Interconnect Layout Optimization Under Higher-Order RLC Model*

Jason Cong and Cheng-Kok Koh
UCLA Computer Science Dept., Los Angeles, CA 90095-1596

Abstract

In this paper, we study the interconnect layout optimization problem under a higher-order RLC model to optimize not just *delay*, but also *waveform* for RLC circuits with non-monotone signal response. We propose a unified approach that considers topology optimization, wiresizing optimization, and waveform optimization simultaneously. Our algorithm considers a large class of routing topologies, ranging from shortest-path Steiner trees to bounded-radius Steiner trees and Steiner routings. We construct a set of *required-arrival-time Steiner trees* or RATS-trees, providing a smooth trade-off among signal delay, waveform, and routing area. Using a new *incremental* moment computation algorithm, we interleave topology construction with moment computation to facilitate accurate delay calculation and evaluation of waveform quality. Experimental results show that our algorithm is able to construct a set of topologies providing a smooth trade-off among signal delay, signal settling time, voltage overshoot, and routing cost.

*This work is partially supported by DARPA/ETO under Contract DAAL01-96-K-3600 and a grant from Intel Corporation under the California MICRO Program.

1 Introduction

As VLSI circuitry reaches deep submicron device dimension, operates at giga-Hertz clock frequencies, and is packaged in highly integrated multichip modules, the performance of interconnect structures has become the dominating factor in determining system performance. Recent studies show that interconnect delay can be reduced by interconnect topology optimization, wiresizing optimization, and/or buffer optimization. A comprehensive survey of these optimization techniques can be found in [1]. In this paper, we study the interconnect layout optimization problem under a higher-order RLC model. Our objective is to perform topology and wiresizing optimization for performance and signal integrity under a higher-order moment-based interconnect model. It has the following advantages:

(1) Most of the previous works on interconnect optimization were achieved under RC interconnect models only, including recent studies on topology optimization [2, 3], wiresizing optimization [4, 5, 6, 7], as well as the most recent works that combined topology construction with buffer insertion and/or wiresizing [8, 9, 10]. They did not consider the inductance effect, which may be an important factor in high-speed MCM/PCB designs and high frequency deep sub-micron designs. In this study, we overcome this shortcoming by considering a higher-order moment-based RLC interconnect model in the optimization process. We also present a new algorithm to *incrementally* compute moments of sinks in an RLC tree in a bottom-up manner.

(2) A few approaches have proposed to use higher-order RLC models for topology optimization [11, 12], wiresizing optimization [13], and termination optimization [14, 15, 16]. But none of them can consider all three optimization simultaneously for both delay and signal integrity optimization. Our optimization algorithm takes a more global approach; it considers wiresizing during routing topology construction, optimizes for both signal delays and response waveforms, and returns a set of routing topologies with appropriate wire width assignments.

(3) Our method is capable of constructing a large class of routing topologies, ranging from shortest-path Steiner trees to bounded-radius Steiner trees and Steiner routings. All of the previous topology construction algorithms are limited to a single class of routing topology. For example, the A-tree algorithm considered shortest-path Steiner trees only [2], and the set of topologies generated by the P-tree algorithm [9] is restricted to a permutation-induced abstract topology pre-determined by a Traveling Salesman heuristic. By considering a large class of routing topologies, our algorithm is able to produce a set of topologies with a smooth trade-off among signal delay, waveform, and routing area.

In order to optimize signal delay, waveform, and wiring area, we construct a set of *required-arrival-time Steiner trees* (RATS-trees) by recursively merging subtrees in a bottom-up fashion. Using this bottom-up tree construction, we can easily compute sink moments in an incremental manner. For a given set of sinks, we maintain a set of topologies which correspond to a trade-off among routing costs, time domain waveforms, and signal delays. Moreover, we also consider interconnect sizing (and possibly, buffer insertion) in the subtree merging process. At the end of the bottom-up process, the designer would have a set of solutions to choose from. In this paper, we also introduce a space-efficient *reduced tree representation* to capture all wiresized topologies constructed during the execution of our algorithm.

The rest of the paper is organized as follows: Section 2 formulates the interconnect layout optimization problem that we aim to solve. In Section 3, we present the RATS-tree algorithm in detail, including the reduced tree representation and the incremental bottom-up moment computation algorithm. We present the experimental results in Section 4 and conclude the paper in Section 5.

2 Problem Formulation

Given a net of pins or terminals $\{s_0, s_1, s_2, \cdots, s_n\}$ to be electrically connected, we assume that s_0 denotes the source (or driver) of the net and the rest of the pins are sinks (or receivers). In this paper, we are interested in interconnect trees We use $P_T(u,v)$ to denote the unique path from u to v in an interconnect tree T, $d_T(u,v)$ the pathlength of $P_T(u,v)$, and $d(u,v)$ the Manhattan distance between u and v. We denote the signal delay from u to v by $t_T(u,v)$. The source node s_0 will generally be referred to as the root of an interconnect tree, and each node v in the tree is connected to its parent by edge e_v. We use T_v to denote the subtree that is rooted at v. Given an edge e, we use T_e to denote the subtree with e being its root edge. $l(e)$ denotes the length of e.

Let q_i be the required arrival time of sink s_i. A minimal *required-arrival-time Steiner tree* (RATS-tree) is a minimum-cost Steiner tree such that $q_i \geq t_T(s_0, s_i)$ for all sinks. RATS-trees include several commonly used topologies when we use the pathlength delay formulation with $t_T(u,v) = d_T(u,v)$. For example, by setting $q_i = d(s_0, s_i)$ for all sinks, an optimal RATS-tree is an optimal Steiner arborescence [17]. If we relax the requirement such that all sinks have the same required arrival time, then an optimal RATS-tree is an optimal bounded-radius Steiner tree [18]. Lastly, an optimal RATS-tree with unbounded q_i's is an optimal Steiner tree [19].

The pathlength formulation captures the delay for unloaded lossless transmission lines in MCM/PCB designs perfectly. In this case, the output response is a replica of the input signal delayed by the time-of-flight (or propagation delay) $t_f = \sqrt{LC}l$ where L and C are the unit-length interconnect inductance and capacitance, respectively, and l is the length of the interconnect. In general, it is assumed that $\sqrt{LC}$ is a constant unchanged by wiresizing. Therefore, the time-of-flight (or propagation delay) on the path from the source to sink s_i is $t_f(s_0, s_i) = \sum_{e_v \in P_T(s_0, s_i)} \sqrt{LC}l(e_v)$, which is proportional to the pathlength $d_T(s_0, s_i)$.

A more general formulation is to model MCM/PCB and IC interconnect structures as lossy transmission lines. Under this formulation, the delay at each sink is the sum of the propagation delay and the rising/falling (or transition) delay of the signal response waveform [13]. The transition delay can be estimated by a higher-order moment-based delay model as follows: Let $h_i(t)$ be the impulse response at a node of interest, say sink s_i, in an interconnect. Let $v_{in}(t)$ be the input voltage of the linear circuit, $v_i(t)$ be the output voltage of sink s_i in the circuit, $V_{in}(s)$ and $V_i(s)$ be the Laplace transform of $v_{in}(t)$ and $v_i(t)$, respectively. Then, $H_i(s) = V_i(s)/V_{in}(s)$ is the transfer function, which can be written as:

$$\begin{aligned} H_i(s) &= \int_0^\infty h_i(t)e^{-st}dt = \sum_{j=0}^{\infty} \frac{(-1)^j}{j!} s^j \int_0^\infty t^j h_i(t)dt \\ &= \sum_{j=0}^{\infty} (-1)^j \cdot m_i^j \cdot s^j. \end{aligned} \tag{1}$$

where m_i^j is the j-th *moment* of the transfer function.

