Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 474601, 15 pages
http://dx.doi.org/10.1155/2013/474601

Research Article

Fast and Near-Optimal Timing-Driven Cell Sizing under Cell Area and Leakage Power Constraints Using a Simplified Discrete Network Flow Algorithm

Huan Ren and Shantanu Dutt

Department of ECE, University of Illinois at Chicago, Chicago, IL 60607, USA

Correspondence should be addressed to Shantanu Dutt; dutt@ece.uic.edu

Received 24 May 2012; Revised 6 November 2012; Accepted 21 November 2012

Academic Editor: Gi-Joon Nam

We propose a timing-driven discrete cell-sizing algorithm that can address total cell size and/or leakage power constraints. We model cell sizing as a "discretized" mincost network flow problem, wherein available sizes of each cell are modeled as nodes. Flow passing through a node indicates the choice of the corresponding cell size, and the total flow cost reflects the timing objective function value corresponding to these choices. Compared to other discrete optimization methods for cell sizing, our method can obtain near-optimal solutions in a time-efficient manner. We tested our algorithm on ISCAS'85 benchmarks, and compared our results to those produced by an optimal dynamic programming- (DP-) based method. The results show that compared to the optimal method, the improvements to an initial sizing solution obtained by our method is only 1% (3%) worse when using a 180 nm (90 nm) library, while being 40–60 times faster. We also obtained results for ISPD'12 cell-sizing benchmarks, under leakage power constraint, and compared them to those of a state-of-the-art approximate DP method (optimal DP runs out of memory for the smallest of these circuits). Our results show that we are only 0.9% worse than the approximate DP method, while being more than twice as fast.

1. Introduction

In order to achieve a balance between design quality and time-to-market, cell library-based design is becoming the dominant design methodology over the custom design method even for high-performance ICs. Usually in a cell library, several different cell implementations are available for the same function with different sizes, intrinsic delays, driving resistances, and input capacitances. Choosing the cell with an appropriate size, that is, *cell sizing*, is a very effective approach to improve timing.

The cell-sizing problem has been studied for a long time. Many methods [1, 2] assume the availability of a continuous range of cell sizes; that is, the size of a cell can take any value in a range. Then, the obtained gate size is rounded to the nearest available size in the library. However, a large number of realistic cell libraries are "sparse," for example, geometrically spaced instead of uniformly spaced [3]. Geometrically spaced gate sizes are desired in order to cover a large size range with a relatively small number of cell instances. Also it has been proved in [4] that, under certain conditions, the set of optimal gate sizes must satisfy the geometric progression. With a sparse library, the simple rounding scheme can introduce huge deterioration from the continuous solution, which often causes the sizing results to fail to meet given timing requirements [3].

On the other hand, few time-efficient methods are known that they can directly handle timing optimization with discrete cell sizes since this problem is NP complete. The technique in [5] uses multidimensional descent optimization that iteratively improves a current solution by changing the size choices of a set of cells that produces the largest improvement. It is not clear how well this method can avoid being trapped in a local optimum. In [3], a new rounding method is developed based on an initial continuous solution. Instead of only rounding to the nearest available size, the method visits cells in topological order in the circuit, and tries several discrete sizes around the continuous solution

for each cell. In order to reduce the search space, after a new cell is visited, it performs a pruning step that discards obviously inferior solutions and a merging step that keeps only several representative solutions within a certain quality region. The run time of this method is significantly larger than the method in [5].

More recently, [6] has proposed a method that uses a combination of continuous and discrete optimization techniques for the power-driven timing-constrained cell-sizing problem. The problem is first simplified to an unconstrained problem using Lagrange relaxation. Then, the resulting unconstrained discrete sizing problem is solved using dynamic programming. Through Lagrange relaxation-based simplification, it is able to handle complex delay constraints of current industry designs. However, as a continuous convex optimization technique, the quality and convergence of the Lagrange relaxation method is not guaranteed when applied to discrete problems. On the other hand, our proposed method directly handles constraint satisfaction simultaneously with objective optimization in a unified and specially designed mincost network flow model.

In this paper, we propose a network-flow-based method for the discrete gate-sizing problem. In our method, the different size options of a cell are modeled by nodes in the network graph. The flow cost represents the change in the timing objective function value when the cell sizes corresponding to the nodes in the graph which have flows through them are chosen. Hence, by solving a mincost flow in the network graph, near-optimal cell sizing can be determined by choosing cell sizes whose corresponding nodes have the mincost flow through them—the near optimality comes from having to constraint the flow to adhere to certain discrete requirements like going through exactly one size-option node per cell.

By modeling the gate-sizing problem as a mincost network flow problem, we can solve it using standard network flow algorithms, which are very time efficient. Also problem constraints, like the maximum allowable total cell area, can be handled efficiently by making the flow amount proportional to the chosen cell area and using an arc with an appropriate capacity to limit the total flow amount.

However, network flow is a continuous optimization method. Thus when applying it to solve the discrete option selection problem, invalid solutions may be produced. For example, the mincost flow may pass through two sizing options for the same cell. We solve such problems by using *min lookahead-cost* or *max flow* selection heuristics. Network flow has been used to solve various EDA problems including placement [7–9] and placement legalization [10]. In the recent technique of [10], only the network graph modeling the legalization problem, and which nodes (representing bins of cells) are overfull (these are then supply nodes) and under full (these are demand nodes) have been determined a priori to solve the mincost flow problem. Costs of arcs, which represent shipment of cells, between adjacent bins are not known in advance since this depends on the exact set of cells being shipped from one bin to an adjacent one. This set of cells and the cost of arcs out of the "current" bin are determined dynamically within Dijstra's shortest path computation that is used iteratively to solve an approximate mincost flow problem. Our problem in using network flow to solve the cell-sizing problem is different: while we know the exact cost and capacities of arcs in the network graph modeling the problem, the issue we need to tackle is that of preventing splitting of the flow among each subset of arcs that represent the selection of the sizes of each cell, so that flow can go through only one of these arcs, thus providing a unique selection of a size for each cell. We solve this problem in an outer loop enveloping the mincost flow computation, as opposed to the technique in [10], which solves it within an inner loop of the mincost flow computation.

This paper is an extended version of the workshop paper [11] that appeared in a internal workshop compendium of papers, which is not considered a published proceedings. Thus it is not strictly necessary to discuss the issue of extensions of that paper. However, there are extensions, which include an updated discussion of previous work, results for benchmark circuits using Synopsys's 90 nm library (in addition to the 180 nm library used in [11]), and an analysis of the differences in the two sets of results for the two libraries.

The rest of the paper is organized as follows. Section 3 provides an overview of our method. A general view of our size selection network graph (SSG) is presented in Section 4. In Sections 5.1–5.4, we discuss various issues of the SSG. In Section 6, we show how to obtain a valid mincost flow in the SSG. Section 7 briefly describes an optimal exhaustive search method to which we compare our network-flow-based technique. Section 8 presents experimental results and we conclude in Section 9.

2. New Algorithmic Approach Used

In this paper we use a simplified version of *discretized network flow* (DNF) that has been recently introduced as a versatile optimization technique for CAD problems over the last four years [7, 9, 12–14]. The DNF technique imposes certain discrete requirements on the flow through the network graph G like a mutual exclusivity constraint on certain arc sets, called *mutually exclusive arc* (MEA) sets, in which at most one arc can have a nonzero flow in each MEA set. In the above cited works, the DNF problems is modeled as a fixed-charge network flow problem [15] in order to solve complex physical synthesis problems using multiple transforms and constraints with significant efficacy. However, it is possible to solve the discrete cell-sizing problem near optimally with a simple version of DNF in which max-flow or min-lookahead-cost (subsequently termed "mincost" for brevity) heuristics for determining which arc in each MEA set should have nonzero flow. We present this simpler version of DNF as the new algorithmic technique in this paper.

3. Overview of Our Method

Our cell-sizing method starts from an initial sizing solution that may be far from the optimal. The objective is to improve the critical path delay of the circuit by resizing cells. We define $\mathcal{P}_\alpha$ to be the set of paths with a delay greater than $1 - \alpha$

fraction ($\alpha < 1$) of the most critical path delay. In order to reduce the complexity of the problem we only consider changing the sizes of cells in $\mathscr{P}_\alpha$. In our experiments, α is set to be 0.1. This simplification does not limit the optimization potential of our method, since our method is incremental in its nature. Thus we can iterate it several times to take more paths into consideration.

