEED) NEAR-OPTIMAL SOLUTIONS IS ALWAYS 10%.

CHAPTER 6

A NEW DYNAMIC PROGRAMMING METHOD WITH WEAK DOMINATION FOR TACKLING MANY CONSTRAINTS

One important issue that often arises for general discrete optimization problems are the guaranteed optimality gap. Many such problems cannot be solved optimally, but when a feasible solution is generated, sometimes we want a guarantee that this solution is worse than the optimal one by no more than certain percentage. The C'-based concave min-cost network flow cannot produce such near-optimality guarantee, though generally good solution quality is obtained. In this chapter, we will discusses our dynamic programming (DP) based method to solve the 0/1 integer programming problem that provides a near-optimality guarantee. It is especially suitable for solving 0/1 integer non-linear programming problems.

6.1 Weak-Domination Based Dynamic-Programming

We propose a novel *weak-domination* based dynamic-programming (DP) type method to solve 0-1 INLP problems[1] near-optimally, i.e., obtain a solution that is *guaranteed* to be within $(1+\epsilon)$ of the optimal solution, where ϵ is the given optimality-gap bound. A strictly better dominating condition usually becomes very ineffective in pruning solutions when the number

[1]One advantage of our DP method for solving INLPs is that the lower bound for the objective or constraint function is calculated with min-cost network flow, while in most INLP solvers the bounding is done by solving a more difficult non-linear optimization problem. However, this advantage is nullified for ILPs , since the lower bounding there is naturally a linear programming problem, which is easy to solve. Thus the current version of DP is most suitable for INLPs but no so much for ILPs.

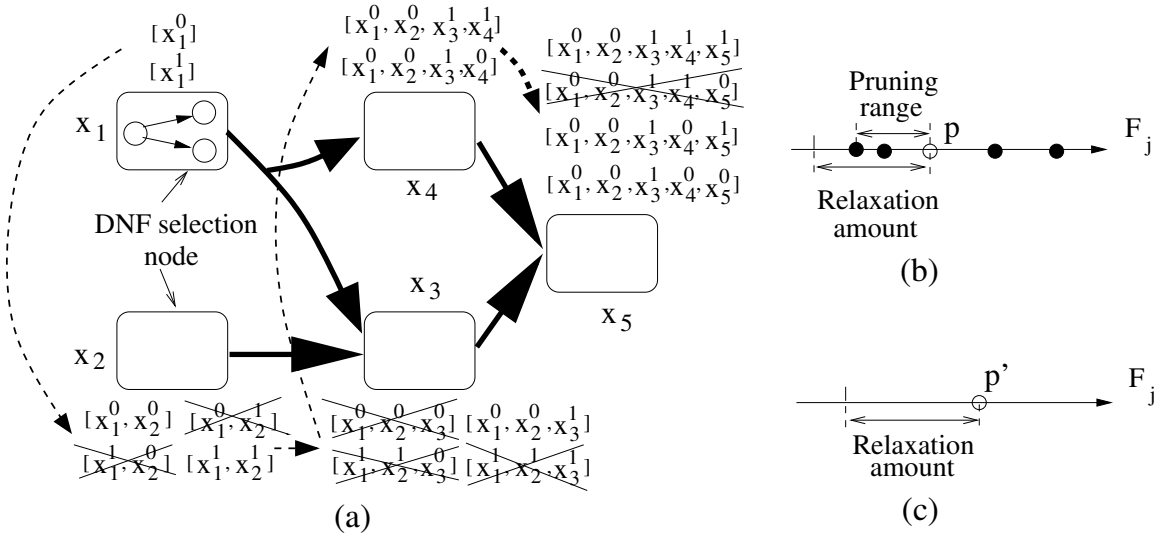

Figure 31. (a) The DP process: The DNF optimization graph G_{opt} shown at the level of MEA meta-nodes and meta-arcs (see Chapter 3). Dashed arcs indicate the processing order of variables. For each variable x_i, the PS's option vectors after combining options of x_i are shown. The pruned PS's are crossed off. (b) The weak domination condition: Dark nodes are the F_j values of PS's pruned by PS p. (c) For $p' \succ p$, its relaxation amount must cover the pruning range of p.

of constraints becomes large. Hence, we need a relaxed "weak dominating" condition that can effectively pruning solutions for problems with many constraints, while maintaining required near-optimality.

