A Network-Flow Based Cell Sizing Algorithm

Huan Ren and Shantanu Dutt
Dept. of ECE, University of Illinois-Chicago

***Abstract*—We propose a timing-driven discrete cell-sizing algorithm that can incorporate total cell size constraints. We model cell sizing as a min-cost network flow problem. In the network flow graph, 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 change with the chosen sizes of cells. Compared to other discrete optimization methods for cell sizing, our method can obtain a near-optimal solution in a very time-efficient manner. We tested our algorithm on the ISCAS'85 benchmark, and compared our results with an optimal solution produced by an exhaustive search method with exponential time complexity. The results show that given the same initial sizing, the improvement obtained by our method is only 1% worse (11.9% v.s. 12.9%) than the optimal solution, while satisfying a given total cell area constraint. Furthermore, our method is 60 times faster than the optimal method.**

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, i.e., *cell sizing*, is a very effective approach to improve timing.

The cell sizing problem has been studied for a long time. Many methods [5], [8] assume the availability of a continuous range of cell sizes, i.e., 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", e.g., geometrically spaced instead of uniformly spaced [9]. 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 [9].

On the other hand, few time-efficient methods are know that can directly handle timing optimization with discrete cell sizes, since this problem is NP-complete. The technique in [6] uses multi-dimensional 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 [9], a new rounding method is developed based on an initial continuous solution. Instead of only rounding to the nearest available size, it 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 [6].

This work was supported by NSF grant CCR-0204097.

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 that have flows through them are chosen. Hence, by solving a min-cost flow in the network graph, near-optimal cell sizing can be determined by choosing cell sizes whose corresponding nodes have the min-cost 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 min-cost 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 min-cost flow may pass through two sizing options for the same cell. We solve such problems by using *min cost* or *max flow* selection heuristics.

The rest of the paper is organized as follows. Section 2 provides an overview of our method. A general view of our size selection network graph (*SSG*) is presented in Sec. 3. In Secs. 4-7, we discuss various issues of the SSG. In Sec. 8, we show how to obtain a valid min-cost flow in the SSG. Section 9 briefly describes an optimal exhaustive search method to which we compare our network-flow based technique. Section 10 presents experimental results and we conclude in Sec. 11.

2. 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 $\mathcal{P}_\alpha$. 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 $CS(n_j)$ to be the set of critical sinks of a net n_j, which are: 1) all sinks in $\mathcal{P}_\alpha$ if the net is in $\mathcal{P}_\alpha$, or 2) the sink with the minimum slack otherwise. Our method tries

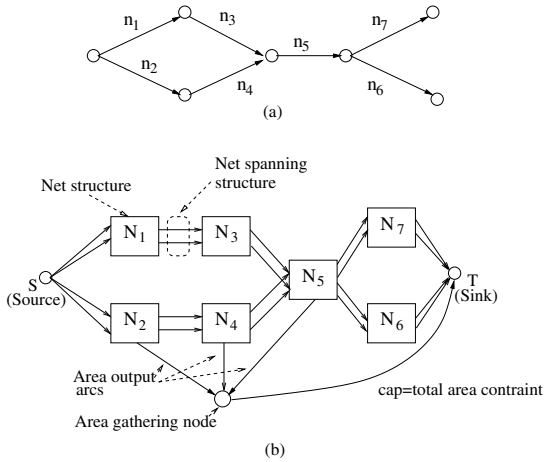

Fig. 1. *(a) Nets in P_α of a circuit. (b) The corresponding SSG, which includes net structures and net spanning structures. N_i is the net structure corresponding to net n_i. Each net structure will send a flow of amount equal to the total cell size it chooses through the area output arc to the area 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.*

to improve the critical path delay by minimizing the objective function proposed in [7]. A timing cost $t_c(n_i)$ of a net n_i is defined as [7]:

$$t_c(n_i) = \sum_{u_j \in CS(n_i)} D(u_j, n_i)/S_a^{\beta}(n_i) \quad (1)$$

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 non-critical paths. Based on experimental results, $\beta = 1$ works best in this scenario, since only nets in P_α and those connected to it are considered in the optimization function (see below). Let $\hat{P_\alpha}$ be the set of nets that are either in P_α or connected to nodes in it. The timing objective function F_t is the summation of $t_c(n_i)$ of all nets in $\hat{P_\alpha}$ given as:

$$F_t = \sum_{n_i \in \hat{P_\alpha}} t_c(n_i) \quad (2)$$

Here we only consider nets in $\hat{P_\alpha}$, 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.