We define $\mathrm{CS}(n_j)$ to be the set of critical sinks of a net n_j, which are (1) all sinks in $\mathscr{P}_\alpha$ if the net is in $\mathscr{P}_\alpha$, or (2) the sink with the minimum slack otherwise. Our method tries to improve the critical path delay by minimizing the objective function proposed in [9]. A timing cost $t_c(n_i)$ of a net n_i is defined as [9]:

$$t_c(n_i) = \sum_{u_j \in \mathrm{CS}(n_i)} \frac{D(u_j, n_i)}{S_a^\beta(n_i)}, \tag{1}$$

where u_j is a sink cell of net n_i, $D(u_j, n_i)$ is the net delay of n_i to u_j, and $S_a(n_i)$ is the *allocated slack* of a net in the initial sizing solution; the allocated slack is defined as the path slack divided by the number of nets in the path. β is the exponent of the allocated slack used to adjust the weight difference between costs of nets on critical and noncritical paths. Based on experimental results, $\beta = 1$ works best in this scenario, since only nets in $\mathscr{P}_\alpha$ and those connected to it are considered in the optimization function (see below). Let $\widehat{\mathscr{P}}_\alpha$ be the set of nets that are either in $\mathscr{P}_\alpha$ or connected to nodes in it. The timing objective function F_t is the summation of $t_c(n_i)$ of all nets in $\widehat{\mathscr{P}}_\alpha$ given as

$$F_t = \sum_{n_i \in \widehat{\mathscr{P}}_\alpha} t_c(n_i). \tag{2}$$

Here we only consider nets in $\widehat{\mathscr{P}}_\alpha$, since the delays of only these nets will change with our selection of resizable cells. Note that in this objective function the nets on more critical paths have a higher contribution since the net cost is inversely proportional to the allocated slack and thus will be optimized more—a desirable outcome, especially in a scenario where there is a quota on the resource (e.g., total area) available for optimization.

Usually, the total area is given as a constraint for timing driven cell sizing. Our algorithm can also handle this constrained optimization problem.

4. Overview of the Size Selection Graph (SSG)

We model the timing-minimization cell-size selection problem as a mincost network flow problem. A network flow graph called *the size selection graph* (SSG) is constructed in which the set of sizing options of each cell is modeled as a set of *sizing option nodes*, and each sizing option node corresponds to one sizing option. We use flows in the SSG to model cell-size selection; that is, a size option of a cell is chosen by a flow in the SSG if its corresponding option node in the SSG is on the path of the flow. Hence, each flow through the SSG that passes through one option node in every set of sizing options in the SSG corresponds to a sizing selection for the cells in $\mathscr{P}_\alpha$. Furthermore, if we can set the costs of arcs between option nodes in the SSG in such a way that the total cost incurred by a flow through the SSG is equal to the change in the timing objective function F_t (2) corresponding to the sizing scheme selected by the flow, then the mincost flow in the SSG selects the optimal cell-sizing scheme for F_t. Hence the problem of finding the optimal cell sizing is converted to the problem of finding the mincost flow in a graph for which several efficient algorithms such as the network simplex algorithm and the enhanced scaling algorithm [16] are available. The general structure of our SSG is described below.

4.1. The SSG Structure. Since F_t is the summation of the timing costs t_c of nets, we employ a divide-and-conquer approach in constructing the SSG; that is, first a mininetwork flow graph called a *net structure* is constructed for each net in $\mathscr{P}_\alpha$, and then net structures of connected nets are connected by *net spanning structures* to form the complete SSG as shown in Figure 1. Thus, the SSG has a similar topology as the paths in $\mathscr{P}_\alpha$. Note that the source node S is the only supply node with total flow of $F(S)$ (whose determination is discussed in Section 5.4), and the sink T is the only demand node of flow amount $F(S)$ in the SSG. A net structure N_j is a *child net structure* of N_i if they are connected, and N_j follows N_i in the flow direction in the SSG (i.e., the signal direction in the circuit is from net n_i to net n_j); correspondingly N_i is also a *parent net structure* of N_j. Each net structure contains the sets of option nodes for cells in the corresponding net.

Let us denote the set of sizing options of a cell u as S_u, and the lth sizing option in it as S_u^l. For a net n_i with a driving cell u_d, the net structure is constructed by connecting each sizing option $S_{u_d}^l$ of u_d to all sizing option nodes in S_{u_k} of every sink cell u_k to form a complete bipartite subgraph between S_{u_d} and S_{u_k}. An example is shown in Figure 2. The net has one driving cell u_d and two critical sink cells u_j and u_k; see Figure 2(a). The corresponding net structure is shown in Figure 2(b). Here, we only show two sizing options in each option set. A flow f through the net structure is also shown, which selects size option $S_{u_d}^1$ for u_d, $S_{u_k}^1$ for u_k, and $S_{u_j}^2$ for u_j. With the complete bipartite subgraphs between the sizing option sets of the drive cell and sink cells in a net structure, every possible combination of size choices of cells in the net has a corresponding flow through the net structure and hence is considered in our size selection process.

There are two major issues that need to be tackled in constructing the SSG in order to correctly map the mincost flow in the SSG to an optimal sizing choice. They are as follow.

(1) In each net structure, the cost of each arc needs to be determined so the total cost of a flow through the net structure can accurately capture the change value of the timing cost t_c for the net corresponding to the sizing options chosen by the flow. A detailed description of this issue is given in Section 5.1.

(2) The flow that is determined must be a valid flow, that is, can be converted to a valid sizing option selection. A valid sizing option selection requires the satisfaction of two types of *consistencies*. First, sizing option nodes of a particular cell

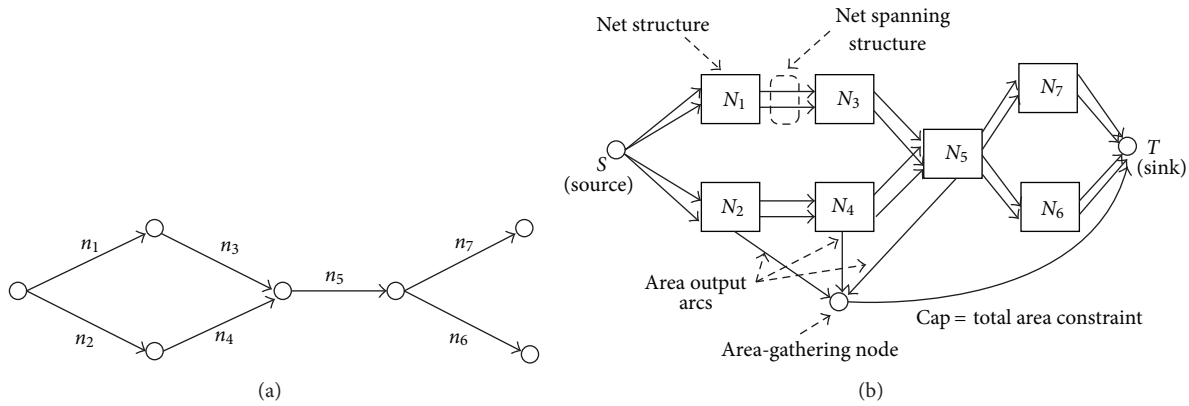

FIGURE 1: (a) Nets in $\mathscr{P}_\alpha$ of a circuit. (b) The corresponding SSG, which includes net structures and net spanning structures. N_i is the net structure corresponding to net n_i. Each net structure will send a flow of amount equal to the total cell size it chooses through the area output arc to the area-gathering node. All these flows are routed to the sink through an arc from the area-gathering node with a capacity equal to the given total cell area constraint.

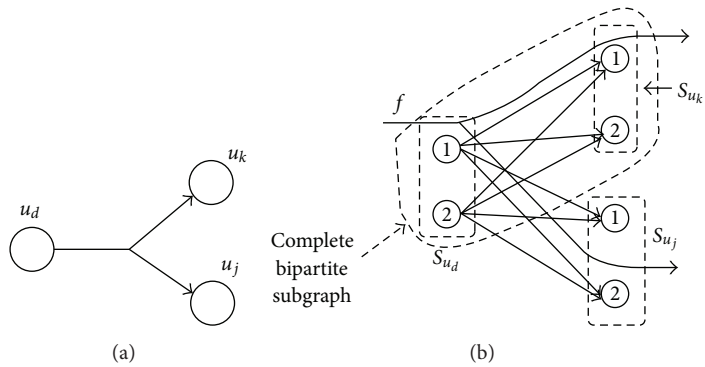

FIGURE 2: (a) A net in critical paths. (b) The net structure corresponding to the net; f is a flow through the net structure.

have to be chosen in a mutually exclusive manner (consistency in a sizing option set). Second, if a cell is connected to multiple nets, its sizing options will also be included in each of the corresponding net structures. In such cases, the selected sizing options for the cell must be consistent across all these net structures (consistency across net structures). As described in Section 5.2, the net spanning structure is designed to guide flows between net structures in the SSG to satisfy the second type of consistency. To guarantee the first type of consistency, two heuristic methods are proposed that prune flows corresponding to invalid option selections when determining the mincost flow; these are discussed in Section 6.

4.2. Handling Cumulative Constraints. We define a *cumulative* metric as one, that is, the sum over all relevant circuit components (cells in our case) of a function $f(s_{i,j})$ of the chosen option (cell size in our case) $s_{i,j}$ of component/cell c_j. A cumulative constraint is then an upper-bound or lower-bound constraint on a cumulative metric. We tackle upper-bound cumulative metrics, specifically, total cell area and total leakage power in this paper.

To handle any given total cell area constraint, each net structure has an outgoing arc called the *area output arc*. A *shunting structure* as explained in Section 5.3 is present in each net structure and connects the area output arc with the option nodes in the net structure. The function of the shunting structure is that when an option node is chosen, the shunting structure diverts a flow of amount that equals the chosen size to the area output arc from the incoming flow through the option node. All these flows are gathered at the *area-gathering node* as shown in Figure 1(b) which is connected to the sink. By setting the capacity of the arc between the area gathering node and the sink to be the given total cell area limit, we make sure the total selected cell area, which is equal to the total incoming flow amount to the area-gathering node, is smaller than or equal to the given limit. The shunting structure is discussed in detail in Section 5.3. Note that, as mentioned before, the sizing option of a cell can be contained in more than one net structure; in this case, the flow amount equal to its selected area will be sent to the area output arc in only one of the net structures that contain the cell.

