Efficient Timing-Driven Incremental Routing for VLSI Circuits Using DFS and Localized Slack-Satisfaction Computations *

Shantanu Dutt and Hasan Arslan
Dept. of ECE, University of Illinois-Chicago
{dutt, arslan}@ece.uic.edu

Abstract: **In current very deep submicron (VDSM) circuits, incremental routing is crucial to incorporating engineering change orders (ECOs) late in the design cycle. In this paper, we address the important incremental routing objective of satisfying timing constraints in high-speed designs while minimizing wirelength, vias and routing layers. We develop an effective timing-driven (TD) incremental routing algorithm TIDE for ASIC circuits that addresses the dual goals of time-efficiency, and slack satisfaction coupled with effective optimizations. There are three main novelties in our approach: (i) a technique for locally determining slack satisfaction of the entire routing tree when either a new pin is added to the tree or an interconnect in it is re-routed—this technique is used in both the global and detailed routing phases; (ii) an interval-intersection and tree-truncation algorithm, used in global routing, for quickly determining a near-minimum-length slack-satisfying interconnection of a pin to a partial routing tree; (iii) a depth-first-search process, used in detailed routing, that allows new nets to bump and re-route existing nets in a controlled manner in order to obtain better optimized designs. 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]. When a new net cannot be optimally routed without perturbing any existing net, some 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. 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. 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 in a controlled manner and without changing their topologies was proposed in [4] and was extended for ECO routing and for FPGAs with complex switchboxes in [3]. Finally, the algorithm presented in [5] uses a depth-first search (DFS) controlled B&R process to find good-quality incremental routing solutions using a gridless framework for VLSI circuits that require variable width and variable spacing on interconnects.

*This work was supported by NSF grant CCR-0204097. We acknowledge the help of Fenghua Yuan and Vishal Suthar in drawing some of the figures and some re-coding to collect additional experimental results.

The above approaches optimize one or more of wirelength (WL), the total number of vias, 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 have developed a novel technique in which only local slack-related information is used 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, but with the added (non-trivial) constraint of satisfying all slack constraints.

In Sec. 2, we define the TD incremental routing problem that we will solve, and then present the high-level flow of the global and detailed routing phases of our algorithm. Sections 3-5 present the various new techniques that are used in the global and detailed routers to make TIDE a very effective TD incremental router. Experimental results are given in Sec. 6 and we conclude in Sec. 7.

2 Timing-Driven Incremental Routing

2.1 Delay Model and Objective

To determine the signal delay of an interconnect, we employ the Elmore delay model that has been used in most TD routing work because of its fidelity [7]. A routing tree T for net n_i is described by a set $V = \{v_0, v_1, \ldots, v_k\}$ of a source/driver pin (v_0), sink pins and Steiner points, and a set of interconnects or edges E. An edge in E is uniquely identified by the node pair (v_i, v_j) that it connects directly (the notation e_{ij} is also used to identify this edge) where v_i is the up-stream end of this edge, i.e., v_i is closer to the source node. A subset $P \subset V$ 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:

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

The *slack* $S(v_k)$ of a connected sink pin v_k is defined as the amount of delay that can be added to the path from the source node to the sink pin without violating the delay upper bound of

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

that pin. For an unconnected pin its slack is its upper bound delay specification.

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 latter is the exact TD problem that needs to be solved, although possibly with added optimization criteria such a minimizing the number of vias and routing layers.

A good and stable incremental routing methodology is one which not only optimizes the routing of ECO-generated or *new* nets, but which also minimizes or bounds the deterioration in various metrics due to the re-routings, if any, of existing nets. The TD incremental routing problem that we will address is formally stated as follows.

`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.

2.2 Overview of TIDE's Global and Detailed Routing

Here we present the high-level flow of our global and detailed incremental routing phases wherein several innovations are used; these innovations are described in later sections.

Global Routing Flow Let n_i be the new net we need to route incrementally. Routing is done in two layers (and easily extendible to mutliple layers), one for horizontal wires and the other for vertical wires.

