Timing-Driven Incremental Routing for VLSI Circuits Via Localized Slack Satisfaction Computations

ABSTRACT

Incremental routing is crucial to incorporating engineering change orders (ECOs) late in the design cycle to correct problems ranging from design errors to timing violations to crosstalk in current very deep submicron (VDSM) circuits. Most previous work in incremental routing addressed some combination of wirelength (WL), via and routing-layer minimization. In this paper, we address the critical objective of satisfying timing constraints in current high-speed designs by developing an effective timing-driven (TD) incremental routing algorithm TIDE for ASIC circuits. There are two main novelties in our approach. The first, at the global routing level, is a novel interval-intersection and tree-truncation algorithm for quickly determining a near-minimum-length slack-satisfying interconnection of a pin to a partial routing tree using only local slack-type information. The second is a mechanism for keeping relevant slack tolerance information at every Steiner node of a net's routing tree that helps to quickly and locally determine in $\log k$ time (for balanced trees; k is the number of pins in the net) whether any re-route of a segment satisfies all the net's pin slacks. In the detailed routing phase, using a depth-first-search process, we allow new nets to bump existing nets in order to have a richer solution space, while also controlling any metric deterioration due to net bumpings. Experimental results show that within the constraint of routing all nets in only two metal layers, TIDE succeeds in routing more than 94% of ECO-generated nets, and also that its failure rate is 7 and 6.7 times less than that of the TD versions of previous incremental routers Standard (Std) and Ripup&Reroute (R&R), respectively. It is also able to route nets with very little (3.4%) slack violations, while the other two methods have appreciable slack violations (16-19%). TIDE is about 2 times slower than the simple TD-Std method, but more than 3 times faster than TD-R&R.

1. INTRODUCTION

Past work tackling the incremental routing problem include [1, 2, 3, 4, 5, 10]. In [1], re-routing is done in a standard single-net routing mode in the available routing space without disturbing existing net routes; we term this incremental routing approach *Standard (Std)*. Limited search space is the major disadvantage of this scheme. The *ripup and reroute (R&R)* approach to incremental physical design was presented in [2]. Routings of new nets are initially attempted without perturbing any existing net. When a new net cannot be routed, some of the existing nets are ripped-up to free up routing resources. The routing is then re-done for the new nets followed by the ripped-up nets. If R&R fails to route all the nets, the given floorplan and placement configurations are needed to be modified. The main disadvantage of a R&R scheme is that the routing is no longer truly incremental, as there is no limit to the extent of ripups of existing nets, and there is little control on the quality of their re-routes. In [10] a two-stage algorithm is proposed to eliminate crosstalk violations for a given routing as well as minimizing the total deviation for two-layer routing. An incremental routing algorithm for FPGAs that uses a *bump-and-refit* (B&R) approach which routes the new nets by "bumping" less critical existing nets without changing their topologies was proposed in [4] and was extended for ECO routing and for FPGAs with complex switchboxes in [3]. This approach eliminates the net ordering problem by changing the track assignments for previously routed nets and gives an optimal solution in the number of used tracks. Finally, the algorithm presented in [5] finds incremental routing solutions using a gridless framework for VLSI circuits that requires variable width and variable spacing on interconnects. It uses a depth-first-search (DFS) controlled process that optimizes the WL, vias used, and routing completion rate of new nets by allowing overlaps with existing nets that congest their optimal routing regions. The overlapped or bumped segments of these existing nets are re-routed in a similar recursive manner. The high-level DFS control of this routing process ensures that the overlapped nets' lengths and other properties do not change beyond preset limits, thus achieving a balance between the conflicting requirements of near-optimal routing of new nets and bounding any deterioration resulting from the re-routes of existing nets.

Mostly, the above approaches optimize one or more of wirelength (WL), the total number of vias, the number of vias per net, and the number of routing or metal layers. However, in recent years, interconnection delay has become the dominant factor in determining the speed of VLSI systems [11]. In this paper, we address this issue by presenting an incremental timing-driven (TD) routing algorithm TIDE for complex VLSI circuits which have predefined delay constraints (slacks) for all sink pins. We use local slack-related information in determining if segment re-routes of nets or new pin connections satisfy all slack constraints of the net. We also use a depth-first-search (DFS) control similar to [5], in order to route the new nets by minimally perturbing[1] existing nets and only allowing perturbations in which their slacks are not violated and their WL increments are bounded.

The rest of the paper is organized as follows. In Sec. 2,

[1]Throughout this paper we will use the terms *perturb, bump* and *overlap* interchangeably.

we define the timing-driven incremental routing problem for VLSI circuits. We explain our TD global routing algorithm in Sec. 3. In Sec. 4, we discuss the detailed routing part of our algorithm which includes a DFS-controlled bump-and-reroute technique and an efficient method for determining slack satisfaction of bumped nets. Experimental results comparing our incremental router to the TD extensions (based on [9]) of the type of incremental routers proposed in [1, 2] and that we have implemented, are given in Sec. 5. We conclude in Sec. 6.

2. TIMING-DRIVEN INCREMENTAL ROUTING

The objective of previous timing-driven (TD) routers have been varied, ranging from minimizing the average source-to-sink delay to minimizing the maximum source-to-sink delay over all sinks [8, 9] to minimizing interconnect length subject to satisfying all timing specifications [6]. The exact TD problem that needs to be solved is one of satisfying given timing constraints (slack satisfaction) while minimizing WL as in [6]. We will, however, solve this aspect of TD incremental routing more efficiently than in the MVERT algorithm of [6], and also more effectively by successively finding the next nearest branch of the partial routing tree to connect a new pin to, if there does not exist any slack-satisfying connection point on the previous nearest branch.

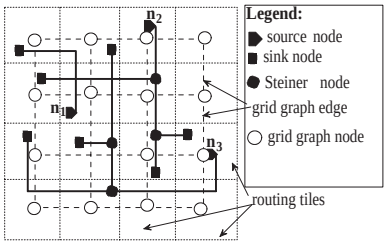

Figure 1: *Routing tiles and the GR-graph.*

While incrementally routing the ECO- generated or *new* nets, we use the B&R paradigm to allow their routes to overlap (bump) some existing nets in order to have a richer solution space (i.e., a solution space that allows more near-optimal routing and higher routing completion rate of the new nets). However, this may result in deterioration of some properties (net delay, number of vias, WL etc.) of the bumped nets. A good and stable incremental routing methodology is one which not only optimizes the routing of new nets, but which also minimizes or bounds the deterioration in various metrics of the bumped nets. The TD incremental routing problem can thus be formally stated as:

`Input`: A set of routed nets of a circuit and a set of new unrouted nets.
`Output`: A completely routed circuit in which:
- All pin slack constraints are satisfied.
- The routing of new nets is optimized (e.g., wrt WL, vias).
- The deterioration in the metrics of interest due to the re-routing of existing nets is minimized or bounded.

In the next two sections we discuss at length our timing-driven incremental global and detailed routers that address the above objectives.