A leakage power constraint is handled similarly, by having a flow equal to the leakage power of a cell corresponding to its chosen size option go through the shunting structure into a *power-gathering node*, and having an arc from this node to the sink with capacity equal to the leakage power upper-bound constraint.

Algorithm FlowSize
(1) Construct a net structure as depicted in Figure 2 for each net in $\mathscr{P}_\alpha$.
(2) Determine the cost of each arc in the net structures, so that the change in timing cost is accurately incurred by flows through them.
(3) Connect net structures with net spanning structures that maintain consistency of size selection of common cells across multiple net structures; see Section 5.2.
(4) Add the shunting structure and an area output arc (Section 5.3) to each net structure to divert flow of amount equal to the selected size options (nodes) to the area output arc.
(5) Connect area output arcs to the area gathering node that has only one outgoing arc with capacity equal to the area-constraint to limit total selected cell area (= flow amount into the node).
(6) Determine a valid min-cost flow in the resulting SSG by applying the standard min-cost flow algorithm and the min-cost/max-flow heuristics (Section 6) iteratively.
(7) Select the size options chosen by the valid min-cost flow as cell sizes to obtain a near-optimal critical path delay for the circuit.

Algorithm 1: Network flow algorithm for the cell sizing problem for optimizing timing under a given area constraint.

The high-level pseudocode of our cell-sizing method FlowSize is given in Algorithm 1.

5. Further Details of the SSG

In this section, we discuss important details of the SSG, including determining arc costs and capacities, net spanning structures, and further details of the structures for constraint satisfaction.

5.1. Arc Cost Determination. To explain our arc cost formulation that accurately captures the timing cost change, we assume a lumped capacitance and resistance net delay model, which is widely used in cell sizing [2, 4, 17]. For net n_i, the delay $D(u_j, n_i)$ to a sink cell u_j of n_i is

$$D\left(u_j, n_i\right) = R_d\left(c \cdot L_{n_i} + \sum_{u_k \in \mathrm{SC}(n_i)} C_{u_k}\right), \tag{3}$$

where R_d is the driving resistance of n_i, c is the unit WL capacitance, L_{n_i} is the WL of n_i, $\mathrm{SC}(n_i)$ is the set of sink cells of n_i, and C_{u_k} is the input capacitance of cell u_k. In the pre-placement sizing stage, the WL of a net is usually estimated according to the fan-out number of the net. In the postplacement resizing stage, it can be estimated using one of several well-know models, for example, HPBB. With this delay model, for a critical net n_i with m critical sinks, the timing cost $t_c(n_i)$ of n_i is

$$t_c\left(n_i\right) = \frac{m}{S_a^\beta\left(n_i\right)} \cdot R_d\left(c \cdot L_{n_i} + \sum_{u_k \in \mathrm{SC}(n_i)} C_{u_k}\right). \tag{4}$$

In the above expression, let us denote the coefficient $m/S_a^\beta(n_i)$ as α. The parameters affected by cell-sizing options are R_d and C_{u_k}. Hence, if a term in the formula includes R_d, its value is determined by the size of u_d, and if a term includes C_{u_k}, its value is determined by the size of u_k. For example, the term $\alpha R_d \cdot C_{u_k}$ is determined by choices in sizes of both u_d and u_k, and the value change of this term $\Delta\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k})$ when the two sizing options $S^l_{u_d}$ and $S^m_{u_k}$ are chosen can be written as

$$\begin{aligned}&\Delta\alpha R_d C_{u_k}\left(S^l_{u_d}, S^m_{u_k}\right)\\ &\quad = \alpha\left[R_d\left(S^l_{u_d}\right) \cdot C_{u_k}\left(S^m_{u_k}\right) - R_d\left(S'_{u_d}\right) \cdot C_{u_k}\left(S'_{u_k}\right)\right],\end{aligned} \tag{5}$$

where $R_d(S^l_{u_d})$ is the driving resistant corresponding to the driver cell u_d with size option $S^l_{u_d}$, $C_{u_k}(S^m_{u_k})$ is the input capacitance of u_k with size option $S^m_{u_k}$, and S'_{u_d} and S'_{u_k} are the original sizes of u_d and u_k, respectively. The term $\alpha R_d \cdot c \cdot L_{n_i}$ is determined only by the size of u_d, and its value change $\Delta\alpha R_d c L_{n_i}(S^l_{u_d})$ when the option $S^l_{u_d}$ is selected is

$$\Delta\alpha R_d c L_{n_i}\left(S^l_{u_d}\right) = \alpha \cdot c \cdot L_{n_i}\left(R_d\left(S^l_{u_d}\right) - R_d\left(S'_{u_d}\right)\right). \tag{6}$$

We denote the set of arcs between S_{u_d} and S_{u_k} as $E(S_{u_d}, S_{u_k})$, where u_d is the driving cell, and u_k is a sink cell. A valid flow will pass through only one arc in each such arc set, since only one option in S_{u_d} and S_{u_k} can be meaningfully chosen. Hence, in order to make the valid flow cost equal to the change of the timing cost, the cost of an arc in an arc set is set as the sum of the changes of all terms in the timing cost function that is functions of the cell-size options represented by the arc. Furthermore, if a term in t_c is a function of only the size of one cell v, then we arbitrarily choose an arc set $E(S_v, S_w)$ among all those connected to S_v, and the term value change determined by each S^l_v is added to all arcs starting from option node S^l_v in the arc set.

Thus, the value change of term $\alpha R_d C_{u_k}$ is included in the cost of the arc set $E(S_{u_d}, S_{u_k})$. Specifically, the cost of an arc $(S^l_{u_d}, S^m_{u_k})$ in the set includes $\Delta\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k})$. The value change of term $\alpha R_d c \cdot L(n_i)$ is a function only of the driving cell size and thus is included in the cost of arcs in only one arc set $E(S_{u_d}, S_{u_j})$, where u_j is an arbitrary chosen sink cell of n_i. Specifically, the cost of an arc $(S^l_{u_d}, S^m_{u_j})$ in the set includes $\Delta\alpha R_d c L_{n_i}(S^l_{u_d})$.

Finally, the size change of a cell in a critical net also affects the timing cost of noncritical nets that are connected

to the cell. Instead of also constructing net structures for these affected noncritical nets, we use a much simpler method, which is including the timing cost changes of noncritical nets in the net structures of their connected critical nets. Let u be a cell in $\mathcal{P}_\alpha$. We have two cases with respect to u and any noncritical net (a net not in $\mathcal{P}_\alpha$) it may be connected to: (1) if u is the driver of a noncritical net n_j, the timing cost change of n_j is $(R_d(S_u^l) - R_d(S_u'))(c \cdot L_{n_j} + C_{\text{load}}(n_j))/S_a^\beta(n_j)$, where S_u^l is the chosen option of u, S_u' is the initial size of u, and $C_{\text{load}}(n_j)$ is the total load capacitance of n_j. (2) Otherwise, if u is a sink cell of a noncritical net n_j, the timing cost change of n_j is $R_d(C_u(S_u^l) - C_u(S_u'))/S_a^\beta(n_j)$. Let $\Delta t_c^u(S_u^l)$ denote the total timing cost change of all noncritical nets connected to u. If $u = u_d$ is the driving cell of the critical net n_i, $\Delta t_c^u(S_u^l)$ is included in arcs in one arc set $E(S_{u_d}, S_{u_j})$, where, again, u_j is the arbitrarily chosen sink cell of n_i. Otherwise, if $u = u_k$ is a sink cell, $\Delta t_c^u(S_u^l)$ is included in the arc set $E(S_{u_d}, S_{u_k})$. Specifically, the cost of an arc $(S_{u_d}^m, S_{u_k}^l)$ includes $\Delta t_c^u(S_u^l)$.

To sum up, for an arc $(S_{u_d}^l, S_{u_k}^m)$ in a net structure, if $u_k = u_j$ (u_j is the chosen sink in whose arc set $E(S_{u_d}, S_{u_j})$ cost, that is, costs of the arcs in this set, and we include the change in the terms in F_t that are only dependent on the cell size of driver u_d), its cost $(S_{u_d}^l, S_{u_k}^m)$ is

$$\begin{aligned}\text{cost}\left(S_{u_d}^l, S_{u_k}^m\right) = {} & \Delta\alpha R_d C_{u_k}\left(S_{u_d}^l, S_{u_k}^m\right) + \Delta\alpha R_d c L_{n_i}\left(S_{u_d}^l\right) \\ & + \Delta t_c^{u_d}\left(S_{u_d}^l\right) + \Delta t_c^{u_k}\left(S_{u_k}^m\right).\end{aligned} \tag{7}$$

Otherwise if $u_k \neq u_j$, its cost is

$$\text{cost}\left(S_{u_d}^l, S_{u_k}^m\right) = \Delta\alpha R_d C_{u_k}\left(S_{u_d}^l, S_{u_k}^m\right) + \Delta t_c^{u_k}\left(S_{u_k}^m\right). \tag{8}$$

We should note that our cost formulation is accurate under the assumption that the delay is linearly proportional to the load capacitance change. Only under such assumption, for a multiple fanout net, we can sum up the cost for the size change of each fanout to obtain the total delay change of the net. Unfortunately, in modern libraries with small feature sizes, the delay of a cell usually shows a nonlinear relationship with load capacitance. To handle this issue of a nonlinear delay function (as a function of capacitive load) in the ISPD'12 library (where the non-linearity is very pronounced), we determine in each iteration of the mincost flow computation (see Section 6), a min-square error linear approximation around the current design point.