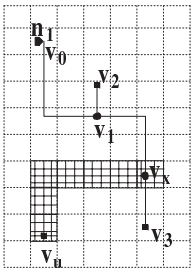

Figure 1: *Randomly generated grid-lines within global routing tiles for the detailed routing of interconnect* (v_u, v_x).

Nets are globally routed in an area that is conceptually divided into rectangular regions called *routing tiles*. This routing structure is modeled as a grid graph $G_{gr}(V_g, E_g)$ called the *GR-graph*, where V_g 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 global routing flow is given in Fig. 2. The main innovation in this phase is the *interval intersection and tree-truncation* algorithm `IntAl` that is discussed in Sec. 4. `IntAl` itself utilizes another innovation discussed in Sec. 3—slack-type *tolerance* determinations at every Steiner node in the routing tree T, and use of these tolerances for locally determining slack satisfaction for all sinks in T when a new pin as connected to it.

Detailed Routing Flow In order to reduce the search space of the huge number of detailed-routing grid lines that typically occupy a tile in current VDSM technology but still explore good routing solutions, detailed routing is performed on a sparse *random grid graph* termed the *DR-graph* (for detailed routing graph), obtained simply by randomly generating a small % (e.g. 3%) of all vertical and horizontal grid lines in the routing tiles assigned to the interconnects of T by the global router; see Fig. 1. After n_i is globally routed, it is detailed routed according to the flow given next in Fig. 3. The main innovations in

`Global-TIDE`(n_i);
/* Let the pins of n_i be $\{v_0, v_1, \ldots, v_m\}$, where v_0 is the source pin. */

1. Determine the min-cost path in G_{gr} from v_0 to v_k using Dijkstra's algorithm, where v_k is the most-critical sink (e.g., sink with smallest slack $S(v_j)$ or smallest value of $S(v_j)/d(v_0, v_j)^2$ [$d(v_0, v_j)$ is the Manhattan distance between v_0 and v_j], among all sinks v_j in n_i). Let this path be the current partially routed tree T;
   /* The cost of an edge between two nodes in G_{gr} is the weighted sum of three metrics: base cost (controls interconnect length), congestion cost (average of horizontal and vertical occupancy densities of its two tiles) and distance-to-target cost (minimizes vias as well as the solution time). */
2. For each remaining sink pin v_u in decreasing order of criticality do the following:
   (a) Determine using the `IntAl` algorithm (Sec. 4) the connection point v_x on T such that slacks of all pins on T and of v_u are satisfied, and the interconnect length from v_u to T is minimized.
   (b) Perform min-cost routing on G_{gr} from v_u to v_x using the above 3-metric cost function.

Figure 2: *TIDE's global routing phase.*

the detailed routing phase are the depth-first search (DFS) based control of a *bump-and-reroute* process for routing new nets near-optimally, but without deteriorating the WL's and # of vias of bumped nets significantly, and without violating any slack constraints (see Sec. 5.1), and using the tolerance concepts of Sec. 3 to quickly determine if a re-route of a bumped interconnect of a net satisfies all its slack constraints (see Sec. 5.2).

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.

Referring to Fig. 4b, 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 $C_g(v_u)$ be the gate capacitance of pin v_u, and r and c the unit-length wire resistance and capacitance, respectively. Recall that $D(v_i)$ is the delay from the source/driver pin to the Steiner-node or sink-pin v_i in the routing tree T; see Sec. 2.1. We next define several tolerance concepts that are useful for computing pin-slack-related information (tolerances) at each Steiner node in T, which are useful for locally evaluating slack satisfaction of each sink in T whenever T is changed in some way.

Definition 1: $Tol_{del}(v_i)$ is the maximum delay increase that can be tolerated at v_i without violating the slacks of sink pins in the subtree T_{v_i} rooted at v_i. Thus $Tol_{del}(v_i) = \min_{x \in sinks(T_{v_i})}\{S(x)\}$ or recursively,

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

Note that for a sink pin x, $Tol_{del}(x) = S(x)$ where $S(x)$ is the slack of x. As seen in Fig. 4b, the delay tolerance for node v_4 is 5 ps which is the minimum of the slack (or delay tolerances) of its two children nodes v_6 and v_7.

