SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm

Gengjie Chen and Evangeline F. Y. Young

***Abstract*—In a weighted undirected graph, a spanning/Steiner shallow-light tree (SLT) simultaneously approximates: 1) shortest distances from a root to the other vertices and 2) the minimum tree weight. The Steiner SLT has been proved to be exponentially lighter than the spanning one. In this paper, we propose a novel Steiner SLT construction method called Steiner SLT (SALT), which is efficient and has the tightest bound over all the state-of-the-art general-graph SLT algorithms. Applying SALT to Manhattan space offers a smooth tradeoff between rectilinear Steiner minimum tree and rectilinear Steiner minimum arborescence for VLSI routing. The adaption also reduces the time complexity from $O(n^2)$ to $O(n \log n)$. Besides, several effective post-processing methods, including safe refinement and shallowness-constrained edge substitution, are proposed to further improve the result. The experimental results show that SALT can achieve not only short path lengths and wirelength but also small delay, compared to both classical and recent routing tree construction methods.**

***Index Terms*—Global routing, physical design, Steiner tree, timing optimization.**

I. Introduction

TIMING and power have been being crucial issues in chip design since more than two decades ago. They also become increasingly more critical as technology scales and application evolves. For example, 50%–80% of gates in high-performance ICs today are repeaters, which do not perform useful computation but work for timing closure [3]; over 50% of the chip at 8 nm will be powered off and cannot be utilized due to the power constraint [4]; the power-sensitive applications in mobile and Internet of Things become ubiquitous [5].

Interconnect, as the carrier of signals, determines the timing quality of ICs directly. It also significantly influences power and consumes more power than computation nowadays. In routing tree construction, both path length and tree weight (i.e., wirelength) and are thus important. Essentially, tree weight implies routing resource usage (routability), power consumption, cell delay, and wire delay, while path length implies wire delay [6].

Manuscript received June 15, 2018; revised October 19, 2018; accepted December 28, 2018. Date of publication January 23, 2019; date of current version May 22, 2020. This work was supported in part by the Research Grants Council of Hong Kong under Project CUHK14208914, and in part by the Hong Kong Ph.D. Fellowship Scheme. The preliminary version was presented at the International Conference on Computer-Aided Design (ICCAD) in 2017. This paper was recommended by Associate Editors I. Bustany and C. Chu.
(Corresponding author: Gengjie Chen.)

The authors are with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong (e-mail: gjchen@cse.cuhk.edu.hk; fyyoung@cse.cuhk.edu.hk).
Digital Object Identifier 10.1109/TCAD.2019.2894653

A. Light Trees or Shallow Trees

It is a well-studied problem if only one of the objectives between light tree weight and shallow path length is pursued, whether the domain is the spanning tree or the rectilinear Steiner one. For spanning trees, the minimum spanning tree (MST) can be obtained by various classical algorithms like Prim's and Kruskal's algorithms in $O(m + n \log n)$ time; the shortest-path tree (SPT) can be constructed by Dijkstra's algorithm in $O(m + n \log n)$ time [7]. For rectilinear Steiner trees, the one with minimum tree weight is called a rectilinear Steiner minimum tree (RSMT), while the lightest one with all paths from root being shortest is a rectilinear Steiner minimum arborescence (RSMA).

The RSMT construction can be achieved by using Steiner nodes on Hanan grid [8] and is NP-hard [9]. Besides the exponential-time exact algorithms (e.g., GeoSteiner [10]), there are, however, many efficient heuristics achieving good or even near optimal quality. Rectilinear MST (RMST) achieves an 1.5-approximation [11] and can be constructed in $O(n \log n)$ time [12]. Many fast algorithms (e.g., [13]–[19]) are also proposed in order to pursue a smaller tree weight.

The RSMA construction is also NP-hard [20]. Approaches for optimal RSMAs include integer programming [21] and dynamic programming [22]. An $O(n \log n)$-time 2-approximation is first proposed by Rao *et al.* [23] and later generalized to all four quadrants by Córdova and Lee [24]. Examples of efficient heuristics are [25]–[27].