5.2. Maintaining Consistency across Net Spanning Structures. As we mentioned before, the net spanning structure is designed to maintain consistency across net structures. Note that if multiple nets have a common cell, their net structures must be connected in the SSG by net spanning structures.

We first consider the situation in which a cell u is a sink cell of a net n_i, and the driving cell of k nets $n_{j_1}, \ldots, n_{j_k}$ ($k \geq 1$); see Figure 3(a). The size option set S_u is present in all the net structures corresponding to these connected nets. We connect these net structures by adding arcs from each option node in S_u of N_i to the equivalent option node (indicating the same size choice) in the S_u's of $N_{j_1}, \ldots, N_{j_k}$, where N_i is the net structure corresponding to net n_i. The resulting spanning structure is shown in Figure 3(b). With this structure, it is easy to see that if one size option of u is chosen by the flow through N_i, then this flow will also pass through the equivalent option nodes in $N_{j_1}, \ldots, N_{j_k}$. Thus, the required consistency is maintained.

The other situation is that the common cell u is the sink cell of more than one net $n_{i_1}, \ldots, n_{i_l}$ ($l > 1$) and the driver of at least one net n_j, as shown in Figure 3(c). In this situation, we will treat each N_{i_m} ($1 \leq m \leq l$) individually and make the connections between each N_{i_r} and N_j as stated in the first situation. However, in this case the spanning structure cannot guarantee the consistency of the option selected for u in $N_{i_1}, \ldots, N_{i_l}$. We use a *mincost* heuristic to tackle this problem; this is described in Section 6.

The costs of all arcs in the spanning structure are 0.

5.3. Structures for Cumulative Constraint Satisfaction. In this subsection we discuss further details on the structures for satisfying the cell area constraint. As described earlier in Section 4.2, a leakage power constraint is handled by a similar structure.

As we mentioned before, for each net structure, there is an *area output arc*. Once a sizing option node is chosen, a flow with an amount equal to the chosen size needs to be sent to this arc—this flow is then sent to the sink via a common *area-gathering node* and its capacity-constrained arc in order to satisfy the total cell area constraint. The simplest way to achieve this is adding to each option node an arc leaving the net structure to the area output arc with a capacity equal to the option size. Then if enough flow comes into the option node, the desired amount will be sent to the area output arc.

An example is shown in Figure 4. A flow f selects two sizing options $S_{u_d}^l$ and $S_{u_k}^m$ and incurs the corresponding cost of arc $(S_{u_d}^l, S_{u_k}^m)$ in a net structure. $A(S_u^l)$ is the size of an option S_u^l. At each of the two option nodes, a branch of flow diverges a part of f to the area output arc with amount equal to the corresponding option sizes.

However, with this structure, the amount of flow that leaves a net structure is dependent on the option selected and is thus a variable. This is not desirable for determining the capacities of arcs in the spanning structures—the capacities of arcs in a spanning structure from net structure N_i to N_j need to be a constant, that is, independent of the size choices made in N_i by the flow entering N_i. We thus need a structure to diverge a constant amount of the flow that enters N_i; this amount is $\sum_{u \in n_i} A_{\max}(S_u)$, where $A_{\max}(S_u)$ is the maximum size of options in S_u.

Our complete structure for diverting flows to the area output arc is shown in Figure 5. In the structure, the capacity of the arc leaving towards the sink (called the *leaving arc*) from each option node in an option set S_u is set to be $A_{\max}(S_u)$. Therefore, the amount of flow leaving the net structure from any option node $\in S_u$ is always $A_{\max}(S_u)$. However, not all this amount is sent to the area output arc. A *shunting node* is connected to each of these leaving arcs

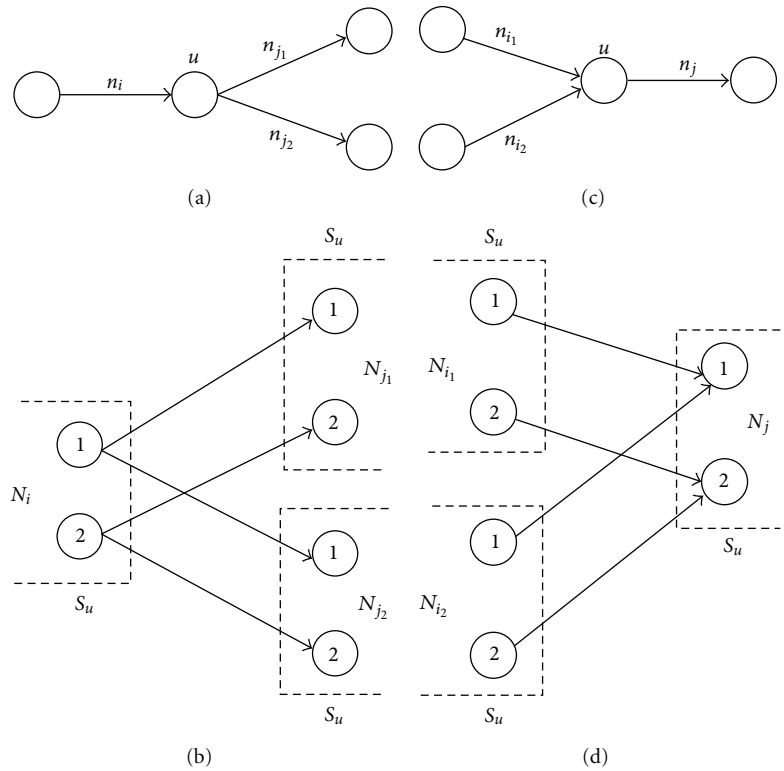

FIGURE 3: The net spanning structure. (a) The common cell u is a sink cell of one net n_i, and the driver of multiple nets n_{j_1} and n_{j_2}. (b) The corresponding spanning structure of (a), where N_k is the net structure corresponding to net n_k. (c) u is a sink cell of multiple nets n_{i_1} and n_{i_2} and the driver of n_j. (d) The corresponding spanning structure of (c).

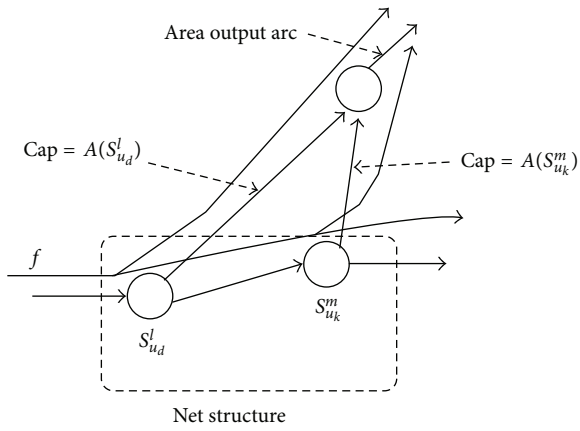

FIGURE 4: Flows to the area output arc of a net structure.

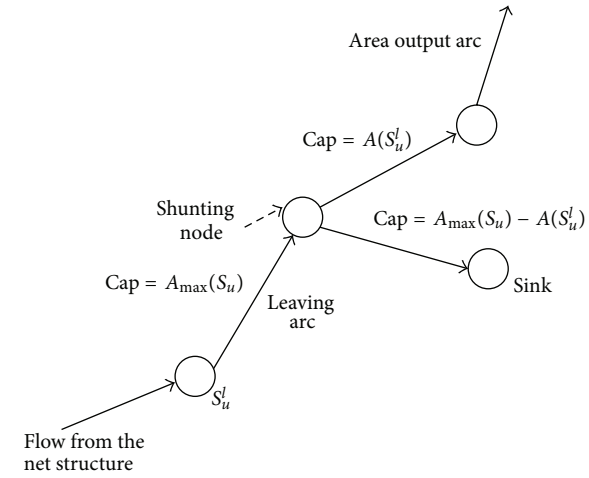

FIGURE 5: The shunting structure for outgoing flows from a net structure.

and divides the flow into two parts, one to the area output arc and the other one *shunted* (i.e., sent directly) to the sink. If the leaving arc is from an option node S^l_u, the capacity of the arc between the shunting node and the area output arc is then $A(S^l_u)$, and the capacity of the arc to the sink is $A_{\max}(S_u) - A(S^l_u)$. Therefore, if S^l_u is chosen, the amount of flow sent to the area output arc is exactly $A(S^l_u)$, and the rest of the amount $A_{\max}(S_u) - A(S^l_u)$ is shunted to the sink. In this way, we send the correct amount of flow into the area output arc of each net structure N_i and also make the total amount of flow leaving N_i to the sink be $\sum_{u \in n_i} A_{\max}(S_u)$. The costs of all arcs in this structure between an option node and the area output arc are 0.

