Delay-Driven Rectilinear Steiner Tree Construction

Hongxi Wu, Xingquan Li, Liang Chen, Bei Yu, *Senior Member, IEEE*, and Wenxing Zhu

***Abstract*—Timing-driven routing is crucial in complex circuit design. Existing shallow-light Steiner tree construction methods balance between wire** length **(WL) and source-sink path** length **(PL) but lack in delay. Conversely, previous delay-driven methods prioritize delay but result in longer WL and PL, making them suboptimal. In this article, we show that simultaneously reducing the WL and PL can effectively reduce the delay. Furthermore, we investigate how delay changes during the reduction of PL. Guided by the theoretical findings, we develop a rectilinear shallow-light Steiner tree construction algorithm designed to reduce delay meanwhile maintaining a bounded WL. Furthermore, a delay-driven edge shifting algorithm is proposed to fine tune the tree's topology, further reducing delay. We show that our proposed edge shifting algorithm can return a local Pareto optimal solution when repeatedly applied. Experimental results show that our algorithm achieves the lowest total delay compared to previous methods while maintaining competitive WL. Moreover, for nets with pins that have timing information, our algorithm can generate the most suitable Steiner Tree based on the timing information. In addition, extended experiments highlight the positive impact of constructing rectilinear Steiner trees with minimized total delay. Our codes will be available at https://github.com/Whx97/Delay-driven-Steiner-Tree.**

***Index Terms*—Elmore delay, rectilinear Steiner tree, timing optimization.**

I. Introduction

With the scaling of VLSI technology, interconnection delay has emerged as a critical concern in the design of complex and high-performance circuits [5], [18]. Consequently, there has been a significant focus on timing-driven routing, aiming to minimize the average or maximum source-sink delay within given signal nets [9], [32]. With the growing importance of rectilinear Steiner trees in VLSI routing [27], developing delay-driven rectilinear Steiner trees becomes crucial, as it significantly supports the objective of timing-driven routing by strategically minimizing delays.

Received 28 April 2024; revised 8 September 2024; accepted 11 November 2024. Date of publication 18 November 2024; date of current version 23 April 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFA1011302; in part by the National Natural Science Foundation of China under Grant 62174033; in part by the Major Key Project of PCL under Grant PCL2023A03; and in part by the Natural Science Foundation of Fujian Province under Grant 2024J09045. This article was recommended by Associate Editor L. Behjat. *(Corresponding author: Wenxing Zhu.)*

Hongxi Wu and Wenxing Zhu are with the Center for Discrete Mathematics and Theoretical Computer Science, Fuzhou University, Fuzhou 350002, China (e-mail: wxzhu@fzu.edu.cn).

Xingquan Li is with the Department of Circuits and Systems, Peng Cheng Laboratory, Shenzhen 518000, China.

Liang Chen is with the School of Microelectronics, Shanghai University, Shanghai 201804, China.

Bei Yu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR, China.

This article has supplementary downloadable material available at https://doi.org/10.1109/TCAD.2024.3501932, provided by the authors.

Digital Object Identifier 10.1109/TCAD.2024.3501932

The Elmore delay in an RC tree [21], [29], [31] is computationally efficient for approximating signal delay. Its reliability in predicting physical (SPICE-computed) delay has been demonstrated in supporting investigations in [8] (i.e., near-optimal Elmore delay implies near-optimal SPICE delay). Given the advantages, the objective of this article is to commence a delay-driven process for constructing a rectilinear Steiner tree by simultaneously considering wire length (WL), source-sink path length (PL), and Elmore delay.

Earlier work implicitly equated an optimal routing tree (ORT) with the rectilinear Steiner minimum tree (RSMT) [19], [27]. However, it has been recognized that one cannot simply use the RSMT for timing-driven routing [2], [15], [16], since routing using the RSMT typically results in longer source-sink PL, which may increase the delay. Moreover, it is not feasible for each sink to use a shortest path tree that satisfies the optimal source-sink PL for routing, since such an approach might lead to an increase in WL, thereby exacerbating the delay. Therefore, constructing routing trees with balanced WL and PL becomes a crucial technique for timing-driven routing.

A theoretical resolution is the construction of shallow-light Steiner tree, i.e., short Steiner tree with bounded source-sink PL [4], [20], [24], [25]. This approach typically starts from a preconstructed short Steiner tree and utilizes a parameter $\varepsilon \geq 1$ to generate a new tree, in which the PL from the source to each sink in the tree is at most ε times the source-sink Manhattan distance. Prim–Dijkstra (PD) algorithm [3] generates a spanning tree effectively, meanwhile balances both WL and PL by combining the classical Prim [28] and Dijkstra [17] algorithms. Thanks to its simplicity and efficiency, PD has gained popularity in the field of semiconductor design and electronic design automation for over two decades, and was further improved in [2] to alleviate the problem of suboptimal tradeoff between WL and PL that may be encountered.

Recently, a provably good routing topology of shallow-light Steiner tree was developed in SALT [11], which offers an excellent tradeoff between both WL and PL, compared to other state-of-the-art shallow-light Steiner tree construction methods. However, although SALT obtains a good tradeoff between WL and PL, it performs less satisfactorily on delay. This observation is supported by the results presented in [11], where KRY [25] produces a Steiner tree with lower delay despite not achieving a balance between WL and PL as optimal as SALT does.

Another category of research focuses on minimizing delay directly when generating a spanning tree. Cong et al. [14] and Cohoon and Randall [13] are the first to propose performance-oriented routing techniques utilizing a linear

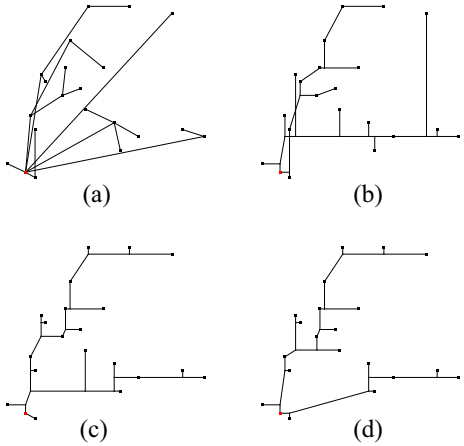


Fig. 1. Comparison of different constructions for a 21-pin net: (a) ERT [7] with WL = 756770, Avg. PL = 78393, and Avg. normalized Elmore delay of 1.199; (b) SERT [7] with WL = 424270, Avg. PL = 72542, and Avg. normalized Elmore delay of 0.986; (c) SALT [11] with WL = 342180, Avg. PL = 71858, and Avg. normalized Elmore delay of 0.951; and (d) a tree with the best delay, WL = 348650, Avg. PL = 71858, and Avg. normalized Elmore delay of 0.877.

delay model. Although the linear delay model offers convenience in terms of calculation, it was widely recognized that it may not accurately capture the delay behavior of modern integrated circuits [10]. Boese et al. [7] developed a heuristic routing algorithm that minimizes delay based on Elmore delay. This algorithm comes in two variants: 1) Steiner point-based Elmore delay routing (SERT) and 2) non-Steiner point-based Elmore delay routing (ERT). However, the delay performance of these methods can be further optimized, since they greedily insert edges to a tree, which may also lead to longer WL and source-sink PL.

Fig. 1 gives an example of topologies obtained by different constructions and various metrics of these topologies. From the figure, it is apparent that methods directly minimizing Elmore delay [Fig. 1(a) and (b)] cannot obtain optimal delay, because they produce longer WLs. The shallow-light method [Fig. 1(c)] balances WL and source-sink PL well, but does not perform as well in terms of delay. Fig. 1(d) shows the best delay result, although the WL is slightly longer than that of Fig. 1(c).

Building upon the aforementioned observations and routing methodologies, we propose a heuristic algorithm designed for the construction of a rectilinear Steiner tree with the primary objective of minimizing the Elmore delay while maintaining competitive WL. The main contributions of this work are summarized as follows.

1) We investigate the impact of WL variations resulting from PL optimization on delay and propose a novel delay-driven tree construction process. This includes the shallow-light tree construction and delay-driven refinement to reduce delay while controlling WL.
2) Applying our theoretical findings on the optimal PL optimization method for delay, the proposed shallow-light algorithm employs edge replacement to actively optimize PL, aiming to minimize delay without excessively increasing WL. We prove that our algorithm can output a tree with bounded WL.

TABLE I
NOTATIONS

Notation	Definitions
V	Set of pins in a signal net, $V = \{n_0, n_1, \ldots, n_k\}$ having one source and k sinks.
n_0	Source of a signal net, which is the root of a tree.
T	Routing tree $T = (V, E)$.
$w(e)$	Wire length of an edge $e \in E$.
$path(u, v)$	The unique path from u to v in a tree T.
$pl(u)$	The unique path length from n_0 to u in a tree T.
$dist(u, v)$	The Manhattan distance between u and v.
$d[u]$	Current estimated path length from n_0 to u.
r_d	Output driver resistance at the source of a signal net.
$T(a)$	The subtree rooted at node a under tree T.
C_a	Total capacitance of subtree $T(a)$, namely, the sum of sink and edge capacitances in $T(a)$.
$length(T)$	The total wire length of tree T, i.e., $\sum_{e \in E} w(e)$.
$d_T(n_i)$	The Elmore delay from n_0 to sink n_i under tree T.
$d_T(a \to b)$	The Elmore delay from a to b under tree T.

3) To improve the limited delay performance of the proposed shallow-light algorithm due to not considering capacitance, the proposed delay-driven refinement modifies the tree's topology locally. This approach optimizes the distribution of load capacitance and achieves a local Pareto optimal solution, when repeatedly applied.
4) Experimental results show that, our algorithm achieves an improvement of average delay by 5.87% and a reduction of WL by 20.23% on 48K high-fanout nets, compared to the previous method with the lowest delay. Moreover, our algorithm offers flexibility in balancing WL and delay priorities through adjustable parameters and search spaces.

The remainder of this article is organized as follows. Section II introduces the Elmore delay model and gives the problem formulation. Moreover, several key findings related to delay reduction are presented. Section III gives our proposed approach for delay-driven rectilinear Steiner tree construction. Experimental results and analysis are presented in Section IV. Finally, Section V concludes this work.

II. Elmore Delay Model, Delay-Driven RSMT Problem, and Delay Reduction Theorems

This section analyzes the Elmore delay model and gives the delay-driven RSMT problem formulation. Moreover, we provide several key findings related to Elmore delay reduction. All notations used in this article are listed in Table I.

