Utilizing Redundancy for Timing Critical Interconnect

Shiyan Hu, Qiuyang Li, Jiang Hu, Peng Li

***Abstract*— Conventionally, the topology of signal net routing is almost always restricted to Steiner trees, either unbuffered or buffered. However, introducing redundant paths into the topology (which leads to non-tree) may significantly improve timing performance as well as tolerance to open faults and variations. These advantages are particularly appealing for timing critical net routings in nano-scale VLSI designs where interconnect delay is a performance bottleneck and variation effects are increasingly remarkable. We propose Steiner network construction heuristics which can generate either tree or non-tree with different slack-wirelength tradeoff, and handle both long path and short path constraints. We also propose heuristics for simultaneous Steiner network construction and buffering, which may provide further improvement in slack and resistance to variations. Furthermore, incremental non-tree delay update techniques are developed to facilitate fast Steiner network evaluations. Extensive experiments in different scenarios show that our heuristics usually improve timing slack by hundreds of pico seconds compared to traditional approaches. When process variations are considered, our heuristics can significantly improve timing yield because of nominal slack improvement and delay variability reduction.**

***Index Terms*— Routing, Steiner network, non-tree, variation, redundancy.**

I. Introduction

Signal net routing is an important part of VLSI circuit designs since it directly affects interconnect delay which is a well-known performance bottleneck. In practice, people almost always use Steiner tree [1], either buffered or unbuffered, for signal net routing because it is cost-effective and its delay is relatively easy to compute. However, introducing redundant paths into the topology (which leads to non-tree) has some remarkable advantages compared to trees. Non-tree topology can often obtain significantly better timing performance than trees [2], [3]. In addition, the redundant paths in a non-tree network provide certain tolerance to open fault and therefore can improve manufacturing yield and reliability [4]. Moreover, non-tree topology sometimes can reduce delay variations [4]. Even though non-tree delay computation is more expensive than that of trees, the increasingly strong need for improving interconnect performance and fault/variation tolerance may force us to adopt non-tree topology for the timing critical nets. On the other hand, the computation overhead can usually be alleviated by the advancement on computation techniques and facilities. Previous transitions from power tree to power grid and from clock tree to clock mesh indicate that design needs often eventually outweigh computation overhead if the overhead is not prohibitively large.

The authors are with Department of Electrical and Computer Engineering, Texas A&M University, College Station, Texas 77843, Email: {hushiyan, qiuyang, jianghu, pli}@ece.tamu.edu.

Perhaps the first non-tree routing work is [2] which greedily adds extra wires on given trees to minimize source-sink delay. The later work of [3] inserts a link between the source and the sink having the maximum delay. The recent work of [4] is focused on another aspect of non-tree routing - reliability and manufacturing yield. It augments extra edges to an existing tree to increase the percentage of 2-connected wires, which implies tolerance to open faults. The previous works [2], [3] on timing driven non-tree routing have two main weaknesses. Since they add wires to existing trees, the performance of the resulting non-trees depend on the initial trees. Starting with an arbitrary tree cannot ensure if this tree can facilitate a good non-tree solution. The other weakness is that they [2], [3] optimize only delay without considering timing constraints. In reality, maximizing slack or minimizing wire cost subject to timing constraints is a more common and useful problem formulation [5].

The timing constraints in previous works [5] are almost always upper bounds for sink delays. In fact, there are delay lower bounds due to the short path (hold time) constraints in synchronous circuits. Some gate sizing works [6] consider both delay upper bound and lower bound at the same time. To the best of our knowledge, there is no signal net routing work considering the double-sided timing constraints yet. This is perhaps due to the reason that delay lower bound can be easily satisfied by padding extra delay. The delay padding can be implemented by wire detour, adding dummy capacitors or inserting redundant buffers. The former two approaches may increase the delay along the long path. The latter approach of redundant buffers may intensify the leakage power problem. Thus, the short path constraints need to be handled in a more careful manner.

We propose Steiner network construction heuristics which consider delay upper bound and lower bound simultaneously for timing critical nets. We will show that sometimes a link insertion can simultaneously reduce long path delay and increase short path delay. The first heuristic is a greedy link insertion in an existing tree, which is similar as [2] but the solution search is trimmed for the double-sided timing constraints. The second one is a dynamic programming based constructive algorithm which can generate a set of solutions with different slack-wirelength tradeoff and can reach either tree or non-tree topology. The third one extends the above heuristic to handle simultaneous Steiner network construction and buffer insertion, which may provide further improvement in slack and resistance to variations. Incremental non-tree delay update techniques are also developed for improving the efficiency of our algorithms. Extensive experimental results

show that our Steiner network construction, either buffered or unbuffered, usually improves slack by hundreds of pico seconds compared to the traditional tree results. Moreover, our constructive method almost always outperforms the greedy approach like [2]. The non-tree approach may bring some wirelength and runtime overhead, but the impact to overall chip design is very limited considering that it is applied to only a small amount of timing critical nets. When process variations are considered, Monte Carlo simulation results show that our methods can improve timing yield greatly because of both nominal slack improvement and delay variability (standard deviation) reduction.

The rest of the paper is organized as follows: Section II formulates the Steiner network construction problem. Section III presents the fast incremental non-tree delay update procedure. Section IV describes the proposed Steiner network construction algorithms without buffers. Section V describes the simultaneous Steiner network construction and buffering algorithm. Section VI presents the experimental results with analysis. A summary of work is given in Section VII.

II. Preliminary

We will show that link insertion in an existing tree or non-tree may simultaneously reduce long path delay and increase short path delay under certain condition. Refer to Figure 1 for an RC network, which can be either a tree or a non-tree. The Elmore delay from the source to a node i in an RC network is given by $t_i = \sum_j R_{i,j}C_j$ where C_j is the ground capacitance at j, and $R_{i,j}$ is the transfer resistance which equals the voltage at node i when $1A$ current is injected into node j and all the other node capacitances are set to zero [3]. Consider inserting a link between two nodes i and j in the RC network. Let the link resistance be R and link capacitance be C. This link insertion is equivalent to adding capacitance $C/2$ at node i and j, respectively, and inserting resistance R between i and j.

Let $t_{i,lc}$ ($t_{j,lc}$) denote the delay increase at i (j) due to adding link capacitor C. Then $t_{i,lc} = \frac{C}{2}(R_{i,i} + R_{i,j})$ and $t_{j,lc} = \frac{C}{2}(R_{i,j} + R_{j,j})$, where $R_{i,i}(R_{j,j})$ is the path resistance from the source to node $i(j)$ and $R_{i,j}$ is their shared path resistance. After the link insertion, the delay to i and j are changed from t_i and t_j to $\tilde{t}_i$ and $\tilde{t}_j$ according to the following equations [7]:

$$\tilde{t}_i = (1-\alpha)(t_i + t_{i,lc}) + \alpha(t_j + t_{j,lc}) \quad (1)$$

$$\tilde{t}_j = (1-\beta)(t_j + t_{j,lc}) + \beta(t_i + t_{i,lc}), \quad (2)$$

where $\alpha = \frac{r_i}{R+r_i-r_j}$ and $\beta = \frac{r_j}{R+r_i-r_j}$. In general, r_i and r_j are equal to the Elmore delays at i and j, respectively, when node capacitance $C_i = 1$, $C_j = -1$ and the other node capacitances are set to zero [3].

The above equations show that the link capacitance always increases signal delay while the link resistance attempts to average the delay between i and j. It is straightforward to derive the following condition on the simultaneous improvement for both long path and short path delay.

Lemma: *If a link with resistance R and capacitance C is inserted between a node i on a long path and a node j on a short path in a Steiner network, the necessary and sufficient condition of simultaneously reducing delay to node i and increasing delay to node j is $t_i \geq (\frac{1}{\alpha} - 1)t_{i,lc} + t_j + t_{j,lc}$.*

Proof: First note that both α and β are positive. To reduce long path delay, i.e., $\tilde{t}_i \leq t_i$, we need $(1-\alpha)(t_i + t_{i,lc}) + \alpha(t_j + t_{j,lc}) \leq t_i$, which is $t_i \geq (\frac{1}{\alpha} - 1)t_{i,lc} + t_j + t_{j,lc}$.

To increase the short path delay, we need $\tilde{t}_j \geq t_j$. This is true since $t_i \geq (\frac{1}{\alpha} - 1)t_{i,lc} + t_j + t_{j,lc}$ and

$$\begin{aligned} & \tilde{t}_j \\ = \; & (1-\beta)(t_j + t_{j,lc}) + \beta(t_i + t_{i,lc}) \\ \geq \; & (1-\beta)(t_j + t_{j,lc}) + \beta((\frac{1}{\alpha} - 1)t_{i,lc} + t_j + t_{j,lc}) + \beta t_{i,lc} \\ = \; & t_j + t_{j,lc} + \frac{\beta}{\alpha}t_{i,lc} \\ \geq \; & t_j \end{aligned}$$

■

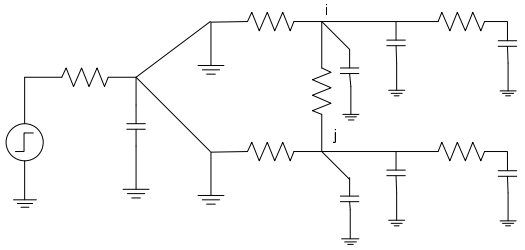

Fig. 1. Inserting a link between node i and node j in an existing network.