5.4. Arc Capacity Determination. Setting proper capacities for arcs is very important for the correct functioning of the SSG. The capacities should be set such that (1) sufficient flow can be sent to each net structure in the SSG to meet

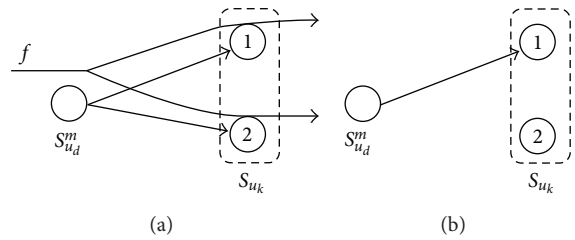

FIGURE 6: (a) Invalid flow f through two sizing options of the same cell u_k. (b) Selecting only one option of S_{u_k} for the next iteration based on some criteria of the previous "split flow."

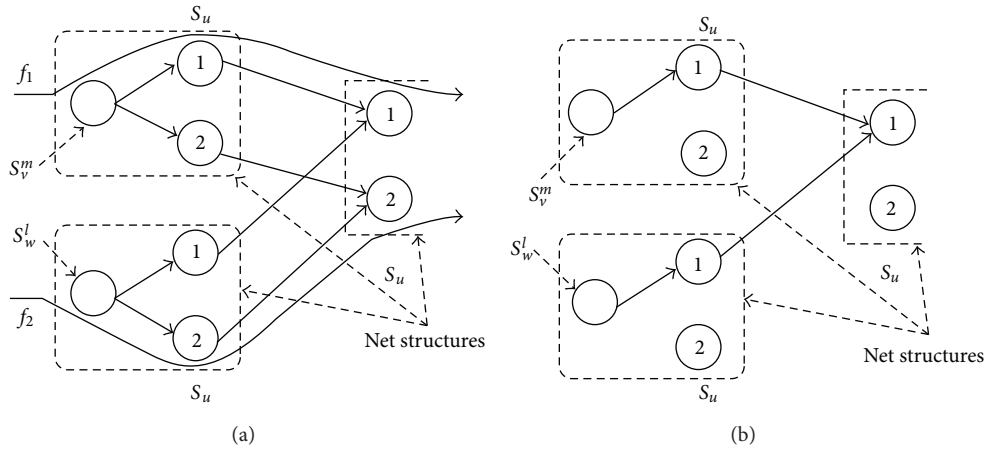

FIGURE 7: (a) Inconsistent option selections of the same cell u in different net structures. f_1 chooses option S^1_u; f_2 chooses option S^2_u. (b) The same option S^1_u of u is selected in the next iteration for all net structures contains u as a sink cell based on some criteria of the previous flow of (a).

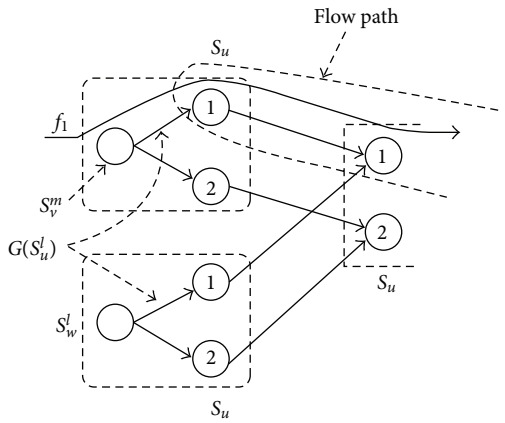

FIGURE 8: The flow path for determining the path cost of S^1_u, and the arcs for determining the arc cost of S^1_u.

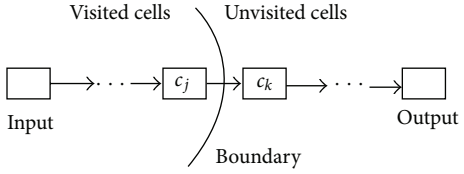

FIGURE 9: Visited cells, unvisited cells, and the boundary between them.

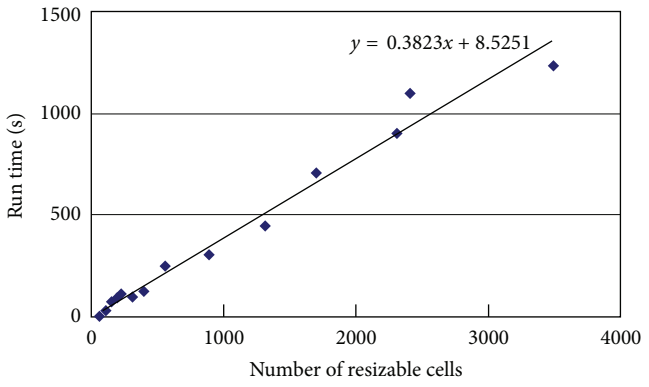

FIGURE 10: Run time versus number of resizable cells.

the flow demand on its area/power output arc; (2) within a net structure, the total incoming flow can be distributed to all sink and the driver cell option sets.

In order to determine the arc capacity, we first determine how much incoming flow is needed for each net structure. A net structure has two types of outgoing flows: (1) into the area output arc for area constraint satisfaction and (2) supply flow to its child net structures. The first type of outgoing flow has a fixed amount $\sum_{u \in n_i} A_{\max}(S_u)$ for a net structure N_i, irrespective of the chosen sizing options, as discussed in Section 5.3. The incoming flow amount must be sufficient to

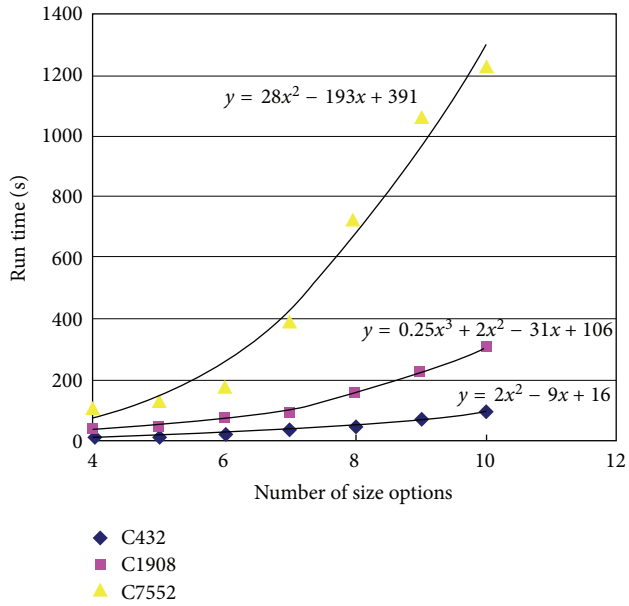

Figure 11: Run time versus number of available size options for each cell for circuits C432, C1908, and C7552. The approximation functions for the empirical time complexity of our method on these circuits are shown.

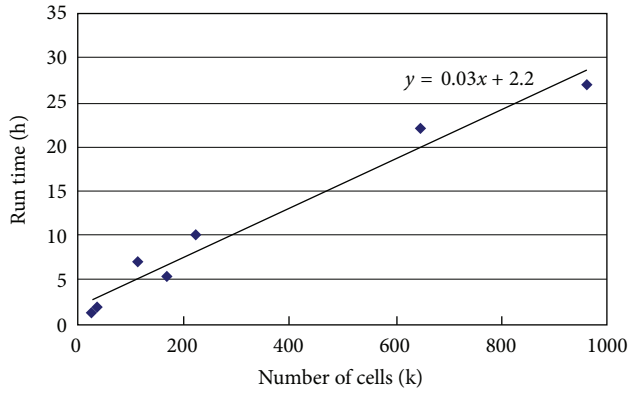

Figure 12: Run time versus number of cells for ISPD 2012 benchmark circuits.

cover the two outgoing flows, and thus the required incoming amount $f_{\text{in}}(N_i)$ of a net structure N_i is recursively given as

$$f_{\text{in}}(N_i) = \sum_{u \in n_i} A_{\max}(S_u) + \sum_{N_j \in \text{child}(N_i)} \frac{f_{\text{in}}(N_j)}{d_{\text{in}}(N_j)}$$

if N_i has any child net structure

$$f_{\text{in}}(N_i) = \sum_{u \in n_i} A_{\max}(S_u),$$

otherwise (N_i) is a "leaf" net structure, (9)

where child (N_i) is the set of child net structures of N_i, and $d_{\text{in}}(N_j)$ is the incoming degree of N_j, that is, the number of parent net structures of N_j. In (9), we assume that the required flow amount for a net structure is sent uniformly from all its parent net structures. The determination of the flow needed in each structure starts from the boundary condition of "leaf" net structures given in (9) that are directly connected to the sink. The incoming flow amount needed for these net structures is the total of their first type of outgoing flows. Starting from the leaf net structures, we visit other net structures in a reverse topological order and determine their required incoming flow amount according to the formulation in (9). The total flow $F(S)$ to be supplied from the source node S is then

$$F(S) = \sum_{N_i \text{ is a "root" net structure}} f_{\text{in}}(N_i), \quad (10)$$

where a root net structure is a net structure of a net that is driven by an I/O cell in the circuit and thus has no "parent" net in the circuit; in the SSG, the parent of all root net structures is S.

