Discretized Network Flow Techniques for Timing and Wire-Length Driven Incremental Placement with White-Space Satisfaction

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

Abstract— **We present a novel incremental placement methodology called FlowPlace for significantly reducing critical path delays of placed standard-cell circuits without appreciable increase in wire length (WL). FlowPlace includes: a) a timing-driven (TD) analytical global placer TAN that uses accurate pre-route delay functions and minimizes a combination of linear and quadratic objective functions; b) a discretized network-flow-based detailed placer TIF that has new and effective techniques for performing TD/WL-driven incremental placement while satisfying row-width (white space) constraints; (c) new and accurate unrouted net delay models that are suitable for an analytical placer; and (d) an effective probability-based WL-cost function in detailed placement for reducing WL deterioration while performing TD-incremental placement. We ran Flowplace on three sets of benchmarks with up to 210k cells. Starting from WL-optimized placements (done by Dragon 2.23) and using purely timing-driven incremental placement, we are able to obtain up to 33.4% and an average of 17.3% improvement in circuit delays at an average of 9.0% WL increase. When incorporating both timing and WL costs in the objective functions of global and detailed placement, the average WL increase reduces to 5.8%, a 35% relative reduction, while the average delay improvement is 15.7%, which is only relatively 9% worse. The run time of our incremental placement method is only about 10% of the run time of Dragon 2.23. Furthermore, starting from an already timing-optimized placement (done by TD-Dragon), we can still obtain up to 10% and an average of 6.5% delay improvement with a 6.1% WL deterioration; the run times are about 6% of TD-Dragon's.**

I. Introduction

Due to the increasing ratio of interconnect to gate delays in very deep submicron (VDSM) designs, and the large impact that placement plays on the final wire length (WL) as well as performance, WL and timing consideration during placement is critical. Timing-driven (TD) placement algorithms can be divided into 3 categories. 1) partition based [13], [17], 2) simulated annealing (SA) based [15], [21], and 3) analytical [12]. Timing optimization is basically a path-based problem, though it is impractical to track delays of all the paths, since the number of paths is generally exponential in circuit size [12]. Hence, timing constraints on paths are usually converted to either net/edge weights or constraints such as net delay bounds, yielding more tractable *net-based* methods. In a recent work [12], a novel edge weight function was proposed that, together with its new objective function, solves the convergence problem[1]. In [21], a SA approach is used along with delay bounds on nets. The slack assignment approach in the paper ensures that estimated long nets are assigned a larger delay bound, so that they are not overly constrained. The objective function there is to minimize the sum of delay violations across all nets.

Another approach to TD placement is *targeted TD incremental placement*. On an initial base placement, an incremental TD placer can focus on reducing the most critical path delays. This greatly reduces the number of paths that need to be considered. A significant advantage of TD incremental placement is that accurate timing information can be derived from the initial placement. TD incremental placement is also important in Engineering Change Order (ECO) scenarios where changes in stages above the physical design level percolate down to required changes in the placement and routing stages. In such applications, TD incremental placement would make the required placement changes while satisfying given timing constraints. In the rest of this paper we describe our TD incremental placer in the context of the first application.

A TD incremental placer was proposed in [20] that directly controls the delay of critical paths. It explicitly sets delay constraints for a certain number of most critical paths. It then finds a solution to these constraints while minimizing total HPBB WL change in the circuit using linear programming. This method only takes the linear part of interconnect delay into consideration, and is thus not an accurate estimation of timing. Also, non-critical nets are ignored. Hence, though it uses a path-based method, the convergence problem can still occur.

The method FlowPlace proposed in this paper consists of two placers. First, an incremental TD analytical placer TAN is used to find an initial placement, possibly with overlaps. Then a TD detailed placer TIF is used to get a legal placement that minimizes critical path delay increase over that of TAN's placement without an appreciable increase in WL. Our TD analytical placer extends the basic techniques of Gordian [14] and Gordian-L [19] to optimize a TD objective function with quadratic as well as linear terms. The objective function is designed to include the delay changes of nets on both critical and non-critical paths into consideration with suitable weights for nets on paths with different criticalities to avoid the convergence problem. The detailed placement algorithm uses a network-flow-based method. Network flow has been used previously for solving the detailed placement problem in standard-cell circuits [4], [5]. In both these works, the network flow modeling is similar to ours in some high-level issues: cells are represented by nodes and possible movements of

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

[1]The convergence problem is that in net-based methods, delay decrease along critical paths is sometimes obtained at the expense of delay increase in non-critical paths to the extent that circuit delay reduction is small, if any at all.

cells are represented by arcs from the cell to their destinations. Their goal was to remove cell overlaps while minimizing total WL. However, they use simple WL models so that the WL change can be easily represented by sum of flow costs. Our objective, on the other hand, is to minimize the delay of critical paths rather than the sum of delays. To this end, we use more complex cost functions and flow graph structures to make sure that the sum of flow costs is a good indicator of the critical path delay change. Besides the purely timing-driven cost, we also develop a probability-based WL cost function to accurately model WL change in a net due to multiple cell movements. Experiments show that using a combination of timing and WL cost can significantly improve critical path delay while keeping the WL deterioration small.

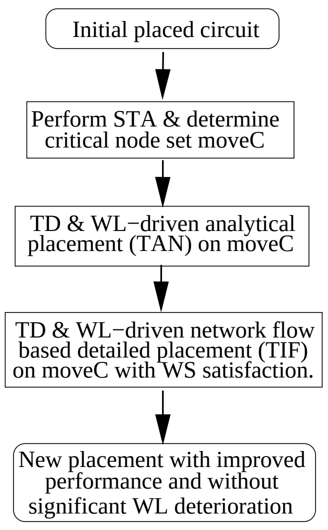
The work reported here is a significant extension of [7] in which only TD costs were considered. The extension to including HPBB-based WL-driven costs is quite involved, and helps in limiting routability and area deterioration while improving circuit performance. We also provide empirical and theoretical results to justify our flow discretization techniques used to discretize the continuous solution yielded by classical network flow so that a valid incremental placement solution (which needs to be a discrete configuration) is obtained. Finally, we provide here a detailed exposition of our network flow based white space (WS) satisfaction techniques that are critical to the success of our entire incremental placement flow.

The rest of the paper is organized as follows. Section II discusses basic issues in incremental TD placement and the flow of our methodology. Our new and accurate pre-route net delay model is introduced in Sec. III. In Sec. IV we present various aspects of our TD analytical placer. In Sec. V, the basic network flow model for TD detailed placement is introduced. Three specific issues about our detailed placer, WL-driven arc costs, flow discretization techniques, and WS satisfaction techniques are discussed in Secs. VI, VII and VIII, respectively. Section IX presents experimental results, and we conclude in Sec. X.

II. TD Incremental Placement and Methodology Flow

The TD incremental placement problem can be stated as:
Input: A placed circuit $\mathcal{PC}$, a set $moveC$ of new unplaced cells (the scenario of modified cells is handled by deleting them from $\mathcal{PC}$ and adding them to $moveC$).
Output: A completely placed circuit $\mathcal{PC}'$ in which: (1) there are minimal changes made to the existing placement $\mathcal{PC}$ in terms of both movements of cells in $\mathcal{PC}$ and deterioration of placement metrics like total wire length (WL) and chip area, and (2) the critical path delay in $\mathcal{PC}'$ is significantly improved compared to the one in $\mathcal{PC}$.

Fig. 1 shows the flow of our TD incremental placer. We start from a placed circuit and identify all critical and near-critical paths using static timing analysis (STA). Let this set of paths be $\mathcal{P}$ (we take paths with delay larger than 0.9 of the circuit delay as near-critical paths in our experiments). After $\mathcal{P}$ is identified, we remove all cells in all nets that lie in $\mathcal{P}$ from the layout. The removed cells form the cell set $moveC$ (nets connected to cells in $moveC$ are denoted by $moveN$) that will be replaced by our TD incremental placer with the goal of reducing the critical path delay. As in standard placement flows, our incremental placement method also consists of two stages, global and detailed. In the global stage, our TD analytical placer TAN is used in which $moveC$ constitutes the set of moveable cells. Our main contribution in this part is two-fold. The first is developing accurate and detailed pre-routing net-delay models, and determining net weights so that the net-delays of critical paths have the highest minimization priority. The second is performing both quadratic and linear optimization simultaneously.



Initial placed circuit
Perform STA & determine critical node set moveC
TD & WL−driven analytical placement (TAN) on moveC
TD & WL−driven network flow based detailed placement (TIF) on moveC with WS satisfaction.
New placement with improved performance and without significant WL deterioration

Fig. 1. *The flow of our TD incremental placer FlowPlace. The detailed content of each timing-driven step including WL consideration will be introduced in different sections.*

The output of our TD analytical placer will generally be an illegal placement for cells in $moveC$—the cell positions determined will generally not be in cell rows and/or there may be overlaps with each other or with cells in $\mathcal{PC}$. However, these cell positions provide starting points for our detailed TD placer, which uses a novel network flow based approach for placing new cells in legal positions and moving existing cells minimally to accommodate them so that the critical path delay is optimized and row-length (i.e., row white space) constraints are satisfied. This TD min-cost max-flow white-space satisfying algorithm called TIF is the major contribution of this paper.

III. The γ Net Delay Model

In our TD incremental placement, we use a new net delay model for unrouted nets that we refer to as the *γ net delay model*. Starting with an unrouted placement, we assume a net routing pattern with a single trunk to the furthest sink with branches off this trunk to the other sinks, as shown in Fig. 2. For a net n_j with driver u_d, and $k-1 \geq 1$ sinks (k is the total number of pins in n_j), let R_d be the driving resistance, C_g the load capacitance of a sink pin[2], r (c) the unit wire resistance (capacitance), $L(n_j)$ the total WL of n_j, and $l_{d,i}$ the interconnect length connecting driver u_d to sink u_i; see Fig. 2. Referring to this figure and considering a sink u_i in n_j, the delay $D(u_i, n_j)$ to it (using the Elmore delay model [8]) from the driver u_d, consists of three parts:

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

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

[2]For simplicity of exposition, we assume uniform loads for all sink pins, though clearly our net-delay modeling and methods also apply to non-uniform loads.

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

TABLE I
The critical path delay calculated using the γ net delay model coupled with the multi-star WL model [6]. The actual routed delay is listed for comparison.