Moments of an RLC interconnect can be computed by the methods proposed in [20, 21]. From the first $2q-1$ moments, one can construct a q-pole transfer function $\hat{H}_i(s)$ to approximate the actual transfer function $H_i(s)$ as follows:

$$\hat{H}_i(s) = \sum_{j=1}^{q} \frac{k_j}{s - p_j}, \tag{2}$$

where p_j's are poles and k_j's are residues, all of which can be determined uniquely by matching the initial boundary conditions, denoted m_i^{-1}, and the first $2q-1$ moments m_i^j of $H_i(s)$ to those of $\hat{H}_i(s)$ [22]. The choice of order q depends on the accuracy required but is always much less than the order of the circuit. In practice, $q \leq 5$ is commonly used. When q is chosen to be two, it is known as the *two-pole model* [23, 11, 24]. From $\hat{H}_i(s)$, one can derive the approximated output voltage $\hat{v}_i(t)$ and solve for the transition delay $t_x(i)$ of the output signal at s_i to reach $x\%$ of V_{DD} (assuming a rising output) for the first time. In this paper, we use the two-pole delay model proposed in [24] to approximate the rise/fall delays of a signal as follows:

$$t_{90}(i) \quad = \quad \begin{cases} 2.36 \cdot \frac{m_i^1 + \sqrt{4m_i^2 - 3(m_i^1)^2}}{2} & \text{if } 4m_i^2 - 3(m_i^1)^2 > 0 \\ 1.66 \cdot \frac{2((m_i^1)^2 - m_i^2)}{\sqrt{3(m_i^1)^2 - 4m_i^2}} & \text{if } 4m_i^2 - 3(m_i^1)^2 < 0 \\ 3.90 \cdot \frac{m_i^1}{2} & \text{if } 4m_i^2 - 3(m_i^1)^2 = 0 \end{cases} \tag{3}$$

Signal response waveform is another important factor in interconnect design. Under the ideal situation, one would prefer the transmission of the input signal to the output not to be distorted. However, due to impedance mismatch and reflection, ringing may occur at the output node, resulting in excessive settling time and voltage overshoot or undershoot, and adversely affecting the circuit performance. Similar to [14, 15, 16], we propose to use moments as an indirect metric to measure signal quality.

For example, if we use the two-pole model to model the interconnect, then ringing can be attributed to the existence of complex poles in $\hat{H}_i(t)$. The condition for the poles to be complex (i.e., for the output response to be non-monotonic, or *underdamped*) is for $\lambda_i = 4m_i^2 - 3(m_i^1)^2$ to be negative. When λ_i is strictly positive, we have a *overdamped* and monotone response. When λ_i is exactly zero, the signal is said to be *critically damped*. On the other hand, an underdamped signal generally has a faster rise delay when compared to a damped signal. Other metrics that involved even higher order moments include the third *central moment* proposed in [15]. With other attributes (such as the signal delay and the wiring length) being the same, we would prefer a RATS-tree with λ_i's as close to zero as possible.

To summarize, we propose to solve the following problem:

Required-Arrival-Time Steiner Tree Routing: Given a set of terminals s_i's and required arrival times q_i's, construct a required-arrival-time Steiner tree T such that $q_i \geq t_T(s_0, s_i)$, critical damping is achieved for good signal quality (i.e., λ_i's are as close to zero as possible), and the total wirelength or wiring area is minimized.

3 RATS-Tree Algorithm

We construct required-arrival-time Steiner trees (RATS-trees) by recursively merging subtrees in a bottom-up fashion. Using this bottom-up tree construction, we can compute moments of sinks and consider interconnect sizing and possibly buffer insertion in the subtree merging process easily.

3.1 Overview of RATS-Tree Algorithm

In the following discussion, we assume that the source is at the origin and the sinks are in the first quadrant. We generalize the algorithm to handle the case where the terminals are at arbitrary locations in Section 3.7. Given a set of terminals, the RATS-Tree algorithm operates on a Hanan grid [25] induced by the terminals. All Hanan grid points are ordered according to their distance from the source, with arbitrary tie breaking.

Let (x_m, y_m) denote the coordinates of grid point m, and $|m| = x_m + y_m$. Given two distinct points p and q, we define the Steiner merging point for p and q, denoted $\langle p, q\rangle$, as the point with coordinates We say that q is dominated by p, or $q \preceq p$ if and only if $x_q \leq x_p$, $y_q \leq y_p$, and $p \neq q$.

In general, the RATS-Tree algorithm follows a branch-and-bound paradigm, by considering *merging* and *skipping* (of merging) of subtrees at a Steiner merging point as introduced in [17], as well as *re-rooting* of a subtree, a concept introduced in [26], at Hanan grid points. Each node in the branch-and-bound (B&B) search tree is associated with a *peer topology set* $\mathcal{T}$ which contains a forest of subtrees constructed so far and a scan level $K = |m|$ where m is the Steiner merging point last considered in the subtree merging process. Not to be confused with a node in a constructed topology, we refer to a node in the B&B search tree as a B&B node.

Starting with a B&B node where the peer topology set contains all single-terminal trees and the scan level $K = \infty$, we expand a B&B node characterized by $(\mathcal{T}, K)$ by considering a new Steiner merging point m with the highest order among all Steiner merging points of tree roots in $\mathcal{T}$ such that $|m| < K$. If the new Steiner merging point is a terminal, then such a merging is called a *terminal merging*. Otherwise, it is called a *Steiner merging*. These operations are similar to those in [17].

Let $P_{\succeq}(m) = \{p|m \preceq p, T_p \in \mathcal{T}\}$ denote the set of tree roots in $\mathcal{T}$ that dominates m. Both terminal and Steiner merging connect from m to each node p in $P_{\succeq}(m)$ and eliminate T_p from $\mathcal{T}$. We make a shortest path connection between m and p by *growing* T_p along the Hanan grid points from p to m in a bottom-up fashion. A new subtree T_m is added to $\mathcal{T}$ and K is updated to $|m|$. Note that when $x_m \neq x_p$ and $y_m \neq y_p$, there are several shortest Manhattan paths from m to p. We consider only the two shortest Manhattan paths that correspond to the boundary of the smallest bounding box containing m and p.

As in the branch-and-bound-based minimum rectilinear Steiner arborescence (MRSA) algorithms in [17], we also consider *skipping* a Steiner merging. In this case, while K is updated to $|m|$, we keep $\mathcal{T}$ unchanged. Therefore, skipping a Steiner merging operation generates an additional child B&B node in the B&B search tree.

Both terminal merging and Steiner merging consider merging at the roots of sub-trees only, thereby producing shortest path trees only. In order to consider a large class of routing solutions, we allow merging at non-root nodes of the sub-trees. We achieve non-root merging by *re-rooting*. After each terminal merging or Steiner merging, the resultant topology T is re-rooted at various Hanan grid points in T, creating several topologies that consist of the same nodes and edges in T, but rooted at different nodes. These re-rooted topologies have identical routing structure, but different sink delays and signal waveforms. Therefore, re-rooting creates several sibling B&B nodes for the newly generated child B&B node.

Figure 1 shows a partial B&B search tree corresponding to the application of the RATS-tree algorithm applied to a 5-terminal net. In the B&B search tree, root nodes of the peer topology set are depicted by empty circles, and non-root nodes by filled circles. For illustrative purpose, the scan level K is shown as a dash line. Note that after the first Steiner merging, the newly created subtree and its re-rooted topologies include a new Hanan grid point.

From the above discussion, it is obvious that growing of a subtree and its subsequent merger with sibling subtrees are bottom-up in nature. As we shall see in Section 3.3, topologies are also re-rooted in a bottom-up manner. It is therefore natural to incorporate

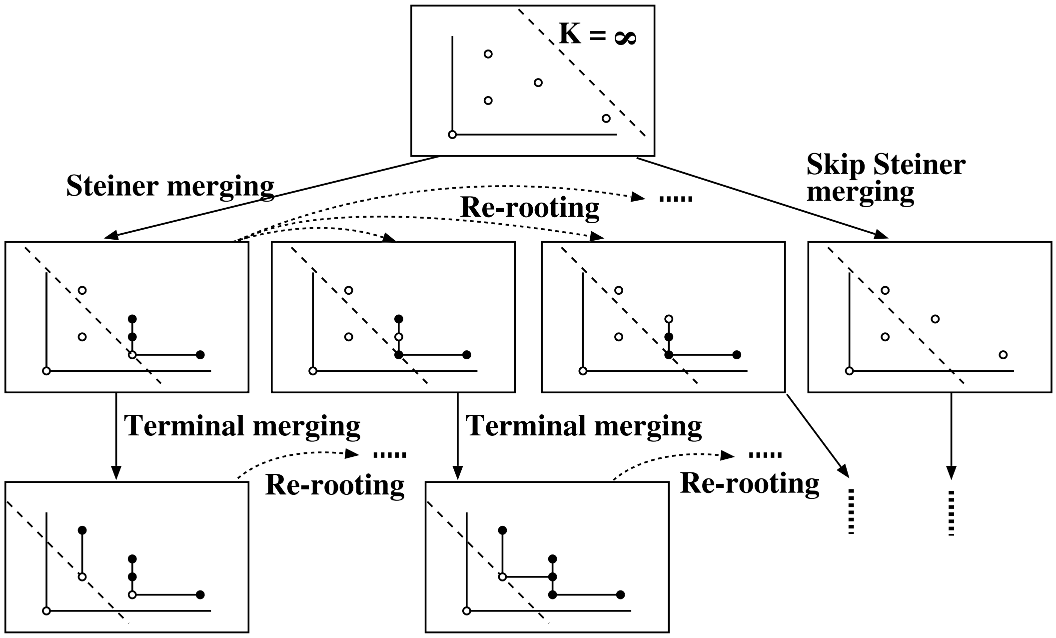


Figure 1: A snapshot of the B&B search tree illustrating the RATS-algorithm applied to a 5-terminal net.

bottom-up wiresizing (and buffer insertion for IC designs) optimization as in [9, 8] during the subtree merging and re-rooting process (see Section 3.4). For an effective evaluation of different wiresizing solutions, as well as topologies, we present in Section 3.5 an efficient incremental bottom-up moment computation method to compute higher-order sink moments for a more accurate delay computation and signal waveform evaluation.