`Detail-TIDE`(n_i, T); /* T is the global-routing tree of n_i. */
for each interconnect e_{jk} of T **do**
repeat

1. Find a min-cost detailed path for interconnect e_{jk} (from v_j to v_k) in the DR-graph generated within the global tiles assigned to e_{jk} by `Global-TIDE`;
   /* The cost of an edge in the DR-graph includes a base cost for controlling path length, a via-estimate cost and a *bumping cost*. The latter is an estimate of how difficult it will be to re-route any overlapped interconnect of some other net n_j that occupies the current edge, and is based on the # of nets that intersect n_j's bounding box. */
2. **for** each interconnect e' "bumped" by the current e_{jk} path **do**:
   (a) Recursively perform a detailed routing of e' (bumpings of other nets except for ancestors in the current depth-first tree of re-routings are allowed).
   (b) If the depth-first search (DFS) exploration of bumpings and re-routings starting from e' terminates with the leaves of the depth-first tree representing non-bumping routes of the latest bumped interconnects, and all slacks of all nets with re-routed interconnects are satisfied, then record *success* for e_{jk}'s current path.
   (c) Otherwise, increase the cost of the edges in the DR-graph containing those interconnects e' bumped by e_{jk}'s current path, that could not be successfully re-routed, to ∞, and find another path for e_{jk}.

until (*success* or a pre-set # of paths for e_{jk} have been explored);
if no *success* then **return**(*failure*) [for routing n_i].
end for.
return(*success*, n_i's detailed route);

Figure 3: *TIDE's detailed routing phase.*

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:

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

Only the $\Delta C_x = c \cdot \Delta l_x$ 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. 4b where the delay effects on other sinks due to an interconnect length increase to sink pin $x = v_6$ is shown by sequentially numbered dashed arrows. The delay change $\Delta D(v_i)$ at x's parent node v_i is $\Delta D(v_i) = R_{v_i}^{up} \cdot \Delta C_x$. 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 $\Delta C_x \leq \frac{S_{min}(T_{v_i})}{R_{v_i}^{up}} = Tol_{cap}(y) \cdot \frac{R_y^{up}}{R_{v_i}^{up}}$ where y is the sink in T_{v_i} with minimum slack. In other words,

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

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

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. Thus,

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

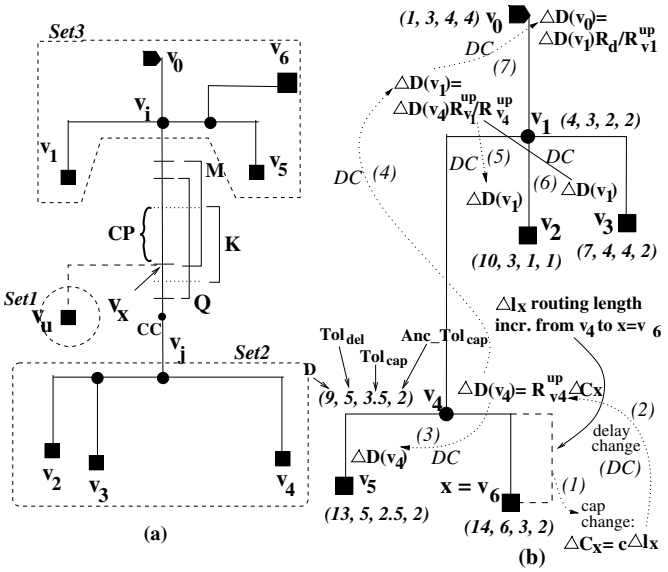

Figure 4: *(a) Q, K, M are the intervals on edge e_{ij} where slack requirements are satisfied for the sinks in Set_1, Set_2 and Set_3, respectively. CP is the interval where Q, K and M intersect. (b) Routing tree with various tolerances shown at each node. The #'s in brackets are $(D, Tol_{del}, Tol_{cap}, Anc\_Tol_{cap})$; note that Tol_{del} of a sink pin is its slack. When the interconn. length to $x = v_6$ increases by Δl_x, and thus the cap. by $\Delta C_x = c \cdot \Delta l_x$, delay changes percolate to various nodes as shown by the #'ed seq. of dashed arrows (labeled as DC (#)).*