3. TD INCREMENTAL GLOBAL ROUTING

The objective of the global routing step is to elaborate a routing plan to determine how nets will finally be routed while attempting to optimize a given objective function which is usually an estimate of the overall wire length, net delay and congestion [9]. The routing cost function that we use takes care of these parameters and is described shortly.

The input to global routing is a set of nets $N = \{n_1, n_2, \ldots, n_m\}$ with each net n_i being defined by a set of pins $V_i = \{v_0, v_1, v_2, \ldots, v_k\}$, wherein the source or driver is denoted by v_0. We consider routing in two layers, one for horizontal wires and the other for vertical wires. We route these nets in an area that is conceptually divided into rectangular regions called *routing tiles*, as shown in Fig. 1 by areas marked by the finely dashed lines. We represent this routing structure as a grid graph $G(V_g, E_g)$, where $V_g = \{g_1, g_2, \ldots, g_l\}$ is the set of grid nodes representing routing tiles, and E_g is the set of grid edges that are present between node pairs representing adjacent routing tiles. The problem of routing a net in G can be described as a Steiner tree routing problem on the nodes in G that are the locations of the pins of the net. We term the grid graph $G(V_g, E_g)$ as the *GR-Graph (Global Routing Graph)*. Our cost function of an edge in G is a weighted sum of congestion, a base-cost to control net length, and the distance-to-target which helps to steer Dijkstra's shortest path wavefront towards the target pin, thereby pruning the search space.

3.1 The Elmore Delay Model

To determine the signal delay of an interconnect, we employ the Elmore delay model which is briefly described here. It is an appropriate delay model because of its fidelity [7] and simplicity. A routing tree T for net n_i is described by a set of nodes $V = \{v_0, v_1, \ldots, v_k\}$ and a set of edges $E = \{e_1, e_2, \ldots, e_p\}$. The location for node v_i is specified by its coordinates x_i and y_i, and any edge in E is uniquely identified by the node pair (v_i, v_j) or the notation e_{ij}, used interchangeably, where v_i is the up-stream end of this edge, i.e., v_i is closer to the source node. A subset $P \subseteq V$ are the sink pins of the net and are the leaf nodes in T. The resistance and capacitance of edge e_{ij} are denoted r_{ij} and c_{ij}, respectively, and R_d denotes the driver (source node) resistance. Let T_{v_i} denote the subtree of T rooted at node v_i, and $C^d_{v_i}$ the downstream capacitance of T_{v_i}, which is the sum of sink capacitance and edge capacitances in T_{v_i}. The Elmore delay $D(v_k)$ at sink v_k is given by:

$$D(v_k) = R_d \cdot C^d_{v_0} + \sum_{\forall e_{ij} \in path\ v_0 - v_k} r_{ij} \cdot (c_{ij}/2 + C^d_{v_j}).$$

Slacks of the sink pins of a net are used to guide our timing-driven incremental routing algorithm. The $slack\,S(v_k)$ at sink pin v_k is defined as the amount of delay that can be added to the interconnect from the driver/source node to the sink pin without increasing the maximum delay limit of that pin. The primary goal in timing-driven incremental routing is to ensure that for each net n_i that is re-routed: $\forall v_k \in n_i, \Delta D(v_k) \leq S(v_k)$, where $\Delta D(v_k)$ is the delay change at sink pin v_k due to the re-routing of n_i. For an unconnected pin v_u, its slack is its total delay specification and $\Delta D(v_u)$ is its delay after it is interconnected.

3.2 TD Connections of New Pins

An important issue during the global routing of a new net n_i is to decide where the next pin should be connected on the partially constructed routing tree T of n_i so that the slack constraints of connected sink pins and the new pin are not violated. Using the terminology of [6], let CC be the closest

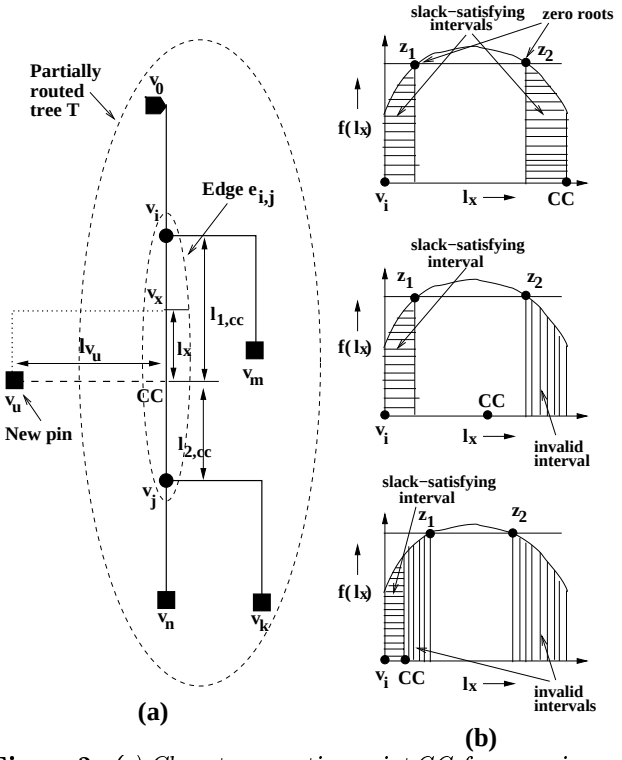

Figure 2: *(a) Closest connection point CC for new pin v_u to partial routing tree T. Connecting new pin v_u to any downstream to CC point v'_x is suboptimal in delay. (b) Different possible slack-satisfying interval(s) on edge e_{ij} obtained from solutions of concave inequalities.*

connection point on T where new pin v_u can be connected, and let edge e_{ij} be the segment of T which contains CC; see Fig. 2a. The optimality of a connection point can be measured by using different metrics such as a weighted sum of the delay from source to sinks, maximum source-to-sink delay [8, 9] and meeting the specified slack constraints [6]. In this paper, optimality is measured by meeting the specified slack constraint of each sink pin of T while minimizing the wire length of the interconnect from v_u to T (as in [6]). Thus, the optimal or near-optimal connection point for v_u will be any point on the tree which is close to and upstream or CC. Note that a connection that is downstream of CC cannot lead to an optimal solution [8].

The optimal connection point is, in general, likely to be a non-Hannan point [6]. The work of [6] showed the advantage of using non-Hannan points and proposed an algorithm MVERT to perform routes based on non-Hannan points for the routing tree. MVERT finds the optimal connection point (in terms of WL) that satisfies all slacks through a binary search and obtains significant WL reductions compared to a SERT-like algorithm [8]. References [6, 8] showed that the delay at every sink node is a concave function[2] of the distance l_x of the connection point for v_u from CC on branch $e_{i,j}$.