We explore the B&B search tree in a breadth-first traversal order. In other words, nodes at the same level of the B&B search tree are expanded first before any of their children are expanded. The reason for the breadth-first expansion order is to facilitate as much pruning as possible (see Section 3.4). A breadth-first traversal essentially trades space for time. We would like to point out that our algorithm can potentially generate an exponential number of topologies and wiresizing solutions. However, for interconnect optimization problems in practice, most nets have no more than 10 sinks, So the run-time and space requirement of our algorithm are not a problem. For very large nets, our algorithm can selectively store a subset of topologies and their wiresizing solutions for runtime and memory efficiency. Moreover, since these topologies share many sub-topologies, we develop a space efficient *reduced tree representation* which supports sub-tree sharing in different routing topologies. The reduced tree representation will be presented in Section 3.2.

Although the RATS-tree algorithm follows the branch-and-bound paradigm, we would like to point out that in general, it does not guarantee optimal RATS-tree construction even under the pathlength formulation for two reasons. First, an optimal RATS-tree may not lie on the Hanan grid. An example of such a RATS-tree is given in Figure 2(a). Second, our algorithm may miss the optimal solutions due to the greedy nature of terminal merging (See Figure 2(b). Nonetheless, if it is required that all the sinks have a shortest path to the source, then our RATS-tree algorithm will always return an optimal rectilinear Steiner arborescence.

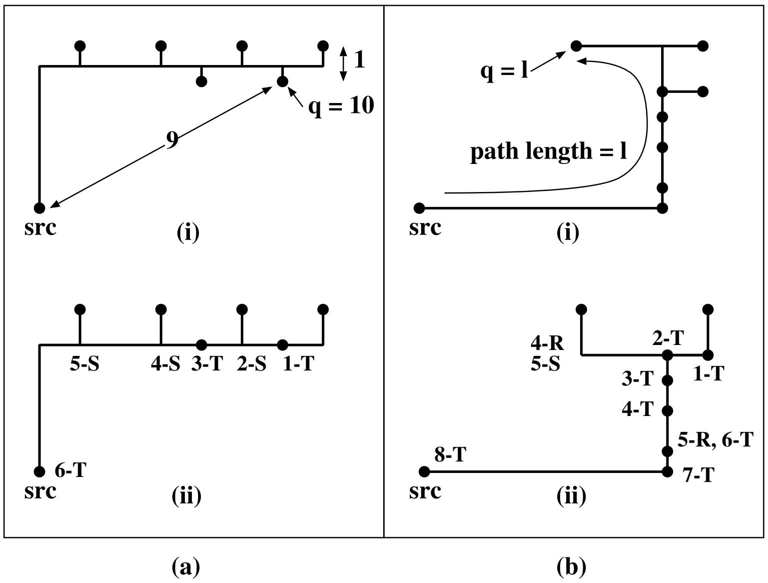


Figure 2: (a) Sub-optimality of the Hanan-grid based RATS-tree algorithm. (b) Sub-optimality of our RATS-tree algorithm due to the greedy nature of terminal merging.

3.2 Reduced Tree Representation

To achieve an efficient implementation, we introduce the notion of a *reduced tree representation* to allow topologies to share common sub-topologies. This is analogous to the concept of *reduced ordered binary decision diagram* (ROBDD) [27] that is commonly used in logic synthesis.

We illustrate the idea of a reduced tree representation in Figure 3. Consider two topologies T_1 and T_2' that correspond to the same interconnect structure with 8 nodes in Figure 3(a). T_1 is rooted at node 1 and T_2' is rooted at node 2. Suppose we already have a reduced tree representation of T_1 as shown by the filled and shaded circles and solid line edges in Figure 3(b). To avoid confusion, any new node created for T_2' to represent node i of the interconnect structure in Figure 3(a) is labeled i'. Since T_2' shares the subtree T_5, and leaf nodes 3 and 4 in T_1, to create a reduced tree representation of the topology T_2', we only need to create two additional nodes $2'$ and $1'$ (empty circles in Figure 3(b)) to represent T_2' completely.

Sharing is allowed not just among re-rooted topologies which cover the same set of nodes, as depicted in the above example. It is also possible to share sub-topologies in topologies that cover different sets of nodes. In Figure 4, we could merge T_u and T_v at Steiner point r. Alternatively, we could skip the merging and in the next iteration, merge T_v and T_w at Steiner point r'. In this case, we would create T_r and $T_{r'}$ in different B&B nodes in the algorithm, and T_r and $T_{r'}$ share a common sub-topology T_v. This, again, is analogous to multi-root ROBDD's for multiple-output functions where instead of keeping one distinct ROBDD for each output function, different outputs share common structures (co-factors) in the ROBDDs. Note that a topology can be uniquely identified by the root node and the child topologies of the root note. In our implementation, we keep each topology in a hash table so that each topology will be created only once.

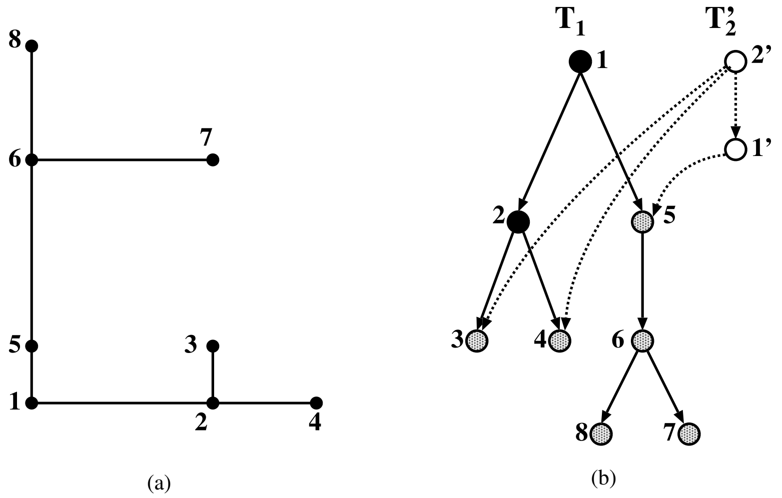


Figure 3: (a) An interconnect with 8 nodes. (b) Reduced tree representations for topologies T_1 and T_2^l rooted at node 1 (filled circle) and 2 (empty circle) of the same interconnect in (a), respectively. The shared nodes are shaded.

3.3 Re-rooting of Topology

Now, we present a linear time algorithm to re-root a topology T_r rooted at node r. Let T_w denote the sub-tree rooted at node w in T_r. Let the k children of r be u_i for $i = 1 \cdots k$. Suppose we want to re-root T_r at one of the child nodes, say u_j where $1 \leq j \leq k$. Let $T^l_{u_j}$ denote the new topology. Let T^l_w be the sub-tree rooted at node w in $T^l_{u_j}$. Re-rooting of T_r at node u_j effectively swaps the parent-child relationship of r and u_j. The parent-child relationships of other nodes are not affected. For example, in Figure 3 where T_1 is being re-rooted at node 2, only two changes are required to reflect the re-rooting operation. First, node 1 becomes a child node of node 2 in T^l_2, and nodes 3 and 4 which are child nodes of 2 in T_1 remain as children of node 2 after the re-rooting operation. Moreover, while the child nodes of 1 in T_1 are nodes 5 and 2, node 5 is the only child node of node 1 in T^l_2 after re-rooting.

We note that the original topology can be re-rooted at all nodes in the topology by a simple depth-first traversal. Using the tree T_r in the preceding discussion as an example, where u_j is the next node in the traversal order, we first create T^l_r, which takes time linear to the degree of T_r. T^l_r differs from T_r in that it does not contain T_{u_j} as its child topology. Next, we create $T^l_{u_j}$, which again takes time linear to the degree of T_{u_j}. $T^l_{u_j}$ differs from T_{u_j} in that it contains T^l_r. Only two additional nodes and $O(k)$ edges are created where k is the maximum degree of a node in the topology. This process is repeated for all child nodes of T_r in the depth-first traversal. Therefore, the total time complexity to re-root an topology with g grid points in it is $O(g \cdot k)$. Since we are dealing with planar single-layer Manhattan routing, the degree of each node is bounded by 4. We can conclude that it takes linear time to re-root a topology. Moreover, instead of requiring $O(g^2)$ nodes and edges to represent all possible re-rooted topologies under an explicit representation, a reduced tree representation can represent all re-rooted topologies using only $O(g)$ nodes and edges.