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

where v_p is the parent node of v_i in T. The boundary condition for this formulation is $Anc\_Tol_{cap}(v_0) = Tol_{cap}(v_0)$. As shown in Fig. 4b by sequentially numbered dashed arrows emanating from the change in the interconnect length to sink pin $x = v_6$, any change in an interconnect of T_{v_4} also affects the delay of each node, say, $v_2 \notin T_{v_4}$. $Tol_{cap}(v_2) = 1\,fF$ is reflected in v_2's parent v_1 as $Tol_{cap}(v_1) = 2\,fF$ (see Eqn. 2–the reflection of a child's Tol_{cap} on a parent results in a larger Tol_{cap} for the parent as its up-resistance is smaller than the child's), which in turn is reflected in v_1's child node v_4 as $Anc\_Tol_{cap}(v_4) = 2$ *fF* (see Eqn. 3). This means that if the capacitance increase seen in T_{v_4} is more than 2 *fF*, this will violate the slack of v_2. Thus $Anc\_Tol_{cap}(v_i)$ captures the maximum allowable capacitance increase in T_{v_i} beyond which some slack in T will be violated. Hence $Anc\_Tol_{cap}(v_i)$ 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.

4 Interval Intersection Algorithm (IntAl)

As shown in Fig. 2, the incremental global router constructs a routing for net n_i by iteratively connecting the sinks in decreasing order of criticality to the partially routed tree T. We now discuss the `IntAl` algorithm for determining the interval(s) on the edge e_{ij} of T where new pin v_u can be connected without violating slack constraints in the sinks of T. The initial candidate e_{ij} is the nearest edge of T to v_u; how subsequent e_{ij}s are selected when the current candidate does not have a slack-satisfying connection point is shown in Fig. 6 and will be discussed later. Using the terminology of [6], let CC be the closest connection point on e_{ij} from new pin v_u. The min-length slack-satisfying connection point for v_u will be some point on e_{ij} which is close

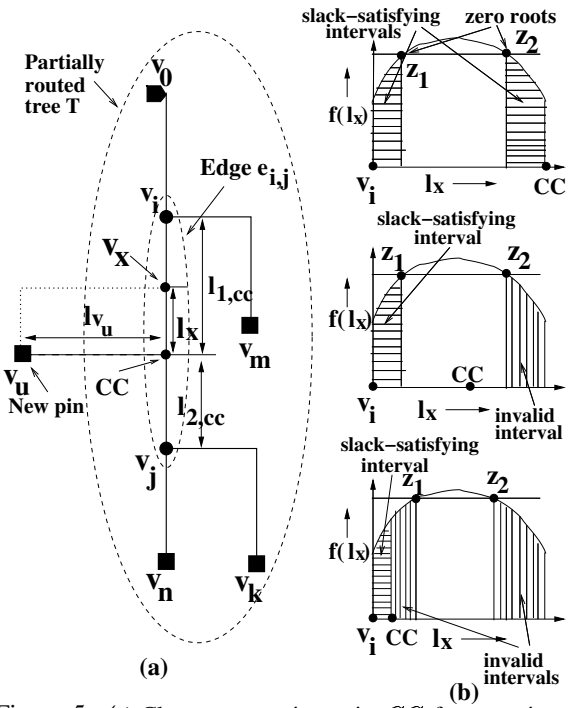

Figure 5: *(a) Closest connection point CC for new pin v_u to partial routing tree T. Connecting new pin v_u to any downstream to CC point v'_x is suboptimal in delay. (b) Different possible slack-satisfying interval(s) on edge e_{ij} obtained from solutions of concave inequalities.*

to and upstream of CC; a connection that is downstream of CC cannot lead to an optimal solution [8]. An important contribution of our work is recognizing that the valid Steiner points for connecting v_u to edge e_{ij} are located in the intersection CP of the slack-satisfying interval(s) of only three "satisfaction functions" corresponding 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. 4a. We refer to Fig. 5a for the notations used in the rest of this section.