It seems that MVERT considers all the k delay functions for a net with k sink nodes and performs a binary search over their intersection points on the span from CC to the upstream Steiner node v_j (Fig. 2b) on $e_{i,j}$. The time complexity of this computation would be $\Theta(k^2 \log k)$ per sink pin as there are $2k(k-1)$ intersection points to compute and sort (into two lists, one by y-axis and another by x-axis values) among the k functions before the binary search process can be started. One of our significant contributions in this work is recognizing that in reality it is enough to satisfy *propagated slack-type constraints* at only three nodes, the new sink v_u and the two Steiner nodes v_j, v_i at the ends of $e_{i,j}$; how different types of slack values (called "tolerances") are computed at Steiner nodes is the subject of the next subsection (and is another innovation useful for locally determining slack satisfaction in different scenarios). This slack-satisfaction evaluation at the above three nodes takes constant time. However, every time a new sink is added to the current partial tree, it takes $\Theta(\log k)$ time to update the tolerance values along the path from the new sink to the source node for a balanced routing tree and $\Theta(h)$ time in general, where $h = O(k)$ is the height of the tree (see Sec. 3.4.2). Thus per sink pin added, our time complexity is $\Theta(\log k)$ for a balanced tree and $\Theta(k)$ for an extremely unbalanced tree.

The valid Steiner points for connecting v_u are located in the intersection CP of the slack-satisfying interval(s) for each of the three "satisfaction functions". These functions correspond to the following three subsets of sink pins connected to T: $Set_1 = \{v_u\}$, $Set_2 = \{v_z | v_z \in P \cap T_{v_j}\}$ and $Set_3 = \{v_k | v_k \in P \cap (T - \{T_{v_j} \cup \{e_{ij}\})\}$, where P is the set of sink pins of T; these subsets are illustrated in Fig. 3a. For each subset we have a different function to determine whether slacks of any sinks in these subsets will be violated if pin v_u is connected to Steiner point $v_x \in e_{ij}$. The slack-satisfying intervals on e_{ij} for each of these functions is determined for the above subsets of sink pins (Fig. 2b shows that for concave functions there will be one to two slack-satisfying intervals, if any), the intersection CP of these intervals is obtained, and if the intersection is not empty, the closest point to CC in it is determined for making a connection; note that there will only be a constant number of disjoint intervals in CP. The algorithm and satisfaction functions are elaborated in Sec. 3.4.

3.3 Tolerance Concepts for Local Evaluation of Slack Satisfaction

Connecting a new pin to a partially constructed tree T and ripping-up and rerouting some segment of T can cause both the total length of T and the slack at each sink pin to be changed. We next define some concepts that capture allowable delay and capacitance changes at Steiner points and sink nodes of T, and are useful in quickly determining if all slack constraints in T are satisfied when it is incrementally modified in the two scenarios mentioned above: (a) connection of a new sink and (b) partial rip-up and reroute of a segment of T.

Referring to Fig. 3b, let $R^{up}_{v_i}$ denote the upstream tree resistance of node v_i, which is the sum of all the edge resistances in the path from v_i to the source node v_0 plus the driver resistance R_d; also, $R^{up}_{v_0} = R_d$. Let $Cg(v_u)$ be the gate capacitance of pin v_u. Recall that $D(v_i)$ is the delay from the source/driver pin to the Steiner-node/sink-pin v_i in the routing tree T; see Sec. 3.1.

Definition 1: $Tol_{del}(v_i)$ is the maximum delay increase that

[2] A concave function of x is a one which initially increases with x (slope is positive but decreasing) reaches a global maximum (slope is 0) and then decreases (slope is negative and its magnitude is increasing); see Fig. 2b for a pictorial example of a concave function.

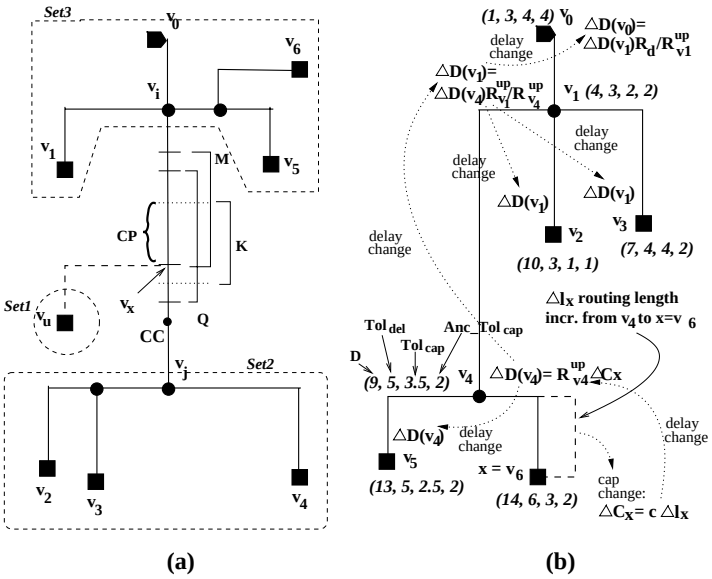

Figure 3: *(a) Q, K, M are the intervals on edge e_{ij} where slack requirements are satisfied for the sinks in Set_1, Set_2 and Set_3, respectively. CP is the interval where Q, K and M intersect. (b) Routing tree with various tolerances shown at each node. The numbers in brackets are $(D, Tol_{del}, Tol_{cap}, Anc\_Tol_{cap})$; note that Tol_{del} of a sink pin is its slack. When the interconnect length to $x = v_6$ increases by Δl_x, and thus the capacitance by $c \cdot \Delta l_x$, delay changes percolate to various nodes as shown by the dashed arrows.*

can be tolerated at v_i without violating the slacks of sink pins in the subtree T_{v_i} rooted at v_i.

$$\text{Thus} \quad Tol_{del}(v_i) = \min_{x \in sinks(T_{v_i})} \{S(x)\} \text{ or recursively,}$$

$$Tol_{del}(v_i) = \min_{v_j \in children(v_i)} \{Tol_{del}(v_j)\}. \qquad (1)$$

Note that for a sink pin x, $Tol_{del}(x) = S(x)$ where $S(x)$ is the slack of x. As seen in Fig. 3b, the delay tolerance for node v_4 is $5ps$ which is the minimum of the slack (or delay tolerances) of its two children nodes v_6 and v_7.