After obtaining f_{in} for each net structure, we can determine its arc capacities as follows.

(i) For an arc in the net spanning structure from N_i to N_j, its capacity is $f_{\text{in}}(N_j)/d_{\text{in}}(N_j)$, which equals the flow amount sent from N_i to N_j according to (9). This makes the total incoming flow amount to a net structure N_i exactly $f_{\text{in}}(N_i)$. As mentioned before, the cost of this arc is 0.

(ii) Within a net structure, the capacities of arcs in each arc set $E(S_{u_d}, S_{u_k})$ are the same as is derived below; u_d is the driving cell and u_k is any sink cell of the corresponding net.

For an arc set $E(S_{u_d}, S_{u_k})$, if S_{u_k} is not connected to any arc in the outgoing spanning structure of N_i, the capacity of each arc in it is set as $A_{\max}(S_{u_k})$, so that sufficient flow can be sent to the leaving arc for constraint satisfaction. Otherwise, let $N_{j_1}, \ldots, N_{j_t}$ be the child net structures of N_i that S_{u_k} is connected to (via spanning structures). Note that this means that u_k is a common cell in nets n_i and $n_{j_1}, \ldots, n_{j_t}$. Then, the capacity of each arc in the set is set to be $A_{\max}(S_{u_k}) + \sum_{r=1}^{t} f_{\text{in}}(N_{j_r})/d_{\text{in}}(N_{j_r})$.

(a) *Unit Flow Arc Cost.* When we gave the costs of arcs in a net structure in Section 5.1, we assumed that any flow on the arc will incur the cost. However, in a standard network flow graph, the cost of a flow on an arc is determined as the flow amount multiplied by the unit flow cost of the arc. With the above capacity assignment, a valid flow will always be a full flow on each arc it passes through in a net structure, since the summation of the arc capacity of a single arc in each arc set is equal to the incoming flow amount, and a valid flow uses exactly one arc in each arc set. In order to incur the same cost for a valid flow as determined in Section 5.1, the unit flow cost $C(S^l_{u_d}, S^m_{u_k})$ of an arc $(S^l_{u_d}, S^m_{u_k})$ is set to be the corresponding arc cost given in Section 5.1 divided by its capacity as determined above, that is,

$$C\left(S^l_{u_d}, S^m_{u_k}\right) = \frac{\text{cost}\left(S^l_{u_d}, S^m_{u_k}\right)}{\text{cap}\left(S^l_{u_d}, S^m_{u_k}\right)}. \quad (11)$$

6. Finding a Valid (Discretized) Mincost Flow

The standard mincost network flow solves a linear programming problem (a continuous optimization method). Hence, it cannot automatically handle the consistency (mutual exclusiveness) requirement in a size option set for a particular cell, which may result in an invalid mincost flow for size selection. Therefore, after one iteration of a mincost network flow process, in a net structure, the obtained mincost flow may pass through an option node $S^m_{u_d}$ for the driving cell u_d and then to two or more sizing options in S_{u_k} for a sink cell u_k as shown in Figure 6(a). Furthermore, as explained in Section 5.2, the resulting flow may also violate the consistency requirement for the size option selection across net structures that have a common sink cell.

In the above two cases, we will start a new iteration of the network flow process by pruning out some options that lead to an invalid flow based on certain criteria of the flow so that a near-optimal size selection is obtained. In the new iteration, for the first case, $S^m_{u_d}$ will only connect to one of the option nodes in S_{u_k} that had flow through them in the first iteration as shown in Figure 6(b); the selection criterion is discussed shortly. Similarly for the second case, in all net structures whose corresponding nets have u as a sink cell, the chosen driving cell options in the first iteration, for example, S^m_v and S^l_w as shown in Figure 7(b), will only connect to the same sizing option of u selected from those that had flow through them in the first iteration. In this way, the same invalid flow will not occur in the second iteration.

We have used two alternative selection heuristics to choose a good option node from an illegal selection in each option set S_u to be part of the new iteration.

(i) Max-flow heuristic: always choose the option node with the largest flow amount through it.

(ii) Mincost heuristic: for the first situation, starting from $S^m_{u_d}$, follow the path of each branch of the mincost flow up to a length of l, and choose the option node that is on the branch that has the mincost path.

For the second situation, for each currently selected option of u, the cost that we use to determine whether it is a good option consists of two parts, output path cost and incoming arc cost. As shown in Figure 8, similar to the first situation, the outgoing path cost of an option S^l_u of u is the cost of the flow path starting from the option node. The incoming arc cost of an option S^l_u of u is the total cost of the set of incoming arcs $G(S^l_u)$ to all S^l_u's across all net structures that contain the option set S_u; see Figure 8. The summation of these two costs is a good estimation of the cost of a valid flow that chooses only option S^l_u for cell u. Hence, we choose the option with smallest summation of the path cost and the arc cost. Note that due to run time consideration we cannot always follow each branch flow to the sink. We thus set a limit l (in number of net structures) on the length of paths we follow.

The percentage timing improvement of four representative circuits in Table 1 for the ISCAS85 benchmarks reveals that the mincost heuristics with path length limits of 2, 3, 4, and 5 perform consistently better than the max-flow heuristic and have a relatively better performance in the range of 24–39%. The mincost heuristic is thus implemented in our algorithms, and we set $l = 3$ for a balance between computational complexity and accuracy.

Table 1: Percentage timing improvements of max-flow heuristic and min-cost heuristic with different path length limits.

Ckt	Max flow	Min cost of length			
		2	3	4	5
	%ΔT	%ΔT	%ΔT	%ΔT	%ΔT
C432	8.4	10.0	11.9	12.2	12.2
C1355	4.2	6.0	6.8	6.9	7.2
C3540	12.9	16.1	16.8	17.0	17.1
C7552	8.3	9.3	9.9	10.3	10.5
Average	8.4	10.4	11.4	11.6	11.7

6.1. Time Complexity of FlowSize. It is easy to see that, with the two pruning heuristics, if the option selection for a cell is inconsistent according to the mincost flow obtained in one iteration, in the following iterations the mincost flow will select a valid size option for the cell. Thus, the number of iterations required to reach a valid mincost flow is no more than the total number of cells in $\mathcal{P}_\alpha$.

In each iteration, we use the network simplex algorithm to solve the mincost flow. Given a graph with m arcs, if the capacities and costs of arcs are all integers, with U being the maximum arc capacity and C being the maximum arc cost, then the time complexity of the network simplex method is $O(Um^2 \log C)$ [18]; however, it is well known that the average-case run time of the simplex method is much lower than this worst-case complexity [19]. As described in Section 5.4, the capacities of arcs in our SSG, being the summation of cell sizes, are integers (note that cell sizes in a standard cell design are integer in the unit of the technique feature size of the library). On the other hand, while our cost is not integer, it can be converted to integer values by proper scaling. Hence though we do not actually do the scaling, we use this assumption here in order to derive an upper bound on the time complexity of our algorithm.

Let us first consider the total number of arcs in our SSG. It is dependent on the number of cells N, the number of nets M in the circuit, and the number of available sizes for each cell S. There are three types of arcs in our SSG: arcs between size option sets, arcs in net spanning structures, and arcs in shunting structures. The number of arcs between two size option sets is S^2, and in each net structure there are $d - 1$ sets of such arcs, where d is the degree of the corresponding net. Thus, the total number of these arcs in the SSG is $S^2(d_{\text{avg}} - 1)M$, where d_{avg} is the average degree of nets in a circuit. Since d_{avg} is usually no more than 4, the total number of arcs between size option sets is $O(S^2M)$. The number of arcs in the shunting structure for each option node is three, and hence the total number is $3NS$. The net spanning structure only connects size option sets for the same cell in multiple net structures, and the number of arcs between two of such size

option sets is S. Since the total number of size option sets is $O(N)$, the total number of arcs in the net spanning structure is thus $O(NS)$. To sum up, the total number of arcs in the SSG is $O(S^2M + NS)$.

The maximum arc capacity is equal to the total cell size when all cells are chosen to be at their maximum width and thus is $O(N)$. The maximum arc cost is less than or equal to the maximum net delay. The delay of a net is dependent on the driving resistance of the driving cell, input capacitances of the sink cells, and the net fanout. Since the driving resistances and input capacitances of cells are constants specified by the library and the average fanout is usually a small constant in a VLSI circuit, C can be viewed as a constant.

Typically in a real circuit, N and M are about the same. Hence, the number of arcs in our SSG can be rewritten as $O(S^2N)$. Therefore, the time complexity of each iteration is $O(S^4N^3)$, and the total time complexity is then $O(S^4N^4)$. The polynomially bounded time complexity is a highlight of our algorithm FlowSize, since other discrete cell sizing methods such as [3, 5] are not polynomially bounded in run time. Also, as we show in Section 8 (Figure 10), its actual run time reveals a much smaller complexity, that is, in keeping with the much smaller average-case run time of the network simplex algorithm compared to its worst-case complexity. Furthermore, our experiments show that the actual number of iterations needed in FlowSize is much less than the number of cells in $\mathscr{P}_\alpha$, for example, eight iterations for a circuit with 300 cells in $\mathscr{P}_\alpha$.