3. The Size Selection Graph (SSG)

We model the timing-minimization cell size selection problem as a min-cost 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

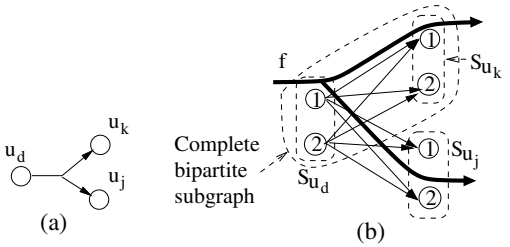

Fig. 2. *(a) A net in critical paths. (b) The net structure corresponding to the net; f is a flow through the net structure.*

to one sizing option. We use flows in the SSG to model cell size selection, i.e., 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 P_α. 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 (Eqn. 2) corresponding to the sizing scheme selected by the flow, then the min-cost 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 min-cost flow in a graph for which several efficient algorithms such as the network simplex algorithm and the enhanced scaling algorithm [2] are available. The general structure of our SSG is described below.

A. 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, i.e., first a mini network flow graph called a *net structure* is constructed for each net in $\mathcal{P}_\alpha$, and then net structures of connected nets are connected by *net spanning structures* to form the complete SSG as shown in Fig. 1. Thus, the SSG has a similar topology as the paths in $\mathcal{P}_\alpha$. 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 l'th 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 Fig. 2. The net has one driving cell u_d and two critical sink cells u_j and u_k; see Fig. 2(a). The corresponding net structure is shown in Fig. 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 min-cost flow in the SSG to an optimal sizing choice. These are:

(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 Sec. 4.

(2) The flow that is determined must be a valid flow, i.e., 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 has 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 Sec. 5, 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 min-cost flow; these are discussed in Sec. 8.

B. Handling the Area Constraint

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 Sec. 6 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 will diverts a flow of amount that equals to the chosen size to the area output arc from the incoming flow to the option node. All these flows are gathered at the *area gathering node* as shown in Fig. 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 Sec. 6. 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 contains the cell.

The high-level pseudo code of our cell sizing method FlowSize is given in Fig. 3.

Algorithm FlowSize

1. Construct a net structure as depicted in Fig. 2 for each net in $\mathcal{P}_\alpha$.
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 Sec. 5.
4. Add the shunting structure and an area output arc (Sec. 6) 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 (Sec. 8) 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.

Fig. 3. *Network flow algorithm for the cell sizing problem for optimizing timing under a given area constraint.*

4. 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 [4], [8], [10]. For net n_i, the delay $D(u_j,n_i)$ to a sink cell u_j of n_i is:

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

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, $SC(n_i)$ 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 post-placement resizing stage, it can be estimated using one of several well know models, e.g., 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:

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

In the above expression, let us denote the coefficient $\frac{m}{S_a^\beta(n_i)}$ 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:

$$\Delta\,\alpha R_d C_{u_k}(S^l_{u_d},S^m_{u_k}) = \alpha[R_d(S^l_{u_d})\cdot C_{u_k}(S^m_{u_k}) - R_d(S'_{u_d})\cdot C_{u_k}(S'_{u_k})]$$

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:

$$\Delta\,\alpha R_d c L_{n_i}(S^l_{u_d}) = \alpha\cdot c\cdot L_{n_i}(R_d(S^l_{u_d}) - R_d(S'_{u_d}))$$

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 are 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 non-critical nets that are connected to the cell. Instead of also constructing net structures for these affected non-critical nets, we use a much simpler method, which is including the timing cost changes of non-critical nets in the net structures of their connected critical nets. Let u be a cell in P_α. We have two cases w.r.t. u and any non-critical net (a net not in P_α) it may be connected to:

(1) If u is the driver of a non-critical net n_j, the timing cost change of n_j is $(R_d(S^l_u) - R_d(S'_u))(c \cdot L_{n_j} + C_{load}(n_j))/S^\beta_a(n_j)$, where S^l_u is the chosen option of u, S'_u is the initial size of u, and $C_{load}(n_j)$ is the total load capacitance of n_j.