1. Slack-satisfying interval for $Set_1 = \{v_u\}$: Let $\Delta cap(CC)$ be the capacitance increase seen at CC if v_u is connected 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, $l_{2,cc}$ be the distance from CC to v_j, 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. 5a. For any connection point $v_x \in e_{ij}$ upstream of CC, the interconnect capacitance $\Delta cap(v_x) = \Delta cap(CC) + c \cdot l_x$ where l_x is the distance between CC and v_x and the downstream capacitance at v_x $C_d(v_x) = C_d(e_{ij}) - c.(l_{1,cc} - l_x)$. Referring to Fig. 5a, using the delay $D(v_i)$ at parent Steiner node v_i, we can calculate $D(v_u)$ as:

$$D(v_u) = D(v_i) + R^{up}_{v_x} \cdot (C_g(v_u) + \Delta cap(v_x))$$

$$+ r_{v_i,v_x} \cdot [\frac{c_{v_i,v_x}}{2} + C_d(v_x)] + r_{v_u,v_x} \cdot [\frac{\Delta cap(v_x)}{2} + C_g(v_u)].$$

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

$$f_1(l_x) = -rc \cdot {l_x}^2 + [R^{up}_{v_i} + 2rc \cdot l_{1,cc} - rC_d(e_{ij}) - \frac{r \cdot \Delta cap(CC)}{2} + \frac{rc \cdot l_{v_u}}{2}] \cdot l_x + H \leq 0 \quad (4)$$

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. 5b. The general procedure for determining the interval(s) Q that satisfy a concave inequality $f_1(l_x) \leq 0$ (as in Ineq. 4) 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 up to the position of v_i and to the right of (and including) z_2 up to the position of CC; see Fig. 5b.
- **If** $(z_1 \geq l_{1,cc})$ (i.e., the z_1-distance node is to the right of CC) **then** $Q = [v_i, CC]$ (see Fig. 5b–bottom). **Exit**.
- **If** $z_1 < l_{1,cc}$ and $z_2 > l_{1,cc}$ **then**
  if $z_1 >= 0$ **then** $Q = [v_i, v_{z_1}]$ (see Fig. 5b–middle), 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** (see Fig. 5b–top)
  { **if** $z_1 >= 0$ **then** $Q = [v_i, v_{z_1}]$ **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 slacks of any sink in Set_2 for v_x to be a valid connection. In other words, $Tol_{cap}(v_x)$ should be greater than the added capacitance; $Tol_{cap}(v_x)$ can be derived from $Tol_{cap}(v_j)$ as:

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

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 the second inequality as a function of l_x:

$$f_2(l_x) = -rc \cdot {l_x}^2 + [R^{up}_{v_j} \cdot c - (\Delta cap(CC) + C_g(v_u))r - rc \cdot l_{2,cc}]l_x$$

$$-Tol_{cap}(v_j) \cdot R^{up}_{v_j} + (\Delta cap(CC) + C_g(v_u)) \cdot (R^{up}_{v_j} - r \cdot l_{2,cc}) \leq 0 \quad (5)$$

$f_2(l_x)$ too is a concave function, and we find its satisfying interval(s) K using a similar procedure to the one 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 $Anc\_Tol_{cap}(v_i)$. Hence we need

$$f_3(l_x) = Anc\_Tol_{cap}(v_i) - [\Delta cap(CC) + c \cdot l_x + C_g(v_u)] \geq 0. \quad (6)$$

By solving this linear inequality we get the valid interval M for Set_3.