In the standard DP process, solutions are pruned based on *domination*. A solution dominates the other if it is better in all objective and constraint metrics. However, when the number of constraint metrics is large, it will be hard of a solution to dominate the other. Hence, the pruning efficiency of the domination based pruning is low for problems with large number of constraints, which results in long run time. Our weak-domination based pruning is a relaxed pruning criterion compared to domination based pruning. It targets achieving a reasonable pruning ratio even with large number of constraints. We will explain in more details in the following sections.

This DP technique operates on the optimization DNF graph G_{opt}, and also uses classical min-cost flow on parts of it to obtain certain lower bounds it needs to perform solution pruning. The following definitions and notations will be used. A *level-k partial solution* is denoted by p_k^i (indicating the i'th level-k partial solution), and is a partial solution (PS) that includes options selected for the first k variables $X_{1,k} = \{x_1, \ldots, x_k\}$. p_k^i is represented by its option vector $[x_1^{i_1}, \ldots, x_k^{i_k}]$, where x_j^t is an option or value of x_j (in 0/1 INLPs, there are only 2 options, 0 and 1, for each x_j) . For p_k^i, an objective or constraint function F_j can be divided into three parts: 1) the sum of *dependent p-terms* that are functions of $X_{1,k}$. The value of this sum (denoted by $F_j^d(p_k^i)$) is fixed for a p_k^i; 2) the sum (denoted by $F_j^u(p_k^i, X_{k+1,n})$) of *undetermined p-terms* that are functions of variables $X_{k+1,n} = \{x_{k+1}, \ldots, x_n\}$, where n is the total number of variables; 3)

the sum of the *boundary p-terms* that are functions of variables in both $X_{1,k}$ and $X_{k+1,n}$. Given a p_k^i, the latter's value (denoted by $F_j^b(p_k^i, X_{k+1,n})$) depends only on $X_{k+1,n}$. Finally, we will use $lb_j(p_k^i)$ to denote a lower bound on $F_j^b(p_k^i, X_{k+1,n}) + F_j^u(p_k^i, X_{k+1,n})$ over all $X_{k+1,n}$. $lb_j(p_k^i)$ can be determined by a standard/classical min-cost flow in the DNF graph for boundary and undetermined p-terms.

The general flow of our method is as follows. We maintain a PS set P. Initially, P includes all level-1 PS's, each of which contains one option for x_1. Full solutions are obtained by alternatively applying two procedures on PS's in P: 1) Partial solution expansion and 2) Partial solution pruning. The expansion process will combine each level-k PS $p_k^i \in P$ with each option x_{k+1}^j for x_{k+1} to form a level-$(k+1)$ PS $p_k^i \cup x_{k+1}^j$. In the pruning process, we will delete from P a subset of PS's based on two criteria: 1) weak domination (wd) criterion and 2) constraint violation criterion. An example of expansion and pruning is shown in Figure 31(a). Unlike many "safe" domination based pruning methods which only prune out PS's that cannot be expanded to the final optimal solution, our wd criteria is more aggressive in order to achieve a significantly higher pruning ratio. We will perform a near-optimality check after the final expansion and pruning iteration to determine if we have found the solution that meets the near-optimality requirement. If not, a *restoration process* will follow, in which we will iteratively restore pruned solutions in a best-first manner until the near-optimality check identifies a desired solution.