(2) Otherwise, if u is a sink cell of a non-critical net n_j, the timing cost change of n_j is $R_d(C_u(S^l_u) - C_u(S'_u))/S^\beta_a(n_j)$. Let $\Delta\, t^u_c(S^l_u)$ denote the total timing cost change of all non-critical nets connected to u. If $u = u_d$ is the driving cell of the critical net n_i, $\Delta\, t^u_c(S^l_u)$ 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, $\Delta t^u_c(S^l_u)$ is included in the arc set $E(S_{u_d}, S_{u_k})$. Specifically, the cost of an arc $(S^m_{u_d}, S^l_{u_k})$ includes $\Delta\, t^u_c(S^l_u)$.

To sum up, for an arc $(S^l_{u_d}, S^m_{u_k})$ 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, i.e., costs of the arcs in this set, we include the change in the terms in F_t that are only dependent on the cell size of driver u_d), its cost $cost(S^l_{u_d}, S^m_{u_k})$ is:

$$\begin{aligned} cost(S^l_{u_d}, S^m_{u_k}) &= \Delta\,\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k}) + \Delta\,\alpha R_d c L_{n_i}(S^l_{u_d}) \\ &+ \Delta\, t^{u_d}_c(S^l_{u_d}) + \Delta\, t^{u_k}_c(S^m_{u_k}). \end{aligned}$$

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

$$cost(S^l_{u_d}, S^m_{u_k}) = \Delta\,\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k}) + \Delta\, t^{u_k}_c(S^m_{u_k}).$$

5. 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 Fig. 4(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 Fig. 4(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 Fig. 4(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 *min-cost* heuristic to tackle this problem; this is described in Sec. 8.

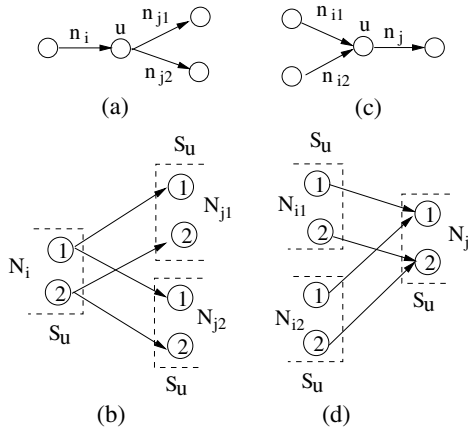

Fig. 4. *The net spanning structure. (a) The common cell u is a sink cell of one net n_i, and the driver of multiple nets n_{j_1} and n_{j_2}. (b) The corresponding spanning structure of [a], where N_k is the net structure corresponding to net n_k. (c) u is a sink cell of multiple nets n_{i_1} and n_{i_2}, and the driver of n_j. (d) The corresponding spanning structure of [c].*

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

6. Tackling the Cell Area Constraint

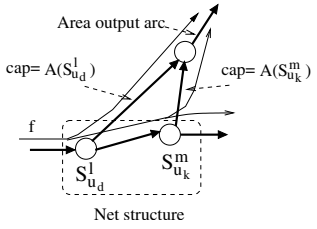

Fig. 5. *Flows to the area output arc of a net structure.*

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.