When consider double sided timing constraints, each sink v_i has a delay upper bound $\overline{q}_i$ and a delay lower bound $\underline{q}_i$. The delay upper bound is the same as the required arrival time (RAT) in traditional methods. We define the **late slack** of a sink v_i as $\overline{s}_i = \overline{q}_i - t_i$ where t_i is the delay. Similarly, the **early slack** of a sink v_i is defined as $\underline{s}_i = t_i - \underline{q}_i$. The **slack** of a sink v_i is $s_i = \min(\overline{s}_i, \underline{s}_i)$. The late slack, early slack and slack of a network (or subnetwork) are the minimum late slack, early slack and slack among all sinks in the network, respectively. For a network (or subnetwork), the sink having the minimum late (early) slack is called **late (early) critical sink**. Here is our problem formulation:

Timing Driven Steiner Network Construction: *Given a source node v_0, a set of sink nodes $\{v_1, v_2, ..., v_n\}$ with each sink v_i having load capacitance c_i, lower delay bound $\underline{q}_i$ and upper delay bound $\overline{q}_i$, construct a rectlinear Steiner network spanning the source and the sinks such that the slack of the network is maximized.*

The above problem is solved in Section IV. We also consider the problem of simultaneous buffer insertion and Steiner network construction in this paper. That is, the network construction process also involves buffer insertion. Such an algorithm is given in Section V.

III. Incremental Non-tree Delay Update

One hurdle of using non-tree topology is the complexity of computing its delay. The first order delay model for an RC network is $\mathbf{t} = \mathcal{G}^{-1}\mathbf{C}$ where $\mathbf{t}$ is the node delay vector,

$\mathcal{G}$ is the conductance matrix and $\mathbf{C}$ is the node capacitance vector. Frequently performing matrix inverse operations in an optimization procedure may consume a lot of runtime. In this section, we propose incremental non-tree delay update techniques. If the node capacitors are treated as current sources, the first order delay computation is equivalent to DC analysis in a linear circuit. Therefore, we introduce these techniques in the language of DC analysis for the convenience of presentation.

We start with a simple example of linear DC circuit in Figure 2(a) without loss of generality. Using the standard MNA (Modified Nodal Analysis) analysis, a set of KCL/KVL equations can be derived for the circuit as

$$Gx = Bu, \tag{3}$$

where

$$G = \begin{bmatrix} G_1 & G_1 & 0 & 0 & 1 \\ -G_1 & G_1+G_2+G_3 & -G_2 & -G_3 & 0 \\ 0 & -G_2 & G_2 & 0 & 0 \\ 0 & -G_3 & 0 & G_3 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

is the conductance matrix, $x = [V_1\ V_2\ V_3\ V_4\ I_{s1}]^T$ is the DC solution, $u = [I_{in1}\ I_{in2}\ I_{in3}\ V_{s1}]^T$ is the input, and

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

links the inputs to the circuit. The number of circuit unknowns is denoted as N which is 5 in this case. The current of the voltage source can be expressed as $I_{s1} = L^T x$, where $L = [0\ 0\ 0\ 0\ 1]^T$. In the following subsections, we use the notation $(G/B/L)$ to describe a linear DC circuit. Given a linear DC circuit and the LU factor of the conductance matrix, we will show how to efficiently update the DC solution if an incremental change is made to the circuit. Since the update after link insertion has been discussed in [8], we focus on the update for the other incremental changes.

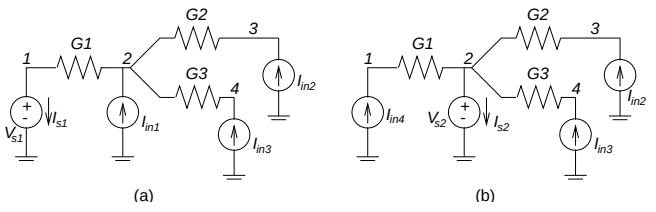

Fig. 2. A linear DC circuit. The root node is changed from 1 to 2 in (b).

A. Reroot

Reroot means the root (voltage input) of a subnetwork is changed from one node to another. We illustrate this operation by moving the voltage source from node 1 (in Figure 2(a)) to node 2 (in Figure 2(b)). In Figure 2(b), a current source is added to node 1 to reflect the node capacitance there. To solve the new DC circuit, an LU factor will be needed for the updated G matrix. The updated conductance matrix G_r can be obtained from G via a rank-4 update $G_r = G + PQ$ where P and Q are $N \times 4$ and $4 \times N$ matrices, respectively, given by

$$P = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Using the well-known matrix inversion formula [9], the inverse of G_r can be obtained from that of G as

$$G_r^{-1} = G^{-1} - G^{-1}P(I + QG^{-1}P)^{-1}QG^{-1}, \tag{5}$$

where $I + QG^{-1}P$ has a dimension of 4×4, therefore it is inexpensive to invert. Notice that the inversions of G and G_r are never formed explicitly. Instead, only the LU factor of G is reused to solve any linear system defined by G_r efficiently.

B. Adding Wire

In this case, a wire (modeled as a conductance G_a and a current source I_a corresponding to wire capacitance) is inserted into the jth internal node of the circuit. To solve the updated circuit, we need to consider the modified conductance:

$$G' = \begin{bmatrix} \tilde{G} & E_{a,j} \\ E_{a,j}^T & G_a \end{bmatrix}, \tag{6}$$

where $\tilde{G}$ is almost identical to G except that the (j,j) location increases by G_a, $E_{a,j} = -G_a E_j$ and E_j is a column vector and has a 1 at the jth location and zeros otherwise. Since $\tilde{G}$ can be obtained from G via low-rank update: $\tilde{G} = G + G_a E_j E_j^T$, an LU factor of G' can be implicitly obtained by using the matrix inversion lemma as in (5). Now consider an arbitrary linear system solution with respect to G' and a right hand side b. The set of circuit equations are partitioned into

$$G'x = \begin{bmatrix} \tilde{G} & E_{a,j} \\ E_{a,j}^T & G_a \end{bmatrix} \begin{bmatrix} x_I \\ x_a \end{bmatrix} = \begin{bmatrix} b_I \\ b_a \end{bmatrix}. \tag{7}$$

Substituting the upper part of (7) into the lower part gives

$$x_a = \left[G_a + E_g^T \tilde{G}^{-1} E_g\right]^{-1} [b_a - E_g^T \tilde{G}^{-1} b_I], \tag{8}$$

where $G_a + E_g^T \tilde{G}^{-1} E_g$ is a scalar therefore its inversion is trivial. Once x_a is obtained, x_I can be solved using the upper part of (7).

C. Merging Two Networks

Now consider the delay update after merging two subnetworks. As shown in Figure 3, a network described by system matrices G_1, B_1, L_1 is being merged into another network (G_2/B_2) at its jth internal node. For the first network, it is assumed that the system matrices are constructed by treating the merging point as an input voltage port. Therefore, L_1 can be used to select the port current I_{merge} when the voltage of the jth internal node of the second network is provided as

$$I_{merge} = L_1^T G_1^{-1} B_1 V_j. \tag{9}$$

If the merging point for the first network varies, a reroot procedure (Section III-A) can be used to implicitly construct

an LU factor of the G_1 matrix. After the merging, the DC system equation for the network (G_2/B_2) becomes

$$G_2 x = B_2 u + e_j I_{merge}, \tag{10}$$

where e_j is a column vector with a proper length and has a 1 at the jth location and zeros otherwise. The above equation gives the voltage at the jth node as $V_j = e_j^T G_2^{-1}(B_2 u + e_j I_{merge})$. Substituting this relationship into (9) leads to

$$V_j = \frac{e_j^T G_2^{-1} B_2 u}{1 - e_j^T G_2^{-1} L_1^T G_1^{-1} B_1}. \tag{11}$$

From (11), the rest of circuit responses in both networks can be subsequently solved.

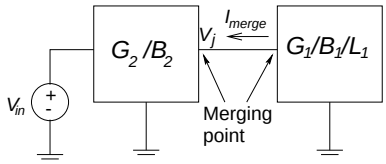

Fig. 3. Merging two subnetworks.

IV. Unbuffered Steiner Network

A. Greedy Link Insertion

In this section, we introduce a greedy heuristic which inserts links in an existing Steiner network such that the slack is maximized considering the double-sided timing constraints. This heuristic can be applied either for obtaining a non-tree network from a tree or as a subroutine in a constructive Steiner network algorithm.