6.1.1 Partial Solution Pruning

To compare the quality of two PS's p_k^a and p_k^b of the same level-k, we define the *maximal inferiority* $I_j(p_a^k, p_b^k)$ of p_k^a to p_k^b for metric F_j as

$$I_j(p_k^a, p_k^b) = \max_{X_{k+1,n}} (F_j^d(p_k^a) + F_j^b(p_k^a, X_{k+1,n}) - (F_j^d(p_k^b) + F_j^b(p_k^b, X_{k+1,n}))) \quad (6.1)$$

$I_j(p_k^a, p_k^b)$ is the maximal difference between p_k^a and p_k^b for the sum of dependent and boundary p-terms of F_j (note that the dependent and boundary p-terms for p_k^a and p_k^b are the same—though not necessarily of the same value—since they are at the same level). The undetermined p-terms are not concerned here since their values are independent of p_k^a and p_k^b.

The meaning of $I_j(p_k^a, p_k^b)$ is that p_k^a is at most $I_j(p_k^a, p_k^b)$ worse than p_k^b for a metric F_j, i.e, for any option (i.e., value) combination O for variables in $V_{k+1,n}$, $F_j(p_k^a \cup O) - F_j(p_k^b \cup O) = \Delta^j_{p_k^a, p_k^b}(O) \leq I_j(p_k^a, p_k^b)$, where $F_j(p_k^a \cup O)$ denotes the value of F_j for options (i.e., variable values) in $p_k^a \cup O$ are selected. The maximization problem in Equation 6.1 can be solved approximately by first finding the maximal difference between p_k^a and p_k^b for each boundary p-term through enumeration. For example, in a boundary p-term T, we can enumerate all possible option selection combinations for variables in T that are not determined in p_k^a and p_k^b. For each such combination O_T we can determine the difference between $T(p_k^a \cup O_T)$ and $T(p_k^b \cup O_T)$, where $T(p_k^a \cup O_T)$ is the value of T with options selected in p_k^a and O_T. The maximal difference across all O_T is the maximal difference between p_k^a and p_k^b for T. $I_j(p_k^a, p_k^b)$ can be determined by summing up the maximal differences for all boundary p-terms as well as $(F^d_{j,p_k^a} - F^d_{j,p_k^b})$.

$I_j(p_k^a, p_k^b)$ will be used frequently in our following discussion of pruning criteria, and it possesses several important properties.

Property 1: $I_j(p_k^a, p_k^b) + I_j(p_k^b, p_k^c) \geq I_j(p_k^a, p_k^c)$.

Proof: This follows from the fundamental inequality property that $\max a + \max b \geq \max(a+b)$. $I_j(p_k^a, p_k^b) + I_j(p_k^b, p_k^c) = \max_{V_{k+1,n}} \Delta^j_{p_k^a, p_k^b}(V_{k+1,n}) + \max_{V_{k+1,n}} \Delta^j_{p_k^b, p_k^c}(V_{k+1,n}) \geq \max_{V_{k+1,n}} (\Delta^j_{p_k^a, p_k^b}(V_{k+1,n}) + \Delta^j_{p_k^b, p_k^c}(V_{k+1,n})) = \max_{V_{k+1,n}} \Delta^j_{p_a, p_c}(V_{k+1,n}) = I_j(p_k^a, p_k^c)$. $\diamondsuit$

6.1.2 Weak Domination Criterion

Our *weak domination (wd)* condition for PS pruning can be stated as follows: a level-k PS p_k^a *weak dominates* another level-k PS p_k^b (denoted $p_k^a \succ p_k^b$) if and only if for each metric F_j:

$$I_j(p_k^a, p_k^b) \leq \delta_j \tag{6.2}$$

where $\delta_j \geq 0$ is a relaxation amount that is a function of the optimality gap bound ϵ. Unlike a "strictly better" domination condition, this condition allows $p_k^a \succ p_k^b$ even if it is δ_j worse than p_k^b in metric F_j. In this way, the wd probability between PS pairs is greatly increased. Furthermore, we also keep track of a *pruning range* $d_j(p_k^a)$ for each PS p_k^a and each metric F_j, which records the difference between p_k^a and the best PS for F_j pruned due to weak domination by p_k^a (or pruned by p_k^a for short); see Figure 31(b). Assume that a PS p_k^b has pruned some partial solution p_k^c that is better in F_j, i.e., $d_j(p_k^b) > 0$. Then, for another PS p_k^a to prune p_k^b, we must ensure that it weak dominates both p_k^b and p_k^c, so that the pruned PS p_k^c is weak dominated by at least one remaining (unpruned) PS. Accordingly, the wd condition for PS pruning is modified to: $p_k^a \succ p_k^b$ if and only if for each metric F_j:

$$I_j(p_k^a, p_k^b) + d_j(p_k^b) \leq \delta_j \tag{6.3}$$

Figure 31(c) shows an example of weak domination considering the pruning range. We should also note that with weak domination, we can have $p_k^a \succ p_k^b$ and $p_k^b \succ p_k^a$ at the same time. In such a situation, out of p_k^a, p_k^b we will keep that PS that weak dominates more PS's (and thus prune the other PS) since it gives us more pruning potential, and generally it also has higher quality. To relate to the ϵ near-optimality requirement, δ_j can be determined as $\epsilon\times$ [a lower bound on the optimal solution value of F_j].

Theorem 7 *With the wd condition in Equation 6.3, there will be a full solution in P that is at most ϵ times worse than the optimal solution for each metric .*

Proof Outline: Let p_k^o denote the level-k PS in the optimal solution. We can prove by induction on k that there is a PS p_k^i in P s.t. $I_j(p_k^i, p_k^o) \leq d_j(p_k^i)$ and $I_j(p_k^i, p_k^o) \leq \delta_j$. Then, finally p_n^i is the stated solution. $\diamondsuit$

For the objective function, Theorem 7 establishes the near-optimality of the obtained full solutions. However, p_n^i being ϵ worse in the constraint metrics F_j than the optimal solution may render p_n^i infeasible. Hence, it is possible that among the final full solutions obtained, there is no feasible solution that is ϵ near-optimal. Such a situation happens due to the *over pruning* phenomenon, i.e, a PS that can be expanded to a feasible solution is pruned by a PS that cannot be expanded to a feasible solution due to the relaxation allowed in the wd condition on constraint functions. The method for tackling over pruning is discussed in Sec. 6.1.4.

6.1.3 Constraint Violation Criterion

Let c_j be the given upper bound on a constraint function F_j. For a PS p_k^i of level-k, the *constraint violation based pruning criterion* for p_k^i has two parts: 1) p_k^i cannot be expanded to a feasible solution, i.e., $F_j^d(p_k^i) + lb_j(p_k^i) > c_j$, and 2) p_k^i does not over-prune any other PS. Note that if the second condition is not true, we need to keep p_k^i for restoration of those over-pruned PS's (see Sec. 6.1.4). A necessary condition for p_k^i to over prune is that for each constraint function F_j:

$$F_j^d(p_k^i) + lb_j(p_k^i) - d_j(p_k^i) \leq c_j \tag{6.4}$$

The rationale is that $d_j(p_k^i)$ is an upper bound on how much a pruned PS is better than p_k^i for F_j.

6.1.4 Over-Pruning Restoration

After expansion and pruning iterations, finally P contains a set P_f of feasible full solutions and a set P_{inf} of infeasible full solutions with possible over pruning. Let B denote the lower bound on the optimal F_0 value, and can be determined as $B = \min_{S \in P}(F_0(S) - d_0(S))$, since, according to the proof of Theorem 7 there is a solution in P, that is at most $d_0(S)$ worse than the optimal solution. Let S_b be the best solution in P_f w.r.t. F_0. Then, we can check whether the near-optimality condition is satisfied by S_b, i.e., if $F_0(S_b) \leq B \cdot (1 + \epsilon)$. If this check fails,