Ineqs. 4, 5 and 6 provide us the three respective slack-satisfying intervals Q, K, M shown in Fig. 4a. The final valid interval CP of *candidate points* for connecting the new pin is the intersection of these intervals; see Fig. 4a. 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.

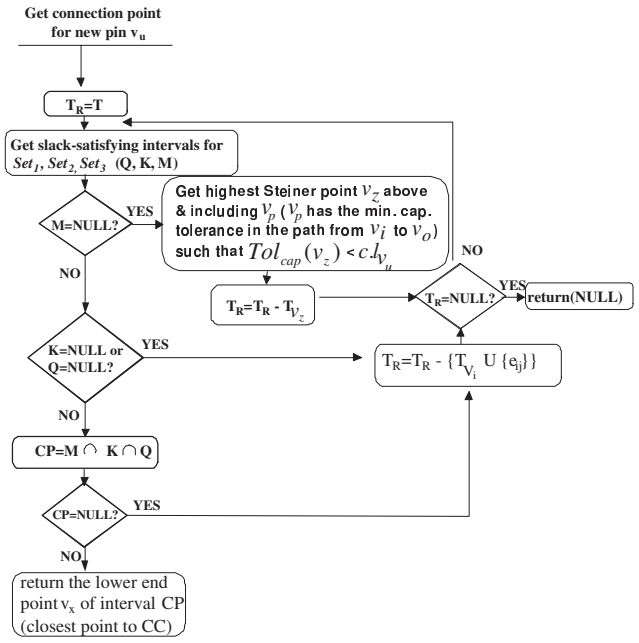

Figure 6: *The IntAl algorithm with tree truncation for obtaining the valid Steiner point set CP for connecting new pin v_u to the partially routed tree T.*

Connection Failure and Tree Truncation Sometimes there may not be a connection point on e_{ij} to connect new pin v_u, i.e. $CP = \emptyset$. We then try another connection point on another segment of T. Fig. 6 gives the flow chart for interval intersection and tree truncation algorithm `IntAl`. As seen in Fig. 6, based on the failure condition, some portion of the routing tree T is truncated in order to eliminate edges that are also guaranteed to not have valid connection points. 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_{ij} 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 1 *The tree truncation method given in Fig. 6 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.*

Finally, tolerance updates are performed in an "as -needed" basis and takes $O(h)$ time per change in T, where h is the height of the tree.

5 New Methods in Incr. TD Detailed Routing

The high-level flow of our detailed router was given in Fig. 3. Here we elaborate on the two innovations embedded in this flow.

5.1 DFS-Controlled Bump-and-Reroute

During detailed routing, sometimes the needed routing resources may not be available in the globally-assigned routing tiles to complete a valid route. In this case, our detailed router will explore a path that overlaps with some existing nets. The *overlapped segments (o-segs)* of these nets in turn will have to be 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 interconnect overlaps and reroutes among some existing nets.

To find a solution for the originally bumped o-seg, we thus perform a depth first search (DFS) of interconnect bumps-and-reroutes; the core non-TD DFS algorithm for this process is developed in detail in [5]. This DFS tree terminates successfully if all its leaf nodes represent non-overlapped routes for the last o-segs on each path in the tree. A DFS path terminates in failure if the route selected for the last o-seg overlaps already overlapped existing nets in the current path or overlaps obstacles or causes constraint violation(s) (e.g. a slack violation on any pin of the net containing the o-seg). If a particular path fails in this manner, the search backtracks and tries another unexplored path 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 for it. This DFS-controlled constraint-satisfying bump-and-reroute process is an important aspect of our TD incremental router. We thus use the acronym *TIDE* (*TIming-driven DEpth-first search controlled routing*) for our algorithm.

5.2 Slack-Satisfying Bumping of Existing Nets

A crucial aspect of the detailed incremental routing phase is determining 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) stored at the Steiner nodes of n_i's routing tree T allow us to quickly and locally determine if a re-routing of the o-seg will satisfy all slack constraints of n_i. There are 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.