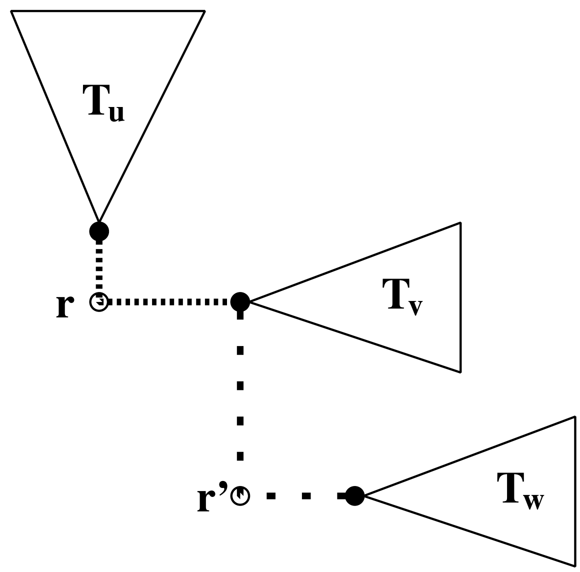


Figure 4: T_r and T_r' share a common sub-topology T_v. T_r is created by merging T_u and T_v, whereas $T_{r'}$ is created by skipping the Steiner merging of T_u and T_v and merging T_v and T_w instead.

3.4 Wiresizing Optimization

From the above description of the re-rooting operation, the basic operations to create re-rooted topologies are still growing and merging of subtrees in a bottom-up manner (albeit a top-down depth-first traversal). Therefore, it is natural to incorporate bottom-up wiresizing optimization into the algorithm as in [9, 8]. Note that it is easy to extend it to consider buffer insertion as well for IC designs.

Basically, given a subtree with a set of I wiresizing configurations. When we grow a subtree, we consider a given set of $|W|$ candidate wire widths for the new root edge e, resulting in $I \times |W|$ combinations of wire width assignments for T_e, the new subtree tree rooted at e. Similarly, merging of two subtrees with I and J wiresizing configurations, respectively, produces a total of $I \times J$ possible wire width assignments for the new subtree.

Clearly, such an approach generates an exponential number of wiresizing configuration. Similar to the previous bottom-up wiresizing algorithms [6, 8], we perform pruning of wiresizing solutions of a topology T. Each wiresizing solution $\mathcal{W}$ of T is associated with a triple $(Cap(T,\mathcal{W}), Slack(T,\mathcal{W}), SQ(T,\mathcal{W}))$, where $Cap(T,\mathcal{W})$ is the total capacitance, $Slack(T,\mathcal{W}) = \min_{s_i \in T}(q_i - t_{T,\mathcal{W}}(s_i))$ with $t_{T,\mathcal{W}}(s_i)$ being the two-pole sink delay of s_i in T (Section 2), and $SQ(T,\mathcal{W})$ is the signal quality of the topology and wiresizing solution defined as follows: Recall that $\lambda_i = 4m_i^2 - 3(m_i^1)^2$ is a measure of the degree of damping for sink s_i. If $\lambda_i > 0$, s_i is overdamped. If $\lambda_i < 0$, s_i is underdamped. Otherwise, s_i is critically damped. We propose to measure the signal quality of a tree T with wiresizing solution $\mathcal{W}$ by $SQ(T,\mathcal{W}) = \min_{s_i \in T} \lambda_i$. In other words, the signal quality is measured by the worst signal response waveform among all sinks.

Given two wiresizing solutions $\mathcal{W}$ and $\mathcal{W}'$ for T, we say that $\mathcal{W}'$ is redundant if $Cap(T,\mathcal{W}) \leq Cap(T,\mathcal{W}')$, $Slack(T,\mathcal{W}) \geq Slack(T,\mathcal{W}')$, $SQ(T,\mathcal{W}) \geq SQ(T,\mathcal{W}')$, *and* at least one of the three inequalities is a strict inequality. Note that the pruning criterion is similar to that in [6, 8] if we do not consider $SQ(T,\mathcal{W})$. The basic idea of our pruning criterion is as follows: with other attributes such as the *Cap*'s and *SQ*'s being identical, we would prefer a solution with a larger *Slack* in order to reduce the possibility of constructing an infeasible topology (with negative *Slack*). Similarly, with *Cap*'s and *Slack*'s being identical, we would always pick the solution with a better signal quality, i.e., larger *SQ*. Finally, we would always choose a solution with a smaller *Cap* if other attributes of the two solutions are identical. Essentially, we are trading-off total capacitance for timing slack or signal quality and vice versa.

We prune all redundant wiresizing solutions from the solution space. To perform such a pruning, we have to compute the higher order moments and evaluate the sink delays and signal quality for each wiresizing solution. We present an incremental bottom-up moment computation algorithm in the following.

3.5 Incremental Bottom-Up Moment Computation

Moments can be computed by the polynomial-time algorithms in [20, 21]. However, these works compute moments by traversing the entire tree iteratively, and do not allow incremental computation of moments. As the topology changes, another round of iterative tree traversals is needed to re-compute the moments. Even when we restrict the topology change to a simple addition of an RLC segment to the root of the original tree, which is the basis of our bottom-up topology construction algorithm, moments cannot be incrementally updated with the previous methods. Therefore, these previous approaches are not suitable for our RATS-tree algorithm. A method to incrementally compute moment in a bottom-up fashion for RC interconnects was recently presented in [10]. However, it did not consider the inductance effect. In the following, we generalize the algorithm to handle moment computation of a RLC tree.

Consider an RLC tree T_v rooted by node v. For any node w in T_v, let m_w^p be the p-th moment of node w and $C_{T_w}^p = \sum_{j \in T_w} m_j^p \cdot C_j$ be the total p-th order moment weighted capacitance of T_w [20] where C_j is the capacitance connected to node j. Now, we add a new edge uv at the root of T_v to obtain a new tree T_u rooted at u. Let $\overline{C}_{T_w}^p$ be the new total p-th order moment weighted capacitances of T_w for w in T_u. Similarly, let $\overline{m}_w^p$ be the new p-th moment of node w in T_u. Let R_v, L_v, and C_v be the total resistance, inductance, and capacitance of the edge uv, respectively. The following theorem illustrates how we can express $\overline{m}_w^p$ for $p \geq 0$ in terms of $\overline{m}_v^q$ and m_w^q for $q = 0 \cdots p$.

Theorem 1 *For root node u,*

$$\overline{m}_u^p = \begin{cases} 0 & \textit{if } p = -1 \textit{ or } p > 0 \\ 1 & \textit{if } p = 0. \end{cases} \tag{4}$$

For $p \geq 0$*,*

$$\overline{C}_{T_v}^{p-1} = \overline{m}_v^{p-1} C_v + \sum_{q=0}^{p-1} \overline{m}_v^{p-1-q} C_{T_v}^q \tag{5}$$

$$\overline{m}_v^p = R_v \overline{C}_{T_v}^{p-1} - L_v \overline{C}_{T_v}^{p-2} \tag{6}$$

$$\overline{m}_w^p = \sum_{q=0}^{p} \overline{m}_v^{p-q} m_w^q \ \forall\, w \in T_v \tag{7}$$

The proof of the theorem can be found in the Appendix. The theorem implies that if we keep $C_{T_v}^i$ for $i = 0 \cdots p-1$, we can compute up to the p-th order the new moments for v by computing $\overline{C}_{T_v}^{i-1}$ followed by $\overline{m}_v^i$ for $i = 0 \cdots p$ by Eqns. (5) and (6). Then, we can update the moments of all the sinks in the topology by Eqn. (7). From the above theorem, we can state the following corollary which allows us to incrementally update sink moments during the merging operation and compute the total p-th order moment weighted capacitance at the new root u:

Corollary 1 *Consider k topologies, denoted T_{v_i} for $i = 1 \cdots k$. If the k topologies are merged at a new node u, then for $p \geq 0$ and $i = 1 \cdots k$, $\overline{C}^{p-1}_{T_{v_i}}$, $\overline{m}^p_{v_i}$ and $\overline{m}^p_w$ $\forall$ $w \in T_{v_i}$ can be computed by Eqns. (5–7), respectively. Let c^s_u be the sink capacitance at the root node u, i.e., $c^s_u = 0$ if u is not a sink. For $p \geq 0$,*

$$\overline{C}^{p-1}_{T_u} \quad = \quad \overline{m}^{p-1}_u c^s_u + \sum_{i=0}^{k} \overline{C}^{p-1}_{T_{v_i}} \tag{8}$$

From Theorem 1 and Corollary 1, the time complexity to update the moments of n sinks in a tree is $O(n \cdot p^2)$. The auxiliary space requirement is $O(p)$. On the other hand, the time complexity of the method proposed by [20] is $O(g \cdot p)$ where g is the total number of grid nodes in the tree and the auxiliary space requirement is $O(g)$. Since our RATS-tree algorithm is based on the Hanan-grid, g could be in the order of $O(n^2)$.