For the given network G, which can be either tree or non-tree, we first identify its early critical sink v_e and late critical sink v_l (both defined in Section II). Next we find the shortest path $p_{e,l} \in G$ which connects the two critical sinks (using only existing edges). For each node $v_i \in p_{e,l}$, we tentatively insert a link between v_i and each edge $e_j \in p_{e,l}$ with the shortest connection. If node v_i is at coordinate (x_i, y_i), and the two ending nodes of e_j are at (x_j, y_j) and (x_k, y_k), respectively, the link is inserted between node v_i and location (x_c, y_c) where $x_c = median(x_i, x_j, x_k)$ and $y_c = median(y_i, y_j, y_k)$. For each link insertion result, we evaluate the slack S of the network. The link which gives the maximum slack improvement is finally chosen to be inserted. After one link insertion is completed, we repeat this procedure till there is no slack improvement. In the case when short path constraints are neglected, the role of the early critical sink is played by the sink with the maximum late slack.

B. Discussion on Topology

The effect of link insertion depends on the initial tree topology. Previously, there were many discussions on the area-radius tradeoff among tree topologies [1]. The area refers to the total wirelength and the radius is the maximum source-sink path length in a tree. The two extreme cases of this trade-off are: (1) chain-like topology (Figure 4(a)), which has small area and large radius, and is usually derived from minimum spanning tree algorithms; (2) star-like topology (Figure 4(b)) with relatively large area and small radius, and can be obtained from the shortest path tree or Rectilinear Steiner Aborescence (RSA) algorithms [1].

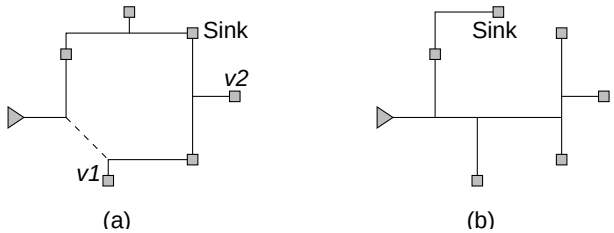

Fig. 4. (a) Chain-like topology and (b) star-like topology. Link is indicated by dashed lines.

The major weakness of a tree with chain-like topology is that the delay of some sink may suffer from the long path length. For example, if v_1 in Figure 4(a) is the late critical sink with tight delay upper bound, the long detour may cause large delay constraint violation. If we include non-tree topology into consideration, we may reach different conclusions. If a link (dashed line) is inserted in the chain-like topology as in Figure 4(a), the long detour problem is eliminated and the small wirelength is still enjoyed. However, if the late critical sink is v_2 instead, perhaps the star-like topology in Figure 4(b) is still better. Thus, it is not clear which tree topology can facilitate a good non-tree solution in general. A main effort in our constructive algorithm is to probe different topologies so that the chance of capturing good non-tree solutions can be increased.

C. Constructive Steiner Network Heuristic

If we treat a network as a tree plus links, the problem of network construction can be accordingly decomposed into finding a proper tree topology and link insertions. We combine these two concerns into a dynamic programming based heuristic. This heuristic is a bottom-up merging procedure where multiple candidate solutions are generated to probe good topologies and link insertions. At the beginning, a set of subnetworks are initialized with the sink nodes. In each iteration, a pair of subnetworks are selected to be merged. Different merging solutions are generated. For each new subnetwork resulting from a merging, another candidate solution is generated by inserting a link in it. These candidate solutions are propagated toward the source.

Solution characterization and pruning. A candidate solution $O_{i,j}$ is a subnetwork $G_{i,j}$ rooted at node v_i. It can be characterized by the total load capacitance $C_{i,j}$, delay lower bound $\underline{q}_{i,j}$ and delay upper bound $\overline{q}_{i,j}$ at v_i. It is easy to derive that the delay upper bound $\overline{q}_{i,j}$ is same as the late slack of $G_{i,j}$. Similarly, the delay lower bound $\underline{q}_{i,j}$ is equal to the negative of early slack. If there is another candidate solution $O_{i,k}$ at node v_i and it has the exactly same sink set as $O_{i,j}$, the two solutions can be compared for pruning. If $C_{i,j} \leq C_{i,k}$, $\underline{q}_{i,j} \leq \underline{q}_{i,k}$ and $\overline{q}_{i,j} \geq \overline{q}_{i,k}$, solution $O_{i,k}$ is *inferior*. During

the solution propagation, inferior solutions are pruned out. The procedure for checking whether solutions are inferior is called *inferiority check*.

Merging selection. We propose two merging selection criteria for two different scenarios: (1) long path constraints and short path constraints are almost equally tight, and (2) long path constraints dominate.

For the first scenario, we use a merging scheme similar to prescribed skew clock tree routing [10]. In fact, when the delay upper bound of each sink is equal to its delay lower bound, i.e., the delay constraints degenerate to a single value target, this problem is equivalent to prescribed skew clock routing. In prescribed skew clock routing, the subtree with the maximum delay target is merged first to reduce the chance of wire detour [10]. Since we have delay upper and lower bound instead of a single delay target, we use the average $(\overline{q} + \underline{q})/2 + t'$ as the criterion. The t' is the anticipated wire delay from the root of the subnetwork to the source node. This is to encourage subnetworks with roots far away from the source to be merged early. In each iteration, we first choose the subnetwork with the maximum $(\overline{q} + \underline{q})/2 + t'$, and then merge it with its nearest neighboring subnetwork.

The second scenario is more like traditional signal routing [1]. Therefore, we adopt a merging criterion similar as that of Rectilinear Steiner Aborescence (RSA) [11]. That is, we choose a pair of subnetworks whose merging root is farthest from the source among all pairs. If we consider merging subnetworks rooted at (x_i, y_i) and (x_j, y_j), then the merging root is at $(x_m = median(x_i, x_j, x_0), y_m = median(y_i, y_j, y_0))$ where (x_0, y_0) is the location of the source node. Then, the pair with the maximum value $|x_m - x_0| + |y_m - y_0|$ is selected for a merging. Our method is different from the well-known RSA algorithm [11] which restricts all sinks in one quadrant if the source is at $(0, 0)$. Our merging selection can handle the cases that sinks are distributed in multiple quadrants.

Merging. After a pair of subnetworks are selected, we consider two types of mergings between them. One is the root-root merging as in Figure 5(a) where subnetwork G_5 and G_6 are merged at node v_7. The other is the shortest merging where two nodes from the two subnetworks with the minimum distance are connected directly. After the merging, the node closest to the source is selected as the root for the merged network. For example, in Figure 5(b), the merging between G_5 and G_6 is obtained by connecting v_2 and v_3 where reroot occurs. Then, v_5 is chosen as the root.

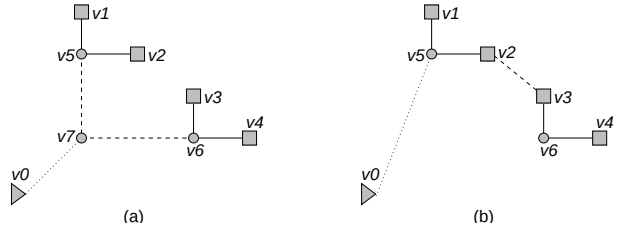

Fig. 5. (a) Root-root merging and (b) the shortest merging. Reroot occurs in (b).

The root-root merging is very similar as the RSA [11] heuristic which leads to star-like topology. The shortest merging is more likely to result in chain-like topology. By having these two different types of mergings, various topologies can be generated to compete for the best slack solution.

Non-disjoint merging. If two *subtrees* contain some common sink nodes, i.e., they have non-disjoint sink sets, the merging between them results in a non-tree even without link insertion. In previous work on dynamic programming based buffered Steiner tree [12], such merging is forbidden since a tree topology needs to be retained. In general Steiner network construction, whether or not to allow the non-disjoint merging becomes less clear. According to our experience, allowing non-disjoint mergings increases runtime dramatically with insignificant gain on timing slack.

Link insertion. For the two subnetworks obtained from mergings, we insert a link in each of them. The link insertion procedure is almost the same as that described in Section IV-A except that only one link is inserted for each subnetwork. There are two reasons for this difference. First, the link insertion here is an intermediate step and trying multiple links in one step is computationally expensive. Second, the size of a subnetwork is relatively small and therefore the number of necessary links is normally small.

Solutions at the source. At the source, there are a set of solutions with different capacitance and slack trade-off. One can choose either the maximum slack solution or the minimum capacitance solution without negative slack.

We say that a solution is *incomplete* if it has more than one subnetworks. Otherwise, it is *complete*. The pseudo-code for the Steiner network construction algorithm is given in Figure 6.

V. Buffered Steiner Network

In this section, we are to describe the constructive method for simultaneous Steiner network construction and buffer insertion. As we will deal with a net driven by multiple drivers, two issues, which do not exist in link insertion for unbuffered Steiner network, have to be considered. One is the risk of short circuit between different drivers. The other is the fast estimation of signal delays in multi-driver nets.

A. Short Circuit Avoidance

In a buffered non-tree, it is likely that multiple buffers drive the same subnet. Then, there is a risk of short circuit from one buffer to another, especially when the signal arrival times to the two buffers are quite different[1]. It is noticed in [13] that the short circuit can be avoided if the arrival time difference is smaller than the signal propagation delay between the two buffers. To be safer, we can require that the upper bound of the arrival time difference is smaller than the lower bound of the delay between the two buffers.