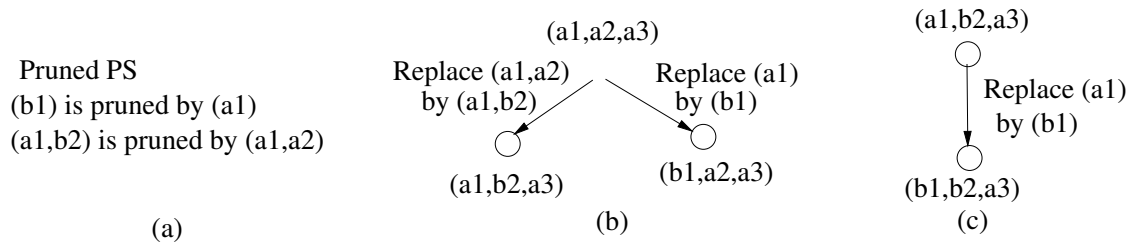

Figure 32. An example restoration process for a problem with three variables v_1, v_2, v_3. Each variable v_i has two options a_i and b_i. (a) The pruned partial solutions that are recorded. (b) The restoration process on solution (a_1, a_2, a_3) with the pruned PS's in [a]. (c) A further restoration process on a solution (a_1, b_2, a_3) generated from the restoration in [b].

we need to perform restoration on solutions in P_{inf} to recover more feasible solutions, as well as tighten the bound B.

Let p_k^S $(1 \leq k \leq n)$ be a level-k PS of an infeasible full solution S. Restoring the solution pruned by S can be performed by replacing each PS p_k^S with each PS pruned by it. Note that in order to do this restoration, we need to record for each PS the set of PS's pruned by it. An example of restoration is shown in Figure 32. After performing restoration on S, S can be deleted from P_{inf}. Among the restored solutions, the feasible ones are added to P_f, the infeasible ones with possible over pruning are added to P_{inf}, and then B and S_b are updated. The restoration process terminates when the near-optimality check $F_0(S_b) \leq B \cdot (1 + \epsilon)$ is satisfied. In order to reduce the number of solutions restored before termination, we will follow a best first order, i.e, each time we can choose $S \in P_{inf}$ with the best objective function value for restoration. We have the following result.

Theorem 8 *The aforementioned over-pruning restoration technique guarantees that our DP method will find an ϵ near-optimal feasible solution of the given problem if one exists.*

6.2 Results of Applying Dynamic Programming with WD to 0/1 INLP Problems

In Table XV, we also show results of our DP technique for INLPs for $\epsilon = 10\%$ under "Our DP method" column. Again, the results are very promising. The actual optimality gap achieved is 6.3% for non-convex INLPs and 7% for convex INLPs. Compared to the optimal Couenne method [46], DP achieves a speedup of almost 5X, and compared to near-optimal Bonmin [47] which has a similar optimality gap, DP achieves a speedup of almost 2X. The tradeoffs between DP and the DNF method are as follows. DP guarantees a given near-optimality, while the DNF method obtains an empirically observed near-optimality—its average optimality gap is 9-15%. However, the latter is faster than DP by about 2X, and is suitable for both ILPs and INLPs, while DP is only suitable for INLPs.

Non-convex INLPs						
Bench-marks	**Couenne**		**Our DP method**			
	Val.	**Run-time**	**Val.**	**Opt. gap(%)**	**Run-time**	**Spdup over opt**
iran13x13	3258	921s	3516	8	200s	4.6
imisc07	2814	2928s	3010	7	963s	3.0
pb30235	$3.38 \cdot 10^{6}$	4h	$3.5 \cdot 10^{6}$	4	1.1h	3.6
c-imod011	$1.84 \cdot 10^{-4}$	1427s	$1.9 \cdot 10^{-4}$	4	618s	2.3
c-qap	$2.7 \cdot 10^{5}$	3042s	$2.9 \cdot 10^{5}$	8	269s	11
c-pb30235	$1.4 \cdot 10^{6}$	5.4h	$1.5 \cdot 10^{6}$	7	1.4h	3.8
Avg.				6.3		4.7

Convex INLPs										
Bench-marks	**Couenne**		**Bonmin**			**Our DP method**				
	Val.	**Run-time**	**Val.**	**Opt. gap(%)**	**Run-time**	**Val.**	**Opt. gap(%)**	**Run-time**	**Spdup over opt**	**Spdup (10%)**
ibell3a	$8.79 \cdot 10^{5}$	18s	$9.2 \cdot 10^{5}$	4	10s	$9.4 \cdot 10^{5}$	6	6s	3	1.7
imod011	$5.6 \cdot 10^{-4}$	612s	$6.0 \cdot 10^{-4}$	7	187s	$6.0 \cdot 10^{-4}$	7	206s	2.5	0.9
qap	$3.88 \cdot 10^{5}$	1716s	$4.1 \cdot 10^{5}$	5	508s	$4.2 \cdot 10^{5}$	8	195s	8.8	2.6
Avg.				5.3			7.0		4.8	1.7

*Manually terminated after 12h. The best feasible solution found is reported.