A. Elmore Delay Model

The Elmore delay model [21] is used to approximate the distributed RC delay, it is defined as follows. Given a routing tree T rooted at the source n_o, the Elmore delay $d_T(n_i)$ at sink n_i is

$$\begin{aligned} d_T(n_i) &= r_d C_{n_o} + \sum_{e_v \in path(n_0, n_i)} r_{e_v}\left(\frac{c_{e_v}}{2} + C_v\right) \\ &= r_d C_{n_o} + \sum_{e_v \in \text{path}(n_0, n_i)} r_{e_v}\frac{c_{e_v}}{2} + \sum_{e_v \in \text{path}(n_0, n_i)} r_{e_v} C_v \quad (1) \end{aligned}$$

where $r_{e_v} = R_0 w(e_v)$, $c_{e_v} = C_0 w(e_v)$, and R_0 and C_0 are the unit length wire resistance and capacitance, respectively.

A closer look at the final equality in (1) reveals the following facts.

1) The first term $r_d C_{n_o}$ can be minimized when length(T) is minimum. Therefore, when r_d is large enough, an ORT is an RSMT [16].
2) The second term implies that $d_T(n_i)$ has a quadratic relationship to the length of the path path(n_0, n_i). Therefore, an ORT tends to be a shortest path tree when minimizing this term only [16].
3) The cumulative sum of the third term implies that, of all the nodes on the path(n_0, n_i), the subtree capacitance $(C_v \ \forall v \in \text{path}(n_0, n_i))$ of these nodes should be as small as possible for minimizing the delay. This implies that the number of Steiner points on the path(n_0, n_i) should be minimized.

The above observations suggest that, the Elmore delay model is a relatively complex objective function for minimizing delay since it involves optimizing WL, source-sink PL, and the distribution of Steiner points. In the next section, we define the research problems of this work.

B. Problem Definition

In view of the facts stated in Section II-A, and the fact that minimizing the delay directly may introduce suboptimal delays (see Fig. 1), we propose to address the following problem to construct a topology with minimal Elmore delay.

Problem 1 [Minimum Elmore Delay Rectilinear Steiner Tree Problem (MD-RST)]:

Input: A signal net $V = \{n_0, n_1, \ldots, n_k\}$, where n_0 is the source with a position p_0 and resistance r_d, $V\backslash\{n_0\}$ is the set of sinks n_i with associated position p_i and capacitance c_i. Unit length wire resistance R_0 and capacitance C_0.

Output: A directed rectilinear Steiner tree T rooted at n_0 with minimal $f(T) = (f_1(T), f_2(T)) \in \mathbb{R}^2$, where $f_1(T) = \text{length}(T)$ and $f_2(T) = \sum_{n_i \in V\backslash\{n_0\}} d_T(n_i)$. Since this work focuses on the delay-driven rectilinear Steiner tree construction, we focus mainly on minimizing $f_2(T)$.

Problem 1 primarily aims at optimizing the overall source-sink delay. However, this approach can sometimes fall short because timing-driven routing must also consider the timing criticality of individual pins. In timing-driven optimization tasks, the goal is to minimize the latest actual arrival time (or maximize the earliest required arrival time) for each pin [6]. To achieve this, pins with higher timing criticality should be prioritized to ensure shorter delays. Hence, we pose the following problem.

Problem 2 [Critical-Sink Rectilinear Steiner Tree Problem (CS-RST)]:

Input: A signal net $V = \{n_0, n_1, \ldots, n_k\}$, where n_0 is the source with a position p_0 and resistance r_d, $V\backslash\{n_0\}$ is the set of sinks with each sink having an associated position p_i, capacitance c_i and timing slack s_{n_i}. Unit length wire resistance R_0 and capacitance C_0.

Output: A directed rectilinear Steiner tree T rooted at n_0 with minimal $f(T) = (f_1(T), f_2(T)) \in \mathbb{R}^2$, where $f_1(T) = \text{length}(T)$, and

$$f_2(T) = \sum_{n \in V\backslash\{n_0\}} \max\left(0, d_T(n) - s_n\right). \tag{2}$$

This work focuses on the delay-driven rectilinear Steiner tree construction, with the primary goal of minimizing $f_2(T)$. For the objective function $f_2(T)$ in (2), we offer the following explanation. Consider a sink v experiencing a timing violation, indicated by a negative slack s_v. Since the delay $d_T(v)$ is always positive, $d_T(v) - s_v > 0$ holds. Therefore, to minimize $f_2(T)$, the delay $d_T(v)$ should be reduced as much as possible, ideally approaching zero. Furthermore, if there are two timing violation sinks v_1 and v_2, where s_{v_1} is significantly less than s_{v_2}, indicating a more severe violation at v_1, the algorithm should prioritize minimizing the delay $d_T(v_1)$. This prioritization ensures that the sink with the most negative slack receives the highest priority in the delay optimization process.

In fact, the MD-RST problem represents a generic form applicable to various scenarios. On the other hand, the CS-RST problem becomes more significant when specific timing information related to a design is available. Next, we prove the hardness result of the two problems.

Theorem 1: Both MD-RST and CS-RST problems are NP-hard.

Proof:

1) Minimizing the term length(T) is NP-hard, as it corresponds to the RSMT problem [22]. On the other hand, the problem of minimizing the term $\sum_{n \in V\backslash\{n_0\}} d_T(n)$ has also been proven NP-hard [30]. Hence, the MD-RST problem is NP-hard.
2) Consider the case that the timing slack of all sinks in a net $V = \{n_0, n_1, \ldots, n_k\}$ is 0, i.e., $s_n = 0, n \in V\backslash\{n_0\}$. Then, the objective function of the CS-RST problem becomes

$$f(T) = \left(\text{length}(T), \sum_{n \in V\backslash\{n_0\}} \max\left(0, d_T(n)\right)\right).$$

Since the delay $d_T(n) > 0$ naturally holds, minimizing $\sum_{n \in V\backslash\{n_0\}} \max\left(0, d_T(n)\right)$ and $\sum_{n \in V\backslash\{n_0\}} d_T(n)$ are equivalent. Thus, the CS-RST problem can be reduced to the MD-RST problem. So the MD-RST problem is NP-hard. ■

Given that the MD-RST and CS-RST problems are NP-hard and identifying Pareto optimal solutions is challenging, this work focuses on discovering local Pareto optimal solutions. To this end, we provide the following definitions.

Definition 1 (Edge Shifting): Edge shifting refers to the operation in a tree, for which a given node n is disconnected from its current parent node and reconnected to a created new node n' on the bounding box of a given edge $e = (m, m')$. Then, disconnect (m, m') and connect (m, n') and (n', m').

Definition 2 (Pareto Dominance): For two trees T_1 and T_2, if $f_i(T_1) \leq f_i(T_2)$ for all $i = 1, 2$ and $f_i(T_1) < f_i(T_2)$ holds for at least one $i \in \{1, 2\}$, then we say that the tree T_1 Pareto dominates the tree T_2.

Definition 3 (Local Pareto Optimum): A tree T is local Pareto optimal, if it is not dominated by any other trees constructed from T by edge shifting.

In the next section, we show several theoretical results regarding delay reduction when using the Elmore delay model.

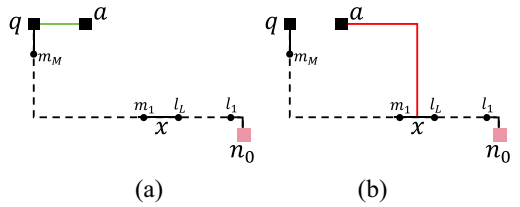


Fig. 2. (a) Initial tree T. (b) Optimized tree T'. There are several nodes on the dotted lines, and there are subtrees rooted at these nodes.

C. Theoretical Results on Delay Reduction

For mathematical simplicity, we normalize the unit wire resistance and the unit wire capacitance so that they are both equal to one. In the following proofs of theorems, C_a, path(n_o, a), and $pl(a)$ are all defined under the initial Steiner tree T, unless otherwise stated.

Previous literature [15], [16] has suggested that WL and PL should be balanced to optimize the delay of a rectilinear Steiner tree. However, these studies did not explicitly identify the combined effect of WL and PL on the delay. Moreover, shallow-light tree algorithms [11], [25] mainly focus on minimizing the WL with the desired source-sink PL. Nonetheless, existing shallow-light tree algorithms exhibit either good WL but less satisfaction on delay [11] or good delay but long WL [25] in the experiments.

In fact, during reducing the source-sink PL in a Steiner tree, the WL of the tree may increase or decrease. This change in WL, in turn, may significantly impact the total delay. Therefore, it is important to investigate how variation in WL, resulting from the optimization of PL, influences the total delay. For example, Fig. 2(a) gives an RSMT T, and a node a in T, where $pl(a) > \text{dist}(n_0, a)$. In the initial tree T, the parent of node a is q. In the optimized tree T' [see Fig. 2(b)], the edge (q, a) is disconnected, and then a is reconnected to another node x on the path(n_0, a) to optimize the PL from n_0 to a.

For Fig. 2, we give the following notations. Suppose there are nodes $l_1, \ldots, l_L$ on path(n_0, x), and nodes $m_1, \ldots, m_M$ on path(x, q). For convenience, we further define $l_1, l_2, \ldots, l_L$ to also represent the lengths of the edges $(n_0, l_1), (l_1, l_2), \ldots, (l_{L-1}, l_L)$ and define similarly for $m_1, \ldots, m_M$. Suppose there are $l'_1, \ldots, l'_L, m'_1, \ldots, m'_M, q'$, and a' sinks in the optimized subtrees $T'(l_1), \ldots, T'(l_L), T'(m_1), \ldots, T'(m_M), T'(q)$, and $T'(a)$, respectively.

In the subsequent discussion, we elucidate how variation in WL, resulting from the optimization of PL, influences the total delay. We have the following lemma.

Lemma 1: Let $\Delta WL = \text{dist}(a, x) - (pl(a) - pl(q))$ and Δd_v denote the amount of change in WL and delay that results from reducing the PL from n_0 to node a, respectively. We have

$$\Delta d_a = (r_d + pl(x))\Delta WL + \text{dist}(a, x)\left(\frac{\text{dist}(a, x)}{2} + C_a\right) - d_T(x \to a) \quad (3)$$

$$\Delta d_{l_j} = \left(r_d + l_1 + \cdots + l_j\right)\Delta WL, \quad l_j \in \text{path}(n_0, x) \quad (4)$$

$$\Delta d_{m_j} = (r_d + pl(x))\Delta WL - \left(pl(m_j) - pl(x)\right) ((pl(a) - pl(q)) + C_a), \quad m_j \in \text{path}(x, q) \quad (5)$$

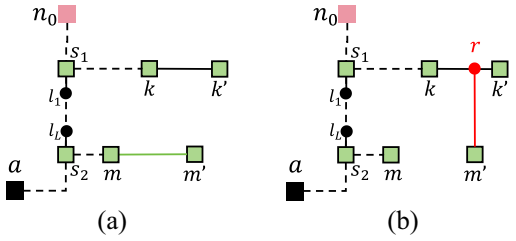


Fig. 3. (a) Initial tree T. (b) Optimized tree T'. There are several nodes on the dotted lines, and others are subtrees rooted at these nodes.

$$\Delta d_q = (r_d + pl(x))\Delta WL - (pl(q) - pl(x)) ((pl(a) - pl(q)) + C_a). \quad (6)$$

Moreover, if

$$\sum_{l_j \in \text{path}(n_0, x)} l'_j \Delta d_{l_j} + \sum_{m_j \in \text{path}(x, q)} m'_j \Delta d_{m_j} + q' \Delta d_q + a' \Delta d_a < 0$$

then reducing the PL from n_0 to node a leads to a lower total delay.