Denote the two buffers by B_i and B_j. The lower bound $\tau_{i \rightsquigarrow j}$ of signal propagation delay from B_i to B_j can be obtained through the method of [14] as follows.

$$\tau_{i \rightsquigarrow j} = \frac{\sum_{(u,v) \in path(B_i \rightsquigarrow B_j)} R_{uv}^2 C_v}{\sum_{(u,v) \in path(B_i \rightsquigarrow B_j)} R_{uv}}, \tag{12}$$

[1]See [13] for more details and illustrations.

Procedure: Steiner network construction without buffers
Input: a source node and a set of sinks
Output: a set of routing solutions (Steiner networks) spanning on the source node and sinks
1. initialize each sink as an individual subnetwork and place this solution into the empty solution set
2. repeat
3. pick and remove an incomplete solution from the solution set
4. choose two subnetworks to merge according to the current scenario
5. perform root-root merging to generate a new solution
6. perform shortest merging to generate another new solution
7. perform link insertion to each solution
8. place the two new solutions into the solution set and perform inferiority check
9. until all solutions in the solution set are complete

Fig. 6. Procedure of Steiner network construction without buffers.

where (u, v) indicates two end nodes of an edge, R_{uv} is the edge resistance and C_v is the total capacitance downstream of node v. Similarly we can obtain the lower bound $\tau_{j\rightsquigarrow i}$ of signal propagation delay from B_j to B_i. Denote by $\Delta_{ij,max}$ the upper bound of the difference between signal arrival time to B_i and B_j considering variations. The criterion for avoiding short circuit between B_i and B_j is [13]:

$$\min(\tau_{i\rightsquigarrow j}, \tau_{j\rightsquigarrow i}) > \alpha\Delta_{ij,max} \tag{13}$$

where $\alpha > 1$ is a constant used for added safety margin. Refer to [13] for further details.

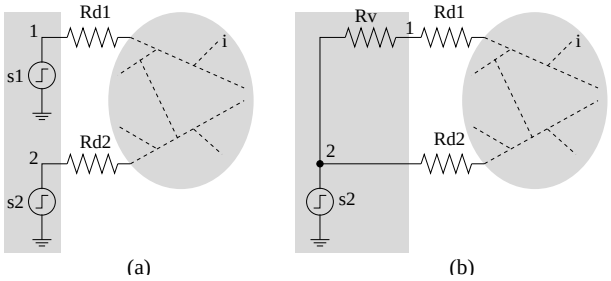

Fig. 7. [13] The dual driver net in (a) can be converted to the single driver net in (b) when signal departure time t_1 at node 1 is no less than the signal departure time t_2 at node 2.

B. Multi-driver Delay Estimation

Although signal delay in a multi-driver net can be computed by SPICE or model order reduction methods such as AWE [15], an Elmore-like method is necessary for fast delay estimation during optimizations. Such a method is very similar to the one proposed in [13]. For completeness, we include it in this paper.

The main idea is to transform a multi-driver net to an equivalent single driver net. For the convenience of presentation, the method in [13] is described using a dual-driver case. If there are two drivers for a net as in Figure 7(a), the term of signal delay is not well defined as the signal departure time from the two drivers may be different. As such, we need to find the signal arrival time t_i to a node i in the net given the signal departure time t_1 and t_2 from node 1 and node 2 in Figure 7(a). Without loss of generality, assume that $t_1 \geq t_2$, i.e., $t_1 = t_2 + \Delta$ and $\Delta \geq 0$. We are to insert a virtual resistance R_v between the signal source s_1 and node 1 such that the signal delay across the virtual resistance is Δ and the signal departure time from s_1 is t_2. Setting the signal departure time at s_1 and s_2 both to t_2 after this insertion, we can merge s_1 with s_2 into a single source as shown in Figure 7(b). It remains to find the value of the virtual resistance R_v such that the delay across it is equal to Δ. Since the net in Figure 7(b) has a single driver, the signal delay t_1 at node 1 is well defined by letting the signal departure time $t_2 = 0$. Clearly, t_1 is a function $t_1(R_v)$ and $t_1(R_v) = \Delta$ [13].

Following [13], for the current non-tree, we can find a single node, which is called joint node, such that the non-tree can be transformed into a tree by tearing the joint node into two separated nodes u and w. For Figure 7(b), node tearing separates the non-tree into subtree T_1, which is driven through R_v and R_{d1}, and subtree T_2 which is driven through R_{d2}. If node $u \in T_1$ and node $w \in T_2$, the delay at node 1 is $R_v C_1$ where C_1 is the total downstream capacitance at node 1 in subtree T_1. After node u and node w are merged back, the delay at node 1 satisfies [7]:

$$t_1(R_v) = R_v C_1 - \frac{t_u - t_w}{r_u - r_w} R_v = \Delta \tag{14}$$

where t_u and t_w are delays at node u and w before the merging. The values of r_u and r_w are equal to the Elmore delay at u and w, respectively, when node capacitance $C_u = 1, C_w = -1$ and the other node capacitance are zero [7].

Following [13], t_u can be decomposed as $t_u = R_v C_1 + t_{1,u}$ where $t_{1,u}$ is the delay from node 1 to u before merging. Similarly, r_u can be decomposed as $r_u = R_v + R_{1,u}$ where $R_{1,u}$ is the total path resistance from node 1 to node u. The value of $-r_w$ is equal to the total path resistance $R_{2,w}$ from node 2 to node w before the merging. The value of R_v can be then derived from Equation (14) as:

$$R_v = \frac{(R_{1,u} + R_{2,w})\Delta}{(R_{1,u} + R_{2,w})C_1 + t_w - t_{1,u} - \Delta} \tag{15}$$

C. Simultaneous Steiner Network Construction and Buffering

The simultaneous buffering and Steiner network construction algorithm goes in the same way as the unbuffered case described in Section IV-C except that at each merging point, we consider the possibility of inserting a buffer there. That is,

at each merging point, both buffered solutions and unbuffered solutions are placed into the solution set. To update the delay/slack of the buffered Steiner subnetworks, the method described in Section V-B is employed. In addition, to avoid short circuit problem, a buffer can be inserted only in "feasible" positions, i.e., those merging points such that the criterion in Equation (13) is satisfied. We find that allowing buffer insertion at each feasible position may result in too many solutions and make the algorithm very inefficient. Speedup techniques are necessary. As such, we set up a threshold T_b and a buffer is inserted when the number of non-inferior solutions is less than T_b. A buffer is inserted also when a random number rand$(0, 1)$ is smaller than a value r set by user, where rand$(0, 1)$ denotes a random number generator with output ranging from 0 to 1. As indicated by our experiments, this strategy improves the efficiency of the algorithm with slight degradation in solution quality. Refer to Figure 8 for the pseudo-code of the simultaneous algorithm.

VI. Experimental Results

All algorithms are implemented in C++ and the experiments are performed on a PC computer with 3.2GHz processor and 1G memory. We generated different testcases with the number of sinks ranging from 5 to 25, which are typical sizes for signal nets in reality. Without loss of generality, we let the source be at coordinate of $(0, 0)$. In some cases, all of the sinks are in one quadrant while some other cases have sinks distributed in four quadrants. For example, in the data tables, the notation of "15s, 2Q" means there are 15 sinks and they are distributed in two quadrants. The $70nm$ technology parameters reported in [16] are employed in the experiments[1]. We compare the following methods:

- **AHHK**. This is a Steiner tree heuristic [1] which can achieve different area-radius tradeoff by varying a parameter $\alpha \in [0, 1]$. When the value of α is shifted from 0 to 1, the resulting tree gradually changes from chain-like to star-like topology [1]. Although it is not directly timing driven, we can achieve very good timing performance by trying different α and choosing the result with the best slack. We tested AHHK trees with $\alpha = 0, 0.5, 1$ in the experiments.
- **AHHK+detour.** If there is short path violation, the edge incident to the early critical sink is elongated to increase the delay till the early slack is close to the late slack, so that the overall slack is maximized.
- **AHHK+link.** Inserting links greedily as described in Section IV-A. This method is similar to [2].
- **Steiner network.** The dynamic programming based unbuffered Steiner network construction proposed in Section IV-C.
- **Buffered tree.** The dynamic programming based buffered Steiner network construction method proposed in Section V-C where "link insertion" is disallowed.
- **Buffered tree+link.** This is to investigate whether simply performing link insertion to buffered trees may result in good routing topologies. A Buffered tree+link is obtained by performing a greedy link insertion heuristic in a Buffered tree, i.e., in the bottom-up order, for each subtree rooted with a buffer, a link is inserted if it achieves the maximum slack gain.
- **Buffered AHHK+link.** This is to compare with optimal buffering followed by greedy link insertion. We start with AHHK tree and perform optimal timing-driven buffer insertion to it. Greedy link insertion is then performed to the resulting buffered tree. Note that Buffered tree is built by simultaneous tree construction and buffer insertion from scratch, while Buffered AHHK is obtained from buffering an AHHK tree.
- **Buffered Steiner network.** The dynamic programming based buffered Steiner network construction proposed in Section V-C.