Definition 2: For a Steiner node v_i, the capacitive tolerance $Tol_{cap}(v_i)$ is the maximum capacitance increase downstream in T_{v_i} that can be tolerated without violating the slack at any sink pin in T_{v_i}. The boundary condition for the capacitive tolerance is at sink pins x, where $Tol_{cap}(x)$ is defined as $S(x)/R_x^{up}$. Let Δl_x be the length increment of interconnect to x due to rip-up and reroute of the segment connecting x to its parent Steiner point v_i. The increase in delay $\Delta D(x)$ at x is:

$$\Delta D(x) = R_{v_i}^{up}(c \cdot \Delta l_x) + \frac{rc}{2} \cdot ((\Delta l_x)^2 + 2l_x \cdot \Delta l_x) + r \cdot \Delta l_x \cdot Cg(x).$$

Only the $c \cdot \Delta l_x$ capacitance increase on the new interconnect from v_i to x has a bearing on the delay increases of the other sink pins in T_{v_i}; see Fig. 3b. The delay change $\Delta D(v_i)$ at x's parent node v_i is

$$\Delta D(v_i) = R_{v_i}^{up} \cdot c \cdot \Delta l_x.$$

The delay change at all sinks of T_{v_i} other than x will also be $\Delta D(v_i)$ and thus we have the constraint $\Delta D(v_i) \leq$ the minimum slack $S_{min}(T_{v_i})$ in T_{v_i} or $c \cdot \Delta l_x \leq \frac{S_{min}(T_{v_i})}{R_{v_i}^{up}} = Tol_{cap}(y) \cdot \frac{R_y^{up}}{R_{v_i}^{up}}$ where y is the sink in T_{v_i} with minimum slack. In other words,

$$Tol_{cap}(v_i) = \min_{x \in sinks(T_{v_i})} (Tol_{cap}(x) \cdot \frac{R_x^{up}}{R_{v_i}^{up}}), \text{or recursively,}$$

$$Tol_{cap}(v_i) = \min_{v_j \in children(v_i)} (Tol_{cap}(v_j) \cdot \frac{R_{v_j}^{up}}{R_{v_i}^{up}}). \qquad (2)$$

Definition 3: Ancestor capacitance tolerance $Anc\_Tol_{cap}(v_i)$ at node v_i is the maximum capacitance increase that can be tolerated in the subtree T_{v_i} without violating the slack of any sink pin in the entire routing tree T.

$$\text{Thus} \quad Anc\_Tol_{cap}(v_i) = \min_{v_k \in ancestor(v_i) \cup \{v_i\}} \{Tol_{cap}(v_k)\}$$

or recursively,

$$Anc\_Tol_{cap}(v_i) = \min\{Anc\_Tol_{cap}(v_p), Tol_{cap}(v_i)\} \qquad (3)$$

where v_p is the parent node of v_i in T.

As seen in Fig. 3b, any change in an interconnect of T_{v_4} also affects the delay of each node, say, $v_2 \notin T_{v_4}$. $Tol_{cap}(v_2) = 1fF$ is reflected in $Tol_{cap}(v_1) = 2fF$ of its parent v_1, which in turn is reflected in $Anc\_Tol_{cap}(v_4) = 2fF$ of its child node v_4; this means that if the capacitance increase seen in T_{v_4} is more than $2fF$, this will violate the slack of v_2. Thus $Anc\_Tol_{cap}(v_i)$ of v_i captures the maximum allowable capacitance increase in T_{v_i} beyond which some slack in T will be violated. Thus $Anc\_Tol_{cap}(v_i)$ is the crucial slack-related tolerance to check; $Tol_{cap}(v_i)$'s use is essentially in computing $Anc\_Tol_{cap}(v_i)$ using Eqn. 3.

3.4 Interval Intersection Algorithm (IntAl)

We now discuss the algorithm for determining the interval(s), if any, on the edge $e_{i,j}$ of the partially routed tree T where new pin v_u can be connected without violating slack constraint in the sinks in the three subsets Set_1, Set_2 and Set_3; see Sec. 3.2. If the reader wishes he/she can skip the step-by-step derivation of the satisfaction functions leading to Ineqs 5,6,7. It will, however, be useful for the reader to take a look at the inequalities themselves, and read the material after each of them in their respective sections.

We refer to Fig. 2a for the notations used here.

1. Slack-satisfying interval for $Set_1 = \{v_u\}$: Let $\Delta cap(CC)$ be the capacitance increase seen at CC caused by connecting v_u to T at CC, l_{v_u} be the distance between v_u and CC, $l_{1,cc}$ be the distance from v_i to CC and $C_d(e_{ij})$ be the total downstream capacitance at v_j plus the capacitance of the interconnect e_{ij} connecting v_i and v_j; see Fig. 2a. For any connection point v_x upstream of CC, the interconnect capacitance $\Delta cap(v_x) = \Delta cap(CC) + c.l_x$ where l_x is the distance between CC and v_x and the downstream capacitance at v_x will be $C_d(v_x) = C_d(e_{ij}) - c.(l_{1,cc} - l_x)$. Referring to Fig. 2a, using the delay $D(v_i)$ at parent Steiner node v_i, we can calculate $D(v_u)$ as:

$$D(v_u) = D(v_i) + R_{v_x}^{up} \cdot (Cg(v_u) + \Delta cap(v_x))$$
$$+ r_{v_i,v_x} \cdot [\frac{c_{v_i,v_x}}{2} + C_d(v_x)] + r_{v_u,v_x} \cdot [\frac{\Delta cap(v_x)}{2} + Cg(v_u)]. \qquad (4)$$

For any valid Steiner connection on edge e_{ij}: $D(v_u) - S(v_u) \leq 0$. After substituting in this inequality for $D(v_u)$ from Eqn. 4 and expanding some of its terms, we get as a function of the variable l_x the following inequality that needs to be satisfied:

$$f_1(l_x) = -r \cdot c \cdot {l_x}^2 + [R_{v_i}^{up} + 2 \cdot r \cdot c \cdot l_{1,cc} - r \cdot C_d(e_{ij}) - \frac{r \cdot \Delta cap(CC)}{2} + \frac{r \cdot c \cdot l_{v_u}}{2}] \cdot l_x + H \leq 0 \quad (5)$$

where H is the constant part of the $D(v_u) - S(v_u)$ expression.

$f_1(l_x)$ is a concave function, an example of which is shown in Fig. 2b. The general procedure for determining the interval(s) Q that satisfy a concave inequality $f_1(l_x) \leq 0$ (as in Ineq. 5) is as follows.

- Obtain the two roots z_1, z_2 of $f_1(l_x) = 0$. Let $z_1 < z_2$. Note that since the function is concave, the *possible* intervals satisfying $f_1(l_x) \leq 0$ are to the left of and including z_1 and to the right of and including z_2; see Fig. 2b.
- **If** $(z_1 \geq l_{1,cc})$ (i.e., the z_1 distance node is to the right of CC) **then** $Q = [v_i, CC]$. **Exit**.
- **If** $z_1 < l_{1,cc}$ and $z_2 > l_{1,cc}$ **then**
  if $z_1 >= 0$ **then** $Q = [v_i, v_{z_1}]$, where v_{z_1} is the node on e_{ij} at a distance of z_1 from v_i.
  else $Q = \emptyset$. **Exit**.
- **If** $z_1 < l_{1,cc}$ and $z_2 \leq l_{1,cc}$ **then**
  {
  if $z_1 >= 0$ $Q = [v_i, v_{z_1}]$ **else** $Q = \emptyset$.
  if $z_2 >= 0$ **then** $Q = Q \cup [v_{z_2}, CC]$ **else** $Q = Q \cup \emptyset$.
  } **Exit**.