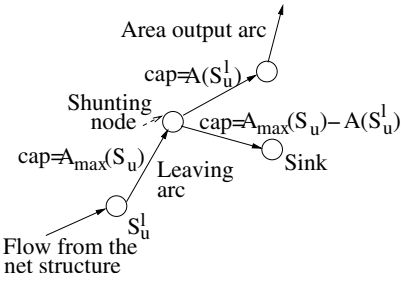

Fig. 6. *The shunting structure for outgoing flows from a net structure.*

An example is shown in Fig. 5. A flow f selects two sizing options $S^l_{u_d}$ and $S^m_{u_k}$ and incurs the corresponding cost of arc $(S^l_{u_d}, S^m_{u_k})$ in a net structure. $A(S^l_u)$ is the size of an option S^l_u. 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 needs to be a constant, i.e., 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 diverging flows to the area output arc is shown in Fig. 6. 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, 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_u^l, the capacity of the arc between the shunting node and the area output arc is then $A(S_u^l)$, and the capacity of the arc to the sink is $A_{max}(S_u) - A(S_u^l)$. Therefore, if S_u^l is chosen, the amount of flow sent to the area output arc is exactly $A(S_u^l)$, and the rest of the amount $A_{max}(S_u) - A(S_u^l)$ 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.

7. 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 the flow demand on its area 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; 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 Sec. 6. The incoming flow amount must be sufficient to cover the two outgoing flows, and thus the required incoming amount $f_{in}(N_i)$ of a net structure N_i is recursively given as:

$$f_{in}(N_i) = \sum_{u \in n_i} A_{max}(S_u) + \sum_{N_j \in child(N_i)} f_{in}(N_j)/d_{in}(N_j)$$
if N_i has any child net structure

$$f_{in}(N_i) = \sum_{u \in n_i} A_{max}(S_u),$$
otherwise (N_i is a "leaf" net structure in the SSG) (3)

where $child(N_i)$ is the set of child net structures of N_i, and $d_{in}(N_j)$ is the incoming degree of N_j, i.e., the number of parent net structures of N_j. In Eqn. 3, 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 Eqn. 3 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 Eqn. 3.

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

- For an arc in the net spanning structure from N_i to N_j, its capacity is $f_{in}(N_j)/d_{in}(N_j)$, which equals the flow amount sent from N_i to N_j according to Eqn. 3. This makes the total incoming flow amount to a net structure N_i exactly $f_{in}(N_i)$. As mentioned before, the cost of this arc is 0.
- Within a net structure, the capacities of arcs in each arc set $E(S_{u_d}, S_{u_k})$ are the same and 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 $A_{max}(S_{u_k}) + \sum_{r=1}^{t} f_{in}(N_{j_r})/d_{in}(N_{j_r})$.

A. Unit Flow Arc Cost

When we gave the costs of arcs in a net structure in Sec. 4, 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 Sec. 4, the unit flow cost $C(S_{u_d}^l, S_{u_k}^m)$ of an arc $(S_{u_d}^l, S_{u_k}^m)$ is set to be the corresponding arc cost given in Sec. 4 divided by its capacity as determined above, i.e.,

$$C(S_{u_d}^l, S_{u_k}^m) = cost(S_{u_d}^l, S_{u_k}^m)/cap(S_{u_d}^l, S_{u_k}^m)$$

8. Finding a Valid (Discretized) Min-cost Flow

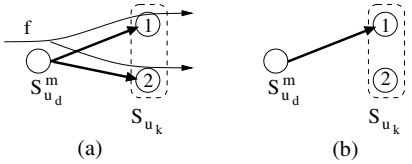

Fig. 7. *(a) Invalid flow f through two sizing options of the same cell u_k. (b) Selecting only one option of S_{u_k} for the next iteration based on some criteria of the previous "split flow".*

The standard min-cost 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 min-cost flow for size selection. Therefore, after one iteration of a min-cost network flow process, in a net structure, the obtained min-cost flow may pass through an option node $S_{u_d}^m$ 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 Fig. 7(a). Furthermore, as explained in Sec. 5, the resulting flow may also violate the consistency requirement for the size option selection across net structures that have a common sink cell.