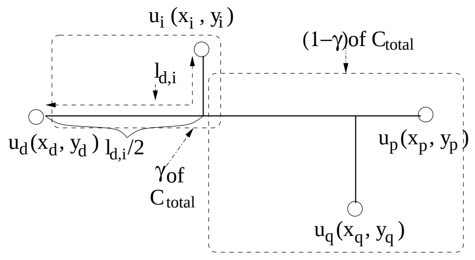


Fig. 2. *The γ net delay model for post-placement unrouted nets; C_{total} is the total (net and load) capacitance seen by the driver.*

$$D_3(u_i, n_j) = r \cdot (l_{d,i}/2)((1-\gamma+\gamma/2)(c \cdot L(n_j) + (k-2)C_g) \quad (3)$$

$$D(u_i, n_j) = D_1(n_j) + D_2(u_i, n_j) + D_3(u_i, n_j) \quad (4)$$

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

The comparison between the delay estimation using our γ-delay model coupled with our new multi-star WL model [6][3], and the actual routed delay is provided in Table I. The delay estimation error is only 5.6%, which shows the accuracy of our γ net delay model. Furthermore, our delay model shows 100% fidelity with routed delay.

IV. TD Analytical Global Placement

Our analytical placer TAN is a timing-driven extension of a combination of Gordian [14] and Gordian-L [19], i.e. we optimize an objective function that contains both linear and quadratic terms.

A. Basic Gordian and Gordian-L

Gordian is a quadratic programming technique for cell placement for minimizing total quadratic WL $L^2(n_j)$ for all net n_j. Gordian-L applies an additional inner-iteration for the optimization in each subregion, which essentially comprises dividing in the $(m+1)$'th inner iteration, each $L^2(n_j)$ part of the objective function by a net-centric linear-length quantity given by $\eta_j^m = \sum_{u_i \in n_j} |x_i^m - x_c^m|$ (for the optimization along the horizontal dimension), where x_i^m is the value of the x-coordinate of u_i after the m'th iteration, and $\eta_i^0 = 1$. This has the effect of linearizing the WL objective function at the end of the inner iteration.

B. Timing-driven objective function

We define the *critical delay* $D_c(n_j)$ of net n_j as:

$$D_c(n_j) = D_1(n_j) + \sum_{u_i \in critical(n_j)} D_2(u_i) + D_3(u_i).$$

where $critical(n_j)$ is the set of critical sinks on critical and near critical paths through n_j. Refer to Sec. III for definition of D_1, D_2 and D_3. The intent here is to include only the delays of critical sinks in D_c, and this is really a delay-criticality measure of n_j rather than an actual delay of some component of this net. We define the *allocated slack* $S_a(n_j)$ of net n_j as $S(P_{max}(n_j))$/(# of nets in path $P_{max}(n_j)$), where $P_{max}(n_j)$ is the maximum-delay path through n_j, and recall that $S(P)$ is the slack of path P (other slack allocation algorithms such as ZSA [11] and that in [10] can also be used).

How much a net n_j's interconnect lengths should be minimized for optimizing the circuit's critical path delay depends not only on the net's D_c value but also on $S(P_{max}(n_j))$—a net with high D_c value but lying on a path with very high slack either does not need to be delay-optimized or should have low delay optimization priority, and similarly for the reverse case. Furthermore, considering two nets n_i, n_j on different max-delay paths with similar slacks and similar D_c values, they should not necessarily be optimized similarly. The important parameter along with D_c is the allocated slack S_a of the nets. The rationale for this is as follows. Let the max-delay path through n_i (n_j) have 10 (5) nets in them. If the delay optimization priority were the same for all the nets on $P_{max}(n_i)$ and $P_{max}(n_j)$ due to their similar D_c and path slack values, then the delays on their critical interconnect (assuming only one critical interconnect from the driver to a single most critical sink on each of the 15 nets) will be made almost equal. This results in $P_{max}(n_i)$ having twice the delay of $P_{max}(n_j)$, and thereby a high probably of slack violation in the former. On the other hand, if the delay cost of each net is made $\propto D_c/S_a$, then in our example, since the S_a for the nets in $P_{max}(n_i)$ are half that of those in $P_{max}(n_j)$, the former will have twice the delay optimization priority (i.e., delay cost) than the latter leading to balanced delays for both critical paths $P_{max}(n_i)$ and $P_{max}(n_j)$.

Based on the above arguments we define the *delay cost* $C_D(n_j)$ of n_j as

$$C_D(n_j) = D_c(n_j)/S_a(n_j)^\rho$$

where ρ is an exponent of the S_a metric that allows magnification (with $\rho > 1$) or shrinking (with $\rho \leq 1$) of differences in optimization priorities of nets on paths with varying allocated slacks; $\rho = 1$ in this stage gives the best results. Hence an appropriate TD objective function F_{TD} is:

$$F_{TD} = \sum_{n_j \in moveN} C_D(n_j). \quad (5)$$

[3]The multi-star model is explained in [6]. The difference between the estimated WL using this model and the routed WL on TD-Dragon benchmarks [21] is only 6.6%. On the other hand, the HPBB, star graph and LSE [16] models differ from the routed WL by 19%, 35% and 23%, respectively.

F_{TD} includes both linear and quadratic WL terms. We use a quadratic solver to optimize F_{TD}[4]. Note that since the analytical placement phase will be followed by a legalizing detailed placer, we do not perform the hierarchical partition-based iteration of the quadratic optimization process as in Gordian and Gordian-L.

C. WL consideration in global placement

Timing-driven placement usually will cause WL increase compared to a WL-optimized placement. To maintain the routability of a design, the WL increase should be limited to a small amount. To reduce the WL increase in global placement, we employ a simple approach here of making the objective function a weighted summation of the TD objective function (Eqn. 5) and a WL oriented objective function. The WL oriented objective function F_{WL} is chosen to be the total net WL of all nets belonging to $moveN$. We use the accurate multi-star model [6] to estimate WL during global placement.

Since WL is linear rather than quadratic, the quadratic approximation approach is used again for the linear terms as in the timing-driven objective function. The combined objective function is:

$$F = w_1 \cdot F_{TD} + (1 - w_1) \cdot F_{WL} \tag{6}$$

where F_{TD} is given in Eqn. 5. Our experiments with different values of w_1 reveal that choosing w_1 to be 0.9 yields good timing improvement with little WL deterioration.

V. Basic TD Network-Flow Based Detailed Placement Model

The output of the TD analytical global placer (TAN) will generally be an illegal placement, but it presents a good starting point for our TD network-flow based detailed placer (TIF) to place the new cells in legal positions to minimize critical path delays. To accommodate new cell placement, existing cells will be moved minimally. In a TD detailed placement, all cell movements are done based on the timing-driven costs. The timing-driven cost is: a) proportional to the *delay sensitivity* $D_s(u)$ which is the delay change per unit displacement of cell u of the most critical interconnect through it, and b) inversely proportional to the *allocated slack* $S_a(u)$ of this cell ($S_a(u) = S_a(n_j)$ where n_j is the net on the max-delay path through u); further details are in Sec. V-D.

Besides the basic model provided in this section, there are also three other issues in TIF: 1) WL cost determination for reducing WL deterioration; 2) Flow discretization for obtaining valid solutions; 3) WS constraint satisfaction with discrete flows. These are discussed in Secs. VI, VII and VIII, respectively.

A. Network flow model

Fig. 3(a) shows the network flow graph used in TIF with arc costs and capacities. Network flow has commonly been used to model transportation of goods from a source to a target destination using a network so that costs are minimized and capacity constraints are met. Network flow has found application in VLSI CAD as well in applications ranging from partitioning to placement [4], [5], [22].

Our network-flow-based incremental placement algorithm TIF is novel in the way it models the timing and WL driven arc costs, and in that it accurately solves white space constraints for standard cell placement by overlaying constraints on the flow process.

There is an arc from the source S to each new cell v (e.g. cells A_1 and A_2 in Fig. 3(a)) of capacity equal to the width $w(v)$ of v, and for each such v, there are two "vertical" arcs from it directed towards cells in rows immediately above and below it (there are more details to these "conceptual" arcs shown in Fig. 3(c)); the capacity of each vertical arc is also $w(v)$. A total flow of $f = \sum_{v \in moveC} w(v)$ emanates from S, and a max-flow solution through the network will result in each new cell being pushed to one of its row-position choices (modeled by the vertical arcs from it).

From each *row cell* (i.e., existing cell such as cells C_{11} and C_{12} in Fig. 3(a)), there are four arcs, one in each of the 2D directions (this is easily extended to 6 arcs, one in each of the 3D dimensions in case of 3D VLSICs). The vertical arcs from u go to cells in adjacent rows and model possible movement of u in the respective vertical directions; the capacity of these arcs is $w(u)$, since only u can move along these arcs. The horizontal arcs from u model possible horizontal movement of u within its row, and are potentially of capacity equal to the width of the row from u to the corresponding end of the row, since u could be moved up to either end of the row. However, since arc cost estimates become more inaccurate for large displacements of the cells, a capacity equal to the maximum of the widths of the cell in adjacent rows or new cells that have vertical arcs into u is imposed on the horizontal arcs. This allows enough horizontal flow through u that causes its required movement to remove overlaps with cells vertically moved to its position (via vertical flows into u). There can be intermediate white space within rows and these are also modeled as nodes with incoming horizontal and vertical arcs, but only one outgoing arc to the total WS node W_i of the row; the arc's cost is zero and capacity equal to amount of that intermediate white space. Finally, the *total white space* $w(W_i)$ of row i (R_i) = (max row size constraint) - ($\sum$ [cell widths in it]) is also modeled as a node W_i at the right end of the row with an incoming horizontal arc from the rightmost cell and an outgoing arc to T of zero cost and capacity = $w(W_i)$. This network flow graph is called the *detailed network flow graph* (as opposed to the global network flow graph to be introduced later).

B. Vertical arc structures

Figures 3(b-c) show the details of the simplified "vertical arcs" of Fig. 3(a). The detailed structures are needed to determine the vertical flow amount from a cell, say, C_{21}, to cells in rows above and below it that it would overlap if it is moved vertically up or down, respectively, to the corresponding row. The total amount of flow emanating from C_{21} corresponding to its vertical movement is $w(C_{21})$, and enters its "outflow node" D_{out} as shown in Fig. 3(b). The outgoing flow from

[4]Both the quadratic and linear terms are optimized in an inner loop in which, in each iteration, the quadratic versions of the linear terms are divided by their current values, while the quadratic terms remain unchanged.