TABLE XV

THE RESULTS OF OUR METHOD, THE DP METHOD WITH WEAK DOMINATION, AND SEVERAL COMPETING METHODS FOR SOLVING CONVEX AND NON-CONVEX 0/1 INLP PROBLEMS.

CITED LITERATURE

1. Choi, W. and Bazargan, K.: Incremental placement for timing optimization. In International Conference on CAD, pages 463–466, 2003.

2. Elmore, W. C.: The transient response of damped linear networks with particular regard to wideband amplifiers. Journal of Applied Physics, 19:55–64, 1948.

3. Silveira, L. M., Kamon, M., and White, J.: Efficient reduced-order modeling of frequency-dependent coupling inductances associated with 3-d interconnect structures. In Design Automation Conference, pages 376–380, 1995.

4. Dutt, S. and Ren, H.: Discretized network flow techniques for timing and wire-length driven incremental placement with white-space satisfaction. IEEE Transactions on VLSI System, 19(7):1277–1290, 2011.

5. Kao, J., Chandrakasan, A., and Ch, A.: Dual-threshold voltage techniques for low-power digital circuits. IEEE Journal of Solid-State Circuits, 35:1009–1018, 2000.

6. Billingsley, P.: Probability and measure. Wiley, 3 edition, 1995.

7. Sinha, D., Shenoy, N. V., and Zhou, H.: Statistical timing yield optimization by gate sizing. IEEE Transactions on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems, 25:1140–1146, 2006.

8. Clark, C. E.: The greatest of a finite set of random variables. Operations Research, 9:145–162, 1961.

9. Wang, J., Das, D., and Zhou, H.: Gate sizing by lagrangian relaxation revisited. IEEE Trans. on CAD of Integrated Circuits and Systems, 28(7):1071–1084, 2009.

10. Hu, S., Ketkar, M., and Hu, J.: Gate sizing for cell library-based designs. In DAC, pages 847–852, 2007.

11. Wu, T. and Davoodi, A.: Pars: fast and near-optimal grid-based cell sizing for library-based design. In ICCAD, pages 107–111, 2008.

CITED LITERATURE (Continued)

12. Agarwal, A., Chopra, K., Blaauw, D., and Zolotov, V.: Circuit optimization using statistical static timing analysis. In DAC, pages 321–324, 2005.

13. Srivastava, A., Kastner, R., Chen, C., and Sarrafzadeh, M.: Timing driven gate duplication. IEEE Trans. VLSI Syst., 12(1):42–51, 2004.

14. Usami, K. and Horowitz, M.: Clustered voltage scaling technique for low-power design. In International Symposium on Low Power Design, pages 3–8, 1995.

15. Wu, H., Wong, M., and Liu, I.: Timing-constrained and voltage-island-aware voltage assignment. In DAC, pages 429–432, 2006.

16. van Ginneken, L. P. P. P.: Buffer placement in distributed rc-tree networks for minimal elmore delay. In IEEE International Symposium on Circuits and Systems, pages 865–868, 1990.

17. Lillis, J., Cheng, C., and Lin, T. Y.: Optimal wire sizing and buffer insertion for low power and a generalized delay model. In International Conference on CAD, pages 138–143, 1995.

18. Dhillon, Y., Diril, A., Chatterjee, A., and Lee, H.: Algorithm for achieving minimum energy consumption in cmos circuits using multiple supply and threshold voltages at the module level. In ICCAD, pages 693–700, 2003.