Proof: The proof is provided in Supplementary Material due to limited space. ■

Next, by applying Lemma 1 we show that, reducing the PL of a Steiner tree can reduce the total delay of the tree, if the WL is reduced or kept unchanged.

Theorem 2: When reducing the source-sink PL reduces or keeps unchanged the WL of a Steiner tree, the total delay will be reduced.

Proof: See Appendix A. ■

Moreover, given a node a in a tree, we show that the delay from n_0 to a can be optimized by shifting the load capacitance on the path(n_0, a). This insight will inspire us to optimize the critical delay by using an appropriate edge shifting strategy.

Fig. 3(a) gives a structure of RSMT T, in which a node a in T will be optimized for the delay. s_1 and s_2 are nodes on path(n_0, a). Suppose there is an edge (m, m') within subtree $T(s_2)$ that is not located on the path(s_2, a). This edge is designated for disconnection. Subsequently, the node m' will be reconnected to subtree $T(s_1)$. Specifically, node m' will be reconnected to an edge (k, k'), which is a part of $T(s_1)$ but not on path(s_1, a). In the optimized tree T' [Fig. 3(b)], m' is reconnected to node r, which is a created new node on the bounding box of the edge (k, k'). Then, disconnect (k, k') and connect (k, r) and (r, k').

In the subsequent discussion, we will show that when $(\text{dist}(m, m') + \text{dist}(k, k')) \geq (\text{dist}(r, m') + \text{dist}(k, r) + \text{dist}(r, k'))$, i.e., $\text{length}(T) \geq \text{length}(T')$, the delay from n_0 to a can be reduced.

Theorem 3: Consider the structure shown in Fig. 3(a). After performing edge shifting on the node m', if $\text{dist}(m, m') + \text{dist}(k, k') \geq \text{dist}(r, m') + \text{dist}(k, r) + \text{dist}(r, k')$, then the delay from n_0 to a can be reduced.

Proof: See Appendix B. ■

Furthermore, given a node a in a tree, we show how the total delay of the tree changes when the parent of node a is reset to an existing node or a newly created node on path(n_0, a) to reduce the PL from the source n_0 to a.

Theorem 4: Suppose b is a node on the path(n_0, a) such that $pl(b) + \text{dist}(a, b) = \text{dist}(n_0, a)$, and dist$(a, b)$ is minimized. When reducing the PL between n_0 and node a along the path(n_0, b), the optimal total delay can be achieved when the parent node of a is b or n_0.

Proof: See Appendix C. ■

Lemma 1 shows that, reducing source-sink PL is still possible to achieve lower delay, even if it leads to an increase in WL. This finding helps explain why KRY [25] achieves lower delay compared to SALT [11], despite with potentially longer WL. Theorem 3 shows that it is necessary to locally refine a tree for further optimizing delay. Furthermore, Theorem 4 will inspire us to optimize the delay by determining an appropriate strategy to reduce the PL.

III. Delay-Driven Rectilinear Steiner Tree Construction

The challenge addressed in this work focuses on two objectives: 1) minimizing WL and 2) delay, with a predominant emphasis on delay minimization. In addition, optimizing the delay involves also optimizing implicitly the WL, therefore, we propose the following optimization process. Our approach begins with an RSMT with minimized WL. Subsequently, iterative edge replacement is utilized to shorten the source-sink PL and improve delay, guided by the theoretical insights detailed in Section II-C. Finally, we apply local edge shifting to refine the tree and enhance delay reduction.

A. Constructing Rectilinear Shallow-Light Steiner Tree Based on Edge Replacement

As mentioned earlier, existing shallow-light tree algorithms exhibit either good WL but less satisfaction on delay [11], or good delay but long WL [25] in the experiments. To address this issue, we propose a new shallow-light tree algorithm that constructs a rectilinear shallow-light Steiner tree to reduce delay while limiting the increase in WL. This approach is based on the theoretical findings of Theorem 4.

The conventional shallow-light tree constructions focus on bounding the shallowness and lightness to optimize the tree cost (e.g., delay) [25]. *Lightness* η of a tree T of a net indicates that the WL of the tree T is at most η times the WL of its minimum spanning tree (MST), i.e., length$(T) \leq \eta \cdot \text{length}(MST)$. A tree has *shallowness* ε if the PL from the source to each sink in the tree T is at most ε times the source-sink Manhattan distance, i.e., $\max\{[(pl(v_i))/(\text{dist}(n_0, v_i))] | v_i \in V \backslash \{n_0\}\} \leq \varepsilon$.

Algorithm 1 gives the pseudocode of the rectilinear shallow-light Steiner tree based on the edge replacement (RSLT-ER) algorithm. Essentially, the basic RSLT-ER algorithm starts with a rectilinear Steiner minimal tree by using the heuristic FLUTE algorithm [12] (line 2), and continues iteratively searching for the edge replacements that result in a tree with a desired shallowness ε.

First, RSLT-ER identifies some breakpoints on the RSMT (line 3), based on performing a depth-first search (Function DFS on line 8) on the RSMT. As the DFS progresses, if a node v violates the shallowness constraint, it is identified as a breakpoint (lines 9 and 10). Subsequently, its estimated PL $d[v]$ is relaxed to dist(n_0, v) to adjust the estimated PLs of its child nodes.

Algorithm 1: RSLT-ER

Input: Nodes V on Manhattan plane, root n_0, parameter $\varepsilon \geq 1$;
Output: Rectilinear shallow-light Steiner tree with shallowness ε.

1 Initialize (Breakpoints $B \leftarrow \emptyset, d[v] = 0\ \forall v \in V$);
2 $T_M \leftarrow$ rectilinear Steiner minimal tree using FLUTE;
3 $B \leftarrow$ perform DFS(n_0, T_M) to find all breakpoints;
4 **for** $n \in B$ **do**
5 $\quad m_M \leftarrow$ closest node to b on $path(n_0, b)$, where b is a node on the $path(n_0, n)$ such that $pl(b) + dist(b, n) = dist(n_0, n)$, and $dist(n, b)$ is minimized;
6 $\quad$ Find the best removed edge (r_1, r_2) on the $path(m_M, n)$;
7 $\quad$ Add edge (m_M, n), remove edge (r_1, r_2);
8 **Function** DFS(v, T)**:**
9 $\quad$ **if** $d[v] > \varepsilon \cdot dist(n_0, v)$ **then**
10 $\quad\quad B \leftarrow B \cup \{v\}$;
11 $\quad\quad d[v] \leftarrow dist(n_0, v)$;
12 $\quad$ **foreach** *child* u *of* v *in* T **do**
13 $\quad\quad d[u] \leftarrow d[v] + dist(u, v)$;
14 $\quad\quad$ DFS(u, T);
15 $\quad$ Return B;
16 **End Function**

The next step is to determine the reconnected node on the path(n_0, n) for a given breakpoint n. According to Theorem 4, the best candidate for reconnection is either node b (b is a node on the path(n_0, a) such that $pl(b) + \text{dist}(a, b) = \text{dist}(n_0, a)$, and dist$(a, b)$ is minimized) or source node n_0. However, we have chosen to reconnect at node m_M, which is the closest node to b on path(n_0, b) (see Fig. 10). This decision is based on three key considerations as follows.

1) *Avoiding Additional Steiner Points:* By selecting m_M, we avoid introducing a new Steiner point, which could increase the load capacitance on other paths and potentially lead to a higher total delay in the subsequent steps.
2) *Minimal Delay Difference:* If b is the optimal reconnection point, the delay difference will be minimal due to the close proximity of m_M to b.
3) *Controlling WL:* If n_0 is the optimal node, connecting to either m_M or b will result in a similar delay, while also preventing a significant increase in WL that would occur if we connect directly to n_0.

Moreover, this approach allows us to maintain a balance between delay minimization and WL control without the need for additional calculations to identify the absolute best node for reconnection.

Finally, we find the best-removed edge on path(m_M, n), because the added edge (m_M, n) will lead to a cycle in the tree. We traverse each edge of path(m_M, n) to find an edge e' such

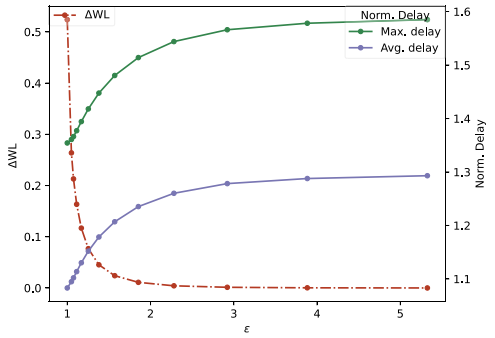


Fig. 4. "ΔWL" in the left axis represents the amounts of increase in normalized WL before and after executing RSLT-ER algorithm, at different values of parameter ε. The right axis represents normalized delay at different values of parameter ε.

that the following conditions are satisfied: 1) after removing e' and adding (m_M, n), it does not result in node $\hat{n}$ ($\hat{n} \in V \backslash B$) violating the shallowness constraint and 2) $w(e')$ is maximized (line 6).

Since the relaxation step (line 11 in Algorithm 1) underestimates the PLs of the child nodes, lines 3–7 of Algorithm 1 need to be executed multiple times until there are no breakpoints. Next, we demonstrate that the RSLT-ER algorithm returns a tree with bounded WL.

Theorem 5: Consider an initial tree T and the parameter ε. If $\varepsilon = 1$, then the RSLT-ER algorithm returns a tree T' with $\text{length}(T') \leq \sum_{n_i \in V \backslash \{n_0\}} \text{dist}(n_0, n_i)$; if $\varepsilon > 1$, then the RSLT-ER algorithm returns a tree T' with $\text{length}(T') < (1 + (2/[\varepsilon - 1]))\text{length}(T)$.

Proof: The proof is provided in Supplementary Material due to limited space. ■

Specifically, we use 1294K nets from the ICCAD Contest benchmarks [26] for experiments, and use the FLUTE [12] algorithm as a reference. Fig. 4 presents the relationship between the amount of increase in normalized WL and normalized delay before and after executing our RSLT-ER algorithm, at different values of parameter ε. It shows the extent of delay reduction by our RSLT-ER algorithm, highlighting the algorithm's effectiveness in optimizing delay. Also, the figure indicates that the WL is increased by at most a factor of 0.524 than FLUTE.