7. An Optimal Dynamic Programming Algorithm

Since we do not have the library or library parameters used in [3], we cannot directly compare our results for the benchmarks they use (ISCAS'85) to their published results for those benchmarks. Thus, we implemented an optimal dynamic programming (DP) method that can produce optimal solutions for the sizing problem of cells in $\mathscr{P}_\alpha$ and compared our solution quality to the optimal one.

In the DP algorithm we propose three pruning methods to reduce the number of partial solutions generated in the DP process that can maintain the optimality of the solution but greatly reduce the run time. We process cells in circuit topological order (from driver cells to sink cells). Each time we process a cell, a new set of partial solutions that involve the cell is generated by combining all partial solutions we have for previously processed cells with possible size choices of the cell. The pruning happens after new partial solutions are generated.

We propose three pruning conditions. A partial solution is pruned when (1) it fails to meet the area constraint; (2) it gives longer delay than the critical path delay produced by our method; (3) there is another better partial solution (generated in the search process or extracted from the complete solution of our method) that gives smaller total area, and better arrival time at the outputs of cells on the boundary of the visited region (connected to unvisited cells as shown in Figure 9) in both of the following cases: (a) the unvisited cells are all at their maximum sizes or (b) the unvisited cells are all at their minimum sizes.

The first two pruning methods obviously do not change the optimality of the exhaustive search method. For the third condition, let us first denote the arrival time at the output of a cell c_i as $A_o(c_i)$. Figure 9 shows a single path situation. c_j is the visited cell at the boundary of the visited region, and c_k is the unvisited cell connected to it. We have two partial solutions τ and τ', and τ' is a better solution according to our third pruning condition. Then for any complete solution τ_{comp} expanded from τ, we can also expand τ' to τ'_{comp} by choosing exactly the same sizes for unvisited cells in τ' as in τ_{comp}. Since τ' has smaller area than τ, if τ_{comp} meets the constraints, so does τ'_{comp}. Then the total delay at the output of c_j will be the same for both complete solutions. Since τ' is better than τ, we have $A_o^{\tau}(c_j) > A_o^{\tau'}(c_j)$, where $A_o^{\tau}(c_j)$ is the $A_o(c_j)$ value according to partial solution τ, and $A_o^{\tau'}(c_j)$ is the $A_o(c_j)$ value according to partial solution τ'. Hence, the total delay of the path for τ_{comp} > the total delay for τ'_{comp}. Therefore, τ cannot be expanded to an optimal solution. Thus, our third pruning method also does not negatively affect optimality of the method.

8. Experimental Results

We tested our algorithm on two sets of benchmarks, the ISCAS'85 suite, and the ISPD 2012 suite [21]. For the ISCAS'85 benchmark suite, we used two different libraries, a 0.18 μm (180 nm) library and Synopsys's 90 nm library. We use the same industrial 0.18 μm standard cell library as in [22], which provides four cell implementations for each function with different areas, driving resistances, input capacities, and intrinsic delays. The interval between the four available sizes $w_1 < w_2 < w_3 < w_4$ for each cell is increased about exponentially, that is, $(w_4 - w_3) \approx 2(w_3 - w - 2) \approx 4(w_2 - w_1)$. Other electrical parameters we use are unit length interconnect resistance $r = 7.6 \times 10^{-2}$ ohm/μm and unit length interconnect capacitance $c = 118 \times 10^{-18}$ f/μm. For ISPD 2012 benchmark, it has its own artificial library, which has high nonlinear dependency between delay and load capacitance. Results were obtained on Pentium IV machines with 1 GB of main memory for ISCAS'85 benchmark and Xeon machine with 72 G memory for ISPD 2012 benchmark. Competing methods (the optimal DP method of Section 7 for the ISCAS'85 benchmarks and an approximate DP method [3] for the ISPD'12 benchmarks) were also run on the respective machines.

The ISCAS'85 benchmarks come with initial sizing solutions. We ran our algorithm with a 10% total cell area increase constraint, which means that the total cell area after cell sizing cannot increase more than 10% from the initial solution. The improvements compared to the initial solution obtained by our net work flow method and the optimal dynamic programming- (DP-) based exhaustive search method are listed in Table 2. Compared to the optimal solution from DP, the improvement obtained by our method is only 1% worse

Table 2: Results of our method and the optimal DP method. Four sizes are available for each cell. $\%\Delta T$ is the percentage timing improvement, $\%\Delta A$ is the percentage change of total cell area (negative value means deterioration).

Ckt	Number of cells	Number of crit. cells	DNF				Opt. DP			
			Delay (ns)	$\%\Delta T$	$\%\Delta A$	Runtime (sec)	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (sec)
C432	160	50	3.2	11.9	−9.9	9	3.1	13.2	−10.0	103
C499	202	51	3.5	11.9	−9.4	14	3.5	12.4	−9.3	119
C880	383	77	3.5	14.8	−9.9	19	3.4	16.2	−10.0	195
C1355	544	85	4.5	6.8	−8.1	22	4.4	7.0	−8.8	489
C1908	880	88	5.5	16.1	−9.8	39	5.4	17.0	−10.0	556
C2670	1.3 K	91	3.4	9.6	−9.5	38	3.4	11.1	−9.7	908
C3540	1.7 K	124	5.0	16.8	−9.0	69	4.8	19.0	−9.4	1724
C5315	2.3 K	138	5.3	10.0	−6.9	70	5.3	10.2	−7.9	3228
C6288	2.4 K	299	14.1	11.5	−8.1	97	14.1	11.7	−9.1	14998
C7552	3.5 K	199	7.3	9.9	−8.2	112	7.2	11.5	−7.5	6847
Average	1336	120		11.9	−8.9	48		12.9	−9.2	2916

Table 3: Results of our method and the optimal DP method with Synopsys 90 nm library [20].

Ckt	DNF				Opt. DP			
	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (secs)	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (secs)
C432	1.7	8.5	−9.1	10	1.7	11.7	−9.9	92
C499	1.0	12.8	−9.9	15	1.0	13.2	−9.9	144
C880	1.5	10.5	−9.8	17	1.4	14.7	−10.0	148
C1355	1.0	7.0	−8.8	40	1.0	8.1	−9.0	705
C1908	1.6	12.1	−10.0	72	1.5	16.2	−10.0	1024
C2670	1.3	9.4	−10.0	44	1.3	13.1	−10.0	884
C3540	2.2	13.5	−10.0	105	2.1	17.9	−10.0	1650
C5315	1.6	9.9	−9.4	280	1.5	11.8	−9.5	8410
C6288	6.0	14.2	−10.0	312	5.8	18.2	−10.0	25689
C7552	1.7	10.2	−9.8	364	1.7	13.8	−9.8	13204
Average		10.9	−9.7	126		13.9	−9.8	5195

(11.9% versus 12.9%); note that the solutions of both methods satisfy the 10% area increase constraint. Furthermore, our run time is 60X less than that of the exhaustive search method even with all three pruning conditions. Note again that we cannot compare our technique directly to that of [3], since we do not have their cell library or parameters.

We have also tested our algorithm using the more advanced and industry-like 90 nm library from Synopsys [20]. The results are given in Table 3. A similar trend to that of Table 2 is obtained is with the same initial sizing and 10% area relaxation, our method is only 3% worse (10.9% versus 13.9%) than the optimal solution. The run time we use is 40X less than the optimal one on average. We observe that there is a slightly increase in the optimality gap (from 1% to 3%) for 90 nm library compared to the 180 nm library. Our conjecture is that this increase is mainly due to the increased sensitivity in the 90 nm library; that is, for the same amount of cell size change, the relative delay change in the 90 nm library can be about 40% more than that of 180 nm library. Thus small errors in the optimal choice of cell sizes in near-optimal methods such as ours can lead to somewhat larger errors in the circuit delay results for libraries with larger sensitivity. However, we should note that, even with such large sensitivity increase as mentioned above, our optimality gap increases by only 2%.

To show the scalability of our algorithm, two additional experiments were performed with the 180 nm library. In the first one, the sizes of all cells were considered for resizing rather than only cells in $\mathcal{P}_\alpha$. The results are listed in Table 4(a). Compared to only focusing on $\mathcal{P}_\alpha$, the average number of cells that are resizable is increased by 10 times from 120 to 1336, the run time is also increased by about 10 times from 48 secs to 525 secs, while the timing improvement % is an absolute of 2% better. The run time plot with respect to the number of resizable cells is shown in Figure 10, which best fits a linear function. This is much lower than the worst-case complexity we derived in Section 6, and in keeping with the well-known much smaller empirical time complexity of the network simplex method compared to its worst-case complexity [19].

In the second experiment, we expanded the cell library by adding six artificial size options with proportional driving resistances and input capacitances for each cell (three size options are added between w_3 and w_4 with uniform spacing between them, and the other three added options are made

Table 4: (a) Results of our method when all circuit cells are considered for resizing with four sizes for each cell. (b) Results of our method when ten size options available for each cell; only cells in $\mathscr{P}_\alpha$ are resizable.