19. Gao, F. and Hayes, J.: Total power reduction in cmos circuits via gate sizing and multiple threshold voltages. In DAC, pages 31–36, 2005.

20. Liu, Y. and Hu, J.: A new algorithm for simultaneous gate sizing and threshold voltage assignment. IEEE Trans. on CAD of Integrated Circuits and Systems, 29(2):223–234, 2010.

21. Jiang, Y., Sapatnekar, S., Bamji, C., and Kim, J.: Interleaving buffer insertion and transistor sizing into a single optimization. IEEE Trans. VLSI Syst., 6(4):625–633, 1998.

22. Donath, W., Kudva, P., Stok, L., Villarrubia, P., Reddy, L., Sullivan, A., and Chakraborty, K.: Transformational placement and synthesis. In DATE, pages 194–201, 2000.

23. Srivastava, A., Sylvester, D., and Blaauw, D.: Power minimization using simultaneous gate sizing, dual-vdd and dual-vth assignment. In DAC, pages 783–787, 2004.

CITED LITERATURE (Continued)

24. Chinnery, D. and Keutzer, K.: Linear programming for sizing, vth and vdd assignment. In Proc. of ISLPED, pages 149–154, 2005.

25. Ahujaa, R. and Orlin, J.: Network flow: theory and application. Pretice Hall: Pretice Hall, 1991.

26. Dutt, S., Dai, Y., Ren, H., and Fontanarosa, J.: Selection of multiple snps in case-control association study using a discretized network flow approach. In Proceedings of the 1st International Conference on Bioinformatics and Computational Biology, BICoB, pages 211–223, 2009.

27. Ren, H. and Dutt, S.: A discretized network flow based method for solving travelling salesman problems. `www.ece.uic.edu/~dutt/tech-reps/TSP_DNF.pdf`, Tech. Report, 2011. (Note: In case you cut and paste this URL please retype the tilde (`~`) character portion of the URL from your keyboard.).

28. Dutt, S. and Ren, H.: Solving boolean satisfiability problems using discretized network flow. `www.ece.uic.edu/~dutt/tech-reps/SAT_DNF.pdf`, Tech. Report, 2011. (Note: In case you cut and paste this URL please retype the tilde (`~`) character portion of the URL from your keyboard.).

29. Dutt, S., Ren, H., Yuan, F., and Suthar, V.: A network-flow approach to timing-driven incremental placement for asics. In ICCAD, pages 375–382, 2006.

30. Ren, H. and Dutt, S.: A provably high-probability white-space satisfaction algorithm with good performance for standard-cell detailed placement. IEEE Trans. VLSI Syst., 19(7):1291–1304, 2011.

31. et. al., S. N. A.: Unification of partitioning, placement and floorplanning. In ICCAD, pages 550–557, 2004.

32. Yang, X., Choi, B., and Sarrafzadeh, M.: Timing-driven placement using design hierarchy guided constraint generation. In ICCAD 2002, pages 177–184, 2002.

33. Chen, T., Jiang, Z., Hsu, T., Chen, H., and Chang, Y.: Ntuplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints. IEEE Trans. on CAD, 27(7):1228–1240, 2003.

34. Nahapetyan, A. and Pardalos, P.: Adaptive dynamic cost updating procedure for solving fixed charge network flow problems. Comput. Optim. Appl., 39:37–50, 2008.

CITED LITERATURE (Continued)

35. Ren, H. and Dutt, S.: Effective power optimization under timing and voltage-island constraints via simultaneous vdd, vth assignments, gate sizing, and placement. IEEE Trans. on CAD, 30(5):746–759, 2011.

36. Ahuja, R. K. and Orlin, J. B.: The scaling network simplex algorithm. Oper. Res., 40:5–13, 1992.

37. Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D.: Linear Programming and Network Flows. Wiley-Interscience, 2004.

38. Ren, H. and Dutt, S.: A network-flow based cell sizing algorithm. In Int'l Workshop on Logic Synthesis, pages 7–14, 2008.