B. Shallow Light Trees

The spanning/Steiner shallow-light tree (SLT) combines the objectives of shallow path length and light tree weight together, as Table I and Fig. 1 show. In a spanning/Steiner tree with *shallowness* α and *lightness* β, each path length is at most α times the shortest-path distance, while the tree weight is β times the minimum tree weight. In an $(\bar{\alpha}, \bar{\beta})$-SLT, $\alpha \leq \bar{\alpha}$, and $\beta \leq \bar{\beta}$.

A spanning SLT approximates SPT and MST simultaneously, where the tradeoff is in the order of $(1 + \epsilon, O(1/\epsilon))$. The ABP algorithm by Awerbuch *et al.* [28] and the BRBC algorithm by Cong *et al.* [29] are in fact identical and provide a bound of $(1 + 2\epsilon, 1 + [2/\epsilon])$. After them, Khuller *et al.* [30] proposed the KRY algorithm with a bound of $(1+\epsilon, 1+[2/\epsilon])$ and proves that the bound is the best possible one for spanning trees. KRY also provides a smooth tradeoff between SPT

0278-0070 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SPANNING AND STEINER SLTs

	Shallowest	Lightest	Shallow light
Spanning	Spanning SPT ($O(m+n\log n)$)	MST ($O(m+n\log n)$)	Spanning SLT
Steiner	Steiner SPT (NP hard)	SMT (NP hard)	Steiner SLT
Rectilinear Steiner	RSMA (NP hard)	RSMT (NP hard)	Rectilinear Steiner SLT

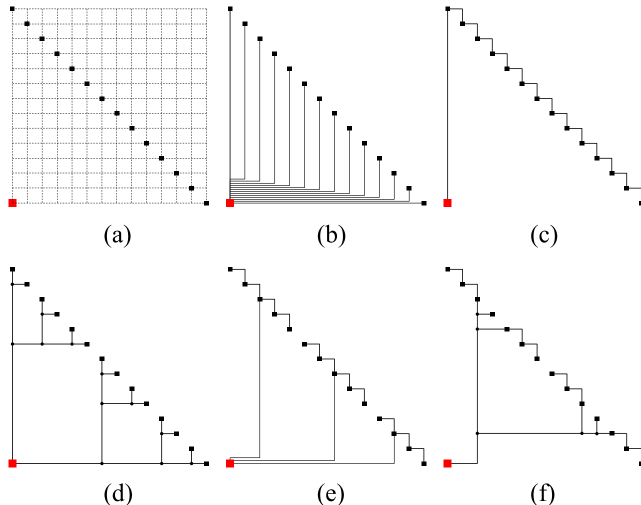

Fig. 1. Different routing topologies on the same net (the root is marked by red; α and β denote shallowness and lightness). (a) Net on a regular grid. (b) Spanning SPT ($\alpha = (13/13)$, $\beta = (182/39)$). (c) RMST/RSMT ($\alpha = (39/13)$, $\beta = (39/39)$). (d) RSMA ($\alpha = (13/13)$, $\beta = (54/39)$). (e) Spanning SLT ($\alpha = (17/13)$, $\beta = (61/39)$). (f) SALT ($\alpha = (17/13)$, $\beta = (44/39)$).

and MST controlled by ϵ, while ABP does not (e.g., an MST is not implied when $\epsilon = +\infty$). Besides, the PD algorithm due to Alpert *et al.* [32] smoothly trades off between SPT and MST, and has been widely used in industry [5]. However, the resulted tree is not guaranteed to be SLT.