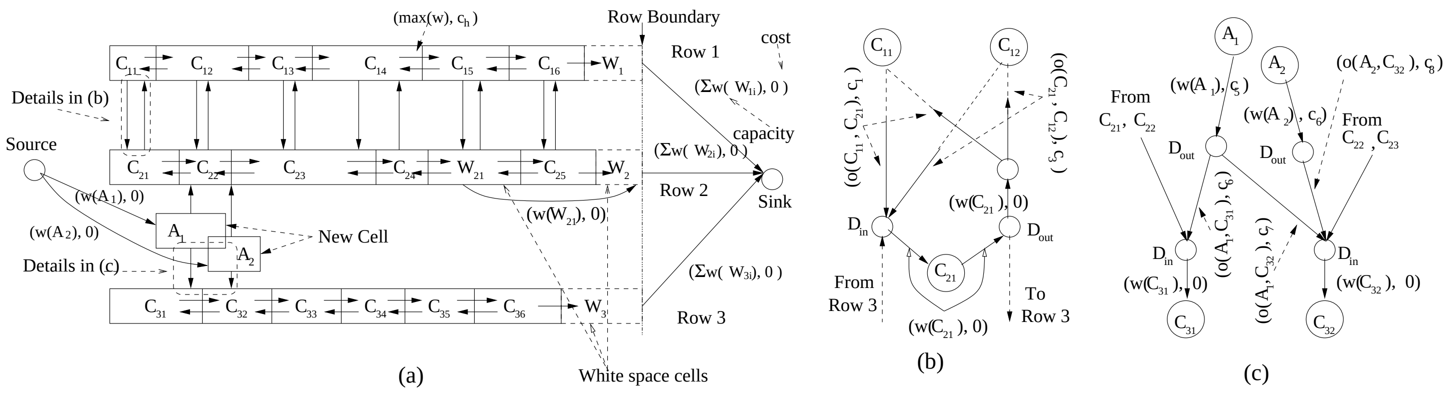


Fig. 3. *(a) The high-level network flow graph for placing cells A_1, A_2 in legal positions; $w(u)$ is the width of a cell u. (b) Details of flow graph structure for vertical flows between cell pairs (C_{11}, C_{21}) and (C_{12}, C_{21}); $o(u, v)$ is the horizontal overlap distance between cells u and v. This flow graph structure only allows a flow of amount $<= w(u)$ into a row cell u, and also the vertical flow out of a cell v can go to all cells in the adjacent row that it horizontally overlaps. (c) Similar details of the flow graph structure for flows from the new cells into vertically adjacent row cells.*

D_{out} going to the row above C_{21}'s (the structure is similar for the flow going to the row below C_{21}) splits into as many arcs as the number of cells C_{21} would overlap when moved to the row above it. The amount of flow in the arc to cell C_{1j} in this row is equal to the overlap amount $o(C_{21}, C_{1j})$ between C_{21} and C_{1j} when C_{21} is moved up; this is appropriate, since this flow amount emanating horizontally from C_{1j} would move it by an amount $o(C_{21}, C_{1j})$ that removes the overlap (if the flow emanates vertically from C_{1j}, then irrespective of the flow amount, as we will see in Sec. VII-A, C_{1j} is moved to the next row in the direction of the chosen vertical arc, thus removing the overlap). Note that the sum of the $o(C_{21}, C_{1j})$'s is equal to $w(C_{21})$. Figure 3(c) shows similar structures for outgoing vertical flows from illegally placed cells A_1, A_2 going down to row R_3.

On the flip side, flows can vertically enter cell C_{21} from all cells in rows above and below it as well as from illegally placed cells near its row that would overlap it when moved to its row. However, a total flow of only $w(C_{21})$ can be allowed to enter it, and this is ensured with vertical flows from the above-mentioned cells entering the "inflow node" D_{in} of C_{21} and from there entering C_{21} via an arc of capacity $w(C_{21})$; see Fig. 3(b). Henceforth, for a more conceptual discussion relating to vertical flows, we will represent vertical movements by the simple vertical arcs of Fig. 3(a); the detailed structures corresponding to these vertical arcs are not necessary for subsequent discussions.

C. The Simplex network flow algorithm

The Simplex method is widely used to solve the min cost network flow problem. Its basic idea is to iteratively improve an initial solution. It starts with a feasible but generally non-optimal flow of the given amount f. After that, it tries to find *negative cycles*, defined as cycles that have negative costs when traveling in a certain direction. For each such cycle, the Simplex method augments or pushes a flow of the maximum possible value in the cycle in the negative-cost direction. It continues doing so until there are no negative cycles, or flows in negative cycles cannot be augmented anymore. Our implementation is based on the Simplex algorithm proposed in [2].

D. Timing-driven cost functions

As mentioned earlier, the TD cost of arc (u, v) should be: i) proportional to the delay change or sensitivity of the most critical interconnect of its start node u to unit length displacements of u in the direction of the arc, and ii) inversely proportional to the allocated slack of its start node u. Delay sensitivity, which is essentially the derivative of the delay function w.r.t. start cell displacement, is a good measure of performance cost when cells are moved by not-very-large displacements from known positions, as in the case of incremental detailed placement.

Eqns. 1-4 gives the delay formulation for a sink u_i on net n_j. The sensitivity of this delay to a displacement of either sink u_i or driver u_d by $\Delta l_{d,i}$ can be obtained by taking derivatives w.r.t. $l_{d,i}$; for the components in Eqns. 1-4, these are:

$$\Delta D_1(u_i, n_j) = R_d c \cdot \Delta L(n_j) \approx R_d c \cdot \Delta l_{d,i} \tag{7}$$

$$\Delta D_2(u_i, n_j) = rc\Delta l_{d,i}^2 + 2rcl_{d,i} \cdot \Delta l_{d,i} + r \cdot \Delta l_{d,i} C_g, \tag{8}$$

$$\Delta D_3(u_i, n_j) = \Delta D_{3a}(u_i, n_j) + \Delta D_{3b}(u_i, n_j), \text{where}$$

$$\Delta D_{3a}(u_i, n_j) = r \cdot (\Delta l_{d,i}/2)((1-\gamma/2)(c \cdot L(n_j) + (k-2)C_g) \tag{9}$$

$$\Delta D_{3b}(u_i, n_j) = r \cdot (l_{d,i}/2)((1-\gamma/2)(c \cdot \Delta l_{d,i}) \tag{10}$$

$$\Delta D(u_i, n_j) = \Delta D_1(u_i, n_j) + \Delta D_2(u_i, n_j) + \Delta D_3(u_i, n_j). \tag{11}$$

Note that the $\Delta l_{d,i}$ can be positive (increasing $l_{d,i}$) or negative (decreasing $l_{d,i}$). The magnitude of $\Delta l_{d,i}$ for a horizontal arc is its capacity (which reflects the maximum displacement of the cell), and for a vertical arc, it is the spacing between the two adjacent rows that the arc spans (this reflects the exact cell displacement if there is any flow along this arc).

The displacement of a cell u in the direction of a flow arc e emanating from it impacts critical nets connected to u in two ways: a) as a sink on its *most critical net*—net connected to u with the minimum allocated slack among all of u's nets, and b) as a driver of the most critical net connected to it.

a) As a sink, there are two cases:
i) u is the most critical sink of its most critical net n_j, in which case its effect on the delay change on n_j is

$$\Delta D_a(u) = \Delta D(u, n_j) \,\text{as explained in Eqns. } 7-11.$$

ii) u is not the most critical sink of its most critical net n_j, in which case its effect on the delay change on n_j is

$$\Delta D_a(u) = \Delta D_1(u, n_j) + \Delta D_{3b}(u_i, n_j),$$

which reflects the displacement's effect on $L(n_j)$ and thereby on $\Delta(u_i, n_j)$ for the most critical sink u_i on n_j.

b) As a driver of its most critical net n_k, the effect of $u's$ displacement on the delay on its most critical interconnect is:

$$\Delta D_b(u) = \Delta D(u, n_k)$$

Based on the above, the timing-driven cost of an arc e (i.e., its unit-flow cost) emanating from u is:

$$cost_t(e) = \frac{\Delta D_a(u) + \Delta D_b(u)}{cap(e) \cdot S_a(u)^\kappa} \quad (12)$$

Note that $S_a(u) = S_a(n_j) = S_a(n_k)$ as n_j and n_k lie on the max-delay path through u, and the exponent κ is used to magnify or shrink cost differences among arcs emanating from cells connected to critical and non-critical nets; $\kappa = 2$ gives us the best overall results.

Purely timing-driven cost can result in a WL increase of over 10%. Thus, to reduce the final WL increase, we use a combined arc cost which is the weighted sum of the delay cost described above and the WL cost which is the total WL change due to the movement of the arc.

VI. WL Cost in Detailed Placement

In this section, we discuss our WL cost formulation for measuring the WL increase caused by a cell movement. In our WL-aware TD detailed placement, the final cost of an arc in the network flow graph is a combination of the timing and the WL costs of the arc's corresponding cell movement.We use the HPBB model to measure WL here.

A. Net WL change with multiple moving cells

When we consider the HPBB-based WL metric, determining the unit flow cost of each arc (HPBB change per unit displacement of the start cell of the arc) is not straightforward. There are basically two problems as discussed below.

The first problem is that the unit flow cost of an arc is dependent on the amount of flow on the arc (as opposed to being a constant in a standard network flow problem). This is illustrated in Fig. 4(a). Cells u_1 and u_2 belong to the same net whose bounding box is shown by dashed lines. Movements of cell u_1 to u_1' and u_1'' correspond to different flow amounts on the same arc from u_1. The flow amount for movement to u_1' is smaller than the one for movement to u_1''. However, because the movement to u_1' does not cause any bounding box change while the other one to u_1'' does, the unit flow cost of the arc calculated according to these two movements will not be the same (they will be zero and non-zero, respectively).