We can integrate the incremental bottom-up moment computation algorithm with our RATS-tree algorithm easily. For each topology constructed by our algorithm, we keep a set of irredundant wiresizing solutions and their corresponding sink moments (up to a pre-specified p-th order) for the topology. Note that all these are stored in the reduced tree representation. As we grow a topology along the path of Hanan grid points towards the new root in the merge operation, we compute the length of each new edge and for each candidate wire width of the new edge, derive the interconnect resistance, inductance and capacitance. These RLC parasitics are used to update the moments of the sinks using Theorem 1. We then use Corollary 1 to compute the weighted capacitances at the new root. The RATS-tree algorithm then prunes the wiresizing solutions of the newly created topology.

3.6 Further Pruning of Solution Space

Removal of redundant wiresizing solutions from a topology is not the only pruning technique employed by the RATS-tree algorithm. We apply several other techniques to prune the B&B search tree. For example, with a clever arrangement of the tree roots in $\mathcal{T}$, it is possible to find the new Steiner merging point m and $P_{\succeq}(m)$ in $O(n)$ time and partially avoid generation of non-planar routings during the merging operation (see [17] for details.)

Due to the re-rooting operation, however, it is possible that we may create many sibling B&B nodes which will generate non-planar routings in subsequent merging operations. While we can prune away B&B nodes that contain non-planar routings, a better approach is to avoid creating them as much as possible. We introduce the *visible set* of a RATS-tree, which is a subset of the Hanan grid points in the RATS-tree defined as follows:

A node p dominates q, denoted $q \preceq p$ if and only if $\langle p, q \rangle = q$ and $p \neq q$. Node p x-dominates q, denoted $q \preceq_x p$, if $q \preceq p$ and $p_y = q_y$. Given a RATS-tree T, $p \in T$ is x-visible if it does not x-dominates any node in T. We define the y-dominance relation and y-visibility similarly. A node p is visible if it is either x-visible or y-visible. In Figure 3(a), for example, except for nodes 3 and 7, the rest of the nodes are visible. We can show that if we order the x-visible nodes by their y-coordinates, then the merging operations can generate the x-visible nodes of the new topology from the x-visible nodes of its child topologies in linear time.

Similarly, the y-visible nodes can be computed in linear time. We do not re-root at all Hanan grid points in the tree but only at the visible nodes, i.e., grid points in the visible set.

Another pruning technique that we employ is to prune wiresizing solutions among *different* topologies. We say that two RATS-trees T and T' share the same *alias* if they are rooted at the same node and cover the same set of sinks. Consider wiresizing solutions $\mathcal{W}$ of T and $\mathcal{W}''$ of T' where T and T' share the same alias, we say that $\mathcal{W}''$ is redundant if $Cap(T, \mathcal{W}) \leq Cap(T', \mathcal{W}'')$, $Slack(T, \mathcal{W}) \geq Slack(T', \mathcal{W}'')$, $SQ(T', \mathcal{W}) \geq SQ(T', \mathcal{W}'')$, *and* at least one of the three inequalities is a strict inequality. A RATS-tree is redundant if all of its wiresizing solutions are redundant with respect to the wiresizing solutions of topologies with the same alias.

We use a hash table to store aliases. For each alias in the table, we maintain a set of irredundant wiresized RATS-trees. For each RATS-tree generated (whether by merging or re-rooting operation), our algorithm updates the set of irredundant wiresized RATS-trees that share the same alias as the newly generated RATS-tree. All B&B nodes that are associated with redundant RATS-trees are pruned.

3.7 Summary

The RATS-tree can be generalized by the following re-definitions to handle the case where the sinks are not restricted to the first quadrant, and the source is not located at the origin. Given two distinct points p and q, we define the Steiner merging point $\langle p, q\rangle$ to be $(\mathrm{med}(x_{s_0}, x_p, x_q), \mathrm{med}(y_{s_0}, y_p, y_q))$ where med() returns the median of three numbers. We say that $q \preceq p$ if and only if $\langle p, q\rangle = q$ and $p \neq q$. $|m| = d(s_0, m)$ is the Manhattan distance between s_0 and m. Lastly, the visible nodes of a topology are ordered and kept in their respective quadrants with respect to the source to facilitate computation of the visible nodes of parent topologies.

To summarize, we use a queue to implement breadth-first traversal of the B&B search tree. At the beginning of the algorithm, the queue contains a B&B node with a peer topology set $\mathcal{T}$ containing all single-terminal subtrees and a scan level $K = \infty$. We use the pair $(\mathcal{T}, K)$ to denote a B&B node. Until the queue is empty, the algorithm iterates the expansion of a B&B node removed from the head of the queue by either performing a terminal/Steiner merging operation, or skipping a Steiner merging operation. In the case of subtree merging, the resultant topology is re-rooted to generate sibling B&B nodes. All newly generated B&B nodes are appended to the end of the queue. A brief outline of our RATS-tree algorithm is given in Figure 5.

The above summary assumes no pruning of the B&B search tree. To consider such pruning, two simple modifications are made to the algorithm. A B&B node with $(\mathcal{T}, K)$ is only expanded if all partial RATS-trees in $\mathcal{T}$ are irredundant. After each partial RATS-tree is constructed (by either merging or re-rooting operation), the irredundant list of RATS-trees with the same alias as the newly created topology is updated.

4 Experimental results

We have implemented the RATS-tree algorithm in C++ language and evaluated the algorithm for MCM designs. We use randomly generated netlists with 6 to 12 terminals on a $10cm \times 10cm$ MCM substrate. In the first set of experiments, we run the RATS-tree algorithm under the pathlength formulation to investigate the trade-off between the pathlength and the routing cost of the

RATS-tree Algorithm

```
B&B-Queue ← {({{s_0},{s_1},{s_2},···,{s_n}},∞)}
while B&B-Queue not empty do
   (𝒯,K) ← Remove-Head-of-Queue(B&B-Queue)
   if all RATS-trees in 𝒯 irredundant then
      (m,m_N,m_E,Merger-Type) ← Find-Merging-Point((𝒯,K))
      if Merger-Type = TERMINAL-MERGING then
         (𝒯',K') ←(Terminal-Merge(m,𝒯),|m|)
      else
         /* STEINER-MERGING */
         (𝒯',K') ←(Steiner-Merge(m,m_N,m_E,𝒯),|m|)
         /* SKIP STEINER-MERGING */
         K ← |m|
         Append (𝒯,K) to B&B-Queue
      end if
      New-B&B-Set ← Re-root((𝒯',K')) + (𝒯',K')
      Append all B&B nodes in New-B&B-Set to B&B-Queue
   end if
end while
```

Figure 5: Outline of the RATS-tree algorithm.

	$k=1$	$k=1.2$		$k=1.5$		$k=2$	
n	CPU	Length	CPU	Length	CPU	Length	CPU
6	0.3	0.996	0.3	0.987	0.3	0.984	0.3
7	0.9	1.000	0.9	0.996	0.9	0.982	0.9
8	0.8	0.989	0.9	0.973	0.9	0.963	0.9
9	4.3	0.991	4.7	0.976	4.2	0.973	4.3
10	8.7	0.972	8.8	0.956	9.3	0.948	8.6
11	37.4	0.988	47.7	0.971	40.4	0.951	36.3
12	96	0.978	152.2	0.958	145.0	0.942	125.0

Table 1: Pathlength requirement and routing cost trade-off.

topologies generated. Ten random n-pin nets are generated for each n ranging from 6 to 12. For each routing instance, we consider four different arrival time (or pathlength) requirements, namely $q_i = k \times d(s_0, s_i)$, where $k = \{1, 1.2, 1.5, 2\}$ for sinks s_i's. Note that when $k = 1$, it is known as the shortest path Steiner routing.