2. Slack-satisfying interval for $Set_2 = \{v_z | v_z \in P \cap T_{v_j}\}$: When pin v_u is connected to point v_x, the delay increase $\Delta D(v_x)$ due to the the extra capacitance $\Delta cap(v_x) + C_g(v_u)$ seen at v_x cannot be greater than the delay tolerances (slacks) of any sink in Set_2 in order for v_x to be a valid connection. In other words, the capacitance tolerance at v_x should be greater than the added capacitance; the capacitance tolerance at v_x can be derived from the tolerance at node v_j as:

$$Tol_{cap}(v_x) = Tol_{cap}(v_j) \cdot \frac{R_{v_j}^{up}}{R_{v_x}^{up}} = Tol_{cap}(v_j) \cdot \frac{R_{v_j}^{up}}{(R_{v_j}^{up} - r.(l_x + l_{2,cc}))}$$

where $l_{2,cc}$ is the distance of CC from v_j. So, in order to ensure that connecting the new pin v_u to v_x will not violate any slack in Set_2, $\Delta cap(v_x) + C_g(v_u) - Tol_{cap}(v_x) \leq 0$. This gives us a second inequality as a function of l_x:

$$f_2(l_x) = \Delta cap(CC) + c \cdot l_x + Cg(v_u) - Tol_{cap}(v_j) \cdot \frac{R_{v_j}^{up}}{(R_{v_j}^{up} - r \cdot (l_x + l_{2,cc}))} \leq 0 \text{ or}$$

$$f_2(l_x) = -rc \cdot {l_x}^2 + [R_{v_j}^{up} \cdot c - (\Delta cap(CC) + Cg(v_u))r - rc \cdot l_{2,cc}]l_x - Tol_{cap}(v_j) \cdot R_{v_j}^{up} + (\Delta cap(CC) + Cg(v_u)) \cdot (R_{v_j}^{up} - r \cdot l_{2,cc}) \leq 0 \quad (6)$$

This too is a concave function, and we find its satisfying interval(s) K, if any, using a similar procedure as that given above for $f_1(l_x)$.

3. Slack-satisfying interval for $Set_3 = \{v_k | v_k \in P \cap (T - \{e_{ij} \cup T_{v_j}\})\}$: To check whether the slack of any pin $v_m \in Set_3$ will be violated, we need to check if the extra capacitance at v_x is no more than the capacitive tolerances of the Steiner nodes on the path from v_x to source pin v_0. Thus it will be enough to compare the extra capacitance with the ancestor capacitance tolerance $Anc\_Tol_{cap}(v_i)$ at v_i. Hence for satisfying the slacks of the sink pins in Set_3: $f(v_x) = Anc\_Tol_{cap}(v_i) - [\Delta cap(v_x) + Cg(v_u)] \geq 0$, i.e.,

$$f_3(l_x) = Anc\_Tol_{cap}(v_i) - [\Delta cap(CC) + c \cdot l_x + Cg(v_u)] \geq 0. \quad (7)$$

By solving the linear inequality $f_3(l_x) \geq 0$ we get the valid interval M for Set_3.

Ineqs. 5, 6 and 7 provide us the three respective slack-satisfying intervals Q, K, M as shown in Fig. 3a. The final valid interval CP of *candidate points* for connecting the new pin is the intersection of these intervals; see Fig. 3a. If CP is not empty then we select the point $p \in CP$ which is the closest point to CC to connect the new pin. This connection will not only satisfy all slack constraints but will also be near-optimal in wire length. Fig. 4 gives the flow chart for finding this intersection.

THEOREM 1. *The IntAl algorithm correctly determines the closest Steiner point on edge $e_{i,j}$ of partially routed tree T that satisfies slack constraints of all sink pins of T, if any such point exists.*

Proof: Follows by construction from the derivation of $f_1(l_x), f_2(l_x)$ and $f_3(l_x)$ and the correct determination of capacitive tolerances Tol_{cap} and Anc_Tol_{cap} of Steiner nodes of a routing tree. $\diamond$

After we decide where the new pin will be connected on T, we apply Dijkstra's shortest path algorithm (with edge costs described in Sec. 3) to find a global path from v_u to v_x.

3.4.1 Connection Failure and Tree Truncation

Sometimes there may not be a connection point on e_{ij} to connect new pin v_u, i.e. CP is an empty set. We then try another connection point on another segment of T. As seen in Fig. 4, based on the failure condition, some portion of the routing tree T is truncated in order to eliminate edges that are also guaranteed not to have a valid connection point. After truncating T, if the remaining routing tree T_R is empty, then it means that because of some slack violation(s), this net cannot be routed and we declare a failure to route this net with the given timing constraints.

Our tree truncation approach is another significant improvement over previous techniques which on a connection failure on $e_{i,j}$ either conducted an exhaustive-type search by attempting a connection at the next closest branch and so forth, or a connection was directly attempted at the source node without trying to find valid connections on any other tree branches.

THEOREM 2. *The tree truncation method given in Fig. 4 will always find a slack-satisfying connection point for new pin v_u on the nearest valid edge, if one exists, of the partial routing tree T.*

Proof Outline: We consider one case here due to space constraints. Referring to Fig. 4, let $M = \emptyset$; this means that the minimum capacitance increase $c \cdot l_{v_u}$ (see Fig. 2a) in T_{v_i} for connecting v_u to CC on branch $e_{i,j}$ is $> Anc\_Tol_{cap}(v_i) = Tol_{cap}(v_p)$ for some ancestor v_p of v_i with the minimum Tol_{cap} among all its ancestors. Thus $c \cdot l_{v_u}$ capacitance increase will violate some slack in T_{v_p}. Since any other connection of v_u to any other branch of T_{v_p} will, by definition of

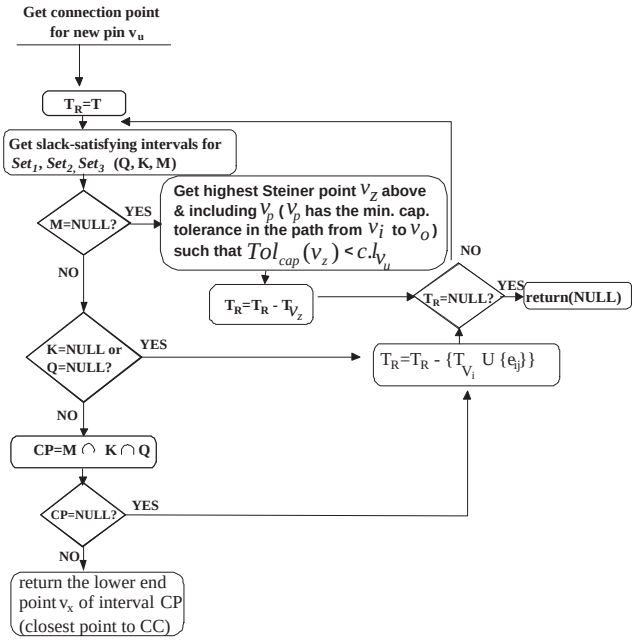

Figure 4: *The IntAl algorithm with tree truncation for obtaining the valid Steiner point set CP for connecting new pin v_u to a partially routed tree T.*