Overlapping a leaf interconnect As seen in Fig. 7, 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*.

In this case, we only re-route the interconnect between pin v_l and its parent Steiner node v_p in T with minimal length increase, if any. If the re-route causes an increase in length Δl_{v_l} of the interconnect, we only need to check if the increased delay to v_l is no more than its slack, and if the capacitive tolerances stored in v_p is at least equal to the capacitive increase $\Delta cap = c \cdot \Delta l_{v_l}$ that it sees. The delay increase at pin v_l is

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

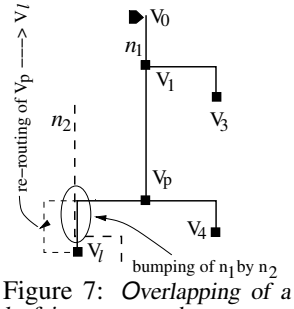

Figure 7: *Overlapping of a leaf interconnect between v_p and v_l, and its rerouting.*

All slack constraints of T will be satisfied if: $\Delta D(v_l) \leq S(v_l)$, and $\Delta cap \leq Anc\_Tol_{cap}(v_p)$.

The last condition guarantees that the extra capacitance does not violate any Steiner node capacitance tolerance and hence any sink pin slacks in T. This is yet another application of node tolerance concepts (the first one was in the `IntAl` algorithm for determining valid connection points for a new pin) which provide a local and efficient means for verifying if interconnect re-routings satisfy all slack constraints of the net without exhaustive checking.

Ckts	num. of nets	num. of pins	% Unrt. nets			Slack viols			Av rout net length ($\times 10^4$)			Vias per new net			Modif nets per new net			Runtime (secs)		
			TD-S	TD-R	TIDE	TD-S	TD-R	TIDE	TD-S	TD-R	TIDE	TD-S	TD-R	TIDE	TD-S	TD-R	TIDE	TD-S	TD-R	TIDE
Ckt1	1643	7200	19.2	3.0	2.8	17.2	3.0	2.3	1.80	2.01	1.58	13.3	14.9	13.2	0	3.7	1.8	62.0	1342.1	90.4
Ckt2	2277	10080	19.7	4.9	3.5	22.6	4.4	3.2	2.53	2.87	2.26	13.2	15.0	13.5	0	4.4	1.8	87.8	1854.0	154.2
Ckt3	2949	12960	25.9	24.2	4.2	28.4	17.5	5.4	3.23	3.69	2.95	12.5	14.4	13.7	0	3.6	2.1	94.0	347.6	203.0
Ckt4	3588	15840	17.6	8.5	3.1	28.0	10.1	5.9	4.00	4.46	3.49	13.2	15.0	13.3	0	3.6	2.3	147.9	1591.2	264.6
Ckt5	4287	18720	22.7	19.5	3.4	37.1	14.1	8.1	4.68	5.33	4.18	12.7	14.6	13.4	0	3.2	2.6	200.5	983.4	212.1
Ckt6	4919	21600	40.2	42.0	3.1	74.7	59.7	8.9	4.94	5.59	4.81	11.3	12.8	13.3	0	5.0	2.2	112.9	202.1	314.9
Ckt7	5572	24480	46.3	47.8	3.1	105.3	95.5	10.7	5.21	5.97	5.49	10.6	12.0	13.4	0	3.7	3.1	128.5	178.2	318.9
Ckt8	6261	27360	23.9	22.5	3.4	53.9	26.6	12.0	6.76	7.71	6.22	12.7	14.6	13.8	0	3.4	2.0	260.5	1501.2	458.5
Ckt9	7613	33120	46.5	48.4	3.2	139.9	127.8	13.7	7.43	8.42	7.58	11.2	12.6	13.7	0	5.8	2.2	196.6	1277.5	449.3
Ckt10	10435	47520	73.1	74.3	15.8	438.0	429.2	98.1	8.76	9.31	8.06	8.8	9.3	12.9	0	10.5	2.3	152.6	167.3	312.6
Avg	*4954*	*21888*	*40.7*	*39.1*	*5.8*	*94.5*	*78.8*	*16.8*	*4.93*	*5.53*	*4.66*	*11.9*	*13.5*	*13.4*	*0*	*4.7*	*2.3*	*144.3*	*944.5*	*277.9*
Factr impr of TIDE			**7.02**	**6.74**	–	**5.63**	**4.69**	–	**1.06**	**1.19**	–	**0.89**	**1.01**	–	**0**	**2.04**	–	**0.52**	**3.40**	–
Avg – glob nets			*31.3*	*30.6*	*3.2*	*56.9*	*44.7*	*10.7*	*20.83*	*42.17*	*6.61*	*45.0*	*75.5*	*17.1*	*0*	*31.3*	*3.3*	–	–	–
Factr impr of TIDE			**9.78**	**9.56**	–	**5.32**	**4.18**	–	**3.15**	**6.38**	–	**2.63**	**4.42**	–	**0**	**9.48**	–	–	–	–