Table 1 shows the trade-off between the pathlength requirement and routing cost. We normalize the length of each resultant topology with respect to that of the shortest path routing. Therefore, we do not show the routing length when $k = 1$. The columns labeled "CPU" give the worst-case CPU seconds incurred among the ten nets for each n and k. In general, the average CPU time is much lower. For example, the average CPU time for 12-pin net is only 22 seconds whereas the worst-case CPU time is 152 seconds. While the average gain in terms of total routing length is a modest 5% when we relaxed k from 1 to 2, in some instances, the gain could be as high as 15%. Such high gains in general occur more frequently in larger nets than in smaller nets. Moreover, the shortest path routings generated by our algorithm is optimal in terms of wirelength, and therefore, it is more difficult to achieve much reduction in wirelength.

Topology	Max-delay (ns)	Settling Time (ns)	Overshoot	Wire Cap. (pF)
$RATS1$	2.15	0.83	0.10	33.4
$RATS2$	2.28	0.76	0.07	38.3
$RATS3$	2.26	0.00	0.04	37.7
$RATS4$	2.24	0.71	0.06	35.7
$RATS5$	2.18	0.82	0.10	31.9

Table 2: Trade-off among maximum sink delay, signal settling time, voltage overshoot, and routing cost for a small $k = 2$.

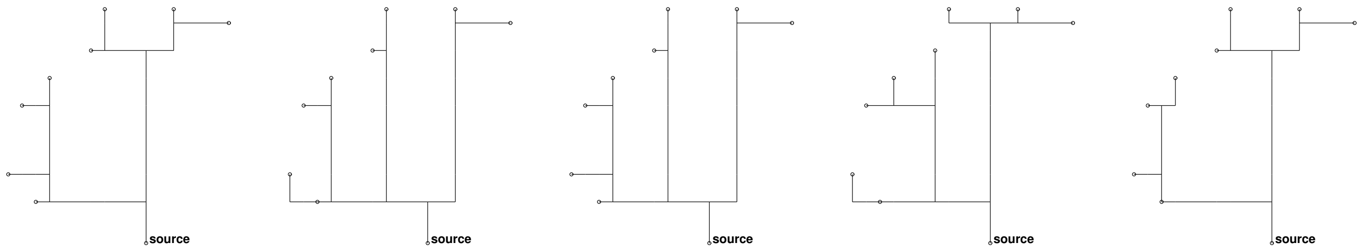
In the second set of experiments, we apply the RATS-tree algorithm without consideration of wiresizing under the two-pole model. The purpose of this experiment is to investigate the impact of routing topology on signal delay and integrity. The interconnect parameters that are used by our algorithms for moment, delay, and signal quality computations are obtained from the Micro Module System (MMS) D500 process on Aluminum offered through MIDAS. Assuming a nominal width of 19μm, the interconnect resistance, inductance, and capacitance are $236.84\Omega/m$, $301.49nH/m$, and $128.99pF/m$, respectively. The load capacitance of each sink is assumed to be $1pF$, and the driver resistance ranges from 10Ω to 30Ω, depending on the size of the net and the proximity of the terminals.

For each net, we set the required arrival time of sink s_i to be $k \times \sqrt{LC}d(s_0, s_i)$ where $k > 1$. Note that k is larger than 1 in order to account for the rise/fall delay. For each net, our algorithm constructs a large class of topologies satisfying the delay requirements. We then run SPICE simulations using the transmission line model to evaluate the sink delay and measure signal integrity in terms of the signal settling time and voltage overshoot of these constructed topologies. We measure the signal delay at the 90% V_{dd} (assuming a rising signal) and the signal settling time is the time taken for the signal to settle above the 90% V_{dd}. In general, the large class of topologies generated by our algorithm is able to provide a trade-off among maximum sink delay, signal settling time, voltage overshoot, and routing cost.

For example, Table 2 shows the maximum delays, signal settling times, voltage overshoots, and routing costs for the topologies generated by our RATS-tree algorithm for one of the randomly generated 9-pin nets. Both the delay and settling time of each topology are in ns, and the total wire capacitance is in pF. The voltage overshoot is normalized with respect to V_{dd}. In this example, due to a small k of 2, most of the RATS-trees generated are shortest-path Steiner tree. As we can see from Figure 6, except for $RATS5$, the topologies $RATS1$–4 are all shortest-path Steiner tree. In fact, $RATS1$ is an optimal Steiner arborescence. While it is the best in terms of maximum delay, its total wire capacitance, settling time, and voltage overshoot are not necessary the smallest.

Table 3(a) and (b) show the routing solutions (for the same net as in Table 2) for the case where we relax k to 3 and 6, respectively. Figure 7 shows an example of the topologies generated for each k. Note that the total wire capacitance of $RATS10$ is 20% smaller than that of $RATS1$. We only show the simulation results of a subset of the topologies generated by our algorithm in Table 3. In total, 16, and 20 topologies are generated for the two different k's, respectively. Note that if we do not consider pruning, the numbers could be much larger. There are also overlaps between the three sets of topologies.

In the final set of experiments, we apply the RATS-tree algorithm with wiresizing under the two-pole model to investigate the impact of wiresizing on signal delay and signal integrity. In addition to the nominal width of 19μm, we allow four other different



(a) $RATS1$ (b) $RATS2$ (c) $RATS3$ (d) $RATS4$ (e) $RATS5$

Figure 6: Topologies generated by the RATS-tree algorithm for a 9-pin net for a small $k = 2$.

Topology	Max-delay (ns)	Settling Time (ns)	Overshoot	Wire Cap. (pF)
$RATS6$	2.65	0.00	0.09	29.9
$RATS7$	2.39	0.58	0.15	30.5

(a) $k = 3$

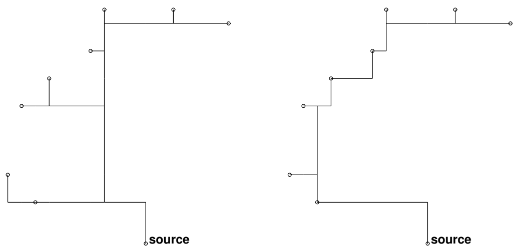
Topology	Max-delay (ns)	Settling Time (ns)	Overshoot	Wire Cap. (pF)
$RATS8$	2.84	0.00	0.12	26.7
$RATS9$	2.81	0.00	0.11	27.4

(b) $k = 6$

Table 3: Trade-off among maximum sink delay, signal integrity, and routing cost for the case where (a) $k = 3$ and (b) $k = 6$.

wire widths $\{39\mu m, 59\mu m, 79\mu m, 99\mu m\}$ for the interconnect. The interconnect parameters for the four additional wire widths are given in Table 4. In this experiment, the RATS-tree algorithm produces a much larger number of solutions when compared to the previous two experiments. For instance, there are a total of 123 wiresized solutions generated for the above 9-pin sample net. The run-times also increase significantly due to the wiresizing optimization. For example, if we keep *all* the irredundant wiresizing solutions for each irredundant topologies generated in the algorithm, the average run-times for 6-pin, 9-pin, and 12-pin nets are 5 seconds, 341 seconds, and 12,702 seconds, respectively.

However, we observe that a majority of the irredundant wiresizing solutions of a topology are very similar to each other in terms of $Cap(T, \mathcal{W})$, $Slack(T, \mathcal{W})$, and $SQ(T, \mathcal{W})$. Therefore, we adopt a more aggressive pruning approach where we keep only a user-specified number, say I, of irredundant wiresizing solutions for each topology. In this approach, we first apply the pruning technique outlined in Section 3.4. If the number of irredundant wiresizing solutions is greater than I, then we always keep the wiresizing solutions that correspond to the smallest $Cap(T, \mathcal{W})$, largest $Slack(T, \mathcal{W})$, and largest $SQ(T, \mathcal{W})$. We select the rest of the solutions uniformly by their ordering based on $Cap(T, \mathcal{W})$'s. Using this approach, we are able to cut down the run-time significantly while maintaining high solution quality. For example, the average run-times for 6-pin, 9-pin, and 12-pin nets are reduced to 4 seconds, 72 seconds, and 604 seconds, respectively when we set I to 15. Note that the run-times for the RATS-tree algorithm in the first two experiments are similar. While the run-times of the RATS-tree algorithm may compare unfavorably with those of algorithms in [9, 8] which also consider wiresizing with topology construction, the higher computational complexity is



(a) $RATS6$ (b) $RATS8$

Figure 7: An example of the topologies generated by the RATS-tree algorithm for a 9-pin net for (a) $k = 3$ and (b) $k = 6$.