CC, have length $\geq l_{v_u}$, we can only connect v_u to a branch above T_{v_p}. However, there may be an ancestor of v_p whose Tol_{cap} while greater than that of v_p is still less than $c \cdot l_{v_u}$. If v_z is the highest such ancestor of v_p, then we cannot connect v_u to T_{v_z} but may be able to do so at some branch above T_{v_z}; hence the remaining tree to consider connecting v_u to is $T - T_{v_z}$. $\diamond$

3.4.2 As-Needed Updates

After we connect the new pin to the routing tree, delay and capacitance tolerances of the Steiner nodes need to be updated. The delay of the sinks in Set_2 will increase by $\Delta D = R_{v_x}^{up} \cdot [\Delta cap(v_x) + Cg(v_u)]$, and the delay of any ancestor Steiner node v_m will increase by $\Delta D(v_m) = R_{v_m}^{up} \cdot [\Delta cap(v_x) + Cg(v_u)]$. It is only necessary to compute and store the respective delay and tolerance changes at the child and all ancestor Steiner nodes of v_x. Subsequently, when we need to determine if a delay and capacitance change at a new Steiner node or at a sink node (due to re-routing a bumped segment of T or connecting a new pin to it) satisfies the tolerances at its child v_y and ancestor nodes, we first update their tolerances by traveling up the tree from v_y to the source node v_0 and accumulate all previously updated delay and tolerance changes in this path, and then apply the checks. Thus it is not necessary to completely update the entire tree after every delay and capacitance changes. This process of *as-needed-updates* takes $O(h)$ time where h is the height of the tree; if the tree is balanced, then this time is $O(\log k)$, where k is the number of pins on the net.

4. TD INCR. DETAILED ROUTING

After the global router assigns routing tiles to the interconnection between pin v_u and point v_x on T, the detailed routing phase begins. The objective of the detailed routing is minimum utilization of available routing resources, including the number of vias.

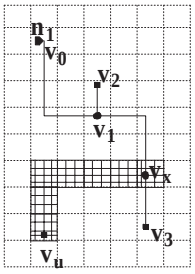

Figure 5: *Randomly generated grid-lines for detailed routing.*

In current VDSM technology the number of grid lines for detailed routing in a tile will be huge, resulting in an extremely large search space for the detailed route from v_u to v_x. In order to reduce the search space but still be able to explore good routing solutions on the average, we route on a random grid graph within the globally assigned routing tiles. The random grid graph, termed the *DR-Graph* (for detailed routing graph), is obtained simply by randomly generating a small % (e.g. 3%) of all vertical and horizontal grid lines in the routing tiles assigned by the global router. These grid-lines also include horizontal and vertical lines passing through the centers of nodes v_u and v_x; see Fig. 5. By again applying Dijkstra's shortest path algorithm and using a cost function that considers via-cost, we find an exact path from v_u to v_x in the DR-Graph.

Bump-and-Reroute and Its DFS Control During detailed routing, we first try to find an appropriate path $P_k \in PV$ in the routing tiles, where PV is the set of the paths between points (nodes) v_u and v_x that are vacant, i.e., they do not overlap any existing nets. Sometimes the routing resources are not available in these routing tiles to complete a valid route. In this case, the detailed router will explore a path $P_k \in PO$ where PO is the set of paths that overlaps with some existing nets [5]. The *overlapped segments* (o-segs) of existing nets in turn will have to be ripped up and rerouted between the closest two Steiner nodes on their respective nets, and their routing in turn may overlap other existing nets. Thus we may get a sequence of rerouting overlaps and partial R&Rs among the existing nets. Note that we do not rip-up and reroute the entire overlapped existing net as in the case of [2]. Here, we are only ripping-up and rerouting the *o-seg* of the overlapped net [5].

Finding a path for a particular o-seg can initiate a set of other o-seg rip-ups and path searchings which finally terminate when a $P_k \in PV$ is found for each leaf o-seg of this tree-structured search. Essentially, we are performing a depth first search (DFS) of partial rip-ups and re-routings, to find a solution for the original o-seg, i.e., the one that was overlapped in the routing of a new net. A DFS path terminates in failure if the route selected for the current o-seg overlaps already overlapped existing nets in the current path or overlaps obstacles or leads to some constraint violation (e.g. a slack violation on any pin of the net containing the o-seg, increase in net length of the o-seg beyond a pre-set upper bound) . If a particular path $P_k \in PO$ fails in this manner, the search backtracks and tries another unexplored path P_l for the o-seg. When all paths explored for the current o-seg fail, the search backtracks to the parent o-seg that overlaps it and tries another path of it. This phase is one of the cruxes of our incremental routing algorithm. We thus use the acronym *TIDE* for our algorithm, which stands for *TIming-driven DEpth-first search controlled segment ripup-&-reroute.*

4.1 Slack-Satisfying Bumping of Existing Nets

A crucial issue addressed in our timing-driven incremental routing algorithm is to decide whether re-routing the o-

seg of an overlapped net n_i will violate any of its sink pins' slacks. The pre-computed tolerances (see Sec. 3.3) stored at the Steiner nodes of n_i's routing tree T allow us to quickly and locally determine re-routings of the o-seg that will satisfy all slack constraints of n_i or determine that such re-routings are not possible in our solution space. We consider three types of possible overlaps of n_i: (1) overlapping a leaf interconnect, (2) overlapping an interior interconnect, and (3) overlapping a Steiner node. Due to space constraints, here we discuss only the first type of overlap.

4.1.1 Overlapping a leaf interconnect

As seen in Fig. 6, during the bump and re-route process, some segment of the current net can overlap an interconnect of an existing net n_i that connects a sink pin v_l; we term this a *leaf interconnect*.

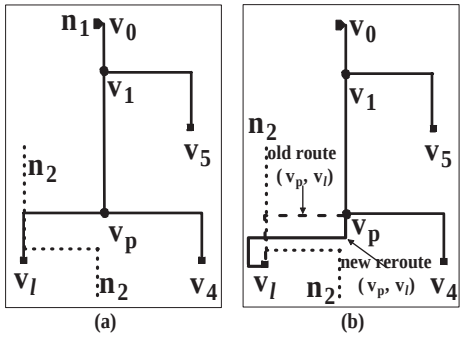

Figure 6: *(a) Overlapping of a leaf interconnect between v_p and v_l, and (b) its rerouting.*

In this case, we only rip-up the interconnect between pin v_l and its parent Steiner node v_p in T and try to find another path between v_l and v_p without violating any slack constraint. This problem is similar to that of finding a connection point for a new pin which was addressed in Sec. 3.4. The difference here is that, we already know that a valid and good connection point for v_l is v_p. Thus, we only need to check whether the tolerances stored at v_l and v_p allow a reroute of the interconnect between v_p and v_l with an increased length. Let Δl_{v_l} be the change in length of interconnect (v_p, v_l). If $\Delta l_{v_l} = 0$, it means that there is no extra capacitance Δcap or delay and thus vacuously all slack constraints of T are satisfied. If, on the other hand, $\Delta l_{v_l} > 0$, the extra capacitance will be $\Delta cap = c \cdot \Delta l_{v_l}$ and it will increase the delay at pin v_l by

$$R^{up}_{v_p} \cdot (c \cdot \Delta l_{v_l}) + \frac{r \cdot c}{2} \cdot ((\Delta l_{v_l})^2 + 2 \cdot l_{v_l} \cdot \Delta l_{v_l}) + r \cdot \Delta l_{v_l} \cdot Cg(v_l).$$

Thus to determine if this re-routing of the leaf interconnect to v_l satisfies all slack constraints of T, we only need to ensure that: $\Delta D(v_l) \leq S(v_l)$, and $\Delta cap \leq Anc\_Tol_{cap}(v_p)$.