In our experiments, to make our results and analysis more general, the timing constraints for nets range from a few dozens of pico seconds to several thousand pico seconds, and thus they can be fitted into design with the clock cycle time ranging from several hundred pico seconds to several nanoseconds.

A. Cases with Single Critical Sink

Our experiments are performed on a large set of nets and in this section, we choose the following nets to present our results and analysis. They are 15 nets with 5, 10, 15, 20, 25 sinks and sinks in 1 quadrant, 2 quadrants and 4 quadrants. Each net has a single critical sink which is often on the long path. Therefore, wire detour is rarely necessary here. In Table I, we compare AHHK and AHHK+link on 10 cases among the 15 nets where links are indeed inserted. The average results in the last row show that link insertion can improve slack by about $428ps$ with about 15% increase on wirelength. The link insertion can also achieve about 37% 2-connected wires, i.e., about 37% of the wires are tolerant to open faults.

TABLE I

Cases with 1 critical sink. Comparison on slack $S(ps)$, total wirelength $W(\mu m)$, the number of inserted links $\#L$ and percentage of 2-connected wires 2-C.

Case	AHHK			AHHK+link			
	α	S	W	S	W	$\#L$	2-C
15s, 1Q	0	-494	8751	-196	10700	1	44%
15s, 1Q	0.5	-56	9055	-4	11004	1	42%
15s, 4Q	0	-709	15333	-455	17799	1	57%
20s, 1Q	0	-704	9887	-248	11963	1	41%
20s, 2Q	0	-2137	12453	-1377	14415	1	35%
20s, 4Q	0.5	-243	17519	77	19491	1	24%
25s, 1Q	0	-746	9596	-442	11290	1	39%
25s, 2Q	1	-49	18183	-1	20411	1	33%
25s, 4Q	0	-3106	19954	-1749	21612	1	20%
Average		-916	13415	-488	15409	1	37%

The comparison between our constructive Steiner network

[1]Nontree routing can be used to improve timing, provide fault tolerance, and ameliorate variation impact. These are the important issues often encountered in practice and thus our approach is of significant practical value. Ideally, we wish to use the realistic designs for experiments. However, it is difficult to get industrial testcases in advanced technology such as $70nm$ technology. According to our knowledge, some industrial companies have already incorporated nontree routing into their physical design flows. However, due to the proprietary issue, we are not able to disclose the details here.

Procedure: simultaneous Steiner network construction and buffer insertion

Input: a source node and a set of sinks

Output: a set of buffered Steiner networks spanning on the source node and sinks

1. initialize each sink as an individual subnetwork and place this solution into the empty solution set
2. repeat
3. pick and remove an incomplete solution from the solution set
4. choose two subnetworks to merge according to the current scenario
5. perform root-root merging to generate a new solution
6. perform shortest merging to generate another new solution
7. perform link insertion to each solution
8. if the number of solutions is less than T_b or rand$(0, 1)$ is less than r
9. for each solution, generate a new solution by adding a buffer at merging point if it will not cause short circuit problem
10. place the new solutions into the solution set and perform inferiority check
11. until all solutions in the solution set are complete

Fig. 8. Procedure of simultaneous Steiner network construction and buffer insertion.

TABLE II

CASES WITH 1 CRITICAL SINK. COMPARISON BETWEEN AHHK+LINK AND STEINER NETWORK.

	AHHK+link						Steiner network				
Case	α	$S(ps)$	$W(\mu m)$	$\#L$	2-C	CPU(s)	$S(ps)$	$W(\mu m)$	$\#L$	2-C	CPU(s)
5s, 1Q	0	-3	5122	0	0	0.01	120	7544	1	58%	0.01
5s, 2Q	1	-15	7442	0	0	0.01	82	8641	1	30%	0.01
5s, 4Q	0	14	9804	0	0	0.01	123	11184	1	25%	0.02
10s, 1Q	1	26	7409	0	0	0.01	124	7743	1	15%	0.05
10s, 2Q	1	54	12831	0	0	0.01	188	14763	1	42%	0.09
10s, 4Q	0.5	112	10120	0	0	0.01	199	13468	3	53%	0.49
15s, 1Q	1	102	9566	0	0	0.01	320	8892	0	0	0.25
15s, 2Q	1	-13	12135	0	0	0.01	283	13195	1	20%	0.38
15s, 4Q	1	-12	18345	0	0	0.01	130	18836	1	22%	0.72
20s, 1Q	1	7	12372	0	0	0.02	177	12429	1	19%	2.28
20s, 2Q	0.5	-3	14214	0	0	0.02	177	17355	2	39%	4.95
20s, 4Q	0.5	77	19491	1	24%	0.02	242	18639	1	19%	0.53
25s, 1Q	1	-5	13869	0	0	0.02	534	14493	2	34%	1.41
25s, 2Q	1	-1	20411	1	33%	0.02	308	17019	1	18%	10.48
25s, 4Q	1	-18	23144	0	0	0.02	211	24128	1	17%	0.66
Average		21	13085		3.8%	0.01	215	13889		27.4%	1.49

heuristic and AHHK+link is made in Table II for the entire 15 nets. For AHHK+link, we pick the results of α with the best timing slack. Among multiple solutions generated by the constructive heuristic, we report the solution with the best slack and largest wirelength, which can improve the slack from $21ps$ to $215ps$ on average according to the last row of Table II. By this, we achieve 7%-18% improvement in the ratio of slack to the timing constraint. The wire increase of our Steiner network heuristic is only 6% over the AHHK+link results.

The dynamic programming based Steiner network construction can generate a set of solutions with different slack-wirelength tradeoff. This is confirmed by a plot in Figure 9 which is obtained from a 25-sink net.

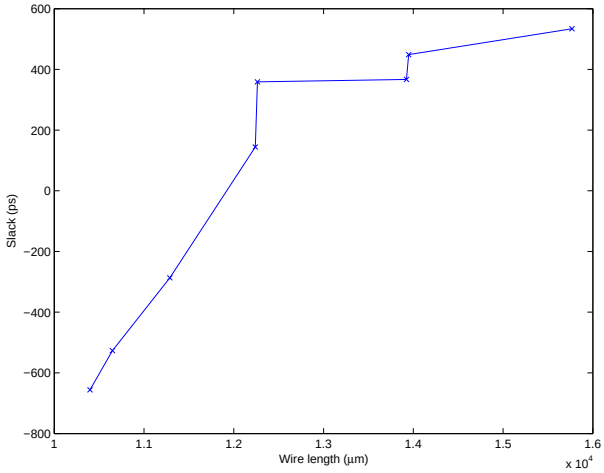

Fig. 9. Slack-wirelength tradeoff curve from case 25s, 1Q and single critical sink.

As routing blockage is an important issue in reality, we also consider to handle blockages in our Steiner network heuristic. For this, the following modifications are made. The algorithms (including Steiner network and AHHK) are the same as before except that during the process of generating candidate solutions, whenever an edge is inside a blockage, it will be rerouted. For this, we first identify the intersection of the edge and the blockage, and then we re-routed the edge along the boundary of the blockage. In the experiments, we randomly place blockages with total area summed to 20% that of the smallest bounding box of each net, which is similar to the way of blockage generation in [17]. The results are summarized in Table III. Clearly, Steiner network heuristic still performs better than AHHK+link when blockage is handled.

TABLE III

HANDLING BLOCKAGE.

Case	AHHK+link				Steiner network		
	α	$S(ps)$	W	CPU	$S(ps)$	W	CPU
15s, 1Q	0	-5	11822	0.02	209	12930	1.37
15s, 2Q	0.5	-11	15839	0.01	273	17013	0.58
15s, 4Q	0.5	7	18729	0.02	225	19283	1.29
20s, 1Q	0	25	15927	0.03	298	16525	2.20
20s, 2Q	0.5	-2	17992	0.03	332	18732	3.85
20s, 4Q	0.5	1	16705	0.04	319	17858	3.89
25s, 1Q	1	-9	18897	0.04	253	20131	7.20
25s, 2Q	0	18	22250	0.04	185	23217	8.53
25s, 4Q	1	11	20719	0.04	238	21822	12.72
Average		3.9	17653	0.03	259.1	18612	4.5

TABLE IV

MONTE CARLO RESULTS CORRESPONDING TO TABLE I ON MEAN SLACK $\mu_S(ps)$, STANDARD DEVIATION OF SLACK $\sigma_S(ps)$ AND TIMING YIELD Y.