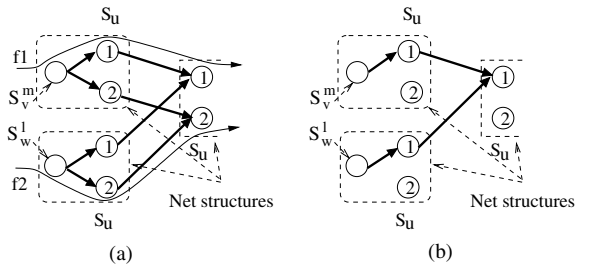

Fig. 8. *(a) Inconsistent option selections of the same cell u in different net structures. f_1 chooses option S_u^1; f_2 chooses option S_u^2. (b) The same option S_u^1 of u is selected in the next iteration for all net structures contains u as a sink cell based on some criteria of the previous flow of [a].*

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, $S_{u_d}^m$ 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 Fig. 7(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, e.g., S_v^m and S_w^l as shown in Fig. 8(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 a illegal selection in each option set S_u to be part of the new iteration:

- *Max-flow heuristic*: Always choose the option node with the largest flow amount through it.
- *Min-cost heuristic*: For the first situation, starting from $S_{u_d}^m$, follow the path of each branch of the min-cost flow up to a length of l, and choose the option node that is on the branch that has the min-cost 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 Fig. 9, similar to the first situation, the outgoing path cost of an option S_u^l of u is the cost of the flow path starting from the option node. The incoming arc cost of an option S_u^l of u is the total cost of the set of incoming arcs $G(S_u^l)$ to all S_u^l's across all net structures that contain the option set S_u; see Fig. 9. The summation of these two costs is a good estimation of the cost of a valid flow that chooses only option S_u^l 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 I for the ISCAS85 benchmarks reveals that the min-cost 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 min-cost heuristic is thus implemented in our algorithms, and we set $l = 3$ for a balance between computational complexity and accuracy.

Ckt	Max-flow	Min-cost of length			
		2	3	4	5
	%Δ T	%Δ T	%Δ T	%Δ T	%Δ T
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
avg.	8.4	10.4	11.4	11.6	11.7

TABLE I
Percentage timing improvements of max-flow heuristic and min-cost heuristic with different path length limits.

A. 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 min-cost flow obtained in one iteration, in the following iterations the min-cost flow will select a valid size option for the cell. Thus, the number of iterations required to reach a valid min-cost flow is no more than the total number of cells in $\mathcal{P}_\alpha$.

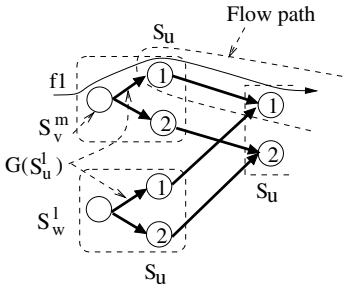

Fig. 9. *The flow path for determining the path cost of S_u^1, and the arcs for determining the arc cost of S_u^1.*

In each iteration, we use the network simplex algorithm to solve the min-cost 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 $O(Um^2 \log C)$ [3]; however, it is well-known that the average-case run time of the simplex method is much lower than this worst-case complexity [1]. As described in Sec. 7, 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 $S^2(d_{avg}-1)M$, 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 [6], [9] are not polynomially bounded in run time. Also, as we show in Sec. 10 (Fig. 11), 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 $\mathcal{P}_\alpha$, e.g., eight iterations for a circuit with 300 cells in $\mathcal{P}_\alpha$.

9. An Optimal Exhaustive-Search Algorithm

Since we do not have the library or library parameters used in [9], we cannot directly compare to their technique. Thus, we implemented an exhaustive search method that can produce optimal solutions for the sizing problem of cells in $\mathcal{P}_\alpha$, and compare our solution quality to the optimal one.

The idea of this exhaustive search is to try every possible cell sizing solution, and choose the one with best critical path delay. However, if no proper solution pruning method is applied, the run time of the search would be unacceptable even for the smallest circuit in the benchmark. We propose three pruning methods that can maintain the optimality of the solution, but greatly reduce the run time. In the exhaustive search method, we visit cells in topological order, and when we are visiting a cell, we will combine all partial solutions we have for all visited cells with possible size choices of the current cell to form new partial solutions. The pruning process happens after new partial solutions are generated.