B. Delay-Driven Edge Shifting

Shallow-light algorithms (e.g., RSLT-ER, SALT [11], and KRY [25]) mainly focus on the tradeoff between WL and source-sink PL while disregarding the load capacitance on the pins and the capacitance on the edges. However, Theorem 3 shows that it is necessary to locally refine a tree for optimizing delay. Hence, in this section, we introduce delay-driven edge shifting (DDES), which aims to consider the load capacitance on pins and the capacitance on the edges, and refine the tree through edge shifting to further reduce delay.

Algorithm 2 gives the pseudocode of the proposed DDES. It is specifically designed to address the MD-RST problem. With minor modifications, this algorithm can also be adapted

Algorithm 2: DDES

Input: Tree $T(V, E)$;
Output: Refined tree T'.
1 Compute $slack(T(v_i))$ for $v_i \in V$;
2 Query candidate edges for V by R-tree;
3 **for** $n_i \in V$ **do**
4 **foreach** *candidate edge* (v_j, v_j') **do**
5 Continue if $v_j \in T(v_i)$;
6 $v_j'' \leftarrow$ closest node to v_i within the bounding box of edge (v_j, v_j');
7 Candidate nodes $CN \leftarrow \{v_j, v_j', v_j''\}$;
8 Find node $c \in CN$ such that the following conditions are satisfied: 1) the total delay is reduced the most; 2) the obtained total delay is lower than the total delay of the previous step; and 3) satisfies $\Delta pl = pl(c) + \text{dist}(c, v_i) - pl(v_i) \leq slack(T(v_i))$;
9 **if** $c \neq \{\}$ **then**
10 Disconnect (v_i, v_i'), connect (c, v_i), (c, v_j) and (v_j', c);
11 Update $slack(T(u))$ for $u \in T(v_i)$;

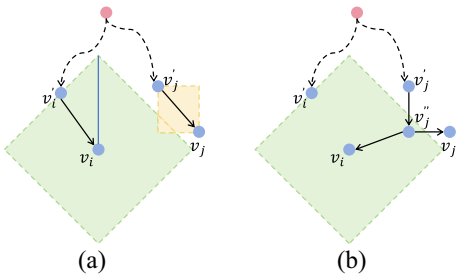


Fig. 5. (a) R-tree query by a diamond-shaped box, edge (v_j', v_j) is a candidate edge of edge (v_i', v_i). (b) Edge shifting.

to solve the CS-RST problem, which will be discussed immediately following the description of the algorithm. First, we reduce the search space by applying shallowness constraints to reduce the runtime. In order to efficiently check whether edge shifting causes nodes to violate the shallowness constraint, we precompute the slack $\text{slack}(v_i)$ of each node v_i and $\text{slack}(T(v_i))$ of the subtree $T(v_i)$ (line 1)

$$\text{slack}(v_i) = \varepsilon \cdot \text{dist}(n_0, v_i) - \text{pl}(v_i) \tag{7}$$

$$\text{slack}(T(v_i)) = \min_{t \in T(v_i)} \text{slack}(t). \tag{8}$$

Equation (7) calculates the remaining amount of PL that the node can add while adhering to the shallowness constraint. To ensure $pl(t) \leq \varepsilon \cdot \text{dist}(n_0, t)$ $(\forall t \in T(v_i))$ holds, (8) calculates the maximum remaining PL that can be added from n_0 to v_i. In this way, if a target node v_i increases its PL by Δpl resulting from a potential edge shifting, then the legality of the edge shifting means $\Delta pl \leq \text{slack}(T(v_i))$.

Then, we identify candidate edges for $\forall v_i \in V$ by R-tree [23] (line 2). In [11], R-tree is used to find suitable edge substitutions to further minimize the WL while ensuring the

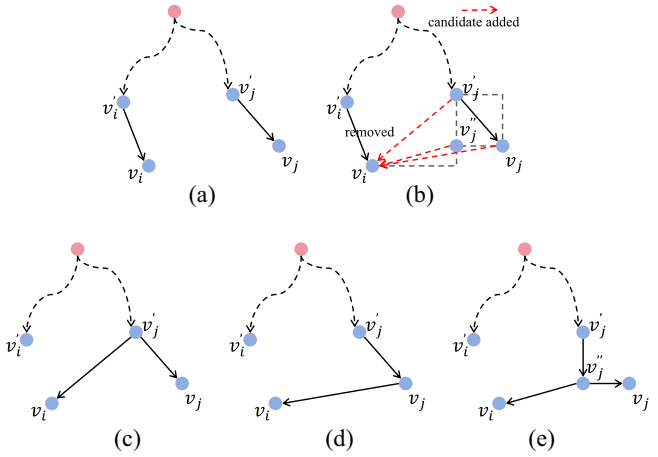


Fig. 6. (a) Before DDES, v_i is the *target node*, v_i' is v_i's parent node, and (v_j, v_j') is a candidate edge. (b) v_j'' is the closest node to n_i within the bounding box of edge (v_j, v_j') and the red dotted lines are candidate edges to be added. (c)–(e) After DDES, edge (v_i, v_i') is shifted to the candidate edge.

PL remains unchanged. This idea can be adapted to construct a delay-driven tree by not only focusing on WL but also taking delay into account. Specifically, R-tree stores the bounding box of every edge of a tree. Subsequently, each edge is processed through a diamond-shaped query box to identify its candidate edges. An edge (v_j', v_j) becomes a candidate of the edge (v_i', v_i) if (v_j', v_j)'s bounding box intersects with the diamond-shaped query box of (v_i', v_i) [see Fig. 5(a)]. Among all the candidate edges of a node v_i, we find the optimal edge shifting scheme from three different candidate edges (red dotted lines in Fig. 6(b) and lines 6–8 in Algorithm 2). Note that, a legal candidate edge (v_j', v_j) cannot be in $T(v_i)$ (line 5), otherwise it will make the tree disconnected.

To guarantee the final effectiveness of the DDES, Algorithm 2 should be run twice in different modes. The first run computes the delay improvement based on the input topology T, while the second run processes the edge shifting in descending order of improvement and takes the edge shifting that is still legal and reduces the total delay.

Next, we outline the application of the DDES algorithm to address the CS-RST problem. To adapt the DDES algorithm for addressing the CS-RST problem, it is sufficient to modify line 8 in Algorithm 2 to align with the objectives specific to CS-RST. That is, modifying the delay object $D(T')$ as

$$D(T') = \sum_{\substack{n_i \in V \setminus \{n_0\} \\ s_{n_i} < 0}} \text{dist}(n_0, n_i) \frac{-s_{n_i}^2}{d_{T'}(n_i)}.$$

To reduce the delay without increasing the WL, we propose DDES-S, which is a variant of the DDES algorithm. In DDES-S, only v_j'' (i.e., $CN \leftarrow \{v_j''\}$) is considered to be the candidate node in line 7 of Algorithm 2, compared with the DDES algorithm. Next, we provide the local Pareto optimality proof of the DDES-S algorithm.

Theorem 6: Consider an initial tree T, repeatedly applying the DDES-S algorithm returns a tree T' which is a local Pareto optimal solution.

Proof: See Appendix D. ■

TABLE II
NET STATISTICS OF BENCHMARK DESIGNS

	small	medium	large	huge
$\lvert V \rvert$	4-7	8-15	16-31	≥ 32
#Nets	799587	253133	193773	48048

IV. EXPERIMENTAL RESULTS

This section presents experimental results to demonstrate the effectiveness of our proposed algorithm. The experimental setup is detailed in Section IV-A. Performance comparisons of our algorithm with others in solving the MD-RST and CS-RST problems are provided in Sections IV-B and IV-F, respectively. Section IV-C verifies the effectiveness of our proposed DDES algorithm. Section IV-D demonstrates the effectiveness of our proposed algorithmic flow. Section IV-E discusses our algorithm's ability to balance WL and Elmore delay. Finally, experiments on real netlists are presented in Section IV-G.

A. Experimental Setup

We use C++ programming language to implement the algorithms given in Section III. Experiments are performed on a Linux server with the Intel Xeon Platinum 8380 CPU in single-threaded mode. Benchmarks of the ICCAD 2015 Contest [26] are used for a comprehensive evaluation and comparison. Given that finding an optimal solution with optimal WL and PL is straightforward for nets with only two or three pins, our experiments primarily concentrate on nets with pins greater than three. Approximately 1294K total nets are divided into four groups (i.e., small, medium, large, and huge) according to the number of pins. The statistical information is presented in Table II.

B. Superior Performance on the MD-RST Problem

Our algorithmic flow performs the RSLT-ER algorithm (Algorithm 1) first and then the DDES algorithm (Algorithm 2), denoted as RSLT-ER+DDES. To see the delay improvement of the constructed rectilinear Steiner tree by our algorithm, we compare it with the RSLT-ER, DDES-S, SALT [11], KRY [25], ERT [7], SERT [7], SALT-1 [11], FLUTE [12], and SALT [11]+DDES. SALT-1 is obtained by setting shallowness as 1 in SALT. We further construct the RSLT-ER+DDES-S and SALT [11]+DDES-S for experimental comparison.

Table III lists the average results of the normalized WL (WL), maximum normalized PL (max_PL), maximum normalized delay (max_Delay), and average normalized delay (avg_Delay), of respective algorithms on the four groups of nets. The normalization for WL is by [(length(T))/(length(FLUTE))], while the normalization for PL is by $[(pl(n_i))/(\text{dist}(n_0, n_i))]$. The normalization for delay is by $[(d_T(n_i))/(\max_{n_i \in V \setminus \{n_0\}} lb(n_i))]$, where $lb(n_i)$ is the lower bound of delay from [30]. For SALT, KRY, RSLT-ER, SALT+DDES, SALT+DDES-S, RSLT-ER+DDES, and RSLT-ER+DDES-S, we report the results with the lowest average delay for 13 values of ε from 1 to 5.32488 (mainly by the geometric sequence $0.05 \times 1.5^n, n = 0, 1, \ldots, 11$).