	39μm	59μm	79μm	99μm
R (Ω/m)	115.38	76.27	59.96	45.45
L (nH/m)	205.88	157.82	128.50	108.62
C (pF/m)	188.89	246.42	302.64	358.03

Table 4: MMS interconnect parameters.

due to its ability to handle a large class of topologies.

Again, we use the same 9-pin net in the above discussion to illustrate the impact of wiresizing. In general, we observe that most of the topologies that resemble shortest-path Steiner tree more have more wiresizing solutions. For example, topologies such as $RATS1$–5 have more wiresizing solutions than $RATS6$–9. We also observe that most of the wiresizing solutions are monotone, i.e., the widths decrease from source to sinks. Note that these solutions are obtained by the RATS-tree algorithm with aggressive pruning of wiresizing solutions. Table 4 shows the impact of wiresizing on $RATS1$-3 and a new shortest-path Steiner tree, denoted $RATS10$. We use $RATSj.a$ and $RATSj.b$ to refer to two different wiresizing solutions of $RATSj$. From the table, we see that it is possible to improve both max-delay and voltage overshoot simultaneously by wiresizing ($RATS1.a$ and $RATS3.a$). On the other hand, we can also use wiresizing to minimize voltage overshoot at the expense of signal delay. In all cases, the total wire capacitance increases.

5 Concluding Remarks

To summarize, this paper describes a RATS-tree construction algorithm under a higher-order RLC interconnect model. Our algorithm considers the impact of routing on signal delays and response waveforms, and returns not one, but a set of appropriately wiresized routing topologies. Our algorithm optimizes signal waveform, not just delay, and due to its capability to handle a large class of topologies, provides a trade-off among routing cost, signal delay, and signal integrity. In the algorithm, we introduce a space-efficient reduced tree representation to capture all topologies and wiresizing solutions constructed during the execution of our algorithm. We also present a new algorithm to incrementally compute moments of sinks in a bottom-up manner.

Currently, we are working towards improving the run-time of the algorithm. Besides the simple yet effective pruning technique that we outlined in Section 4, we are investigating the effectiveness of other pruning techniques to reduce run-time while maintaining high solution quality. For example, we can perform pruning using bin sorting based on $Cap(T, \mathcal{W})$, and keep only

Topology	Max-delay (ns)	Settling Time (ns)	Overshoot	Wire Cap. (pF)
$RATS1.a$	1.82	0.77	0.08	37.6
$RATS1.b$	2.18	0.85	0.04	44.8
$RATS2.a$	2.64	0.92	0.04	57.9
$RATS3.a$	2.19	0.69	0.03	44.2
$RATS3.b$	2.64	0.00	0.01	56.0
$RATS10.a$	2.71	0.00	0.00	64.3

Table 5: Impact of wiresizing on signal delay and integrity.

a representative wiresizing solution from each bin. Furthermore, we observe that the re-rooting operation generates many child B&B nodes. By limiting the number of re-rooting allowed along any branch of the B&B search tree, it is possible to reduce the run-time without any solution quality degradation. We are also focusing our efforts on generalizing the RATS-tree algorithm for general graph routing, which would enable us to handle multi-layer routing more effectively.

Acknowledgment

The authors would like to thank Dr. Dian Zhou of University of North Carolina, and Yean-Yow Hwang, Kei-Yong Khoo, Hardy Leung, and Patrick Madden at UCLA for their helpful discussions and critical comments.

References

[1] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integration, the VLSI Journal*, vol. 21, pp. 1–94, 1996.

[2] J. Cong, K. S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," in *Proc. Design Automation Conf*, pp. 606–611, 1993.

[3] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Near-optimal critical sink routing tree constructions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 1417–1436, Dec. 1995.

[4] J. Cong and K. S. Leung, "Optimal wiresizing under the distributed Elmore delay model," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 321–336, Mar. 1995.

[5] N. Menezes, S. Pullela, F. Dartu, and L. T. Pillage, "RC interconnect synthesis —- a moment fitting appraoch," in *Proc. Int. Conf. on Computer Aided Design*, pp. 418–425, 1994.

[6] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. Int. Conf. on Computer Aided Design*, pp. 138–143, Nov. 1995.

[7] C. P. Chen, Y. P. Chen, and D. F. Wong, "Optimal wire-sizing formula under the Elmore delay model," in *Proc. Design Automation Conf*, pp. 487–490, 1996.

[8] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. Int. Conf. on Computer Aided Design*, pp. 44–49, Nov. 1996.

[9] J. Lillis, C. K. Cheng, T. T. Y. Lin, and C. Y. Ho, "New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing," in *Proc. Design Automation Conf*, pp. 395–400, June 1996.

[10] F.-J. Liu, J. Lillis, and C.-K. Cheng, "A new layout-driven timing model for incremental layout optimization," in *Proc. Asia South Pacific Design Automation Conf.*, 1997.

[11] D. Zhou, F. Tsui, and D. S. Gao, "High performance multichip interconnection design," in *Proc. 4th ACM/SIGDA Physical Design Workshop*, pp. 32–43, Apr. 1993.

[12] M. Sriram and S. M. Kang, "Performance driven MCM routing using a second order RLC tree delay model," in *Proc. Int. Conf. on Wafer Scale Integration*, pp. 262–267, 1993.

[13] T. Xue, E. S. Kuh, and Q. Yu, "A sensitivity-based wiresizing approach to interconnect optimization of lossy transmission line topologies," in *Proc. IEEE Multi-Chip Module Conf.*, pp. 117–121, 1996.

[14] R. Gupta and L. T. Pillage, "OTTER: Optimal termination of transmission lines excluding radiation," in *Proc. Design Automation Conf*, pp. 640–645, 1994.

[15] B. Krauter, R. Gupta, J. Willis, and L. T. Pileggi, "Transmission line synthesis," in *Proc. Design Automation Conf*, pp. 358–363, 1995.

[16] R. Gupta and L. T. Pileggi, "Constrained multivariable optimization of transmission lines with general topologies," in *Proc. Int. Conf. on Computer Aided Design*, pp. 130–137, 1995.

[17] K.-S. Leung and J. Cong, "Fast optimal algorithms for the minimum rectilinear Steiner arborescence problem," in *Proc. IEEE Int. Symp. on Circuits and Systems*, 1997. to appear.

[18] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably good performance-driven global routing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, pp. 739–752, June 1992.

[19] A. B. Kahng and G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, pp. 893–902, July 1992.

[20] Q. Yu and E. S. Kuh, "Exact moment matching model of transmission lines and application to interconnect delay estimation," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 3, pp. 311–322, June 1995.

[21] A. B. Kahng and S. Muddu, "Two-pole analysis of interconnection trees," in *Proc. IEEE Multi-Chip Module Conf.*, pp. 105–110, Jan. 1995.

[22] L. T. Pillage and R. A. Rohrer, "Asymptotic waveform evaluation for timing analysis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, pp. 352–366, Apr. 1990.

[23] M. A. Horowitz, *Timing Models for MOS Circuits*. PhD thesis, Stanford University, 1984.

[24] A. B. Kahng and S. Muddu, "An analytical delay model for RLC interconnects," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 4.237–4.240, May 1996.

[25] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM Journal of Applied Mathematics*, vol. 14, pp. 255–265, 1966.

[26] J. H. Huang, A. B. Kahng, and C.-W. A. Tsao, "On the bounded-skew routing tree problem," in *Proc. Design Automation Conf*, pp. 508–513, June 1995.

[27] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

Appendix: Proof of Theorem 1

First, we present the method proposed in [20] for moment computation of an RLC tree. Consider a lumped RLC tree with n nodes. Let $\overline{k}$ be the parent node of node k, and T_k be the subtree rooted at node k. Let C_k be the capacitance connected to node k, R_k and L_k be the resistance and inductance of the branch between $\overline{k}$ and k. Let m_k^p be the p-th order moment of $H_k(s)$, the transfer function at node k.

Let $C_{T_k}^p = \sum_{j \in T_k} m_j^p \cdot C_j$, which is the total p-th order weighted capacitance of T_k, then m_k^p (for $p > 0$) can be written recursively as [20]:

$$m_k^p = \begin{cases} 0 & \text{if } p = -1, \\ 1 & \text{if } p = 0, \\ 0 & \text{if } p > 0 \text{ and } k \text{ is the root}, \\ m_{\overline{k}}^p + R_k \cdot C_{T_k}^{p-1} - L_k \cdot C_{T_k}^{p-2} & \text{if } p > 0 \text{ and } k \text{ is not the root.} \end{cases} \tag{9}$$

Based on this recursion, given the $(p-1)$-th order and $(p-2)$-th order moments, one can first compute $C_{T_k}^{p-1}$ and $C_{T_k}^{p-2}$ in $O(n)$ time in a bottom-up fashion. Then, m_k^p can be computed in a top-down fashion for all nodes in the interconnect tree in $O(n)$ time. Therefore, the moments up to the p-th order of an RLC tree can be computed in $O(np)$ time.