Case	AHHK				AHHK+link		
	α	μ_S	σ_S	Y	μ_S	σ_S	Y
15s, 1Q	0	-497	34	0	-197	27	0
15s, 1Q	0.5	-57	27	2%	-4	27	44%
15s, 4Q	0	-711	50	0	-456	45	0
20s, 1Q	0	-707	35	0	-249	29	0
20s, 2Q	0	-2144	69	0	-1382	58	0
20s, 4Q	0.5	-245	42	0	77	42	97%
25s, 1Q	0	-751	40	0	-447	41	0
25s, 2Q	1	-51	43	12%	-1	44	49%
25s, 4Q	0	-3117	91	0	-1754	69	0
Average		-920	47.9	1.6%	-490	42.4	21.1%

Monte Carlo simulations (5000 runs for each result) are performed to observe the behaviors of these algorithms under process variations. We consider wire width, sink capacitance and driver resistance variations which are assumed to follow Gaussian distribution with standard deviation equal to 5% of nominal value. The comparison between AHHK trees and AHHK+link results is in Table IV. The mean values μ_S of the slacks are about the same as the deterministic results in Table I. On average, AHHK+link can reduce the standard deviation σ_S of slack by about 10% and increase timing yield from 1.6% to 21.2%. The **timing yield** refers to the probability of non-negative slack. The data in Table V indicate that our constructive method can reduce the standard deviation further by 10% and improve the timing yield from 61% to 100% compared to AHHK+link.

In order to better capture the impact of process variation on circuits, the modeling proposed in [18] is also used to perform Monte Carlo simulations. It is a first order approximation and is able to capture the major component of variations [18]. As an example, we consider two sources of variations for each gate, which are gate length L and threshold voltage V. Note that other sources of variations can be easily incorporated into the model. For each gate, the input capacitance is modeled as

$$C_g = C_{g0} + \frac{\partial C}{\partial L}\Delta L + \frac{\partial C}{\partial V}\Delta V + \delta_g + \epsilon, \qquad (16)$$

where C_{g0} is the nominal value, $\frac{\partial C}{\partial L}$ and $\frac{\partial C}{\partial V}$ refer to the sensitives of gate capacitance to gate length and threshold voltage, respectively, random variables ΔL and ΔV represent variations in gate length and threshold voltage, respectively, and random variables δ_g and ϵ refer to the local and global variations. We similarly model the gate driving resistance R_g.

TABLE V

MONTE CARLO RESULTS CORRESPONDING TO TABLE II.

Case	AHHK+link				Steiner network		
	α	μ_S	σ_S	Y	μ_S	σ_S	Y
5s, 1Q	0	-4	20	43%	119	17	100%
5s, 2Q	1	-15	23	25%	82	19	100%
5s, 4Q	0	14	27	70%	123	26	100%
10s, 1Q	1	24	23	86%	125	22	100%
10s, 2Q	1	52	41	89%	189	34	100%
10s, 4Q	0.5	111	26	100%	199	29	100%
15s, 1Q	1	102	26	100%	318	22	100%
15s, 2Q	1	-14	36	35%	283	28	100%
15s, 4Q	1	-12	44	39%	130	40	100%
20s, 1Q	1	7	29	60%	176	27	100%
20s, 2Q	0.5	-5	37	44%	176	39	100%
20s, 4Q	0.5	77	42	97%	241	39	100%
25s, 1Q	1	-8	38	42%	534	32	100%
25s, 2Q	1	-1	44	49%	308	36	100%
25s, 4Q	1	-20	54	35%	210	50	100%
Average		21	34.0	60.9%	214	30.7	100%

As in [19], to handle spatial correlation, grids are layered upon the circuit and each gate belongs to a grid which is indexed by (a, b). Incorporating spatial correlation into consideration, we have

$$C_g = C_{g0} + \frac{\partial C}{\partial L}\sum_{(a,b)} \alpha_{(a,b)}\Delta L_{(a,b)} + \frac{\partial C}{\partial V}\sum_{(a,b)} \beta_{(a,b)}\Delta V_{(a,b)} + \delta_g + \epsilon, \qquad (17)$$

where α, β are the parameters. $\Delta L_{(a,b)}$ (and $\Delta V_{(a,b)}$) is independent of each other which can be obtained through performing Principal Component Analysis (PCA) to the originally correlated random variations. Refer to [19] for the details. With the above new model, Monte Carol simulation results are summarized in Table VI. These results confirm that our Steiner network method significantly improves the timing yield compared to AHHK+link.

The proposed Steiner network heuristic can be easily incorporated into the existing global routers. As an example, we incorporate it with Labyrinth [20]. The algorithm is as follows. The global routing is first computed and timing analysis is performed on the resulting circuit. Signal nets along the timing critical path are then identified and re-routed using the proposed Steiner network heuristic. We apply the above algorithm to IBM-PLACE benchmark circuits which are the benchmarks converted from ISPD'98 circuits [21], and the results are summarized in Table VII. Since the original circuits (e.g., gate type) for IBM-PLACE benchmark are not known to us, circuit information (e.g., gate resistance and capacitance) is randomly generated. From Table VII, one sees that Steiner network heuristic can still improve the timing yield. In addition, since it is applied only to the nets along the timing critical path, the additional wirelength is quite small and the additional runtime is not large.

B. Other Unbuffered Cases

We tested the algorithms in cases with multiple critical sinks. That is, there may be several sinks with similar timing criticality in each net. In order to see the effect on fixing short path delay constraint violations, these testcases usually have tighter constraints on short path than on long path. The

TABLE VII

INCORPORATING STEINER NETWORK HEURISTIC INTO GLOBAL ROUTING.

Circuit			Labyrinth				Labyrinth+Steiner network				
name	nets	wires	W	μ_S	σ_S	Y	W	μ_S	σ_S	Y	Additional CPU (s)
ibm01	11507	28232	76517	-22	35	25%	76972	5	30	57%	289
ibm02	18429	55649	204734	-30	51	27%	205998	2	48	52%	307
ibm03	21621	45727	185116	-38	33	12%	187033	9	33	61%	358
ibm04	26163	52487	196920	-17	15	13%	198129	3	11	60%	559
ibm05	27777	94304	420583	-18	27	23%	421125	20	25	79%	685
ibm06	33354	82541	346137	-25	27	18%	348721	-3	22	43%	617
ibm07	44394	109365	449213	-52	40	10%	452712	-9	35	40%	795
ibm08	47944	133353	469666	-9	18	31%	472107	11	21	71%	892
ibm09	50393	128708	481176	-15	12	11%	483990	2	10	58%	918
ibm10	64227	182010	679606	-13	19	25%	681181	5	17	61%	1082
Average	34581	91238	350967	-23.9	27.7	19.5%	352797	4.5	25.2	58.2	650.2

TABLE VI

MONTE CARLO RESULTS CORRESPONDING TO TABLE II USING THE MODEL IN [18].

Case	AHHK+link				Steiner network		
	α	μ_S	σ_S	Y	μ_S	σ_S	Y
5s, 1Q	0	-5	15	38%	119	12	100%
5s, 2Q	1	-15	21	25%	80	17	100%
5s, 4Q	0	13	28	67%	123	22	100%
10s, 1Q	1	25	22	87%	122	20	100%
10s, 2Q	1	49	40	89%	185	28	100%
10s, 4Q	0.5	109	24	100%	202	25	100%
15s, 1Q	1	102	18	100%	315	18	100%
15s, 2Q	1	-13	40	38%	280	23	100%
15s, 4Q	1	-10	38	39%	131	39	100%
20s, 1Q	1	9	25	65%	176	25	100%
20s, 2Q	0.5	-7	33	41%	175	38	100%
20s, 4Q	0.5	78	39	98%	240	31	100%
25s, 1Q	1	-8	32	40%	532	29	100%
25s, 2Q	1	-2	41	49%	307	32	100%
25s, 4Q	1	-22	48	32%	211	38	100%
Average		20.2	30.9	60.5%	213.2	26.5	100%

wire detour method can increase the delay to the early critical sink but at the cost of increasing long path delay. This is in contrast to our approach which can increase short path delay and reduce long path delay simultaneously. Moreover, wire detour cannot lead to any tolerance to open faults as in non-tree. The results in Table VIII show that our Steiner network heuristic can improve the slack by about $80ps$ on average when compared to performing wire detour on existing trees. The wirelength increase due to our method is about 4% with respect to the wire detour results.

We also run experiments on cases without delay lower bound which are the same as the conventional timing driven routing. Refer to Table IX for the results. We can see our Steiner network results can improve the slack by $125ps$ with only 5% increase on wirelength.

In Figure 10, we give some plots for AHHK tree, AHHK+link and our Steiner network for comparison, where non-rectilinear edges are "soft edges" [22] which can be realized to rectilinear edges when necessary. These plots are for the "15s, 1Q" case with multiple critical sinks. In this case, the slack and the wirelength of AHHK are $190ps$ and $9206um$, respectively, of AHHK+link are $375ps$ and $10285um$, respectively, and of Steiner network are $497ps$ and $10281um$, respectively. Clearly, compared to AHHK+link, Steiner network improves slack by $> 100ps$ while still saving wires.

Note that non-tree routing could introduce more congestion which sometimes makes the circuit more difficult to route, route with more detour or cause signal integrity issues. These could get worse if the routing is performed on small region. These are the drawbacks of our approach.