The second problem is that the WL metric change caused by a cell movement is dependent on the movements of its adjacent cells. As shown in Fig. 4(a), the movement of cell u_1 to u_1' has no effect on the bounding box, and thus should have a 0 WL cost. However, if u_2 is moved first to u_2', as shown in Fig. 4(b), then u_1 becomes the only cell on the new bounding box. As a result, the same movement of u_1 to u_1' now causes a HPBB change; thus a non-zero WL cost should now be associated with it.

B. Probability-based cell position estimation

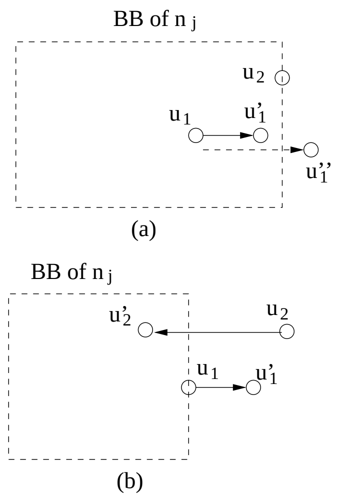


Fig. 4. *Dependency of the WL cost for moving u_1 on (a) the length of the movement, and (b) the movement of other cells, e.g., u_2, on the same net.*

Due to the above two dependencies, our approach here is to determine a probabilistic unit flow WL cost. We first determine the probability of a cell moving to a specific position. This probability can be used to solve both the flow-amount dependency (convert it to the probability that there is a certain amount of flow on an arc) and the adjacent-cell-position dependency (estimate the final positions of adjacent cells probabilistically). To determine the probability of a cell moving to a specific position, we need to use the flow result of a *global network-flow process*. The global network-flow process is a simplified version of the regular network-flow process for detailed placement introduced earlier, in which a node represents a row instead of a cell (see Fig. 9); from each row node, there are vertical arcs to nodes representing adjacent rows and a horizontal arc to the sink. With global network flow we cannot, of course, obtain the flow passing through each cell as in detailed network flow, but we can obtain an estimate of the total flow entering and leaving each row (the flows on the vertical arcs between row nodes) and of the original white space occupied in each row (the flow from a row node to the sink). In order to determine the global flow, we use a purely TD global network-flow graph, and based on the resulting flow, estimate the WL cost of each arc as described below. The purely TD global flow will not be exactly the same as the global flow with a combined cost. However, since in the combined cost function of timing and WL, the former has a dominant weight, this approximation is reasonable. We give a more detailed exposition of the global network-flow process and the costs in it, and discuss its other major application in Sec. VII-C.

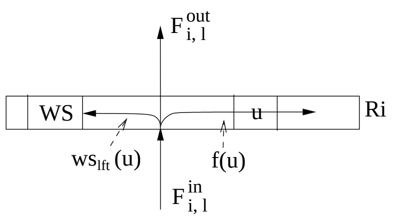


Fig. 5. *Estimating the horizontal movement of cell u.*

We first consider horizontal cell movements. For a cell u in a row R_i, suppose the total flow amount entering the row (as determined in the global network flow process) is F_i^{in}, and the total outgoing flow amount from the row is F_i^{out} in the global flow process. We can divide the total incoming flow into unit flows $f_1, f_2, ..., f_m$, where $m = F_i^{in}$. Assume that each unit incoming flow f_j ($1 \le j \le m$) has an equal probability of entering at any point in the row. We define a function $\mathit{left}(f_j, u)$, which is 1 if the entering point of f_j is to the left of u, and 0 otherwise. The summation of $\mathit{left}(f_j, u)$ of all incoming unit flows gives the total flow amount $F_{i,l}^{in}(u)$ that comes into R_i to the left of u. It is easy to see $\mathit{left}(f_j, u)$ has a 0-1 distribution across all unit incoming flows. The probability $p^{\mathit{lft}}(u)$ of $\mathit{left}(f_j, u)$ being 1 is the ratio of the width of the row segment to the left of the center of u and the maximum allowable row width assuming the unit incoming flow has an equal probability of entering at any point of the row; thus if we set the horizontal coordinate of the left end of each row to be 0 and let the horizontal coordinate increase from left to right, then $p^{\mathit{lft}}(u) = \frac{x_u}{W_{max}}$, where x_u is the horizontal coordinate of the center of u, and W_{max} is the maximum allowable row width. We also denote $w(u)$ to be the width of u

The *central limit theorem* [3] states that the distribution of the summation of k independent variables with the same distribution that has a mean of μ and a standard deviation of σ can be approximated by a normal distribution with a mean of $k\mu$ and a standard deviation of $\sqrt{k}\sigma$. In our case, the mean of $\mathit{left}(f_j, u)$ is $p^{\mathit{lft}}(u) \cdot 1 + (1 - p^{\mathit{lft}}(u)) \cdot 0 = p^{\mathit{lft}}(u)$, and the standard deviation of $\mathit{left}(f_j, u)$ is $[p^{\mathit{lft}}(u) \cdot (1 - p^{\mathit{lft}}(u))^2 + (1 - p^{\mathit{lft}}(u)) \cdot (p^{\mathit{lft}}(u))^2]^{\frac{1}{2}} = [p^{\mathit{lft}}(u)(1 - p^{\mathit{lft}}(u))]^{\frac{1}{2}}$. Therefore, the distribution of $F_{i,l}^{in}(u)$ is a normal distribution with a mean of $p^{\mathit{lft}}(u) \cdot F_i^{in}$ and a standard deviation of $[p^{\mathit{lft}}(u)(1 - p^{\mathit{lft}}(u)) \cdot F_i^{in}]^{\frac{1}{2}}$. A similar conclusion applies to the outgoing flow from R_i, i.e., the total amount of flow $F_{i,l}^{out}$ that comes out of R_i at the left side of u follows a normal distribution with a mean of $p^{\mathit{lft}}(u) \cdot F_i^{out}$, and a standard deviation of $[p^{\mathit{lft}}(u)(1 - p^{\mathit{lft}}(u)) \cdot F_i^{out}]^{\frac{1}{2}}$. The distance of the final movement of u is, by definition of horizontal flow, the flow $f(u)$ passing through it. As shown in Fig. 5, $f(u)$ is obtained as:

$$f(u) = F_{i,l}^{in}(u) - F_{i,l}^{out}(u) - ws_{\mathit{lft}}(u) \tag{13}$$

where $ws_{\mathit{lft}}(u)$ is the amount of flow that goes into white spaces to the left of u; $ws_{\mathit{lft}}(u)$ is obtained by multiplying the total amount of original white space in R_i to the left of u with the fraction of white space in R_i used by the global network-flow process (the amount of flow going to the sink from the R_i node). Eqn. 13 is correct, since, from the definition of the four parameters involved, $F_{i,l}^{in}(u)$ has three components: $F_{i,l}^{out}(u), ws_{\mathit{lft}}(u)$ and $f(u)$ (see Fig. 5).

Recall that x_u is the current horizontal coordinate of u, and the coordinate increases from left to right. The final horizontal coordinate x'_u of u is:

$$x'_u = x_u + f(u) = x_u + F_{i,l}^{in}(u) - F_{i,l}^{out}(u) - ws_{\mathit{lft}}(u)$$

Since $F_{i,l}^{out}(u)$ and $F_{i,l}^{in}(u)$ have normal distributions, and x_u and $ws_{\mathit{lft}}(u)$ are constants (the latter is determined by global network flow as mentioned earlier), x'_u also has a normal distribution with a mean of $x_u + p^{\mathit{lft}}(u) \cdot F_i^{in} - p^{\mathit{lft}}(u) \cdot F_i^{out} - ws_{\mathit{lft}}(u)$, and a standard deviation of $(p^{\mathit{lft}}(u)(1 - p^{\mathit{lft}}(u)) \cdot F_i^{in} + p^{\mathit{lft}}(u)(1 - p^{\mathit{lft}}(u)) \cdot F_i^{out})^{\frac{1}{2}}$. Let us denote the probability density function of x'_u by $\Phi_u(x)$.

The vertical movement is quite different from the horizontal movement. For cell u in R_i, it has only two possible destinations of vertical movement, the two adjacent rows R_{i-1} and R_{i+1}. If there is a large global flow from R_i to R_{i+1}, and the total cell area in R_i is small, then each cell u in R_i has a large probability of moving into R_{i+1}. Thus this probability $p_{i,i+1}(u)$ can be modeled as $p_{i,i+1}(u) = \frac{f_{i,i+1}}{W_{R_i}}$, where $f_{i,i+1}$ is the total flow from R_i to R_{i+1}, and W_{R_i} is the total cell width in R_i. Similarly we can get the probability of u moving to R_{i-1}, and thus the probability that u stays in its original row. We denote the probability density function of the final vertical position of u by $\Phi_u(y)$.

C. Probability-based WL cost function

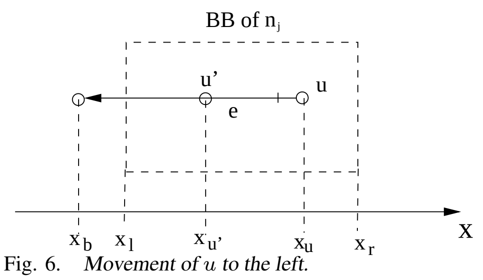


Fig. 6. *Movement of u to the left.*

With the above background, we introduce our probability-based WL cost function. A general case is shown in Fig. 6. The bounding box of a net n_j connected to u is shown by dashed lines with left and right boundary coordinates x_l and x_r, respectively. We consider the unit flow WL cost of arc e from cell u and directed to its left; for arcs to the right and vertical arcs, the WL cost can be obtained similarly. First we determine the WL change of movement of u considering the possible movements of its adjacent cells (solving the adjacent cell position dependency). Let x_b be the bounding horizontal position to where u can be moved due to the finite arc capacity of e. Suppose the final position of u is u' after detailed placement, and the x coordinate $x_{u'}$ of u' satisfies $x_b \le x_{u'} \le x_u$; we consider probabilities of two events here: (1) the right boundary of the net is also moved to $x_{u'}$, so the HPBB is reduced by $x_r - x_{u'}$; (2) if $x_{u'} < x_l$, the left boundary of the net is moved to $x_{u'}$, so the HPBB is increased by $x_l - x_{u'}$. We use $p_e^-(x_{u'}, n_j)$ and $p_e^+(x_{u'}, n_j)$ to denote the two probabilities, respectively. Note that we do not need to consider situations in which the boundary of the net is not on $x_{u'}$, since in these cases, the movement of u does not determine the final HPBB, and hence will have 0 cost. $p_e^-(x_{u'}, n_j)$ is equal to the probability that all other cells in n_j besides u are moved to the left of $x_{u'}$, which can be calculated as:

$$p_e^-(x_{u'}, n_j) = \prod_{u_i \in (n_j - u)} \int_{-\infty}^{x_{u'}} \Phi_{u_i}(x) \tag{14}$$

Note that $\int_{-\infty}^{x_{u'}} \Phi_{u_i}(x)$ is the probability that the final position of a cell u_i is to the left of $x_{u'}$.