In the following, consider an RLC tree T_v rooted by node v. For any node w in T_v, let $C_{T_w}^p$ be the total p-th order moment weighted capacitance of T_w, and m_w^p be the p-th moment of node w. Now, we add a new edge uv at the root of T_v to obtain a new tree T_u rooted at u. Let $\overline{C}^p_{T_w}$ be the new total p-th order moment weighted capacitances of T_w for w in T_u. Similarly, let $\overline{m}^p_w$ be the new p-th moment of node w in T_u. Let R_v, L_v, and C_v be the total resistance, inductance, and capacitance of the edge uv, respectively. The following theorem illustrates how we can express $\overline{m}^p_w$ for $p \geq 0$ in terms of $\overline{m}^q_v$ and m^q_w for $q = 0 \ldots p$,

Theorem 1 *For root node u,*

$$\overline{m}^p_u = \begin{cases} 0 & \text{if } p = -1 \text{ or } p > 0 \\ 1 & \text{if } p = 0. \end{cases} \tag{4}$$

For $p \geq 0$,

$$\overline{C}^{p-1}_{T_v} = \overline{m}^{p-1}_v C_v + \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v C^q_{T_v} \tag{5}$$

$$\overline{m}^p_v = R_v \overline{C}^{p-1}_{T_v} - L_v \overline{C}^{p-2}_{T_v} \tag{6}$$

$$\overline{m}^p_w = \sum_{q=0}^{p} \overline{m}^{p-q}_v m^q_w \;\forall\; w \in T_v \tag{7}$$

Proof of Theorem 1: Eqn. (4) follows from Eqn. (9) directly. For $p = 0$, we can verify that Eqns. (5–7) are true using Eqn. (9). For $p = 1$, Eqns. (5–7) are trivially true since the first moment is equivalent to the Elmore delay, and one can easily verify that the Eqns. compute the Elmore delay. We will prove that the Eqns. hold for $p > 1$ by induction.

Assuming that the Eqns. are valid for $p = k > 1$. Now, consider $p = k+1$. By the definition of $\overline{C}^{p-1}_{T_v}$ given in [20],

$$\begin{aligned}
\overline{C}^{p-1}_{T_v} &= \overline{m}^{p-1}_v C_v + \sum_{w \in T_v, w \neq v} C_w \overline{m}^{p-1}_w \\
&= \overline{m}^{p-1}_v C_v + \sum_{w \in T_v, w \neq v} C_w \left\{ \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v m^q_w \right\} \\
&= \overline{m}^{p-1}_v C_v + \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v \sum_{w \in T_v, w \neq v} C_w m^q_w \\
&= \overline{m}^{p-1}_v C_v + \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v C^q_{T_v}
\end{aligned}$$

Therefore, it follows that Eqn. (6) is true, according to Eqn. (9) since $\overline{m}^p_u = 0$. Now, let w be a child node of v. We first prove that Eqn. (7) holds for w, and then prove that the Eqn. holds for all nodes in T_v by induction. From Eqn. (9),

$$\begin{aligned}
\overline{m}^p_w &= \overline{m}^p_v + R_w \overline{C}^{p-1}_{T_w} - L_w \overline{C}^{p-2}_{T_w} \\
&= \overline{m}^p_v m^0_w + R_w \sum_{k \in T_w} \overline{m}^{p-1}_k C_k - L_w \sum_{k \in T_w} \overline{m}^{p-2}_k C_k \\
&= \overline{m}^p_v m^0_w + R_w \sum_{k \in T_w} C_k \left\{ \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v m^q_k \right\} - L_w \sum_{k \in T_w} C_k \left\{ \sum_{q=0}^{p-2} \overline{m}^{p-2-q}_v m^q_k \right\} \\
&= \overline{m}^p_v m^0_w + \sum_{q=0}^{p-1} \overline{m}^{p-1-q}_v \left\{ R_w \sum_{k \in T_w} C_k m^q_k \right\} - \sum_{q=0}^{p-2} \overline{m}^{p-2-q}_v \left\{ L_w \sum_{k \in T_w} C_k m^q_k \right\} \\
&= \overline{m}^p_v m^0_w + \overline{m}^{p-1}_v R_w \sum_{k \in T_w} C_k + \sum_{q=1}^{p-1} \overline{m}^{p-1-q}_v \left\{ R_w \sum_{k \in T_w} C_k m^q_k \right\} - \sum_{q=0}^{p-2} \overline{m}^{p-2-q}_v \left\{ L_w \sum_{k \in T_w} C_k m^q_k \right\}
\end{aligned}$$

$$
\begin{aligned}
&= \overline{m}_v^p m_w^0 \,+\, \overline{m}_v^{p-1} m_w^1 \,+\, \sum_{q=1}^{p-1} \overline{m}_v^{p-1-q} \left\{ R_w C_{T_w}^q \,-\, L_w C_{T_w}^{q-1} \right\} \\
&= \overline{m}_v^p m_w^0 \,+\, \overline{m}_v^{p-1} m_w^1 \,+\, \sum_{q=1}^{p-1} \overline{m}_v^{p-1-q} m_w^{q+1} \\
&= \sum_{q=0}^{p} \overline{m}_v^{p-q} m_w^q
\end{aligned}
$$

We have shown that Eqn. (7) holds for all child nodes of v, i.e., the Eqn. is true for all descendant nodes one hop from v. Assume that the Eqn. holds for all descendant nodes h hops from v. We will prove that it also holds for all descendant nodes $h+1$ hops from v. Let w be a descendant node h hops from v, and k be a child node of w. From Eqn. (9),

$$
\begin{aligned}
\overline{m}_k^p &= \overline{m}_w^p \,+\, R_k \overline{C}_{T_k}^{p-1} \,-\, L_k \overline{C}_{T_k}^{p-2} \\
&= \sum_{q=0}^{p} \overline{m}_v^{p-q} m_w^q \,+\, R_k \sum_{j \in T_k} \overline{m}_j^{p-1} C_j \,-\, L_k \sum_{j \in T_k} \overline{m}_j^{p-2} C_j \\
&= \overline{m}_v^p \,+\, \sum_{q=1}^{p} \overline{m}_v^{p-q} m_w^q \,+\, R_k \sum_{j \in T_k} C_j \left\{ \sum_{q=0}^{p-1} \overline{m}_v^{p-1-q} m_j^q \right\} \,-\, L_k \sum_{j \in T_k} C_j \left\{ \sum_{q=0}^{p-2} \overline{m}_v^{p-2-q} m_j^q \right\} \\
&= \overline{m}_v^p m_k^0 \,+\, \sum_{q=1}^{p} \overline{m}_v^{p-q} m_w^q \,+\, \sum_{q=0}^{p-1} \overline{m}_v^{p-1-q} \left\{ R_k \sum_{j \in T_k} C_j m_j^q \right\} \,-\, \sum_{q=0}^{p-2} \overline{m}_v^{p-2-q} \left\{ L_k \sum_{k \in T_k} C_j m_j^q \right\} \\
&= \overline{m}_v^p m_k^0 \,+\, \overline{m}_v^{p-1} \left\{ m_w^1 \,+\, R_k \sum_{j \in T_k} C_j \right\} \\
&\quad + \sum_{q=2}^{p} \overline{m}_v^{p-q} m_w^q \,+\, \sum_{q=1}^{p-1} \overline{m}_v^{p-1-q} \left\{ R_k \sum_{j \in T_k} C_j m_j^q \right\} \,-\, \sum_{q=0}^{p-2} \overline{m}_v^{p-2-q} \left\{ L_k \sum_{j \in T_k} C_j m_j^q \right\} \\
&= \overline{m}_v^p m_k^0 \,+\, \overline{m}_v^{p-1} m_k^1 \,+\, \sum_{q=2}^{p} \overline{m}_v^{p-q} \left\{ m_w^q \,+\, R_k C_{T_k}^{q-1} \,-\, L_k C_{T_k}^{q-2} \right\} \\
&= \overline{m}_v^p m_k^0 \,+\, \overline{m}_v^{p-1} m_k^1 \,+\, \sum_{q=2}^{p} \overline{m}_v^{p-q} m_k^q \\
&= \sum_{q=0}^{p} \overline{m}_v^{p-q} m_k^q
\end{aligned}
$$

Therefore, by induction on the number of hops a descendant node is from v, Eqn. (7) holds for all nodes in T_v. Then, by induction on the order p, Eqns. (5–7) hold. □