C. Handling Buffers

For randomly generate test cases, the results for Buffered Steiner network construction are summarized in Table XI. We compare the results with Buffered tree and Buffered tree+link. For fair comparisons, we choose the sum of wire capacitance and buffer input capacitance as a metric to evaluate different topologies. For each case in Table XI, two solutions at driver are reported. One is the best slack ("B.S.") result which is the one with the maximum slack and the other is the best capacitance ("B.C.") result which is the one with positive slack and minimum capacitance. Note that "B.C." results for Buffered tree+link are not shown since they are the same as the "B.C." results for Buffered tree.

From Table XI, one can see that the Buffered Steiner network significantly outperforms the Buffered tree topology in terms of slack (on average $510ps$ v.s. $188ps$ for B.S. and $111ps$ v.s. $49ps$ for B.C.) and it only introduces slightly more capacitance (on average 2.72 v.s. 2.61 for B.C.). When compared to Buffered tree+link, Buffered Steiner network on average obtains $> 100ps$ slack improvement with only $3.47/3.30 - 1 = 5\%$ more capacitance. These demonstrate the advantages of buffered Steiner network routing topology.

Monte Carlo simulations (5000 runs for each result) are performed to observe the behaviors of algorithms under process variations. Wire width, sink/buffer capacitance and driver/buffer resistance variations are assumed to follow Gaussian distribution with standard deviation equal to 5% of nominal value. The comparison results are summarized in Table XII. The mean values μ_S of the slacks are about the same as the deterministic results in Table XI. On average, Buffered tree+link can reduce the standard deviation σ_S of slack by about 6% and increase timing yield from 91% to 100% for B.S. case. Our constructive method can reduce the standard deviation by about 20% and improve the timing yield from 77% to 95% for B.C. case compared to Buffered tree.

We also compare the proposed Buffered Steiner network with Buffered AHHK+link. Recall that Buffered AHHK+link

TABLE VIII

CASES WITH MULTIPLE CRITICAL SINKS. COMPARISON AMONG AHHK, AHHK+DETOUR, AHHK+LINK AND STEINER NETWORK SLACK S, TOTAL WIRELENGTH W AND THE NUMBER OF INSERTED LINKS $\#L$.

Case	AHHK			AHHK+detour		AHHK+link				Steiner network			
	α	$S(ps)$	$W(\mu m)$	$S(ps)$	$W(\mu m)$	$S(ps)$	$W(\mu m)$	$\#L$	CPU(s)	$S(ps)$	$W(\mu m)$	$\#L$	CPU(s)
5s, 2Q	0.5	-97	6952	285	10614	88	8996	1	0.01	357	10589	1	0.02
5s, 4Q	1	-181	8670	41	10527	142	11110	1	0.01	202	10692	2	0.03
10s, 2Q	1	-61	13290	188	15153	165	15593	1	0.01	339	17997	2	0.06
10s, 4Q	1	-83	14630	308	17839	66	17195	1	0.01	405	16531	3	0.24
15s, 2Q	1	-37	11919	204	14947	-37	11919	0	0.01	258	13970	1	1.17
15s, 4Q	1	-32	17217	129	18415	-32	17217	0	0.01	205	18554	1	1.32
20s, 2Q	0.5	-21	15596	363	17827	-21	15596	0	0.02	499	18646	3	2.63
20s, 4Q	1	-26	18336	352	20317	143	19805	1	0.02	368	19790	0	5.58
25s, 2Q	0.5	-161	15539	203	17506	-46	16755	1	0.03	219	22705	4	39.4
25s, 4Q	1	-40	22283	106	23621	-40	22283	0	0.02	120	23440	2	94.5
Average		-74	14443	218	16677	43	15647		0.02	297	17291		14.5

TABLE IX

TESTCASES WITHOUT LOWER DELAY BOUND. COMPARISON BETWEEN AHHK+LINK AND STEINER NETWORK.

Case	AHHK+link						Steiner network				
	α	$S(ps)$	$W(\mu m)$	$\#L$	2-C	CPU(s)	$S(ps)$	$W(\mu m)$	$\#L$	2-C	CPU(s)
5s, 1Q	1	-7	7171	0	0	0.01	100	8080	1	37%	0.01
5s, 4Q	1	-5	8913	0	0	0.01	38	10995	1	48%	0.02
10s, 1Q	1	-4	8858	0	0	0.01	67	8889	1	24%	0.01
10s, 4Q	1	-5	15119	0	0	0.01	64	16833	1	23%	0.02
15s, 1Q	1	-3	11290	1	57%	0.01	39	10578	1	34%	0.16
15s, 4Q	0.5	-2	15607	0	0	0.01	66	16731	1	15%	1.19
20s, 1Q	0.5	-311	10119	0	0	0.02	25	13261	2	43%	0.13
20s, 4Q	1	-107	22923	0	0	0.02	36	22492	2	16%	0.53
25s, 1Q	0.5	-102	10105	0	0	0.03	72	11316	2	58%	1.20
25s, 4Q	1	-119	25633	0	0	0.03	74	23154	2	19%	0.59
Average		-67	13574		6%	0.02	58	14233		32%	0.39

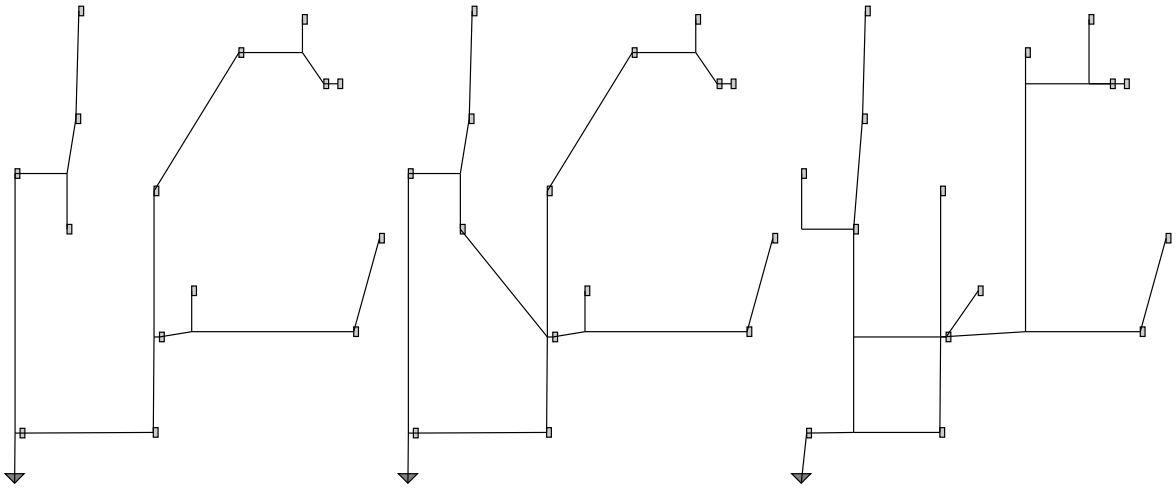

Fig. 10. (a) AHHK ($\alpha = 0.5$), (b) AHHK+link, and (c) Steiner network for the "15s, 1Q" case with multiple critical sinks. Note that non-rectilinear edges are "soft edges" which can be realized to rectilinear edges in many different ways.

is obtained by performing optimal (in terms of slack maximization) buffer insertion to an AHHK tree followed by link insertion. Since the existing buffering algorithm only handles long path constraint, comparison is made on the testcases without lower delay bound. For Buffered Steiner network, the best slack results are chosen for comparison. Refer to Table X for the comparison. From it, one sees that as our Buffered Steiner network builds solutions from scratch (with aggressive solution pruning approach), in general, it outperforms Buffered AHHK+link in terms of slack and variance.

VII. CONCLUSION

This work investigates timing driven routing by using non-tree topology. We propose Steiner network construction heuristics which can generate either tree or non-tree with different slack-wirelength tradeoff, and handle both long path and short path constraints. We also propose heursitcs for simultaneous Steiner network construction and buffering. Furthermore, incremental non-tree delay update techniques are developed to facilitate fast Steiner network evaluations. Experimental results show that our approach is very promising in improve timing

TABLE XI

HANDLING BUFFERS FOR MULTIPLE CRITICAL SINK CASE. CAP REFERS TO THE SUM OF WIRE CAPACITANCE AND BUFFER INPUT CAPACITANCE. B.S. REFERS TO THE RESULTS WITH MAXIMUM SLACK AND B.C. REFERS TO THE RESULTS WITH POSITIVE SLACK AND MINIMUM CAPACITANCE.