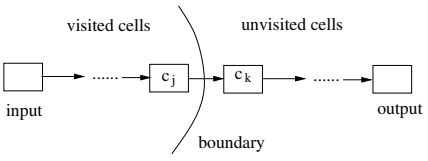

Fig. 10. *Visited cells, unvisited cells and the boundary between them.*

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 Fig. 10) 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. 10 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 does not affect optimality of the method.

Ckt	# cells	# crit. cells	Our method %ΔT	%ΔA	runtime (secs)	Optimal %ΔT	%ΔA	runtime (secs)
C432	160	50	11.9	-9.9	9	13.2	-10.0	103
C499	202	51	11.9	-9.4	14	12.4	-9.3	119
C880	383	77	14.8	-9.9	19	16.2	-10.0	195
C1355	544	85	6.8	-8.1	22	7.0	-8.8	489
C1908	880	88	16.1	-9.8	39	17.0	-10.0	556
C2670	1.3K	91	9.6	-9.5	38	11.1	-9.7	908
C3540	1.7K	124	16.8	-9.0	69	19.0	-9.4	1724
C5315	2.3K	138	10.0	-6.9	70	10.2	-7.9	3228
C6288	2.4K	299	11.5	-8.1	97	11.7	-9.1	14998
C7552	3.5K	199	9.9	-8.2	112	11.5	-7.5	6847
Avg.	1336	**120**	**11.9**	**-8.9**	**48**	**12.9**	**-9.2**	**2916**

TABLE II
Results for our method and optimal exhaustive search method. Four sizes are available for each cell. %ΔT is the percentage timing improvement, %ΔA is the percentage change of total cell area (negative value means deterioration). # of crit. cells means the number of cells in P_α.

ckt	%ΔT	%ΔA	runtime (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
Avg.	**13.9**	**-9.2**	**525**

(a)

ckt	%ΔT	%ΔA	runtime (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
Avg.	**15.6**	**-9.4**	**401**

(b)

TABLE III
(a) Results for our method when all circuit cells are considered for resizing with four sizes for each cell. (b) Results for our method when ten size options available for each cell; only cells in P_α are resizable.

10. Experimental Results

We tested our algorithm on ISCAS'85 benchmarks. We use the same industrial 0.18*um* standard cell library as in [11], 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, i.e., $(w_4 - w_3) \approx 2(w_3 - w - 2) \approx 4(w_2 - w_1)$. Other electrical parameters we use are: unit length interconnect resistance $r = 7.6 \times 10^{-2}$ ohm/μm, unit length interconnect capacitance $c = 118 \times 10^{-18} f/\mu$m. Results were obtained on Pentium IV machines with 1GB of main memory.

We ran our algorithm with a 10% total cell area increase constraint, which means that the total cell area after cell sizing can not increase more than 10%. Given the same initial sizing, the improvements obtained by our net work flow method and the optimal exhaustive search method are listed in Table II.