Recently, Steiner SLTs (SALTs) are proved to be exponentially lighter than spanning ones by Elkin and Solomon [1], [2]. The ES algorithm can efficiently build a Steiner $(1+\epsilon, O(\log[1/\epsilon]))$-SLT with a time complexity of $O(n^2)$. The constants in the shallowness–lightness bound of $(1+2\epsilon, 4+2\lceil\log[2/\epsilon]\rceil)$ are, however, quite large (log denotes $\log_2$ in this paper). Held and Rotter [31] study the problem of SALT with vertex delays (measured by the number of bifurcations). When vertex delays are not taken into account, they tighten the bound of ES to $(1+\epsilon, 2+\lceil\log[2/\epsilon]\rceil)$ in 2-D Manhattan space.

C. Our Contributions

In this paper, we propose an efficient algorithm called SALT for constructing a Steiner SLT and apply it to routing topology construction. This paper is an extension to the preliminary version [33]. Our contributions are summarized as follows.

1) We propose SALT for the Steiner SLT on general graphs, whose shallowness–lightness bound is $(1+\epsilon, 2+\lceil\log[2/\epsilon]\rceil)$. To the best of our knowledge, the bound is tighter than all the previous methods for constructing general-graph spanning/SALTs (see Table II).

TABLE II
HISTORICAL PROGRESS OF SLTs

Algorithm	Shallowness-lightness bound	Metric
ABP/BRBC [28], [29]	$(1+2\epsilon, 1+\frac{2}{\epsilon})$	General
KRY [30]	$(1+\epsilon, 1+\frac{2}{\epsilon})$	General
ES [1], [2]	$(1+2\epsilon, 4+2\lceil\log\frac{2}{\epsilon}\rceil)$	General
HR [31]	$(1+\epsilon, 2+\lceil\log\frac{2}{\epsilon}\rceil)$	Manhattan
SALT	$(1+\epsilon, 2+\lceil\log\frac{2}{\epsilon}\rceil)$	General

2) We simplify SALT and reduce the runtime from $O(n^2)$ to $O(n\log n)$, when applying it to the Manhattan space for VLSI routing. We further decrease path lengths and tree weight in the Manhattan space by integrating SALT with the classical RSMA [23] and RSMT [19] algorithms. The method (rectilinear SALT) provides a smooth tradeoff between RSMA and RSMT controlled by ϵ.[1]
3) We apply several effective safe refinement (SR) techniques to improve the wirelength and path lengths of the tree output by rectilinear SALT.
4) As another post-processing step, we design an edge substitution algorithm to further minimize the wirelength, where slight path length degradation is allowed but is controlled under the shallowness constraint.

Note that we follow the definition in ES [2] for the general-metric Steiner tree. There are, however, some limitations on the generality (e.g., cannot be embedded into a Euclidean metric). The definition will be introduced in detail in Section II-A1.

As a geometric approach for VLSI routing, our method directly targets wirelength and path lengths instead of a highly accurate timing model. However, this is desirable due to three reasons. First, SALT provides a bounded tradeoff and has a strong global view. It can generate high-quality initial solutions for later stage optimization. Second, the linear delay model is reasonable due to buffering [34], [35], wire sizing, and layer assignment, compared to the Elmore delay model. Third, in the experiment, SALT is also comparable in terms of Elmore delay with the state-of-the-art Steiner tree construction method targeting Elmore delay directly [36]–[38].

Last but not least, we want to highlight that even though the bound analysis of SALT is complicated, it can be easily implemented with hundreds of lines of codes. The source code of SALT implementation is also publicly available at `https://github.com/chengengjie/salt`.

The remainder of this paper is organized as follows. The SALT algorithm on general graph (SALT) is presented in Section II. Its adaption to the Manhattan space (rectlinear SALT) is detailed in Section III. The post-processing techniques for further improving constructed trees are illustrated by Sections IV and V. In the end, Section VI shows and analyzes the experimental results, and Section VII concludes this paper.