The last condition guarantees that this extra capacitance does not violate any node capacitance tolerance and hence any sink pin slack in the routing tree T. Thus the concepts of tolerances of Steiner and sink nodes provide a simple, fast and accurate means for verifying if interconnect re-routings satisfy all slack constraints of the net without exhaustive checking.

5. EXPERIMENTAL RESULTS

The TIDE incremental algorithm was tested on a number of benchmarks which were generated by creating magnified cell layout versions of the Mcc1 circuit (an MCM circuit) template with different magnifications, randomly generating the required pins on the cell and chip boundaries (more the magnification, proportionately more is the number of pins and

Unit Wire Resistance	0.076 $\Omega/\mu m$
Unit Wire Capacitance	0.118 $fF/\mu m$
Driver Resistance	180 $\Omega/\mu m$
Gate Capacitance	23.4 fF

Table 1: *Technology Parameters for 0.18μm node*

Ckt	# nets	# pins	Ckt	# nets	# pins
ckt1	1643	7200	ckt2	2277	10080
ckt3	2949	12960	ckt4	3588	15840
ckt5	4287	18720	ckt6	4919	21600
ckt7	5572	24480	ckt8	6261	27360
ckt9	7613	33120	ckt10	10435	47520

Table 2: *Benchmark Circuit Characteristics.*

nets that can be accommodated), and randomly connecting them by nets of 2-14 pins[3]. Their characteristics are shown in Table 2; the number of nets ranges from 1643 to 10,435 and the number of pins from 7200 to 47,520. Pin slacks in each net n_i were determined by a Gaussian distribution in the interval [0, 5% of $D_{max}(n_i)$], where $D_{max}(n_i)$ is the maximum pin delay in n_i. To simulate ECO, we randomly deleted 10% of the original nets, and randomly added the required new nets to connect the unconnected pins (two random variables were used, one for selecting the new net sizes [between 2-14 pins according to the distribution in footnote 3] and the second to determine its pins). For each benchmark circuit, we generated 10 different random ECOs, and the results given for each circuit are averaged over these 10 runs.

We compared TIDE to the timing-driven versions of two prior incremental routing techniques Std [1] and R&R [2] implemented by us overlaid on a timing-driven routing algorithm SERT/SOAR with elements of the SERT-C algorithm of [8] and the SOAR algorithm in [9] in it—essentially this algorithm adds new sink pins to the partially routed tree T in decreasing order of their criticalities or distances from v_0, trying first to minimize WL, and failing that to minimize, by a connection to v_0, the delay increases to all connected pins (at the expense of WL). We will term these incremental TD algorithms TD-Std and TD-R&R.

All routings were required to be completed within the two metal layers of the original circuits; the experiments are thus of the "car crash test" variety, where some failures are guaranteed to happen, and the overall success rate is thus an important metric. We used the parameters in Table 1 for Elmore-delay calculations. We ran all three methods on a 2.6 Ghz Pentium Linux machines with 1GB of RAM. The results are given in Table 3; **Total HP unrt** is the sum of the average half-perimeter (HP) of the bounding box of unrouted nets and the **Modif nets per new net** is the average number of existing net re-routed to accommodate a new net; the other metrics are self-explanatory. TIDE was able to route almost all the nets ($\approx$ 94%) without sacrificing any quality metric. TIDE has $\approx$ 7 and $\approx$ 6.7 times fewer unrouted nets than TD-Std and TD-R&R, respectively. The total length of unrouted nets by TD-Std and TD-R&R are 7.8 and 7.2 times more than that of TIDE. TD-Std and TD-R&R were not able to route 19% and 16% of nets, resp, due to slack violations, while TIDE had only 3.4% slack violations; note that some of these violations could be theoretical certainties as we are not ensuring, especially for new nets[4], if these timing spec-

[3] Generated according to the following distribution: 2 pins: 30%, 3-4 pins: each 20%, 5 pins: 10%, 6-7 pins: each 5%, 8-10 pins: each 2%, 11-14 pins: each 1%.

