Steiner Network Construction for Timing Critical Nets [*]

Shiyan Hu, Qiuyang Li, Jiang Hu and Peng Li
Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843
{hushiyan, qiuyang, jhu, pli}@ece.tamu.edu

ABSTRACT

Conventionally, signal net routing is almost always implemented as Steiner trees. However, non-tree topology is often superior on 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. 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 tree approaches.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: INTEGRATED CIRCUITS—*layout*

General Terms

Algorithms, Performance, Reliability

Keywords

Steiner network, routing, interconnect, redundancy

1. 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] for signal net routing because it is cost-effective and its delay is relatively easy to compute. However, non-tree topology has some remarkable advantages compared to trees. Non-tree topology can often obtain significantly better timing performance than trees [3, 4]. In addition, the redundant paths in a non-tree network provide certain tolerance to open fault and therefore can improve manufacturing yield and reliability [2]. Moreover, non-tree topology sometimes can reduce delay variations [2]. 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.

Perhaps the first non-tree routing work is [3] which greedily adds extra wires on given trees to minimize source-sink delay. The later work of [4] inserts a link between the source and the sink having the maximum delay. The recent work of [2] 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 [3, 4] 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 [3, 4] 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 later 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

[*]This work is partially supported by SRC under contract 2003-TJ-1124.

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. One heuristic is a greedy link insertion in an existing tree, which is similar as [3] but the solution search is trimmed for the double-sided timing constraints. The other 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. Incremental non-tree delay update techniques are developed for improving the efficiency of our algorithms. Extensive experimental results show that our Steiner network construction 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 [3]. 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 method can improve timing yield greatly because of both nominal slack improvement and delay variability (standard deviation) reduction.

2. 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. Consider inserting a link between two nodes i and j in an RC network, which can be either a tree or a non-tree. 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.

The link capacitance always increases the delay by $t_{i,lc} = \frac{C}{2}(R_{i,i} + R_{i,j})$ and $t_{j,lc} = \frac{C}{2}(R_{i,j} + R_{j,j})$. In general, $R_{i,j}$ 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 [4]. 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 delay at i and j, respectively, when node capacitance $C_i = 1$, $C_j = -1$ and the other node capacitances are set to zero [4].

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 $t_i \geq (\frac{1}{\alpha} - 1)t_{i,lc} + t_j + t_{j,lc}$.*

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.*

3. 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 1(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

$$Gx = Bu, \quad (3)$$

where

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

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 unknows 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.

3.1 Reroot

Reroot means the root (voltage input) of a subnetwork is changed from one node to another. We illustrate this

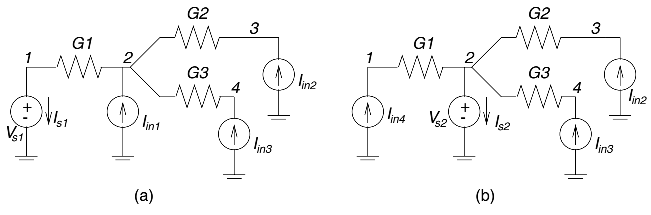

Figure 1: A linear DC circuit. The root node is changed from 1 to 2 in (b).

operation by moving the voltage source from node 1 (in Figure 1(a)) to node 2 (in Figure 1(b)). In Figure 1(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 [11], 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.

3.2 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).

3.3 Merging Two Networks

Now consider the delay update after merging two subnetworks. As shown in Figure 2, 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 merging point for the first network varies, a reroot procedure (Section 3.1) 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.

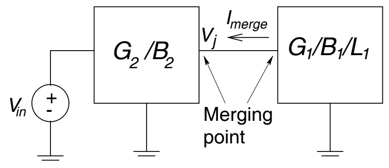

Figure 2: Merging two subnetworks.

4. ALGORITHMS

4.1 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 2). Next we find the shortest path $p_{e,l} \in G$ which connects the two critical sinks. 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.

4.2 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 3(a)), which has small area and large radius, and is usually derived from minimum spanning tree algorithms; (2) star-like topology (Figure 3(b)) with relatively large area and small radius, and can be obtained from the shortest path tree or Rectilinear Steiner Aborescence (RSA) algorithms [1].

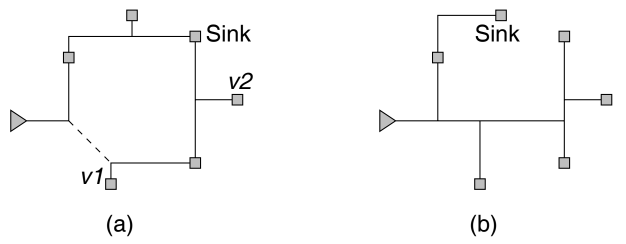

Figure 3: (a) Chain-like topology and (b) star-like topology. Link is indicated by dashed lines.