39. Wu, H., Liu, I., Wong, M. D. F., and Wang, Y.: Post-placement voltage island generation under performance requirement. In International conference on Computer-aided design, pages 309–316, 2005.

40. Kim, H., Matoglu, E., Choi, J., and Swaminathan, M.: Modeling of multi-layered power distribution planes including via effects using transmission matrix method. In Asia and South Pacific Design Automation Conference, page 59, 2002.

41. Nassif, S.: Delay variability: sources, impacts and trends. In Solid-State Circuits Conference, pages 368–366, 2000.

42. Ampl miqp benchmarks. `http://plato.asu.edu/ftp/ampl\_files/miqp\_ampl`.

43. Minlp benchmarks. `http://www.gamsworld.org/minlp/minlplib.htm`.

44. Miplib benchmarks. `http://miplib.zib.de/`.

45. Pseudo-boolean benchmarks. `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/pb-benchmarks.htm`.

46. Couenne solver. `https://projects.coin-or.org/Couenne`.

47. Bonmin solver. `http://www.coin-or.org/Bonmin/`.

48. Scip solver. `http://scip.zib.de/scip.shtml`.

VITA

NAME	Huan Ren
EDUCATION	Ph.D., Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, May, 2012
	M.S., Electrical and Computer Engineering, University of Illinois at Chicago, Chicago,Illinois, May, 2011
	B.S., Electrical Engineering, Zhejiang University, Hangzhou, China, Jun, 2005
EXPERIENCE	Research Assistant, DART Lab, Dept. of ECE, University of Illinois at Chicago, 08/2005 - 2/2011
	Teaching Assistant, Dept. of ECE, University of Illinois at Chicago, 08/2007 - 08/2008
PUBLICATIONS	H. Ren and S. Dutt: Effective Power Optimization Under Timing and Voltage-Island Constraints via Simultaneous Vdd, Vth Assignments, Cell Sizing and Placement. IEEE Trans. on CAD, 30(5), 746-759, 2011.
	H. Ren and S. Dutt: Provably High-Probability White-Space Satisfaction Algorithm with Good Performance for Standard-Cell Detailed Placement. IEEE Trans. on VLSI Systems, 19(7), pp. 1291-1304, 2011.
	S. Dutt and H. Ren: Discretized Network Flow Techniques for Timing and Wire-Length Driven Incremental Placement with White-Space Satisfaction. IEEE Trans. on VLSI Systems, 19(7), pp. 1277-1290, 2011.
	S. Dutt and H. Ren, Timing Yield Optimization via Discrete Gate Sizing using Globally-Informed Delay PDF Functions. Proceedings of International Conference on CAD, San Jose, CA, 2010.

VITA (Continued)

S. Dutt, Y. Dai, H. Ren, and J. Fontanarosa: Selection of Multiple SNPs in Case-Control Association Study Using a Discretized Network Flow Approach. Lecture Notes in Computer Science, 5462, pp. 211-223, 2009.

H. Ren and S. Dutt: Algorithms for Simultaneous Consideration of Multiple Physical Synthesis Transforms for Timing Closure. Proceedings of International Conference on CAD, San Jose, CA, 2008.

H. Ren and S. Dutt: A Network-Flow Based Cell Sizing Algorithm. Proceedings of International Workshop on Logic and Synthesis, Lake Tahoe, CA, 2008.

H. Ren and S. Dutt: Constraint Satisfaction in Incremental Placement with Application to Performance Optimization under Power Constraints. Proceedings of International Conference on Computer Design, Lake Tahoe, CA, 2007.

S. Dutt, H. Ren, F. Yuan and V. Suthar: A Network-Flow Approach to Timing-Driven Incremental Placement for ASICs. Proceedings of International Conference on CAD, San Jose, 2006.

SERVICE

Student member of the Institute of Electrical and Electronics Engineers (IEEE).

Reviewer for 2009 IEEE International Conference on CAD (ICCAD 2009).

Reviewer for 2008 and 2009 Great Lakes Symposium on VLSI (GLSVLSI 2008, 2009).

Reviewer for 2010 Design Automation and Test in Europe Conference (DATE 2010).