TABLE III
AVERAGE RESULTS. FOR MAXIMUM AND AVERAGE DELAY, WE MARK THE TOP THREE IN RED, GREEN, AND BROWN, RESPECTIVELY

$\|V\|$	Obj.	Algorithms										
		shallow-light		delay-driven								
		SALT	KRY	ERT	SERT	SALT-1	FLUTE	RSLT-ER	SALT+DDES	SALT+DDES-S	RSLT-ER+DDES	RSLT-ER+DDES-S
small	WL	1.00819	1.68838	1.86274	1.16825	1.00819	1	1.11404	1.60437	1.0256	1.61161	1.19137
	max_PL	1	1	1.02578	2.32023	1	1.12241	1.00523	1	1	1	1
	max_Delay	1.21284	1.16502	1.15079	1.35024	1.21284	1.25154	1.19719	1.14279	1.20834	1.13554	1.17386
	avg_Delay	1.0208	0.94095	0.90838	1.12314	1.0208	1.05708	0.99728	0.89494	1.01487	0.89114	0.96495
medium	WL	1.06065	2.95887	2.77676	1.41749	1.06953	1	1.38952	1.86685	1.10867	2.12896	1.57045
	max_PL	1.01159	1	1.12628	2.15248	1	1.4175	1.02055	1.00002	1.01112	1.00022	1.0004
	max_Delay	1.53658	1.33622	1.29012	1.61692	1.53924	1.73868	1.45725	1.32787	1.50419	1.26662	1.336
	avg_Delay	1.22998	1.00362	0.95103	1.27199	1.23229	1.39502	1.13836	0.96438	1.19939	0.92109	1.01768
large	WL	1.10041	2.55429	2.84039	1.51962	1.12257	1	1.48715	1.67586	1.16107	2.12081	1.72951
	max_PL	1.03641	1.04836	1.36034	2.00876	1	2.02437	1.05253	1.00026	1.02012	1.00346	1.00479
	max_Delay	1.82524	1.52709	1.42504	1.7958	1.83432	2.37354	1.72622	1.52096	1.75483	1.38500	1.4592
	avg_Delay	1.43343	1.15979	1.09009	1.4168	1.44081	1.85517	1.32919	1.11262	1.37779	1.02889	1.11332
huge	WL	1.12665	2.89377	2.99978	1.63016	1.15889	1	1.75214	1.69613	1.19325	2.39299	1.98207
	max_PL	1.05245	1.09275	1.48651	2.03275	1	2.32766	1.0414	1.00097	1.03157	1.01173	1.01449
	max_Delay	2.31162	1.74177	1.56008	2.13245	2.32556	3.23896	2.12859	1.82222	2.20128	1.53721	1.64777
	avg_Delay	1.77569	1.25889	1.17894	1.65849	1.78781	2.49066	1.579	1.2772	1.68923	1.10976	1.22386

Comparing the results in Table III, we can see that our RSLT-ER+DDES always obtains the lowest delay than the other algorithms. Furthermore, we have the observations as follows.

1) Compared with shallow-light tree construction algorithms SALT and KRY, on average our RSLT-ER+DDES algorithm achieves an improvement of average delay by 25.88% over SALT, and that by 9.16% over KRY.
2) Compared with delay-driven tree construction algorithms ERT and SERT, on average our RSLT-ER+DDES algorithm achieves an improvement of average delay by 4.13% over ERT, and that by 27.18% over SERT.
3) Compared with the two algorithms (KRY and ERT) with delays smaller than other existing algorithms, our RSLT-ER+DDES achieves lower delay while utilizing a reduced WL. Specifically, it reduces the WL by an average of 16.72% over KRY and 20.59% over ERT.
4) For all algorithms, the maximum PL is minimal (close to 1.0) when the delay is low, which suggests that constructing a shallow-light tree from a small ε value is highly effective in achieving a topology with minimal delay.
5) When examining columns 3–9 of Table III, we observe two distinct features: FLUTE has the shortest WL but tends to have the largest delay, while ERT has the lowest delay but tends to have the longest WL. These features indicate that focusing on minimizing WL or delay separately is inadequate for delay-driven rectilinear Steiner tree construction, hence it is preferable to consider the two factors simultaneously.

Fig. 7 illustrates how different values of ε impact the performance of RSLT-ER+DDES, RSLT-ER+DDES-S, RSLT-ER, and SALT. As ε increases, we observe a gradual reduction in WL with the cost of increased delay. In particular, RSLT-ER+DDES consistently has the best delay performance but tends to have a longer WL. In contrast, SALT achieves the shortest WL, yet with a larger delay. Hence, RSLT-ER+DDES-S offers a tradeoff between WL and delay, thereby presenting moderate performance on the two metrics.

We further compare in Table IV the runtimes of SALT [11], ERT [7], SERT [7], our RSLT-ER+DDES-S, and RSLT-ER+DDES. RSLT-ER+DDES spends 5.56×, 4.55×, 0.85×, and 1.49× the runtimes of SALT, ERT, SERT, and RSLT-ER+DDES-S, respectively. The runtime of our algorithm increases notably on large and huge nets, primarily due to the expanded search space of the DDES algorithm.

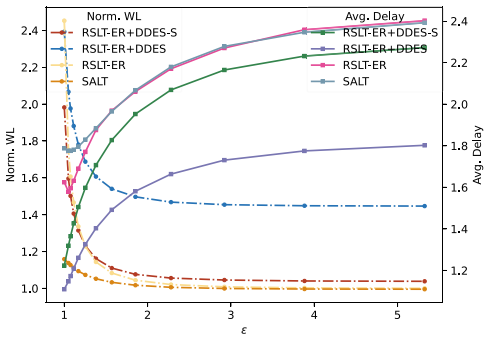


Fig. 7. Comparison of average normalized WL and average normalized delay at different values of parameter ε.

TABLE IV
AVERAGE RUNTIME (10^{-2} S) ON EACH GROUP OF BENCHMARKS

	SALT	ERT	SERT	RSLT-ER+DDES-S	RSLT-ER+DDES
small	0.02	0.00	0.01	0.03	0.05
medium	0.10	0.03	0.15	0.24	0.34
large	0.29	0.23	1.23	0.99	1.42
huge	0.61	0.96	5.34	2.64	3.94
Avg. Ratio	0.18	0.22	1.17	0.67	1.00

C. Effectiveness of Delay-Driven Edge Shifting

To see the effectiveness of our DDES algorithm, we compare our RSLT-ER with RSLT-ER+DDES, and compare SALT with SALT+DDES. The results in Table III show that, the DDES algorithm has a very significant effect in reducing the average Elmore delays, ranging from 10.7% to 29.8%. Particularly on the huge nets, RSLT-ER+DDES improves the average delay by 29.8% over RSLT-ER, and SALT+DDES improves the average delay by 28.1% over SALT.

When comparing SALT+DDES and RSLT-ER+DDES, the latter achieves lower delay but exhibits a longer WL. This is because the RSLT-ER algorithm does not introduce additional Steiner points during the construction of the shallow-light Steiner tree, and hence it leads to longer WL than SALT.

TABLE V
AVERAGE RESULTS OF 1294K NETS AT $\varepsilon = 1$. "RT" DENOTES THE AVERAGE RUNTIME (10^{-4} S)

	WL	max_PL	max_delay	avg_Delay	RT
SERT	1.517	2.584	1.606	1.321	3.03
ERT	2.825	1.060	1.204	0.919	1.40
SERT-SLT+DDES	2.328	1.000	1.201	0.915	8.22
ERT-SLT+DDES	2.477	1.000	1.162	0.894	6.74
RSLT-ER+DDES	2.334	1.000	1.184	0.901	3.62

However, by reducing the number of Steiner points, we can lessen the probability of multiple computational load capacitance occurring on the source-sink path, leading to improved signal propagation and reduced delay. This characteristic explains why the optimal topology for minimizing delay tends to resemble a "star" topology when r_d decreases. The same results can also be observed from the comparison of SALT+DDES-S and RSLT-ER+DDES-S.

D. Robustness of the Algorithm Under Nonuniform Pin Capacitance Distribution

In this section, we verify the validity of the proposed flow when the distribution of the capacitance values on all sinks is not uniform. Benchmarks of the ICCAD 2015 Contest [26] are used for a comprehensive evaluation and comparison. For the benchmarks, each sink within a net is reassigned a capacitance value that ranges from 0 to e-15. We then replace the initial tree of the default RSLT-ER algorithm (line 2 in Algorithm 1) with a delay-driven tree. For simplicity, the modified RSLT-ER algorithm with initial trees generated by SERT [7] and ERT [7] are referred to as "SERT-SLT" and "ERT-SLT," respectively. We compare these modified algorithms with the default RSLT-ER algorithm. The experimental results are presented in Table V.

From Table V, we can see that our default algorithmic flow, RSLT-ER+DDES, maintains excellent delay performance even when the load capacitance values of the pins are uneven. From Table V, we also have the following observations.

1) Compared to ERT-SLT+DDES, RSLT-ER+DDES achieves similar delay performance with shorter WL and much-reduced runtime. The differences of max_Delay and avg_delay are 0.022 and 0.007, respectively.
2) Compared to SERT-SLT+DDES, RSLT-ER+DDES presents better delay performance, similar WL, and much-reduced runtime. Specifically, the reductions in max_Delay and avg_delay are 0.017 and 0.014, respectively.
3) Compared to the previous delay-driven algorithms, SERT and ERT, RSLT-ER+DDES achieve better delay performance and shorter WL than ERT. Moreover, RSLT-ER+DDES has significantly better delay performance than SERT, but with a longer WL.

The experimental results in Table V highlight the effectiveness of our default flow. Hence, although the RSLT-ER algorithm initially ignores the load capacitance on the pins, the subsequent DDES algorithm DDES takes into account the load capacitance on the pins, allowing us to strike a balance between WL and delay performance through these two complementary steps.

E. Flexibility in Balancing Wire Length and Delay

In this section, we show the flexibility of our RSLT-ER+DDES and RSLT-ER+DDES-S algorithms in balancing WL and delay. For different algorithms on each group of nets, Fig. 8 depicts the tradeoff curves for the maximum normalized delay and normalized WL to varying values of ε. From the figure, we have the following observations.

1) When delay is the most important metric and WL is less concerned, then our RSLT-ER+DDES is a better option than other algorithms because it achieves the lowest delay.
2) When WL and delay are equally considered, then our RSLT-ER+DAES-S is a better option than other algorithms since it can reach a smaller delay while ensuring the WL be not too long. In particular, when $\varepsilon = 1.3797$, it can obtain a lower delay than SALT while maintaining a similar WL to SALT. When $\varepsilon = 1.075$, it can achieve a similar delay and a significant reduction in WL, compared to KRY.

F. Superior Performance on the CS-RST Problem

In this section, we present the superior performance of the RSLT-ER+DDES algorithm in solving the CS-RST problem. Benchmarks of the ICCAD 2015 Contest [26] are used for a comprehensive evaluation and comparison. For the benchmarks, each sink within a net is assigned a slack value that ranges from -4 to 0.

We compare RSLT-ER+DDES with RSLT-ER+DDES-S, SALT [11], SERT-C [7], and ERT-C [7]. Furthermore, we develop an algorithm called critical bound delay (CBD), inspired by the definition of the lower bound of delay in [30, Definition 8]. CBD constructs a RSMT connecting the driver pin to pins with 0 slacks, then directly connects pins with negative slacks to the driver pin. The overall results are given in Table VI.