Table 1: *Circuit-wise comparison of metrics across TIDE, TD-Std (TD-S) and TD-R&R (TD-R) for ECO simulated by random deletion of 10% of existing nets and random generations of new nets to connect the resulting unconnected pins. All metrics reported are averaged over ten random runs of net deletions and additions.*

6 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 and randomly connecting them by nets of 2-14 pins[2]. Their characteristics are shown in Table 1; the number of nets ranges from 1643 to 10,435. 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 source to sink-pin delay in n_i [3].

To simulate ECO, we randomly deleted 10% of the original nets, and randomly added the required new nets to connect the unconnected 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 TD routing algorithm SERT/SOAR composed of elements of two well-known prior TD routing algorithms, SERT-C [8] and SOAR [9]. We 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 "crash test" variety, where some failures are guaranteed to happen, and the overall success rate is thus an important metric. We ran all three methods on a 2.6 Ghz Pentium Linux machines with 1GB of RAM. The results are given in Table 1; **Modif nets per new net** is the average number of existing nets re-routed to accommodate a new net, while the other metrics are self-explanatory. TIDE was able to route almost all the nets ($\approx 94\%$) without sacrificing any quality metric. TIDE has ≈ 7 and ≈ 6.7 times fewer unrouted nets than TD-Std and TD-R&R, respectively. 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 if these timing specifications are actually attainable. As shown in Table 1, 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, while TD-Std uses 11% fewer 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 1 show results for *global nets*—nets whose lengths are $\geq 50\%$ of the HP of the chip—which underscore TIDE's efficacy efficacy for both local and global nets, and thus for complex circuits.

[2] Generated according to the foll. distr.: 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%.

[3] The following technology parameters were used assuming a $0.18\mu m$ technology: $r = 0.076\Omega/\mu m$, $c = 0.118fF/\mu m$, driver resistance $R_d = 180\Omega/\mu m$ and gate capacitance $C_g = 23.4fF$.

7 Conclusions

A timing-driven (TD) incremental routing algorithm TIDE was developed which solves the exact TD routing problem — satisfying sink pin slacks while optimizing WL, routing completion and the number of vias. 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, Steiner node tolerance concepts that allow accurate, fast and local determination of slack satisfaction for all pins, and a DFS-controlled bump-and-reroute process that explores a richer solution space for new-net routing without significantly compromising routing metrics 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 and slack violations, when compared to the TD versions of the well known Std and R&R incremental routing methods.

References

[1] J.M. Emmert and D. Bhatia. "Incremental Routing in FPGAs". *Proc. IEEE Int. ASIC Conference and Exhibit*,'98.

[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*, 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] H. Arslan and S. Dutt, "A Depth-First Search Controlled Gridless incremental routing Algorithm for VLSI Circuits", *ICCD*, 2004, pp. 86-92.

[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.