Case	Buffered tree			Buffered tree+link			Buffered Steiner network		
	$S(ps)$	$Cap(pF)$	CPU(s)	$S(ps)$	$Cap(pF)$	CPU(s)	$S(ps)$	$Cap(pF)$	CPU(s)
5s, 2Q (B.S.)	12	1.63	0.05	273	2.70	0.07	349	2.46	0.09
5s, 2Q (B.C.)	12	1.63	0.05				103	1.81	0.09
5s, 4Q (B.S.)	65	2.48	0.03	321	2.78	0.05	338	3.50	0.06
5s, 4Q (B.C.)	65	2.48	0.03				65	2.48	0.06
10s, 2Q (B.S.)	51	1.92	0.06	449	2.23	0.08	576	2.93	0.39
10s, 2Q (B.C.)	5	1.76	0.06				77	2.20	0.39
10s, 4Q (B.S.)	38	3.07	0.04	388	3.43	0.07	564	3.88	0.12
10s, 4Q (B.C.)	38	3.07	0.04				153	2.64	0.12
15s, 2Q (B.S.)	228	2.81	0.15	369	3.46	0.32	555	3.43	1.59
15s, 2Q (B.C.)	16	2.64	0.15				292	2.47	1.59
15s, 4Q (B.S.)	98	2.87	0.23	478	3.23	0.39	522	3.35	1.70
15s, 4Q (B.C.)	11	2.77	0.23				19	2.68	1.70
20s, 2Q (B.S.)	335	3.02	2.32	453	3.10	2.77	569	3.52	20.2
20s, 2Q (B.C.)	15	2.41	2.32				48	2.98	20.2
20s, 4Q (B.S.)	353	3.65	1.93	372	4.34	2.62	376	3.68	17.8
20s, 4Q (B.C.)	53	3.32	1.93				38	3.15	17.8
25s, 2Q (B.S.)	391	3.35	15.8	416	3.77	28.3	639	3.89	210.5
25s, 2Q (B.C.)	198	2.78	15.8				221	3.17	210.5
25s, 4Q (B.S.)	308	3.92	27.2	503	4.02	53.1	612	4.10	325.1
25s, 4Q (B.C.)	73	3.25	27.2				95	3.58	325.1
Average (B.S.)	188	2.87		402	3.30		510	3.47	
Average (B.C.)	49	2.61					111	2.72	

TABLE XII

MONTE CARLO RESULTS CORRESPONDING TO TABLE XI.

Case	Buffered tree			Buffered tree+link			Buffered Steiner network		
	μ_S	σ_S	Y	μ_S	σ_S	Y	μ_S	σ_S	Y
5s, 2Q (B.S.)	12	38	62%	275	33	100%	348	29	100%
5s, 2Q (B.C.)	12	38	62%				103	22	100%
5s, 4Q (B.S.)	65	57	88%	320	55	100%	339	47	100%
5s, 4Q (B.C.)	65	57	88%				67	41	95%
10s, 2Q (B.S.)	52	45	88%	451	50	100%	577	41	100%
10s, 2Q (B.C.)	5	31	56%				79	28	99%
10s, 4Q (B.S.)	39	55	77%	388	56	100%	564	47	100%
10s, 4Q (B.C.)	39	55	77%				153	33	100%
15s, 2Q (B.S.)	227	35	100%	370	41	100%	555	35	100%
15s, 2Q (B.C.)	16	24	75%				293	31	100%
15s, 4Q (B.S.)	98	36	99%	477	32	100%	522	28	100%
15s, 4Q (B.C.)	11	25	67%				18	27	73%
20s, 2Q (B.S.)	333	73	100%	453	45	100%	568	48	100%
20s, 2Q (B.C.)	15	23	73%				48	21	98%
20s, 4Q (B.S.)	353	46	100%	371	41	100%	376	40	100%
20s, 4Q (B.C.)	53	77	76%				38	36	86%
25s, 2Q (B.S.)	390	83	100%	418	80	100%	639	67	100%
25s, 2Q (B.C.)	198	58	99%				221	39	100%
25s, 4Q (B.S.)	307	39	100%	503	47	100%	611	32	100%
25s, 4Q (B.C.)	73	27	99%				95	22	100%
Average (B.S.)	188	51	91%	403	48	100%	510	41	100%
Average (B.C.)	49	41	77%				112	30	95%

slack and handling both long path and short path constraints.

We note the following recommendations for designs adopting nontree routing. First, nontree routing is only applied to global nets. Second, after performing statistical timing analysis, we identify the timing critical nets, and then nontree routing is applied to those timing critical nets. Third, we may also perform statistical sensitivity analysis as proposed in [23], and apply nontree routing to those nets with high sensitivity to variations. In future, we will design non-tree routing method using more accurate delay model.

VIII. ACKNOWLEDGEMENT

The authors are grateful to the anonymous reviewers for their insightful and helpful comments for improving the earlier drafts of the paper.

REFERENCES

[1] A. B. Kahng and G. Robins, "On optimal interconnections for vlsi," *Kluwer Academic Publishers*, 1995.

[2] B. McCoy and G. Robins, "Non-tree routing," *Proceedings of Design, Automation and Test in Europe Conference*, pp. 430–434, 1994.

[3] T. Xue and E. S. Kuh, "Post routing performance optimization via tapered link insertion and wiresizing," *Proceedings of Design, Automation and Test in Europe Conference*, pp. 74–79, 1995.

TABLE X

COMPARISON BETWEEN BUFFERED AHHK+LINK AND BUFFERED STEINER NETWORK FOR THE TESTCASES WITHOUT LOWER DELAY BOUND.

Case	Buffered AHHK+link				Buffered Steiner network			
	μ_S	σ_S	Cap	Y	μ_S	σ_S	Cap	Y
5s, 2Q	201	39	2.02	100%	223	35	2.10	100%
5s, 4Q	158	25	2.51	100%	212	27	2.69	100%
10s, 2Q	328	40	2.27	100%	352	25	2.25	100%
10s, 4Q	373	58	2.93	100%	390	49	3.20	100%
15s, 2Q	357	45	3.12	100%	335	51	3.06	100%
15s, 4Q	419	52	3.69	100%	531	37	3.72	100%
20s, 2Q	482	29	3.05	100%	557	28	3.23	100%
20s, 4Q	450	41	3.67	100%	476	38	3.97	100%
25s, 2Q	529	72	3.80	100%	589	77	3.75	100%
25s, 4Q	538	57	3.72	100%	550	52	3.95	100%
Average	383.5	45.8	3.08	100%	421.5	41.9	3.19	100%

[4] A. B. Kahng, B. Liu, and I. I. Mandoiu, "Non-tree routing for reliability and yield improvement," *ICCAD*, pp. 260–266, 2002.

[5] J. Lillis, C. K. Cheng, T. T. Lin, and C. Y. Ho, "New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing," *DAC*, pp. 395–400, 1996.

[6] W. Chuang, S. S. Sapatnekar, and I. N. Hajj, "Delay and area optimization for discrete gate sizes under double-sided timing constraints," *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 9.4.1–9.4.4, 1993.

[7] P. K. Chan and K. Karplus, "Computing signal delay in general RC networks by tree/link partitioning," *TCAD*, vol. 9, no. 8, pp. 898–902, 1990.

[8] D. Lam, C.-K. Koh, Y. Chen, J. Jain, and V. Balakrishnan, "Statistical based link insertion for robust clock network design," *ICCAD*, pp. 588–591, 2005.

[9] G. H. Golub and C. F. Van Loan, "Matrix computations," *John Hopkins University Press*, 1996.

[10] R. Chaturvedi and J. Hu, "An efficient merging scheme for prescribed skew clock routing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 13, no. 6, pp. 750–754, 2005.

[11] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, pp. 277–288, 1992.

[12] J. Cong and X. Yuan, "Routing tree construction under fixed buffer locations," *DAC*, pp. 379–384, 2000.

[13] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness, and C. J. Alpert, "Practical techniques to reduce skew and its variations in buffered clock networks," *ICCAD*, pp. 592–596, 2005.

[14] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *TCAD*, vol. 2, no. 3, pp. 202–211, 1983.

[15] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *TCAD*, vol. 9, no. 4, pp. 352–366, 1990.

[16] A. B. Kahng and B. Liu., "Q-Tree: a new iterative improvement approach for buffered interconnect optimization," *IEEE Computer Society Annual Symposium on VLSI*, pp. 183–188, 2003.

[17] J. Hu, C. Alpert, S. Quay, and G. Gandham, "Buffer insertion with adaptive blockage avoidance," *TCAD*, vol. 22, no. 4, pp. 492–498, 2003.

[18] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," *DAC*, pp. 331–336, 2004.

[19] H. Chang and S. Sapatnekar, "Statistical timing analysis under spatial correlations," *TCAD*, vol. 24, no. 9, pp. 1467–1482, 2005.

[20] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh, "Pattern routing: Use and theory for increasing predictability and avoiding coupling," *TCAD*, pp. 777–790, 2002.

[21] http://www.ece.ucsb.edu/kastner/labyrinth/benchmarks/.

[22] J. Hu and S. S. Sapatnekar, "Algorithms for non-hanan-based optimization for vlsi interconnect under a higher order awe model," *TCAD*, vol. 19, no. 4, pp. 446–458, 2000.

[23] X. Li, J. Le, M. Celik, and L. Pileggi, "Defining statistical sensitivity for timing optimization of logic circuits with large-scale process and environmental variations," *ICCAD*, pp. 844–851, 2005.