[4] $D_{max}(n_i)$ for new nets is based partly on the HP length

Ckts	# new nets	# new pins	% Unrt. nets TD-S	% Unrt. nets TD-R	% Unrt. nets TIDE	Slack viols TD-S	Slack viols TD-R	Slack viols TIDE	Total HP unrt nets ($\times 10^6$) TD-S	Total HP unrt nets ($\times 10^6$) TD-R	Total HP unrt nets ($\times 10^6$) TIDE	Av rout net length ($\times 10^4$) TD-S	Av rout net length ($\times 10^4$) TD-R	Av rout net length ($\times 10^4$) TIDE	Vias per new net TD-S	Vias per new net TD-R	Vias per new net TIDE	Modif nets per new net TD-S	Modif nets per new net TD-R	Modif nets per new net TIDE	Runtime (secs) TD-S	Runtime (secs) TD-R	Runtime (secs) TIDE
Ckt1	159	664	19.2	3.0	2.8	17.2	3.0	2.3	0.45	0.03	0.03	1.80	2.01	1.58	13.3	14.9	13.2	0	3.7	1.8	62.0	1342.1	90.4
Ckt2	224	923	19.7	4.9	3.5	22.6	4.4	3.2	0.91	0.25	0.07	2.53	2.87	2.26	13.2	15.0	13.5	0	4.4	1.8	87.8	1854.0	154.2
Ckt3	295	1259	25.9	24.2	4.2	28.4	17.5	5.4	1.98	1.91	0.16	3.23	3.69	2.95	12.5	14.4	13.7	0	3.6	2.1	94.0	347.6	203.0
Ckt4	357	1491	17.6	8.5	3.1	28.0	10.1	5.9	2.02	1.03	0.20	4.00	4.46	3.49	13.2	15.0	13.3	0	3.6	2.3	147.9	1591.2	264.6
Ckt5	424	1818	22.7	19.5	3.4	37.1	14.1	8.1	3.68	3.31	0.31	4.68	5.33	4.18	12.7	14.6	13.4	0	3.2	2.6	200.5	983.4	212.1
Ckt6	475	2004	40.2	42.0	3.1	74.7	59.7	8.9	7.97	8.67	0.42	4.94	5.59	4.81	11.3	12.8	13.3	0	5.0	2.2	112.9	202.1	314.9
Ckt7	539	2242	46.3	47.8	3.1	105.3	95.5	10.7	11.70	1.22	0.53	5.21	5.97	5.49	10.6	12.0	13.4	0	3.7	3.1	128.5	178.2	318.9
Ckt8	616	2622	23.9	22.5	3.4	53.9	26.6	12.0	8.22	8.10	0.71	6.76	7.71	6.22	12.7	14.6	13.8	0	3.4	2.0	260.5	1501.2	458.5
Ckt9	748	3174	46.5	48.4	3.2	139.9	127.8	13.7	22.10	23.54	1.07	7.43	8.42	7.58	11.2	12.6	13.7	0	5.8	2.2	196.6	1277.5	449.3
Ckt10	978	4087	73.1	74.3	15.8	438.0	429.2	98.1	61.92	63.13	11.90	8.76	9.31	8.06	8.8	9.3	12.9	0	10.5	2.3	152.6	167.3	312.6
Overall Avg	*481*	*2024*	*40.7*	*39.1*	*5.8*	*94.5*	*78.8*	*16.8*	*12.09*	*11.12*	*1.54*	*4.93*	*5.53*	*4.66*	*11.9*	*13.5*	*13.4*	*0*	*4.7*	*2.3*	*144.3*	*944.5*	*277.9*
Factor improv of TIDE			**7.02**	**6.74**		**5.63**	**4.69**		**7.85**	**7.22**		**1.06**	**1.19**		**0.89**	**1.01**		**0**	**2.04**		**0.52**	**3.40**	
Avg – global nets			*31.3*	*30.6*	*3.2*	*56.9*	*44.7*	*10.7*	*10.63*	*10.84*	*1.29*	*20.83*	*42.17*	*6.61*	*45.0*	*75.5*	*17.1*	*0*	*31.3*	*3.3*			
Factor improv of TIDE			**9.78**	**9.56**		**5.32**	**4.18**		**8.24**	**8.40**		**3.15**	**6.38**		**2.63**	**4.42**		**0**	**9.48**				

Table 3: *Circuit-wise comparison of metrics across TIDE, TD-Std (TD-S) and TD-R&R (TD-R) for random deletion of 10% of existing nets and random generations of new nets to connect the unconnected pins. All metrics reported are averaged over ten random runs of net deletions and additions.*

ifications are actually attainable. As shown in Table 3, the average length of nets routed by TIDE is 6% and 19% shorter than those routed by TD-Std and TD-R&R, resp. The number of vias used by TIDE and TD-R&R are comparable with TD-Std using 11% less vias (TIDE is much superior in via-usage for global nets).

TIDE is 3.4 times faster than TD-R&R, while being about half the speed the simple TD-Std method. The last two rows of Table 3 show results for *global nets*, i.e., nets whose lengths are $\geq 50\%$ of the HP of the chip. The improvements of TIDE over TD-Std and TD-R&R for global nets are even better in all metrics except slack violations (for which it is comparable) than its already significant improvements for all nets. This underscores TIDE's efficacy efficacy for both local and global nets as well as for more complex circuits.

The runtimes for TD-R&R bear some explanation. As can be seen in Table 3, its runtimes sometimes fluctuate significantly from circuit to circuit and in inverse proportion to the circuit size. We observed that TD-R&R sometimes takes an inordinately long time to route some new nets in some circuits, and this has little to do with circuit size. We believe this stems from the potentially out-of-control ripup-&-reroutes that can occur in TD-R&R when it encounters a densely congested routing region. Since the ECO simulations were randomized, it probably escaped such regions for some circuits but got stuck in them for others. Another seeming runtime anomaly that applies to all three methods is that when we go from `ckt9` to the much larger `ckt10`, all runtimes decrease noticeably (this is significantly more pronounced for TD-R&R). However, the % of unrouted nets and slack violations are disproportionately more for `ckt10` than for `ckt9` for all three methods, meaning that if these failures were detected early enough then runtimes could decrease.

6. CONCLUSIONS

A timing-driven (TD) incremental routing algorithm TIDE was developed for VLSI circuits. TIDE solves the exact TD routing problem - satisfying sink pin slacks while minimizing the net length. TIDE has several innovative features including a near-min WL interval-intersection algorithm for a valid connection of a new pin to a partially routed tree using only three local slack-satisfying functions, Steiner node tolerance concepts that allow accurate, fast and local determination of slack satisfaction for all pins during global as well as detailed routing, and a DFS-controlled bump-and-reroute process that explores a richer solution space for new-net routing without significantly compromising on the route qualities of existing nets. Our router was tested on several example benchmarks with up to $> 10,000$ nets, and was shown to produce significantly improved results in terms of the number of successfully routed new nets, number of vias required, wire length of nets and slack violations when compared to the TD versions of the well known Std and R&R routing methods.

estimate of n_i which is generally an underestimate for large-size nets.

7. REFERENCES

[1] J.M. Emmert and D. Bhatia. "Incremental Routing in FPGAs". *Proc. IEEE Int. ASIC Conference and Exhibit*, 1998.

[2] J. Cong and M. Sarrafzadeh. "Incremental Physical Design". *ISPD* , April 2000, pp. 84-92.

[3] S. Dutt, V. Verma and H. Arslan, "A Search-Based Bump-and-Refit Approach to Incremental Routing for ECO Appl. in FPGAs", *ACM TODAES*, 7(4), pp. 664-693, 2002.

[4] S. Dutt, V. Shanmugavel and S. Trimberger, "Efficient Incremental Rerouting for Fault Reconf. in FPGAs",*ICCAD*, pp. 173-176, 1999.

[5] *Reference omitted for blind review.*

[6] H. Hou and S. S. Sapatnekar. "Routing Tree Topology Construction to Meet Interconnect Timing Constraints", *ISPD*, pp. 205-210, 1998.

[7] K.D. Boese and A.B. Kahng and B.A. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions", *ICCD*, pp.81-84, 1993.

[8] K.D. Boese and A.B. Kahng and B.A. McCoy and G. Robins. "Near-Optimal Critical Sink Routing Tree Constructions", *IEEE-TCAD*, pp. 1417-1436, 1995.

[9] D. Wang and E. S. Kuh. "A New Timing-Driven Multilayer MCM/IC Routing Algorithm", *IEEE-MCMC*, pp. 89-94, 1997.

[10] H. Xiang and K-Y. Chao and M. D. F. Wong. "An ECO Algorithm for Eliminating Crosstalk Violations",*ISPD*, 2004.

[11] J. Cong,"Challenges and Opportunities for Design Innovations in Nanometer Technologies," *Frontiers in Semiconductor Research: A Collection of SRC Working Papers,* SRC, 1997.