For $x_{u'} < x_l$, $p_e^+(x_{u'}, n_j)$ is equal to the probability that all the other cells in n_j are moved to the right side of $x_{u'}$. Thus we have the formulation:

$$p_e^+(x_{u'}, n_j) = \prod_{u_i \in (n_j - u)} \int_{x_{u'}}^{\infty} \Phi_{u_i}(x) \tag{15}$$

Bench.	CPD+FAD		CPD		FAD		FP	
sets	%ΔT	%ΔWL	%ΔT	%ΔWL	%ΔT	%ΔWL	%ΔT	%ΔWL
TD-IBM	18.3	-5.0	17.9	-5.6	18.1	-5.9	18.0	-6.6
Faraday	13.8	-7.7	13.5	-8.7	14.0	-9.1	13.5	-9.1
TD-Dragon	6.8	-6.9	6.8	-7.7	6.6	-7.8	6.3	-8.3
Avg.	15.7	-5.8	15.5	-6.5	15.6	-6.8	15.4	-7.3

TABLE II

Average results obtained using different WL cost determination methods with a combined cost of WL and timing in detailed placement for the three sets of benchmarks. Positive numbers in the table indicate improvement, and negative numbers indicate deterioration. This nomenclature will be used in the rest of the tables that show improvement or deterioration of metrics. The initial global placement was done also using a combined WL and timing cost as described in Sec. IV.

Bench.	CPD+FAD		CPD		FAD		FP	
sets	%ΔT	%ΔWL	%ΔT	%ΔWL	%ΔT	%ΔWL	%ΔT	%ΔWL
TD-IBM	12.7	-4.3	12.5	-4.7	12.6	-4.9	12.0	-5.6
Faraday	10.6	-6.5	10.4	-7.5	9.9	-7.6	9.6	-8.2
TD-Dragon	3.6	-6.1	3.3	-7.0	3.5	-7.2	3.2	-7.5
Avg.	10.9	-5.0	10.7	-5.6	10.7	-5.7	10.3	-6.3

TABLE III

Average results obtained using different WL cost determination methods with a purely WL cost in detailed placement for the three sets of benchmarks. The initial global placement was done using a combined WL and timing cost as described in Sec. IV.

If $x_{u'} \geq x_l$, since in this case moving u to u' cannot increase the HPBB of n_j, $p_e^+(x_{u'}, n_j)$ is set to be 0.

The average unit flow cost $cost_e(x_{u'}, n_j)$ of e (flow amount on e is equal to the distance of movement of u) corresponding to HPBB WL increase of n_j when u is moved to u' is thus given as:

$$cost_e(x_{u'}, n_j) = [p_e^+(x_{u'}, n_j) \cdot (x_{u'} - x_l) - p_e^-(x_{u'}, n_j) \cdot (x_r - x_{u'})]/(x_u - x_{u'}) \quad (16)$$

where a negative sign is used for $p_e^-(x, n_j)$ since it represents possible reduction of HPBB. The above equation clearly shows the flow amount dependency of unit flow cost as illustrated before, i.e., the unit flow cost with different final position u' is different. To solve this dependency, the final WL unit flow cost $cost_l(e)$ of e is given by the summation of the probabilistic average of unit flow cost for different final position of e considering each net connected to u:

$$cost_l(e) = \sum_{n_j : u \in n_j} \int_{x_b}^{x_u} \Phi_u(x) \times cost_e(x, n_j) \quad (17)$$

In our detailed placement, to reduce the final WL deterioration, the cost $cost(e)$ of an arc e is a combination of its WL and timing cost as given below:

$$cost(e) = w_2 \cdot cost_t(e) + (1 - w_2) \cdot cost_l(e) \quad (18)$$

Recall that $cost_t(e)$ is the timing cost of e (Eqn. 12). w_2 is the parameter that determines the weight between WL and timing cost. Based on experimental results, w_2 is chosen to be 0.8 for the best timing improvement results with little WL deterioration.

D. Experimental results with four different WL cost functions

To establish the effectiveness of our WL cost calculation method, we also implement three competing methods, each of which modifies one part of our method.

In the first competing method termed FP (for fixed probability), we do not use a global network-flow process to estimate the probability that a cell is moved to a certain position. Instead, for each cell, we assign a fixed probability $\eta < 1/2$ of it moving to the furthest position determined by the arc capacity corresponding to the movement. Thus, a cell u has a probability of η to move to each of the two horizontal and two vertical limit positions, and thus a probability of $1 - 2\eta$ to remain at its original horizontal (vertical) coordinate. With this simplified probability model, we can obtain different probability density functions $\Phi_u^{fix}(x)$ and $\Phi_u^{fix}(y)$ for the final position of u, and thereby determine $cost_l(e)$ (Eqn. 17). Choosing $\eta = 0.4$ gives the least WL deterioration in a purely WL-driven detailed placement.

In the second competing method CPD, only the *adjacent cell position dependency* (CPD) is resolved, while the flow amount dependency is ignored. Thus, to calculate the unit flow cost of an arc e starting from cell u, instead of using Eqn. 17, we will only use the average unit flow cost of a full flow on e, which is $\sum_{\{n_j : u \in n_j\}} cost_e(x_b, n_j)$.

In the third competing method FAD, only the *flow amount dependency* (FAD) is resolved, while the adjacent cell position dependency is ignored. In this method, we take the adjacent cells of u as fixed at their original positions. Thus, no probability is needed to calculate $cost_e(x_{u'}, n_j)$, since we can deterministically check how much a movement of u to u' will change the HPBB of n_j.

The average WL and timing results of using the three competing methods, FP, CPD and FAD, as well as our method (labeled as CPD+FAD) are given in Tables II and III. We ran our experiments on three sets of benchmarks listed in the table, starting from a global placement with a combined WL and timing cost (see Sec. IV). Table II gives the resulting percentage changes on timing and WL when using the combined WL and timing arc cost of Eqn. 18 in detailed placement, while Table III gives the results for using only WL arc cost in detailed placement—the latter allows a more accurate comparison of the WL effect of the four approaches. In both cases, we can see that the final timing improvement does not vary too much across different WL cost formulations, but CPD+FAD outperforms the competing methods in reducing WL deterioration by relative amounts of 10-20%.

VII. Flow Discretization for Legal Incremental Placement Solution

As mentioned earlier, the core incremental detailed placement problem is a DOP, and thus certain illegalities are introduced in it by using a continuous optimization method like network flow to solve it. We discuss two main illegality issues and the *in-processing* discretization techniques that we have developed to deal with them, i.e., techniques that work simultaneously with the network-flow algorithm. The discretization techniques require multiple iterations of the network flow process (as also does the WS satisfaction process, which we discuss shortly in Sec. VIII). We thus introduce a global network-flow process to first quickly guide the flow into appropriate "sectors" of the placement to achieve our objectives (cost minimization, discretization and WS satisfaction) at a global level. This is then followed by the detailed network flow process introduced in Sec. V in which the guidelines of the global flow are followed to reduce time to solution. This results in a faster convergence to a good solution through multiple iterations of alternating global and detailed network flows (note that the global flow process is also used for WL arc-cost determination, as described in Sec. VI).

A. Discrete flow requirement in vertical arcs

Figure 7(b) shows a vertical arc (u, v) from cell u to v of capacity $w(u) = 5$ and unit-flow cost of c_1. This arc is used to model the possible movement of u to the row immediately above it (and thus to the position of v). The physical interpretation of any flow along (u, v) has to be that u is moved to v's location, since any position in between its current position and that of v's is illegal. Thus the exact requirement of the flow amount through (u, v) should be either 0 (no movement of u) or $w(u) = 5$. Furthermore, any flow of $x < w(u)$ through (u, v) will also incur an inaccurate lower cost of $x \cdot c_1$ rather than the "full cost" of $w(u) \cdot c_1$, incurred in actually moving u to $v's$ position. The resulting inaccuracies in cell movements implied by these flows is shown in Fig. 7(c).

We rectify these inaccuracies, by initially having a capacity of 1 and cost $= w(u) \cdot c_1$ (the full cost) for (u, v) as illustrated in Fig. 7(d). When a flow of 1 passes through (u, v) correctly incurring the full cost of (u, v), we update (u, v)'s capacity to $w(u) - 1$ and cost to 0, thus correctly allowing an additional flow of $w(u) - 1$ to pass through it at no cost. Note also that any flow entering u can exit from either the two horizontal arcs or the two vertical arcs including (u, v) that emanate from u. Note that even with a flow of 1 through (u, v), in the physical interpretation we will move u to v's position, and thus v will be shifted to it's left or right by a distance of $w(u)$ to remove its overlap with u. The resulting costs of these movements in the incremental placement of the cells affected by the flow of 1 through (u, v) will thus be incurred, irrespective of whether or not there is any more flow on (u, v). Hence for the rest of the flow coming into u, if any, we encourage $w(u) - 1$ of it to go through (u, v) by the following mechanism: (i) change arc (u, v)'s cost to 0 (this is correct since the entire cost of arc (u, v) corresponding to a full flow has already been incurred); and (ii) maintain positive costs for the two horizontal arcs from u (see Fig. 7(e-f)), as well as for the other vertical arc out of u. Only after a flow of $w(u) - 1$ passes through (u, v), do we make the cost of the horizontal arcs 0 (since u is no longer in this row) and their capacity ∞; see Fig. 7(g). Fig. 7(h) shows the correct cell movements implied by the resulting flow of Fig. 7(g).

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

B. Split Flows

Since a flow on a horizontal or vertical arc out of a cell u represents movement of u in the direction of the arc, as far as the incremental placement DOP is concerned, a flow into u can come out of at most one outgoing arc from u.