(a)

ckt	%ΔT	%ΔA	Run time (secs)
C432	12.9	−9.9	95
C499	13.5	−9.7	97
C880	14.9	−9.8	128
C1355	9.4	−9.5	249
C1908	19.9	−9.7	309
C2670	9.7	−9.4	449
C3540	22.6	−9.9	698
C5315	10.0	−6.9	901
C6288	13.2	−8.9	1097
C7552	12.1	−8.5	1229
Average	13.9	−9.2	525

(b)

ckt	%ΔT	%ΔA	Run time (secs)
C432	14.8	−9.7	72
C499	16.6	−9.8	140
C880	14.9	−9.2	144
C1355	13.3	−9.9	208
C1908	19.9	−9.1	344
C2670	12.7	−9.8	315
C3540	23.1	−9.7	527
C5315	10.8	−8.5	476
C6288	15.0	−9.5	698
C7552	14.9	−8.9	1087
Average	15.6	−9.4	401

larger than w_4. The intervals between the last three newly added size options are the same as between the first three newly added size options. We use linear approximation determined from the four options provided in the original library to calculate the driving resistances and input capacitances of the added options), so that each cell has ten different size options. The results with this larger library and cells in $\mathscr{P}_\alpha$ being resizable are shown in Table 4(b). With the larger library, we can only obtain the optimal results for the two smallest circuits using the exhaustive search method. Our results are only 2.5% worse (14.8% versus 17.3%) for circuit C432, and 2.0% worse (16.8% versus 18.8%) for C499 compared to the optimal solutions. The run times of our method are about 80X less than the exhaustive search method for C432 (72 secs versus 5343 secs) and over 135X less for C499 (140 secs versus 18993 secs). Compared to the four-option results in Table 2, our DNF method's run time is increased by about 7 times, while the timing improvement % is increased by an absolute of 3.7%. We also plot in Figure 11 the run time with respect to the number S of available size options for each cell for three representative circuits C432, C1908, and C7552. The plot for C1908 best matches a cubic function of S, and the plots for C432 and C7552 best match quadratic functions, which are all consistent with the upper bound time complexity derivation given in Section 6 (that the run time is proportional to S^4). However, since, in current VLSI circuits, S and even S^4 are generally much smaller than N, the number of cells being re-sized, the dominant run time function is the one shown in Figure 10 that is linear in N.

Finally, we ran our DNF method on the complex ISPD 2012 contest benchmarks [21]. The sizes of the benchmarks range from 25 K to 959 K cells. For each type of cell, there are 30 different "sizes" (the sizes are combinations of actual sizes and different threshold voltages) in the library. For each circuit, the power constraint is determined by power optimizing the fast version of each ISPD benchmark using an internal tool (this tool uses the more complex DNF formulation modeled as a fixed-charge network flow problem that was alluded to in Section 2) that was part of the ISPD'12 competition and was in the top 6 (out of 17 teams) for 6 of the circuits; see [23]. Then, we perform our timing-driven sizing under this leakage power constraint. We run three rounds of our DNF-based sizing. In the first round, all cells in the circuit are sizable; this is needed as the ISPD'12 circuits also come with an initial sizing, and these correspond to very high delays and low power designs. In the second round, we perform further improvement by focusing on sizing only cells on critical and near critical paths (with delay ≥90% of the max path delay); note that this is the only round we do for the ISCAS'85 circuits. In the last round, we perform final adjustment, again using the DNF method, to only sub-paths in the critical paths that show delay reduction potential. The delay reduction potential is measured by the differences in the delay of the size selection s chosen in the second round, and the adjacent sizes of s, and the ratio of the corresponding power increases to the current positive leakage power slack of the circuit.

Both the circuit sizes of the ISPD'12 benchmark and the number of sizing options per cell are far beyond the capabilities of the optimal DP method (it runs out of memory for the smallest circuit). We thus compared our method to a state-of-the-art cell-sizing method in [3] (implemented by us), which also uses an approximate dynamic programming (DP)—it has a nonoptimal similarity-based partial solution pruning that can significantly reduce the number of partial solution generated. The results are shown in Table 5. As we can see, we achieve almost the same delay quality as the DP-based method (only 0.9% above it on the average) but use less than half of its run time. This highlights the efficiency of our method in achieving high quality results. We also show the run time plot with respect to the number of cells in Figure 12. Again, a linear scalability with respect to the number of cells is seen for our DNF method.

9. Conclusions

We presented a novel and efficient timing-driven network flow-based cell-sizing algorithm. We developed a size option selection graph, in which cell size options are modeled as nodes, and the cost of flows passing through various nodes is equal to the change in the timing objective function when the cell sizes corresponding to these nodes are chosen. Thus,

Table 5: Results of our DNF method and the approximate DP method of [3] for the ISPD 2012 cell-sizing benchmark suite. The "% Delay Impr." column shows the percentage difference of the delay between ours and the Approx. DP method (positive value indicates our method is better, negative indicates that the Approx. DP method is better).

Ckt	Number of of cells	Power Con-str. (w)	DNF				Aprrox. DP		
			Delay (ps)	Power (w)	Run time (h)	% Delay Impr.	Delay (ps)	Power (w)	Run time (h)
DMA	25 k	1.2	770	1.2	1.3	−0.7	765	1.2	2.8
pci_br32	33 k	0.9	676	0.8	2.0	−0.9	670	0.9	3.5
des_perf	111 k	7	795	7	7.1	−0.9	788	7	12.4
vga_lcd	165 k	0.7	673	0.7	5.6	0.6	677	0.7	14
b19	219 k	2.3	2126	2.3	10	−2	2084	2.3	19
leon	649 k	2.1	1632	2.1	22.2	−0.9	1616	2.1	48.4
Netcard	959 k	2.5	2218	2.5	27.2	−1.4	2186	2.5	58
Average					10.7	−0.9			22.6

by solving for a mincost flow, we can determine the cell sizes that can optimize the circuit delay. Various techniques are proposed to ensure that we can obtain, from the continuous optimization of standard mincost flow, a valid "discrete" mincost flow that meets the discrete mutual exclusiveness condition of cell-size selection. Area and leakage power constraint satisfaction is also taken care of by special network flow structures. The results show that the timing improvement obtained using our method is near optimal for ISCAS'85 benchmarks and similar to a state-of-the-art method for the ISPD'12 benchmarks (the near optimality could not be determined for these circuits as the optimal DP method ran out of memory for the smallest circuit) while being more than twice as fast as that technique. Furthermore, our technique scales well with problem size since its worst-case time complexity is polynomially bounded, and the empirical time complexity is linear.

Acknowledgment

This work was supported by NSF Grants CCF-0811855 and CCR-0204097.

References

[1] C. P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.

[2] J. Fishburn and A. Dunlop, "Tilos: a posynomial programming approach to transistor sizing," in *Proceedings of International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[3] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 847–852, June 2007.

[4] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '98)*, pp. 545–550, November 1998.

[5] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 465–472, 1997.

[6] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '11)*, pp. 724–731, November 2011.

[7] S. Dutt and H. Ren, "Discretized network flow techniques for timing and wire-length driven incremental placement with white-space satisfaction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1277–1290, 2011.

[8] H. Ren and S. Dutt, "A provably high-probability white-space satisfaction algorithm with good performance for standard-cell detailed placement," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1291–1304, 2011.

[9] S. Dutt, H. Ren, F. Yuan, and V. Suthar, "A network-flow approach to timing-driven incremental placement for ASICs," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '06)*, pp. 375–382, November 2006.

[10] U. Brenner, "VLSI legalization with minimum perturbation by iterative augmentation," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE '12)*, pp. 1385–1390, March 2012.

[11] H. Ren and S. Dutt, "A network-flow based cell sizing algorithm," in *Proceedings of the 17th International Workshop on Logic & Synthesis*, pp. 7–14, 2008.

[12] S. Dutt and H. Ren, "Timing yield optimization via discrete gate sizing using globally-informed delay PDFs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '10)*, pp. 570–577, November 2010.

[13] H. Ren and S. Dutt, "Effective power optimization under timing and voltage-island constraints via simultaneous VDD, Vth assignments, gate sizing, and placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 746–759, 2011.

[14] H. Ren and S. Dutt, "Algorithms for simultaneous consideration of multiple physical synthesis transforms for timing closure," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '08)*, November 2008.

[15] A. Nahapetyan and P. M. Pardalos, "A bilinear relaxation based algorithm for concave piecewise linear network flow problems," *Journal of Industrial and Management Optimization*, vol. 3, no. 1, pp. 71–85, 2007.

[16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, chapter 10-11, Prentice-Hall, Upper Saddle River, NJ, USA, 1993.

[17] D. Kim and P. Pardalos, "Gate sizing in MOS digital circuits with linear programming," in *Proceedings of the Conference on European Design Automation (EURO-DAC '90)*, pp. 217–221, 1990.

[18] R. K. Ahuja and J. B. Orlin, "Scaling network simplex algorithm," *Operations Research*, vol. 40, supplement 1, pp. S5–S13, 1992.

[19] I. Adler and N. Megiddo, "A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension," *Journal of the ACM*, vol. 32, no. 4, pp. 871–895, 1985.

[20] Synopsys 90 *nm* Library, http://www.synopsys.com/community/universityprogram/pages/library.aspx.

[21] ISPD 2012 cell sizing contest, http://www.ispd.cc/contests/12/ispd2012_contest.html.

[22] X. Yang, B. K. Choi, and M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint generation," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '02)*, pp. 177–180, November 2002.

[23] http://www.ispd.cc/contests/12/ISPD_2012_Contest_Results.pdf.