Since the CS-RST problem requires the algorithm to return a tree such that pins with higher timing criticality are prior to have smaller delays, we give the following metric to evaluate the performance of various algorithms:

$$\sum_{n_i \in V \setminus \{n_0\}} \max\left(0, \frac{d_T(n_i)}{lb(n_i)} - s_{n_i}\right)$$

where $lb(n_i)$ is calculated by the lower bound of delay in [30]. The smaller the value of the metric, the better an algorithm solves the CS-RST problem.

From Table VI, we can draw the following conclusion. Compared to RSLT-ER+DDES-S, SALT [11], SERT-C [7], ERT-C [7], and CBD, on average RSLT-ER+DDES achieves significant improvements on metric (1.06×, 1.30×, 1.29×, 1.49×, and 1.31×, respectively). This indicates that RSLT-ER+DDES achieves the best delay allocation by constructing Steiner trees tailored to the sinks' slack information. In addition, RSLT-ER+DDES uses 1.20×, 1.79×, 1.39×, 0.89×, and 0.59× the WL of RSLT-ER+DDES-S, SALT, ERT-C, SERT-C, and CBD, respectively.

To more clearly illustrate the performance of various algorithms, we select a 21-pin net from superblue1 [26]. The

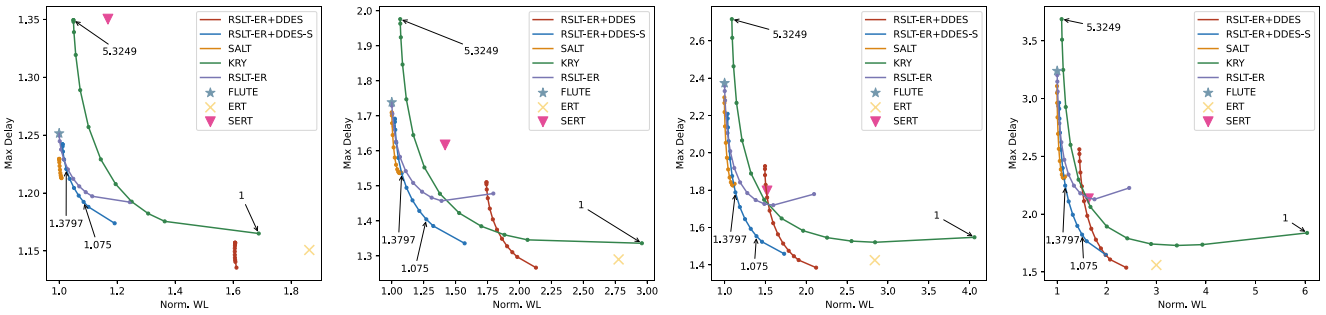


Fig. 8. WL and maximum delay tradeoff on: (a) small nets, (b) medium nets, (c) large nets, and (d) huge nets. Each point in the subfigures represents a result at a specific value of ε, decreasing from the left to right.

TABLE VI
AVERAGE "METRIC" AND "WL" RESULTS

	RSLT-ER+DDES		RSLT-ER+DDES-S		SALT [11]		SERT-C [7]		ERT-C [7]		CBD	
	metric	WL	metric	WL	metric	WL	metric	WL	metric	WL	metric	WL
small	9.16303	1.45087	9.62136	1.20791	10.0299	1.00819	9.39752	1.40547	11.7457	1.53335	9.53162	1.74885
medium	25.4231	1.95030	27.1130	1.58192	30.7125	1.06953	29.2780	1.45869	36.3416	2.08542	29.1083	2.81267
large	55.1497	2.04042	58.1495	1.73167	68.6035	1.12257	68.4034	1.36210	79.5900	2.38997	69.2674	3.63208
huge	99.9048	2.34522	106.589	1.97684	136.225	1.15889	137.652	1.37873	154.615	2.69213	140.682	5.01353
Avg. Ratio	1.00	1.00	1.06	0.83	1.30	0.56	1.29	0.72	1.49	1.12	1.31	1.70

TABLE VII
SLACK AND DELAY VALUES AT $\varepsilon = 1$. RESULTS IN BOLD BLACK HAVE THE SMALLEST SOURCE-SINK DELAY

Node id	Node slack	RSLT-ER+DDES		SALT [11]		SERT-C [7]		ERT-C [7]		CBD	
		delay	metric	delay	metric	delay	metric	delay	metric	delay	metric
1	-0.4	**1.354**	1.754	1.422	1.822	1.729	2.218	1.850	2.25	1.713	2.113
2	-3.4	**1.204**	4.604	1.317	4.717	1.403	4.959	1.482	4.882	1.476	4.876
3	0	**2.114**	2.114	2.164	2.164	2.568	2.777	4.139	4.139	3.918	3.918
4	0	**1.988**	1.988	2.149	2.149	2.110	2.246	5.291	5.291	3.181	3.181
5	0	1.522	1.522	1.115	1.115	**1.083**	1.15	2.420	2.42	2.474	2.474
6	0	**1.617**	1.617	1.776	1.776	1.896	2.108	3.011	3.011	2.507	2.507
7	0	**1.908**	1.908	2.109	2.109	2.261	2.483	3.923	3.923	3.083	3.083
8	0	**1.393**	1.393	1.525	1.525	1.625	1.773	2.519	2.519	2.100	2.1
9	0	**1.723**	1.723	1.852	1.852	2.264	2.388	4.522	4.522	3.214	3.214
10	-0.4	**2.085**	2.485	2.181	2.581	2.535	3.137	2.319	2.719	2.103	2.503
11	0	**1.736**	1.736	1.827	1.827	2.233	2.351	4.383	4.383	3.143	3.143
12	0	**1.433**	1.433	1.963	1.963	2.111	2.299	3.864	3.864	2.874	2.874
13	0	1.458	1.458	1.331	1.331	**1.292**	1.372	2.378	2.378	2.630	2.63
14	0	**1.757**	1.757	1.809	1.809	2.272	2.404	4.830	4.83	3.409	3.409
15	-0.4	**1.970**	2.37	2.331	2.731	2.400	2.981	2.333	2.733	2.091	2.491
16	0	**2.042**	2.042	2.268	2.268	2.442	2.659	3.678	3.678	3.455	3.455
17	0	**1.440**	1.44	1.513	1.513	1.837	1.932	1.983	1.983	2.537	2.537
18	-0.4	**1.652**	2.052	2.190	2.59	2.139	2.663	2.126	2.526	1.990	2.39
19	0	**1.881**	1.881	1.958	1.958	2.489	2.624	4.991	4.991	3.538	3.538
20	-0.4	**1.850**	2.25	2.014	2.414	2.249	2.795	2.348	2.748	2.306	2.706
Total metric			39.527		42.214		49.31		69.79		59.142
Avg. Ratio			1.00		1.07		1.25		1.77		1.50

detailed slack information, along with normalized source-sink delay and metric values, are presented in Table VII. In the table, the "delay" is normalized by $[(d_T(n_i))/(lb(n_i))]$, where $lb(n_i)$ is calculated by the lower bound of delay in [30]. The Steiner trees generated by various algorithms are shown in Fig. 9. As RSLT-ER+DDES and RSLT-ER+DDES-S yield identical results, only the results of RSLT-ER+DDES are presented.

From Table VII, we have the following observations.

1) For all sinks with negative slacks, the Steiner tree generated by RSLT-ER+DDES minimizes the source-sink delay for these nodes, compared to RSLT-ER+DDES-S, SALT [11], SERT-C [7], ERT-C [7], and CBD. This indicates that RSLT-ER+DDES is able to generate more suitable Steiner trees based on timing information.
2) For the 14 sinks with positive slacks, the Steiner tree generated by RSLT-ER+DDES produces comparable source-sink delays, with 12 nodes achieving the minimum source-sink delays.
3) Compared to SALT [11], SERT-C [7], ERT-C [7], and CBD, RSLT-ER+DDES achieves significant improvements on metric (1.07×, 1.25×, 1.77×, and 1.50×, respectively).

G. Effectiveness of Constructing Topology With Minimal Delay

In the previous sections, we have seen that our RSLT-ER+DDES algorithm shows excellent delay performance on a single net. In this section, we analyze the topology generated

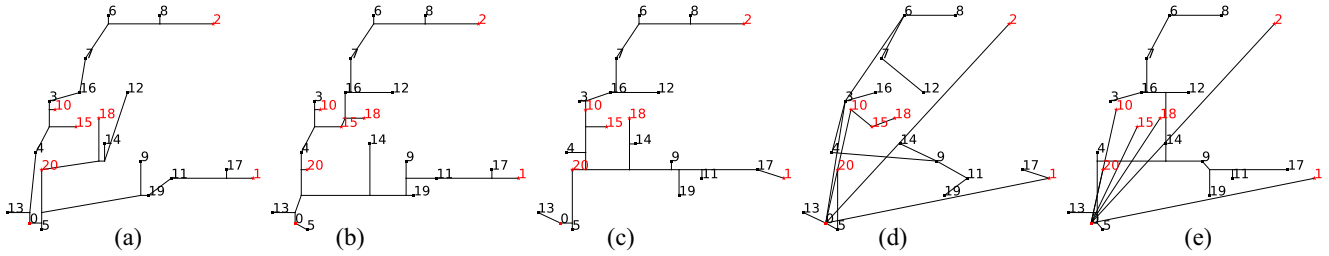


Fig. 9. Topologies of a 21-pin net obtained by: (a) RSLT-ER+DDES, (b) SALT [11], (c) SERT-C [7], (d) ERT-C [7], and (e) CBD. The WLs of the topologies are 414 010, 342 180, 332 510, 735 300, and 729 400, respectively. Sinks with negative slacks are labeled in red.

TABLE VIII
NET STATISTICAL INFORMATION OF EACH DESIGN

Design	#Net				
	Total	small	medium	large	huge
gcd	688	61	15	1	0
aes_core	28882	1919	2484	182	0
apu	4921	845	319	0	0
blabla	26536	3620	1201	65	0
BM64	14370	1586	930	54	0
caravel_upw	1413	82	48	8	0
picorv32a	20306	2755	1302	72	1
s44	244	34	7	1	0
salsa20	34500	3309	2038	163	0
PPU	18240	830	1366	120	0

by RSLT-ER+DDES on real netlists. We select ten real circuits as benchmarks and map them onto the advanced SkyWater [1] technology nodes. The statistical information of the designs is given in Table VIII. The placement results are obtained using an open-source tool. There is no timing information on each design. Table IX lists each worst negative slack (WNS), total negative slack (TNS), total WL, and runtime of the designs, for which the routing topologies are generated by FLUTE and RSLT-ER+DDES, respectively.