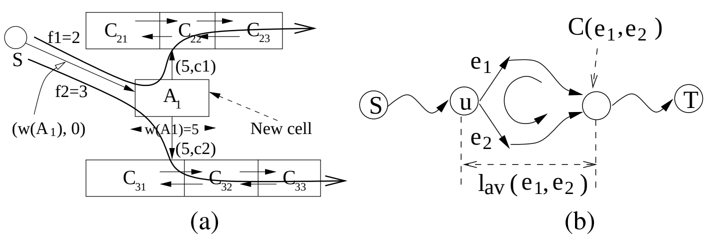


Fig. 8. *(a) Split flow through new cell $A1$. (b) Diverting flow from e_1 to e_2 in a split flow situation.*

In other words, if there are k arcs $e_1^{out}(u), \ldots, e_k^{out}(u)$ out of u ($k \leq 4$), and if we associate 0/1 variables $b_i^{out}(u)$ and real variables $f_i^{out}(u)$ with each $e_i^{out}(u)$, where $b_i^{out}(u) = 0/1$ means there isn't/is a flow on $e_i^{out}(u)$ and $f_i^{out}(u)$ is the amount of flow on $e_i^{out}(u)$ when $b_i^{out}(u) = 1$, then the ILP constraint required to accurately model the incremental placement DOP represented by the flow graph is $\sum_{i=1}^{k} b_i^{out}(u) \leq 1$. Similarly, let $e_1^{in}(u), \ldots, e_k^{in}(u)$ be the input arcs to u with corresponding 0/1 flow controlling variables $b_i^{in}(u)$ and real variables $f_i^{in}(u)$ representing possible flow amount for each $e_i^{in}(u)$. Then an integer quadratic programming (IQP) formulation that conserves flow (input flow into a node = output flow from the node) is

$$\sum_{i=1}^{k} b_i^{in}(u) \cdot f_i^{in}(u) = \sum_{i=1}^{k} b_i^{out}(u) \cdot f_i^{out}(u).$$

The above two formulations (ILP+IQP) together form an IQP and represent an accurate modeling of cell re-placement in the incremental placement DOP.

However, the IQP problem is well-known to be NP-hard [9], and such a formulation of the incremental placement problem would be intractable. The continuous optimization solution we obtain to this problem via the network flow model is in P and much faster. Of course, it has no restriction on how many outgoing arcs from a node can have a flow, resulting in what we term *split flows* when more than one output arc from a node has positive flows; see Fig. 8(a).

If after one iteration of network flow there are split flows from node u, then u will not be moved in this iteration, and in the following iterations (note that usually we cannot obtain a legal placement in one iteration of network flow process due to the split flow and white space constraints that we will mention shortly), we will allow only one branch of the split flows from u and forbid all the other possible outgoing flows by changing the capacity of all outgoing arcs from u to 0 except the arc carrying the allowed flow. We have used two alternative heuristics to choose the allowed branch:

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

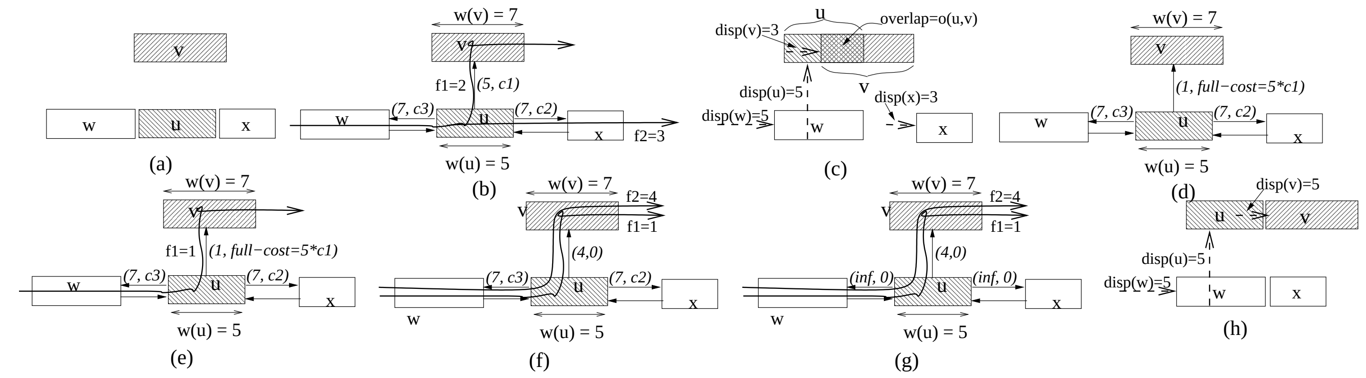


Fig. 7. *(a) Initial placement. (b) "Regular" cost and capacity of vertical arc (u, v) and two flows through cell u. (c) The physical translation of this flow leading to inaccurate incremental placement of affected cells; $disp(v)$ is the displacement distance of v. (d)-(h) New cost, capacity structure of arc (u, v) with dynamic update, resulting in a flow more closely mimicking the corresponding physical movement of cells, and the final accurate incremental placement of affected cells in (h). The dashed arrows in (c) and (h) represent displacements of cells at the end of the arrows.*

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

TABLE IV
Percentage timing improvements of max-flow heuristic, min-cost heuristic with different path length limits and the random selection approach for split flow prevention.

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

Claim 1: Suppose that in a network at any stage of the flow-augmentation process (passing of some of the remaining flow through the n/w), the average unsaturated arc-cost per unit flow is monotonically non-decreasing w.r.t. flow[5], i.e., $d(av\_cost)/df \geq 0$ and is the same (or very similar) across the network, where av_cost is the average cost among all currently unfilled arcs, and f is the current amount of flow. Then the max-flow heuristic for split-flow prevention will always give a solution cost that is smaller than or equal to that yielded by the min-cost heuristic.

Proof: We consider a split flow at any node u with the flow passing through two outgoing arcs from u, e_1, e_2, with $f(e_1) > f(e_2)$. If a flow of value 1 is diverted from e_1 to e_2 the additional cost encountered by that flow will be $[d(av\_cost)/df] \times l_{av}(e_1, e_2)$, and vice versa, where $l_{av}(e_1, e_2)$ is the average length across all nodes u of the subpath from u to the nearest common descendant node $C(e_1, e_2)$ of e_1 and e_2; see Fig. 8(b). However, since there is more flow on e_1, the total cost of diverting all the flow from e_1 to e_2 will be more than the total cost of diverting all the flow from e_2 to e_1. Thus if any other heuristic, including min-cost, chooses to divert flow from e_1 to e_2, then it will result in a larger-cost split-flow prevention solution than that obtained using the max-flow heuristic. On the other hand, if another heuristic chooses to divert flow from e_2 to e_1, then its cost will be the same as that of the max-flow heuristic. $\diamondsuit$

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

Empirical results using the two heuristics shown in Table IV bolsters the above analytical result. The percentage timing improvement of six representative incrementally placed circuits in Table IV for the TD-IBM benchmark [7] reveals that the max-flow heuristic performs consistently better than the min-cost heuristic with path length limits of 3, 4, 5, 20, 50 and 100 arcs, and has a relatively better performance in the range of 13.4-6.0%. Table IV also shows results for a randomized arc selection approach for split flow prevention; it performs significantly worse than the above two heuristics. The max-flow heuristic is thus implemented in our algorithms.

Another very telling statistics that we have collected indicates the efficacy of both the network-flow approach and the discretization technique of the max-flow heuristic for split-flow prevention for obtaining solutions to the VLSI CAD DOPs of TD and timing plus WL-driven incremental placement. If we compare the final flow cost after the split-prevention discretization imposed on the necessary arcs (via the max-flow heuristic) to the final flow cost of a solution with unrestrained splitting (and thus illegal for our DOPs), then the discretized flow is within only 2% of the optimal (but illegal) continuous solution for the timing optimization DOP and only 3% for the timing + WL optimization DOP. These are strong empirical results for the efficacy of our discretized network flow approach for the above two VLSI CAD DOPs, and indicate significant promise for the efficacy of our discretized network flow approach for solving general VLSI CAD DOPs.

C. Global network flow

Flow iterations in the detailed flow graph can be very time-intensive due to the large numbers of nodes and arcs in it. If we can instead find a good direction for *global flows* to go between rows and from rows to the sink T (via their WS arcs), rather than solving the flow problem at the level of detail of the myriad number of individual arcs, then we can follow such a fast global flow with a more precise flow in a subgraph of the detailed flow graph that includes only rows and arcs that span the rows spanned by the global flow.

The *global flow graph* (see Fig. 9) is a directed graph $F_G(V, E)$, where $V = moveC \cup \{v(R_i) | v(R_i)$ is a node representing row $R_i\} \cup \{S, T\}$; recall that $moveC$ is the set of new or moveable cells. There are vertical arcs between the row nodes $v(R_i)$'s of adjacent rows, and from the new cells to the $v(R_i)$'s of their adjacent rows, from the source S to the new cells, as in the detailed flow graph, and to violating row nodes, and finally arcs from row nodes with WS to the sink T. The capacities and costs of relevant arcs are shown in Fig. 9. The capacity of a vertical arc between two row nodes $v(R_i)$ and $v(R_{i-1})$ is W_{max}, the maximum allowable row size, and its cost $C_{i,i-1}$ is the weighted average of the detailed vertical arc costs between the two rows:

$$C_{i,i-1} = \frac{\sum_{e \in (i,i-1)} cost(e) \cdot cap(e)}{\sum_{e \in (i,i-1)} cap(e)}$$

Recall that $cost(e)$ is the cost of arc e, and $cap(e)$ is the capacity of e (its start cell width). The capacity of an arc from $v(R_i)$ to T is the WS width $w(W_i)$ in the row. The cost C_i of this arc is the probabilistic average of all left-to-right detailed horizontal arc costs in R_i. The probability $p_f(e)$ of a horizontal left-to-right arc e in the detailed flow graph being crossed by a flow is given by the ratio between the width of the row segment to the left of the right boundary of the cell and the maximum allowable row width. This probability assumes that a flow into the row can come in at any point with uniform probability. Thus, the cost is given by:

$$C_i = \frac{\sum_{e \in R_i} cost(e) \cdot p_f(e)}{\sum_{e \in R_i} p_f(e)}$$

The iterations of alternating global and detailed flows proceed as follows.

```
While not (all new cells pushed and all rows free of WS violations) do begin
1. Construct a global flow graph;
2. Determine a min-cost max-flow in it;
3. Construct a subgraph of the detailed flow graph induced by the global flow;
4. Determine a min-cost max-flow in it;
5. Perform translation of the detailed flow into corresponding cell movements;
End while.
```

Let N be the number of cells in a circuit. Our detailed network flow graph for the circuit usually contains about $4N$ nodes and $8N$ arcs. Across the iterations, the sizes of global flow graphs are between 10%-50% of those of the detailed network flow graphs. This yielded a run-time reduction by about 65% compared to using only the detailed flow graph, at the cost of about 1-2% delay deterioration. Furthermore as we mentioned in Sec. VI, the global flow process is also useful to determine accurate WL cost of arcs.