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 3(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 3(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 3(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.

4.3 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 and can be pruned.

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 [9]. 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 [9]. 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) [10]. 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 [10] 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 4(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 4(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.

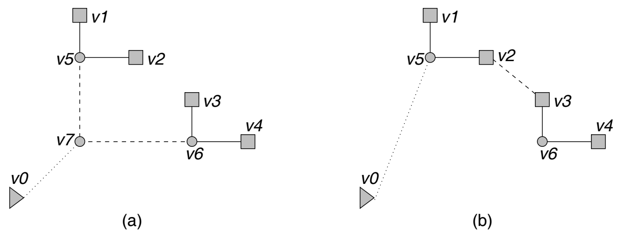

Figure 4: (a) Root-root merging and (b) the shortest merging. Reroot occurs in (b).

The root-root merging is very similar as the RSA [10] 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.

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 4.1 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.

5. 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 ranges from 5 to 25. Without loss of generality, we let the source be at coordinates 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 [12] are employed in the experiments. 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 4.1. This method is similar as [3].
- **Steiner network.** The dynamic programming based Steiner network construction proposed in Section 4.3.

5.1 Cases with Single Critical Sink

We generated 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 1, 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.

The comparison between our constructive Steiner network heuristic and AHHK+link is made in Table 2 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 2. The wire increase of our Steiner network heuristic is only 6% over the AHHK+link results.

Table 1: 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.

	AHHK			AHHK+link			
Case	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%

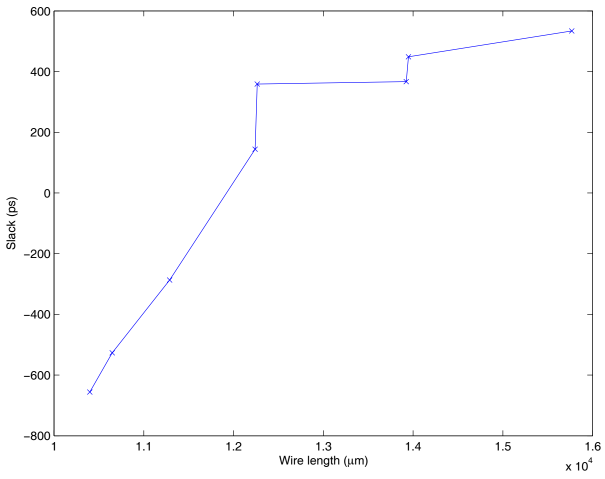

Figure 5: Slack-wirelength tradeoff curve from case 25s, 1Q and single critical sink.

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 5 which is obtained from a 25-sink net.

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 3. The mean values μ_S of the slacks are about the same as the deterministic results in Table 1. 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 4 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.

5.2 Cases with Multiple Critical Sinks

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

Table 2: 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

Table 3: Monte Carlo results corresponding to Table 1 on mean slack $\mu_S(ps)$, standard deviation of slack $\sigma_S(ps)$ and timing yield Y.

		AHHK			AHHK+link		
Case	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%

fixing short path delay constraint violations, these testcases usually have tighter constraints on short path than on long path. The 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 average results in Table 5 show that our Steiner network heuristic can improve the slack by about 80*ps* 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.

6. CONCLUSION AND FUTURE WORK

This work investigates timing driven routing by using non-tree topology. Experimental results show that this is a very promising approach even when both long path and short path constraints are considered. In future, we will find non-tree routing method using more accurate delay model and study buffered non-tree routings.

7. REFERENCES

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

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

[3] B. A. McCoy and G. Robins. Non-tree routing. *DATE*, pages 430–434, 1994.

[4] T. Xue and E. S. Kuh. Post routing performance optimization via tapered link insertion and wiresizing. *DATE*, pages 74–79, 1995.

Table 4: Monte Carlo results corresponding to Table 2.

		AHHK+link			Steiner network		
Case	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%

Table 5: Cases with multiple critical sinks. Comparison on average results (10 nets with 5-25 sinks) of slack $S(ps)$ and wirelength $W(\mu m)$.

AHHK		AHHK+detour		AHHK+link		Steiner network	
S	W	S	W	S	W	S	W
-74	14443	218	16677	43	15647	297	17291

[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*, pages 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. *CICC*, pages 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*, 9(8):898–902, August 1990.

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

[9] R. Chaturvedi and J. Hu. An efficient merging scheme for prescribed skew clock routing. *TVLSI*, 13(6):750–754, June 2005.

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

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

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