In Table IX, the column labeled by "Δ" presents the deviation between the TNS obtained by RSLT-ER+DDES and the TNS obtained by FLUTE. The column labeled by "÷" gives the ratio of the results obtained by RSLT-ER+DDES to the results obtained by FLUTE. Compared to FLUTE, on the ten designs, TNS is reduced by an average of 833.154 ns, with 51.4% decrease in WNS. In addition, the total WL and runtime are increased by 3.5% and 86.1%. The experimental results demonstrate the positive effect of constructing a routing topology with minimized Elmore delay.

V. CONCLUSION

This article has shown the significance of simultaneously optimizing WL and PL in minimizing the Elmore delay, and has investigated how delay changes during PL optimization. Guided by the theoretical findings, this article began with a RSMT and iteratively used edge replacement to optimize PL while controlling WL increase. Subsequently, the Steiner tree was refined locally using edge shifting to further optimize delay. Experimental results demonstrate that the proposed algorithm achieves the lowest delay while maintaining competitive WL. Moreover, it is highlighted that the proposed algorithm has the flexibility to balance WL and delay when constructing a rectilinear Steiner tree. In the future, we plan to extend this work to timing-driven VLSI routing. In addition, we propose to divide large nets into several smaller nets using some clustering approach, to reduce the runtime for large nets.

APPENDIX A
PROOF OF THEOREM 2

Proof: (See Fig. 2) Reducing or keeping unchanged the WL of a Steiner tree implies that $\Delta WL = \text{dist}(a, x) - (pl(a) - pl(q)) \leq 0$. Substituting ΔWL into (3)–(6) in Lemma 1 can get as follows.

1) For (3), we have $(r_d + pl(x))\Delta WL \leq 0$. Moreover, $d_T(x \to a) = pl(a)([(pl(a))/2] + C_a) + K$, where $K \geq 0$ is a constant related to the Steiner tree. Further, since $pl(a) \geq \text{dist}(a)$, we have $\text{dist}(a, x)([(\text{dist}(a, x))/2] + C_a) - d_T(x \to a) \leq 0$. So, $\Delta d_a \leq 0$.
2) For (4), it is clear that $\Delta d_{l_j} \leq 0$, $l_j \in \text{path}(n_0, x)$.
3) For (5), since $pl(m_j) - pl(x) > 0$, and $pl(q) - pl(a) > 0$, we have $\Delta d_{m_j} < 0$, $m_j \in \text{path}(x, q)$.
4) For (6), since $pl(q) - pl(x) > 0$, and $pl(a) - pl(q) > 0$, we have $\Delta d_q < 0$.

Hence, by Lemma 1, Theorem 2 holds. ■

APPENDIX B
PROOF OF THEOREM 3

Proof: (See Fig. 3) Without loss of generality, suppose there are nodes $l_1, \ldots, l_L$ on $\text{path}(s_1, s_2)$. For convenience, we also use $l_1, l_2, \ldots, l_L$ to represent the lengths of the edges $(s_1, l_1), (l_1, l_2), \ldots, (l_{L-1}, l_L)$. Next, we analyze the delays under different trees.

In tree T [Fig. 3(a)], the delay from s_1 to s_2 is

$$d_T(s_1 \to s_2) = l_1\left(\frac{l_1}{2} + C_{l_1}\right) + l_2\left(\frac{l_2}{2} + C_{l_2}\right) + \cdots + \text{dist}(l_L, s_2)\left(\frac{\text{dist}(l_L, s_2)}{2} + C_{s_2}\right).$$

In tree T' [Fig. 3(b)], the delay from s_1 to s_2 is

$$d_{T'}(s_1 \to s_2) = l_1\left(\frac{l_1}{2} + C'_{l_1}\right) + l_2\left(\frac{l_2}{2} + C'_{l_2}\right) + \cdots + \text{dist}(l_L, s_2)\left(\frac{\text{dist}(l_L, s_2)}{2} + C'_{s_2}\right)$$

where $C'_n = C_n - \text{dist}(m, m') - C_{m'}$, $n \in \{l_1, \ldots, l_L, s_2\}$. Since

$$\text{dist}(m, m') + \text{dist}(k, k') \geq \text{dist}(r, m') + \text{dist}(k, r) + \text{dist}(r, k')$$

TABLE IX
WNS (NS), TNS (NS), TOTAL WL (UM), AND RUNTIME (S) UNDER DIFFERENT TOPOLOGIES

Design	WNS			TNS			WL			Runtime		
	FLUTE	RSLT-ER +DDES	÷	FLUTE	RSLT-ER +DDES	Δ	FLUTE	RSLT-ER +DDES	÷	FLUTE	RSLT-ER +DDES	÷
gcd	-0.304	-0.262	0.862	-8.713	-4.686	4.027	17252629	18735054	1.086	3.465	5.249	1.515
aes_core	-2.187	-0.058	0.027	-362.404	-0.191	362.213	2239590750	2335379315	1.043	4.248	10.899	2.564
apu	-0.306	-0.227	0.742	-5.065	-3.170	1.895	140465211	145098013	1.033	3.414	6.111	1.790
blabla	-7.041	-1.262	0.179	-1329.434	-6.913	1322.521	2078775210	2114021211	1.017	4.318	8.338	0.002
BM64	-2.578	-1.256	0.487	-778.608	-87.719	690.889	860865166	881967485	1.025	4.053	6.815	1.681
caravel_upw	-0.052	-0.028	0.538	-0.157	-0.029	0.128	38409321	38411444	1.000	3.594	5.598	1.558
picorv32a	-0.989	-0.914	0.924	-0.989	-1.069	-0.08	1101774795	1140214431	1.035	4.015	6.518	1.623
s44	-0.164	-0.167	1.018	-5.004	-5.017	-0.013	4028511	4073680	1.011	3.446	5.581	1.620
salsa20	-1.734	-0.085	0.049	-17.989	-0.280	17.709	2168846145	2219485001	1.023	4.210	8.537	2.028
PPU	-9.311	-0.355	0.038	-5924.384	-1.038	5923.346	1084602459	1169968888	1.079	3.897	8.976	2.303
Avg. Ratio	-2.467	-0.461	0.486	-843.176	-11.011	833.154	973461019.7	1006735452	1.035	3.866	7.2613	1.861

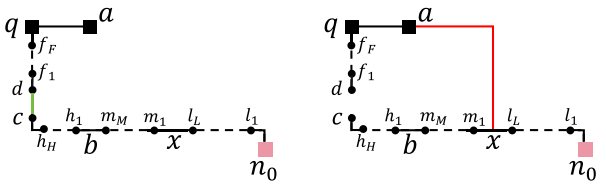


Fig. 10. (a) Initial tree T. (b) Optimized tree T'.

we have $d_T(n_0 \to s_1) \geq d_{T'}(n_0 \to s_1)$. Moreover, the path from s_2 to a is identical in both trees T and T', so we have $d_T(s_2 \to a) = d_{T'}(s_2 \to a)$. Hence, we get

$$\begin{aligned} &d_T(n_0 \to a) - d_{T'}(n_0 \to a) \\ &= d_T(n_0 \to s_1) + d_T(s_1 \to s_2) + d_T(s_2 \to a) \\ &\quad - (d_{T'}(n_0 \to s_1) + d_{T'}(s_1 \to s_2) + d_{T'}(s_2 \to a)) \\ &\geq \big(\text{dist}(m, m') + C_{m'}\big) d_{(s_1,s_2)} > 0 \end{aligned}$$

where $d_{(s_1,s_2)} = l_1 + \cdots + l_L + \text{dist}(l_L, s_2)$, and Theorem 3 holds. ∎

APPENDIX C
PROOF OF THEOREM 4

Proof: (See Fig. 10) Consider a as the node to be optimized. In the initial tree T, the parent node of a is q. In the optimized tree T', the parent of a is reassigned to x, which is a node on path(n_0, b). (c, d) is the edge on path(n_0, a) to be removed.

Let $M_a = \text{dist}(n_0, a)$. As shown in Fig. 10, suppose there are nodes $l_1, \ldots, l_L$ on path(n_0, x) and nodes $m_1, \ldots, m_M$ on path(x, b). Moreover, suppose there are nodes $h_1, \ldots, h_H$ on path(b, c) and nodes $f_1, \ldots, f_F$ on path(d, q). For convenience, we further define a, b, c, d, q, and x to represent the respective PLs from n_0 to these nodes in T. Additionally, we define $l_1, l_2, \ldots, l_L$ to represent the lengths of edges $(n_0, l_1), (l_1, l_2), \ldots, (l_{L-1}, l_L)$ and define similarly for m_j, h_j, and f_j.

We show that the delay function is concave in terms of $0 \leq x \leq b$. Consider the contribution made by replacing the edge (c, d) with the edge (a, x) to Elmore delay at various sinks $n_j \in T'$.

Let $\Delta WL = \text{length}(T') - \text{length}(T) = (M_a - x) - (d - c)$, C'_a denote the total capacitance of subtree $T'(a)$. First, we need to point out that the elements in the set $\{C'_n | n \in \text{path}(m_1, a)\}$ are independent of x, and $d_T(n \to s) = d_{T'}(n \to s)$ $(n \in \text{path}(n_0, a), s \in T'(n) \cap T(n))$ are also independent of x. Next, we analyze the delay of each edge in turn:

1) For $d_{T'}(n_0 \to x)$

$$\begin{aligned} d_{T'}(n_0 \to x) &= r_d C'_{n_0} + l_1\left(\frac{l_1}{2} + C_{l_1} + \Delta WL\right) + \cdots \\ &\quad + d(x, l_L)\left(\frac{d(x, l_l)}{2} + C_{l_L} + \Delta WL - d_{(x,l_L)} - \widetilde{C}_{l_L}\right) \end{aligned}$$

where $d_{(x,l_L)} = x - (l_1 + \cdots + l_L)$, $C'_{n_o} = C_{n_o} + \Delta WL$, and $\widetilde{C}_{l_L} = C_{l_L} - C_{m_1} - \text{dist}(l_L, m_1)$. The coefficient of x^2 in $d_{T'}(n_0 \to x)$ is $-(3/2)$. Consequently, $d_{T'}(n_0 \to x)$ is concave in x.

2) For $d_{T'}(x \to a)$

$$d_{T'}(x \to a) = (M_a - x)\left(\frac{M_a - x}{2} + C'_a\right).$$

The coefficient of x^2 in $d_{T'}(x \to a)$ is $(1/2)$. Consequently, $d_{T'}(n_0 \to a) = d_{T'}(n_0 \to x) + d_{T'}(x \to a)$ has a negative coefficient for x^2.

3) For $d_{T'}(n_0 \to l_j)$

$$\begin{aligned} d_{T'}(n_0 \to l_j) &= r_d C'_{n_0} + l_1\left(\frac{l_1}{2} + C_{l_1} + \Delta WL\right) + \cdots \\ &\quad + l_j\left(\frac{l_j}{2} + C_{l_j} + \Delta WL\right) \forall l_j \in \text{path}(n_0, x). \end{aligned}$$

$d_{T'}(n_0 \to l_j)$ is linear (and thus concave) in terms of x.