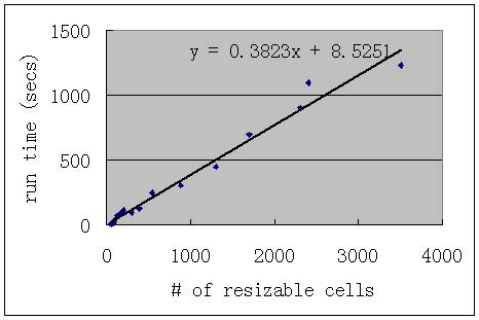

Fig. 11. *Run time vs. number of resizable cells.*

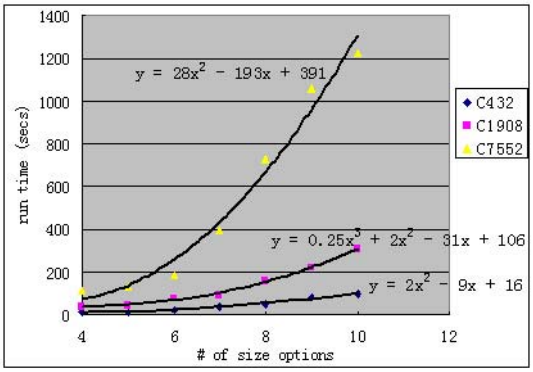

Fig. 12. *Run time vs. number of available size options for each cell for circuits C432, C1908 and C7552. The approximation functions for the empirical time complexity of our method on these circuits are shown.*

Compared to the optimal solution, the improvement obtained by our method is only 1% worse (11.9% v.s. 12.9%); note that the solutions of both methods satisfy the 10% area increase constraint. Furthermore, our runtime 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 [9], since we do not have their cell library or parameters.

To show the scalability of our algorithm, two extra experiments were performed. 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 III(a). Compared to only focusing on P_α, 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 w.r.t. the number of resizable cells is shown in Fig. 11, which best fits a linear function. This is much lower than the worst-case complexity we derived in Sec. 8, and in keeping with the well-known much smaller empirical time complexity of the network simplex method compared to its worst-case complexity [1].

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[1], so that each cell has ten different size options. The results with this larger library and cells in $\mathcal{P}_\alpha$ being resizable are shown in Table III(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% vs. 17.3%) for circuit C432, and 2.0% worse (16.8% vs. 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 vs. 5343 secs) and over 135X less for C499 (140 secs vs. 18993 secs). Compared to the four-option results in Table II, FlowSize'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 Fig. 12 the run time w.r.t. 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 Sec. 8 (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 Fig. 11 that is linear in N.

[1]Three size options are added between w_3 and w_4 with uniform spacing between them, and the other three added options are made 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 added options.

11. Conclusions

We presented a novel timing-driven network flow based cell sizing algorithm. We proposed 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, by solving for a min-cost 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 solution yielded by a standard min-cost flow algorithm, a valid min-cost flow that meets the discrete mutual exclusiveness condition of cell size selection. Constraint satisfaction is also taken care of by the area output arc and the area gathering node structures. The results show that the timing improvement obtained with our method is near optimal. Furthermore, the run time of our technique is small and polynomially bounded, and thus scales well with problem size.

References

[1] 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 (JACM)*, Volume 32 , Issue 4, October 1985, pp. 871 - 895.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network Flows: Theory, Algorithms, and Applications", Chapter 10 pp. 360, and Chapter 11 pp. 402, Prentice Hall.

[3] R. K. Ahuja and J.B. Orlin, "Scaling Network Simplex Algorithm", *Operations Research*, Vol. 40, Supplement 1: Optimization 1992, pp. S5-S13.

[4] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries", *Proc. International Conference on Computer-Aided Design*, 1998, pp. 545-550.

[5] C. Chen, C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation", *IEEE Trans. Computer-Aided Design*, vol. 18, no. 7, pp. 1014-1025, 1999.

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

[7] S. Dutt, H. Ren, F. Yuan and V. Suthar, "A Network-Flow Approach to Timing-Driven Incremental Placement for ASICs,"*Proc. International Conference on Computer-Aided Design*, 2006, pp. 375-382.

[8] J. Fishburn and A. Dunlop, "Tilos: A posynomial programming approach to transistor sizing", *Proc. International Conference on Computer-Aided Design*, 1985, pp. 326-328.

[9] S. Hu, M. Ketkary, J. Hu, "Gate Sizing For Cell Library-Based Designs," *Proc. Design Automation Conference*, 2007, pp. 847-852.

[10] D. Kim, and P. Pardalos, "Gate Sizing in MOS Digital Circuits with Linear Programming", *European Design Automation Conference*, 1990, pp. 217-221.

[11] Yang Xiaojian, Choi Bo-Kyung, M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint generation", *International Conference on Computer-Aided Design*, pp. 177-180, 2002.