VIII. Satisfying White Space Constraints

It would seem that row white space constraints are automatically satisfied due to the structure of flows through white-space cells and the row WS node which have outgoing arcs equal to the amount of WS they represent. However, the problem arises due to the continuous nature of flows through vertical arcs, whereas the requirement for the detailed placement problem

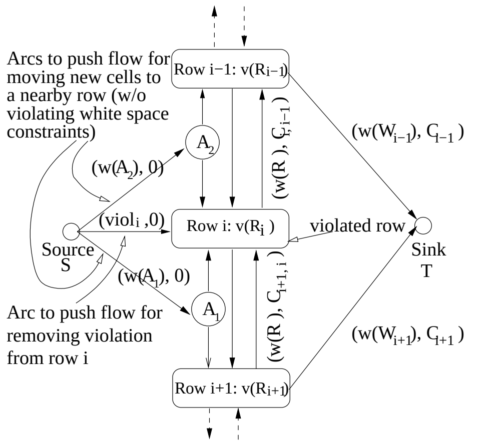


Fig. 9. *The global flow graph corresponding to a row violation correction action and insertion of new cells into legal row positions; arcs are marked with their* $(capacity, cost)$.

is for discrete (2-value) flows through them as discussed in Sec. VII. Referring to Figs. 7(e-f), assume that u is in R_i, v in R_{i-1}, and that the total WS in R_{i-1}, $w(W_{i-1})$, is 3. A total flow of $f = 2$ (note that a total flow of 5 depicted in Figs. 7(e-f) may not be available) coming from the left of R_i into u and then to v and right into the WS node of R_{i-1}, and then to the sink T will be allowed. However, if that is the only flow on arc (u, v), then the problem comes in the translation of this flow into cell movements[6]—when u is actually moved up to R_{i-1}, since $w(u) = 5$, there will actually be a WS violation in R_{i-1} of $w(u) - w(W_{i-1}) = 2$. In other words, the total flow into a row may not always equal the total actual cell area moved into the row in the translation stage.

In our detailed placement technique, we dynamically track the change in the WS amount of each row for every flow augmentation (in some cycle C) in order to detect WS violations. Let the current amount of available WS in row R_i be $w(W_i)$. For a flow augmentation f, let w_i^{out} denote the total area of cells that are moved out of R_i (as per f), and, similarly, let w_i^{in} denote the total area of cells moved into R_i. Due to flow f, $w(W_i)$ is changed by an amount of $w_i^{out} - w_i^{in}$. A WS violation in R_i occurs if $w(W_i)$ becomes negative as a result. Similarly, if $w(W_i)$ changes from negative to positive, then it means that an earlier WS violation in R_i has been removed. An example of $w(W_i)$ change caused by a flow is shown in Fig. 10.

In the rest of this section, we explain the network flow structure used for removing WS constraint violations. We also propose two flow control policies, a *violation control policy* and a *thrashing control policy* to guide the WS constraint satisfaction process. These two polices allow limited but necessary temporary WS violations while proceeding towards a legal placement. With these two policies, we can guarantee the termination of our WS constraint satisfaction process, while providing a large freedom for cell movement in the process. The pseudo code for applying these two policies is

[6]Note that the overlap of u and v shown in Fig. 7(c) is not the issue here, as given enough WS in R_{i-1}, v and subsequent cells to its right can be moved to the right to remove all overlaps without violating the WS constraint in R_{i-1}.

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

Fig. 11. *Pseudo code of the violation policy check that is performed for each flow augmentation in a cycle in each network flow iteration.*

given in Fig. 11.

A. Fixing WS violations in multiple network flow iterations

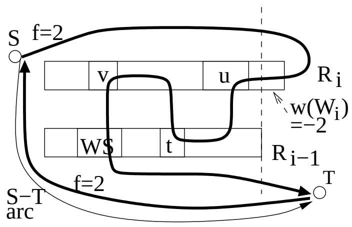


Fig. 10. *A flow f, shown by dark curves, of amount 2 is pushed from S to R_i to remove its WS violation of 2 ($w(W_i)$=-2); see Sec. VIII-A. Let the widths of cells u, t, v be 4, 3, 2, respectively. Flow f implies that u and v are moved to R_{i-1}, and t is moved to R_i. Hence, $w_i^{out} = 6$ and $w_i^{in} = 3$. $w(W_i)$ thus becomes 1 due to flow f, which means that the violation in R_i is removed. The S-T arc shown is an auxiliary arc needed for the Simplex method [2].*

We first introduce the *violation correction structure* to remove a violation in a row. This structure is shown in Fig. 12. An arc is added between the source S and the rightmost cell of each violated row R_i with violation $viol_i$. The capacity of the arc is $viol_i$, and an extra flow of this amount, termed a *violation correction flow*, emanates from S in order to push the violated amount of cell size away from the row. As a result, the detailed flow graph will have a structure that is a combination of that shown in Fig. 3(a) for non-violating rows and Fig. 12 for violating rows. There will also be corresponding violation correction arc(s) in the global network flow graph as shown in Fig. 9 between S and $v(R_i)$ for a violated row R_i.

In order to provide a richer solution space, we allow some intermediate WS violations so that we can arrive at good WS-satisfying solutions. Intermediate WS violations are allowed in a controlled manner using a *violation control policy*, described below, that guarantees that our algorithm will terminate (see Theorem 1 given later). Under this violation control policy, we fix WS violations through multiple iterations of the network flow process. WS violation can be caused by legalizing cells in $moveC$ or by flow discretizations, explained in the previous section, and is allowed or disallowed as per our violation control policy. If after an iteration, there are rows with WS violation, we will attach a violation correction structure to these rows in the network flow graph of the next iteration.

B. Violation and thrashing control policies

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

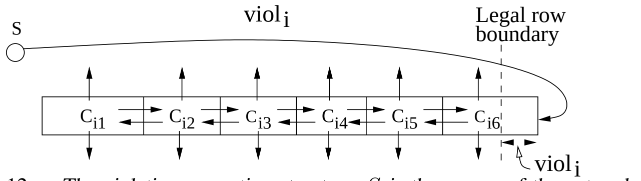


Fig. 12. *The violation correction structure; S is the source of the network graph.*

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

As noted above, we have a two-sided violation control policy: if moving cells into a row from one direction is blocked by the white space violation control policy, moving in cells from the other direction can still take place. This allows a significant amount of flexibility of cell movements that can result in WS satisfaction in other rows, legalization of illegally placed cells, and metric optimization.

One problem of allowing temporary WS violation, even in a limited and controlled way as stated in our violation control policy, is that it may cause infinite *thrashing* in the network flow process, in which some cells may be moved between two rows repeatedly during consecutive network process iterations for violation removal. We define a *forbidden cell* for row R_i to be a cell that when moved into a non-violated R_i causes it to become violated. Our inter-iteration thrashing control policy to cope with this problem is:

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

This policy (unlike the violation control policy) is an inter-iteration policy, i.e., we retain the cell and row states from one iteration to the next. A flow is allowed only if it passes both the violation control policy and the thrashing control policy. Our method terminates either if in the current network flow iteration there is no allowed flow (this is an *unsuccessful termination*), or if all illegally-placed cells are legalized and if all WS violations are eliminated (a *successful termination*). [18] provides a detailed proof that our WS satisfaction process is guaranteed to terminate; this result is stated next.

Theorem 1: [18] With the thrashing control policy and the violation control policy as stated above, our incremental detailed placement method is guaranteed to terminate.

In our experiments our method terminates within 7-8 iterations for each of the 27 circuits we have used, and successfully satisfies all WS constraints. Further, while our method is not guaranteed to find a WS-satisfying solution if one exists, it will, however, find such a solution with very high probability as we prove in [18].

IX. Experimental Results

We used three benchmark suites in our experiments: 1) TD-IBM benchmark suite [7] with circuit size ranging from 13k-210k cells, 2) Faraday benchmarks from [1] with circuit size ranging from 12k-33k cells and 3) The TD-Dragon suite of [21] with circuit size ranging from 3k-26k cells. The third set of benchmarks has complete cell and timing information. The first two sets have no cell delay information. Hence, cell delay is set to be zero for the first two sets. All benchmarks are initially placed by Dragon 2.23 (the TD Dragon benchmarks are also placed by TD-dragon which can only place this suite); see [7] for more information of the benchmarks like delay, total WL and initial placement run time. We then identify paths with delays of at least 90% of the max-path delay as *critical paths*. The γ value (Eqn. 3) is set to 1. Except for TD-Dragon benchmarks which are run with whitespace (WS) of 10% (since these circuits are relatively small), all other benchmarks are allocated a 3% WS. We use typical values for R_d, C_g, r and c for 0.18 μm technology[7]. The unit length for the TD-IBM benchmarks was taken as 0.1125 μm, and that for the Faraday ones as 0.0005 μm. Results were obtained on Pentium IV machines with 1GB of main memory.

Table V shows that if the initial placement is done by a state-of-the-art WL-driven placer (Dragon 2.23 in our experiments), our purely timing-driven method (without any WL cost consideration) can achieve up to 33% and an average of 17% delay improvement. The TD global placement (TAN) results, also given in the table, show an average timing improvement of about 22%. Thus there is an absolute deterioration of only 5% after detailed placement underscoring the efficacy of our network flow based detailed placer. The table also shows the effect of our combined cost function with both timing and WL costs. Compared to purely TD method, the WL increase is reduced from 9.0% to 5.8% (a 35% relative reduction), while the delay improvement is reduced by only 1.6% from 17.3% to 15.7% (a 9% relative reduction). On the other hand, compared to using a purely WL cost in detailed placement (see Table III), the combined cost gives an average timing improvement that is relatively about 44% better (15.7% vs. 10.9%), while the WL deterioration is relatively only 16% worse (-5.8% vs. -5.0%). These results underline the effectiveness of the combined cost formulation in that it provides good timing improvement (little worse than purely TD cost and much better than purely WL-driven cost) and low WL deterioration (little worse than purely WL-driven cost but significantly better than purely TD cost.)