4) Similarly, we have

$$d_{T'}(a \to q) = (a - q)\left(\frac{a - q}{2} + C'_q\right) \tag{9}$$

$$\begin{aligned} d_{T'}(q \to f_j) &= d_{(q,f_F)}\left(\frac{d_{(q,f_F)}}{2} + C'_{f_F}\right) + f_F\left(\frac{f_F}{2} + C'_{f_{F-1}}\right) \\ &\quad + \cdots + f_j\left(\frac{f_j}{2} + C'_{f_{j-1}}\right) \quad \forall f_j \in \text{path}(d, q) \end{aligned} \tag{10}$$

$$d_{T'}(q \to d) = d_{T'}(q \to f_1) + f_1\left(\frac{f_1}{2} + C'_d\right) \tag{11}$$

$$\begin{aligned} d_{T'}(b \to h_j) &= h_1\left(\frac{h_1}{2} + C'_{h_1}\right) + \cdots \\ &\quad + h_j\left(\frac{h_j}{2} + C'_{h_j}\right) \quad \forall h_j \in \text{path}(b, c) \end{aligned} \tag{12}$$

$$d_{T'}(b \to c) = d_{T'}(b \to h_H) + d_{(c,h_H)}\left(\frac{d_{(c,h_H)}}{2} + C'_c\right) \tag{13}$$

where $d_{(q,f_F)} = \text{dist}(q, f_F)$ and $d_{(c,h_H)} = \text{dist}(c, h_H)$; (9)–(13) are all constants independent of x.

5) Moreover

$$d_{T'}(x \to m_j) = m_1\left(\frac{m_1}{2} + C'_{m_1}\right) + \cdots + m_j\left(\frac{m_j}{2} + C'_{m_j}\right) \quad \forall m_j \in \text{path}(x, b) \tag{14}$$

$$d_{T'}(x \to b) = d_{T'}(x \to m_M) + d_{(b,m_M)}\left(\frac{d_{(b,m_M)}}{2} + C'_b\right) \tag{15}$$

where $d_{(b,m_M)} = \text{dist}(b, m_M)$. Since m_1 is actually linear in terms of x with coefficient 1, (14) and (15) are linear (and thus concave) in x.

Based on the above analysis and the fact that the sum of two concave functions is also concave, we can draw the following conclusions. If sink $n_j \in T'(l_j)$, then $d_{T'}(n_j) = d_{T'}(n_0 \to l_j) + d_{T'}(l_j \to n_j)$ is linear (and thus concave) in terms of x. Similarly, if sink $n_j \in T'(a) \cup T'(q) \cup T'(m_j) \cup T'(h_j) \cup T'(f_j)$, then $d_{T'}(n_j)$ has a negative coefficient for x^2. Consequently, $d_{T'}(n_j)$ is concave in x.

In summary, all sinks' delay functions are concave functions over the interval $0 \le x \le b$. Any concave function defined over an interval can be minimized at one of the two endpoints of the interval. Consequently, the total delay can be minimized when $x = 0$ (n_0 is the parent of a) or $x = b$ (b is the parent of a).

Hence, Theorem 4 holds. ■

Appendix D
Proof of Theorem 6

Proof: Since the DDES-S algorithm will preserve the solutions with better delays, we have $f_2(T') < f_2(T)$. First, we prove that $f_1(T') \le f_1(T)$.

In the DDES-S algorithm, the process of identifying candidate edges for each node via an R-tree is as follows. For a target node v_i, its immediate parent node is noted as v'_i. When searching with an R-tree, we construct a diamond-shaped query box for the edge (v'_i, v_i), which is centered at v_i, with each diagonal of length $2 \cdot \text{dist}(v'_i, v_i)$. If an edge (v'_j, v_j) is identified as a candidate edge of node v_i, then the query box of the edge (v'_i, v_i) intersects with the bounding box of the edge (v'_j, v_j), as in Fig. 5(a).

Subsequently, in DDES-S, v_i is reconnected to v''_j [the closest node to v_i within the bounding box of the edge (v_j, v'_j), as in Fig. 5(b)], which indicates that $\text{dist}(v'_i, v_i) \le \text{dist}(v''_j, v_i)$, and $\text{dist}(v'_j, v_j) = \text{dist}(v'_j, v''_j) + \text{dist}(v''_j, v_j)$. This ensures that the WL is reduced after each reconnection, so that $f_1(T') \le f_1(T)$.

The discussion above demonstrates that, for a given input tree T, the DDES-S algorithm can return a new tree T' such that T' Pareto dominates T. By repeatedly applying the DDES-S algorithm until that a tree T' Pareto dominating the input tree T cannot be obtained through edge shifting, this process can return a local Pareto optimal solution. Therefore, Theorem 6 holds. ■

References

[1] "SkyWater." 2021. [Online]. Available: https://github.com/google/skywater-pdk

[2] C. J. Alpert et al., "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2018, pp. 10–17.

[3] C. J. Alpert, T. C. Hu, J.-H. Huang, A. B. Kahng, and D. Karger, "Prim-Dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 7, pp. 890–896, Jul. 1995.

[4] B. Awerbuch, "Efficient broadcast and light-weighted spanners," Dept. Appl. Math. Comput. Sci., Weizmann Inst. Sci., Rehovot, Israel, Rep. CS92-22, 1992.

[5] C. Bartoschek, S. Held, D. Rautenbach, and J. Vygen, "Efficient generation of short and fast repeater tree topologies," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2006, pp. 120–127.

[6] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*. New York, NY, USA: Springer, 2009.

[7] K. D. Boese, A. B. Kahng, and G. Robins, "High-performance routing trees with identified critical sinks," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1993, pp. 182–187.

[8] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Fidelity and near-optimality of Elmore-based routing constructions," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 1993, pp. 81–84.

[9] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Rectilinear Steiner trees with minimum Elmore delay," in *Proc. 31st ACM/IEEE Design Autom. Conf. (DAC)*, 1994, pp. 381–386.

[10] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, "Near-optimal critical sink routing tree constructions," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 12, pp. 1417–1436, Dec. 1995.

[11] G. Chen and E. F. Y. Young, "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1217–1230, Jun. 2020.

[12] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table-based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

[13] J. P. Cohoon and L. J. Randall, "Critical net routing," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 1991, pp. 174–177.

[14] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-driven global routing for cell-based ICs," in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 1991, pp. 170–173.

[15] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C.-K. Wong, "Provably good performance-driven global routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 6, pp. 739–752, Jun. 1992.

[16] J. Cong, K.-S. Leung, and D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 1993, pp. 606–611.

[17] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec. 1959.

[18] W. E. Donath et al., "Timing driven placement using complete path delays," in *Proc. 27th ACM/IEEE Design Autom. Conf. (DAC)*, 1991, pp. 84–89.

[19] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. Jukl, and P. Kazak, "Chip layout optimization using critical path weighting," in *Proc. 25th ACM/IEEE Design Autom. Conf. (DAC)*, 1988, pp. 278–281.

[20] M. Elkin and S. Solomon, "Steiner shallow-light trees are exponentially lighter than spanning ones," *SIAM J. Comput.*, vol. 44, no. 4, pp. 996–1025, 2015.

[21] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55–63, 1948.

[22] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM J. Appl. Math.*, vol. 32, no. 4, pp. 826–834, 1977.

[23] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 1984, pp. 47–57.

[24] S. Held and D. Rotter, "Shallow-light Steiner arborescences with vertex delays," in *Proc. 16th Int. Conf. Integer Program. Comb. Optim.*, 2013, pp. 229–241.

[25] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, pp. 305–321, 1995.

[26] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2015, pp. 921–926.

[27] S.-E. D. Lin and D. H. Kim, "Construction of all rectilinear Steiner minimum trees on the Hanan grid and its applications to VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1165–1176, May 2019.

[28] R. C. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, no. 6, pp. 1389–1401, Nov. 1957.

[29] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 2, no. 3, pp. 202–211, Jul. 1983.
[30] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, pp. 86–109, May 2017.
[31] R.-S. Tsay, "Exact zero skew," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 1991, pp. 336–339.
[32] L. Yang, G. Sun, and H. Ding, "Toward timing-driven routing: An efficient learning-based geometric approach," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2023, pp. 1–9.

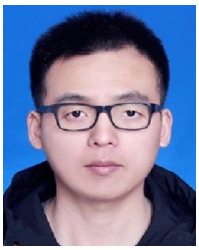
Hongxi Wu received the B.Sc. degree in statistics from Fujian Agriculture and Forestry University, Fuzhou, China, in 2019, and the M.Sc. degree in operations research and cybernetics from Minnan Normal University, Zhangzhou, China, in 2022. He is currently pursuing the Ph.D. degree with Fuzhou University, Fuzhou.

His research interest is VLSI physical design automation.

Xingquan Li received the Ph.D. degree from Fuzhou University, Fuzhou, China, in 2018.

He is an Associate Professor with Peng Cheng Laboratory, Shenzhen, China. He has published over 50 papers. His research interests include EDA and AI for EDA. His team has developed an open-source infrastructure of EDA and toolchain.

Dr. Li received the three First-Place Awards from ICCAD@CAD Contest in 2017, 2018, and 2022, the Application Award of Operations Research from the Operations Research Society of China in 2020, and the Best Paper Award from ISEDA 2023.



Liang Chen received the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2020.

From 2020 to 2022, he was a Postdoctoral Researcher with the University of California at Riverside, Riverside, CA, USA. Since 2023, he has been an Associate Professor with the School of Microelectronics, Shanghai University, Shanghai. His research interests include electronic design automation and machine learning for VLSI reliability analysis.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

Dr. Yu received 11 Best Paper Awards from ICCAD 2013, 2021, and 2024, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ICTAI 2019, Integration the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD Contest Awards, and many other awards, including the IEEE CEDA Ernest S. Kuh Early Career Award in 2022, the DAC Under-40 Innovator Award in 2024, and the Hong Kong RGC Research Fellowship Scheme Award in 2024. He has served as the TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD and in many journal editorial boards and conference committees.

Wenxing Zhu received the Ph.D. degree from Shanghai University, Shanghai, China, in 1996.

He is a Jiaxi Distinguished Professor with Fuzhou University, Fuzhou, China. His main research interests include VLSI physical design and optimization theory and algorithms.

Dr. Zhu received the First Prize of Natural Sciences from the Ministry of Education of China in 2022, the Application Award of Operations Research from the Operations Research Society of China in 2020, the Best Paper Awards from DAC 2017 and ISEDA 2023, and the Second Prize for National Teaching Achievement in 2009. He was nominated for the Best Paper Award at ICCAD 2018 and CCF-DAC 2023.