Tables VI and VII show that starting with circuits placed by a TD placer (TD-Dragon), we can still improve results appreciably. Since the TD-Dragon benchmark circuits are relatively small, we use a larger WS constraint of 10% here. With purely TD cost, we obtain up to 10% and an average of 6.5% timing improvement over TD-Dragon results, while

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

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

TABLE V

Results for circuits initially placed by Dragon. The second and third columns are the global placement (TAN) results for a purely TD cost. The fifth and sixth columns are the final results (after detailed placement) for a purely TD cost. The last two columns are the final results for the combined cost. The WS constraint for all results is 3%, which means that the final maximum row width can increase by no more than 3%. We only show the run time for purely TD placement; the runtime for combined-cost placement is less than 10% different from the TD runtime.

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

TABLE VI

Incremental placement results based on initial placement by TD-Dragon. Both purely TD cost and combined cost results are listed. The WS constraint is 10%.

the average WL increase is 6.1% (see Table VI). However, as shown in the same table, the WL increase can be reduced to 5.3% with the combined cost (an average relative reduction of 13.1% compared to the purely TD cost), and the corresponding average timing improvement is about 6.3% (an average relative reduction of only 3% compared to the purely TD cost).

For the TD-Dragon circuits, we also used the linear-delay model of [20] to compare our results to them. The WS constraint is also set to 10% as in [20], and the results are listed in Table VII. With a purely TD cost, under this delay model our technique obtains up to 9% and an average of 4.5% timing improvement over TD-Dragon using purely TD cost, while the average WL increase is about 6.3%. The WL increase can be reduced to about 5.5% with the combined cost, and the corresponding average timing improvement is about 4.0%. For the same set of benchmarks, an average of only

Ckt (TD)	**10%WS (TD)**		**10%WS (Comb.)**		**Results in [20] w/ 10% WS**	
	%ΔT	**% ΔWL**	**%ΔT**	**%ΔWL**	**%ΔT**	**%ΔWL**
matrix	0.10	-8.20	0.10	-7.63	no impr.	N/A
mac32	4.77	-2.51	3.95	-2.33	3.8	-10.5
mac64	9.01	-4.05	8.35	-3.47	7.5	-10.9
vp2	4.12	-10.51	3.39	-8.7	no impr.	N/A
Avg.	**4.5**	**-6.32**	**4.0**	**-5.52**	**2.8**	**-10.7**

TABLE VII

Results using the same delay model as [20]. The runtime is similar to using our delay model as shown in Table VI.

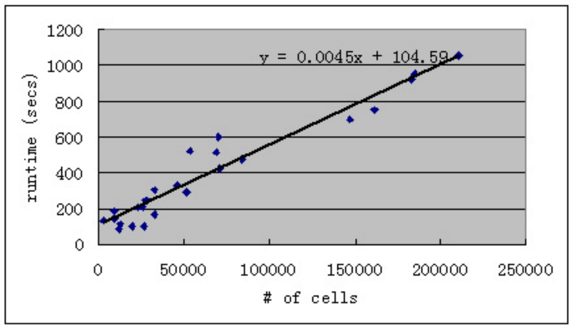


Fig. 13. *Run time vs. total number of cells*

2.8% timing improvement is obtained with around 10% WL increase in [20]. Thus our results are 43-60% relatively better in the timing metric than that of [20], and 41-48% relatively better than it in the WL metric.

Our methods are also quite fast compared to a full placement; we complete incremental placement in about 10% of the initial placement time of Dragon 2.23 and in about 6% of the time of TD-Dragon. Figures 13 and 14 show the empirical asymptotic time complexity of our incremental placer w.r.t. both the number of cells and the number of moved cells; we find that both run time plots best match linear functions, which underscores the timing efficiency and scalability of our techniques, as well as the advantage of using our discretized models of network flow to solve the complex DOP of TD/WL-driven detailed placement.

X. Conclusions

We presented various novel and effective techniques for TD/WL-driven incremental placement. These include: 1) pre-routed net-delay estimate functions; 2) performing both quadratic and linear optimization simultaneously in our TD analytical placer; 3) delay sensitivity and allocated-slack-based timing-driven flow arc costs in network flow based detailed placement; 4) probability-based WL flow arc cost suitable for the HPBB model; 5) the global and detailed flow graph structures to perform optimization driven (TD/WL-driven) cell placement; 6) new techniques for WS constraint satisfaction during detailed placement; and 8) network flow discretization techniques for solving a discrete optimization problem (TD/WL-driven incremental placement) by network flow, a fast continuous optimization method. The end-result is a robust, effective and efficient TD/WL-driven incremental placer that achieves significant delay improvements in quick time with small WL deterioration on circuits placed by state-of-the-art WL (Dragon 2.23) and TD (TD-Dragon) placers. Our methods also scale well with circuit size, e.g., in a purely timing-driven mode we obtained more than 36% delay improvement for a 210K cell circuit td-ibm18 in less than 18 minutes.

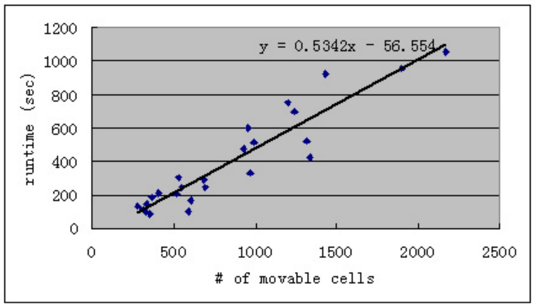


Fig. 14. *Run time vs. total number of movable cells*

XI. Acknowledgements

We thank Fenghua Yuan and Vishal Suthar for some initial codings in TD FlowPlace and related discussions.

References

[1] S. N. Adya, S. Chaturvedi, J. A. Roy, D. Papa and I. L. Markov. "Unification of Partitioning, Floorplanning and Placement",*Proc. Int'l Conf. CAD (ICCAD)*, 2004, pp. 550-557.
[2] R. Ahuja, J. Orlin, P. Shama and P. Sokkalingam, "A network simplex algorithm with O(n) consecutive degenerate pivots", *Operations Research Letters*, 1995, pp. 1417-1436.
[3] D. P. Bertsekas and J. N. Tsitsiklis, "Introduction to Probability", American Mathematical Society, 2002.
[4] U. Brenner, A. Pauli and J. Vygen. "Almost optimum placement legalization by minimum cost flow and dynamic programming", *Proc. Intl. Symp. on Physical Design (ISPD)*, 2004, pp. 2-9.
[5] K. Doll, F. Johannes and K. Antreich, "Iterative placement improvement by network flow methods", *IEEE Trans. Computer-Aided Design*, 1994, pp.1189-1200.
[6] S. Dutt and H. Ren, "An Accurate Multi-Star Based Pre-route Wire Length Model", Technical Report, UIC, 2008. Available at www.ece.uic.edu/~dutt/papers/multistar-wl.pdf
[7] S. Dutt, H. Ren, F. Yuan and V. Suthar, "A Network-Flow Approach to Timing-Driven Incremental Placement for ASICs", *Proc. Int'l Conf. CAD (ICCAD)*, 2006, pp. 375-382.
[8] W. C. Elmore. "The Transient Analysis of Damped Linear Networks with Particular Regard to Wideband Amplifiers", *J. Applied Physics*, vol. 19(1), 1948.
[9] C. A. Floudas and V. Visweswaran, "Quadratic Optimization", in *Handbook of global optimization*, Kluwer Acad. Publ., Dordrecht, 1995, pp. 217–269.
[10] S. Ghiasi, E. Bozorgzadeh, S. Choudhuri and M. Sarrafzadeh, "A unified theory of timing budget management", *Proc. Int'l Conf. CAD (ICCAD)*, 2004, pp. 653 - 659.
[11] P. S. Hauge, R. Nair and E. J. Yoffa, "Circuit placement for predictable performance," *ICCAD*, 1987, pp. 88-91.
[12] A. Kahng, S. Mantik and I. Markov, "Min-Max placement for large scale timing optimization", *ISPD*, 2002, pp. 143-148.
[13] A Kahng, I. Markov and S. Reda "Boosting: min-cut placement with improved signal delay" *DATE*, 2004, pp 1098-1103.
[14] J. Kleinhans, G. Sigl, F. Johannes and K. Antreich, "GORDIAN: VLSI placement by quadratic programming and slicing optimization", *IEEE Trans. Computer-Aided Design*, Mar. 1991, vol. 10, pp.356-365.
[15] A. Marquardt, V. Betz and J. Rose, "Timing-Driven Placement for FPGAs", *ACM/SIGDA Int'l Symp. FPGAs*, 2000, pp. 203-213.
[16] W. Naylor, "Non-linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer". US Patent 6301693, Oct. 2001.
[17] Shih-Lian Ou and M. Pedram,"Timing-driven Placement Based on Partitioning with Dynamic Cut-net Control", *Proc. Design Aut. Conf (DAC)*, 2000, pp. 472-476.
[18] H. Ren and S. Dutt, "A Provably High-Probability White-Space Satisfaction Algorithm with Good Performance for Standard-Cell Detailed Placement", submitted to *IEEE Trans. on VLSI.* Available at www.ece.uic.edu/~dutt/papers/ws_sat.pdf
[19] G. Sigl, K. Doll and F. Johannes. "Analytical placement: A linear or a quadratic objective function?", *Proc. Design Aut. Conf (DAC)*, 1991, pp. 427-432.
[20] Choi Wonjoon and K. Bazargan, "Incremental placement for timing optimization", *ICCAD*, 2003, pp. 463-466.
[21] Yang Xiaojian, Choi Bo-Kyung and M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint generation", *Proc. Int'l Conf. CAD (ICCAD)*, 2002, pp. 177-180.
[22] H. Yang and D. F. Wong,"Efficient Network Flow Based Min-cut Balanced Partitioning", *IEEE Trans. CAD*, 1996, pp. 1533-1540.