[1]It was found after the preliminary version of this paper was published that our rectilinear SALT achieves the same bound as the approach in [31] in the Manhattan space. A minor difference is that rectilinear SALT incorporates a classical RSMA method [24] directly, leading to better Steiner trees in practice. Another small difference is that we break a tie (line 19 of Algorithm 2) to reduce the wirelength.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

TABLE III
NOTATIONS USED IN ES

$MST(G)$	Minimum spanning tree on graph G
$d_G(u, v)$	Distance between vertices u and v in graph G
P_i	i-th vertex on path P

Algorithm 1 ES

Require: Graph $G = (V, E, w)$, root r, trade-off parameter ϵ;
Ensure: Steiner SLT $T = (V', E', w')$ with $V' \supseteq V$ that dominates G;
1: $T_M \leftarrow MST(G)$;
2: $P \leftarrow$ Hamiltonian path based on T_M starting from r;
3: Breakpoint set $B \leftarrow \emptyset$;
4: Breakpoint $b \leftarrow r$;
5: **for** $v \leftarrow P_1$ to P_n **do**
6: **if** $d_P(b, v) > \epsilon \cdot d_G(r, v)$ **then**
7: $b \leftarrow v$;
8: $B \leftarrow B \cup \{b\}$;
9: $T_B \leftarrow$ Steiner SPT on $G[B \cup \{r\}]$ rooted at r;
10: $T \leftarrow$ spanning SPT on graph $T_M \cup T_B$;

II. STEINER SHALLOW-LIGHT TREE ALGORITHM

The exact problem formulation and the ES algorithm [2] for the SALT are first briefly introduced as preliminaries. The framework as well as the light Steiner SPT construction of SALT is then described, followed by the bound analysis.

A. Preliminaries

1) Problem Formulation: Our SALT algorithm on general graphs is under the same problem formulation used in [2]. A spanning/Steiner tree T of a weighted undirected n-vertex graph $G = (V, E, w)$ with respect to a root vertex r is called an SLT if: 1) it approximates all shortest-path distances $d_G(r, v)$ from r to $v \in V$ and 2) its weight $w(T)$ is bounded by that of MST $w(\text{MST}(G))$. For a $(\bar{\alpha}, \bar{\beta})$-SLT: 1) the shallowness $\alpha = \max\{[d_T(r, v)/d_G(r, v)] | v \in V \backslash \{r\}\} \leq \bar{\alpha}$ and 2) lightness $\beta = [w(T)/w(\text{MST}(G))] \leq \bar{\beta}$. Note that on a graph that is metric (i.e., with edge weights satisfying triangle inequality), a lightness bound with respect to MST infers one with respect to SMT because $w(\text{MST}(G)) \leq 2 \cdot w(\text{SMT}(G))$ (i.e., $w(T) \leq \bar{\beta} \cdot w(\text{MST}(G)) \leq 2 \cdot \bar{\beta} \cdot w(\text{SMT}(G))$). For rectilinear Steiner trees, the gap is smaller with $w(\text{RMST}(G)) \leq 1.5 \cdot w(\text{RSMT}(G))$.

Considering the general metric scenario, a Steiner tree for a graph $G = (V, E, w)$ is defined as a tree $T = (V', E', w')$ with $V' \supseteq V$ and $w' : E' \to \mathbb{R}^+$ that *dominates* the metric M_G induced by G, i.e., $\forall u, v \in V, d_T(u, v) \geq d_G(u, v)$.

Even though such SALT cannot be embedded into many metric spaces (e.g., Euclidean space[2] or finite graph metrics), it is applicable to the Manhattan space, which will be shown in Section III. For simplicity of illustration, we henceforth assume all the input graph G is complete and metric. Indeed, any weighted undirected graph G^* defines a metric space and thus implies a graph G that is complete and metric.

2) ES Algorithm: The ES algorithm extends the ABP algorithm for spanning SLTs to Steiner ones. The key steps are shown in Algorithm 1 and Fig. 2 with notations summarized in Table III. Its main idea is to accumulate the distance along a Hamiltonian path P and identify a breakpoint b whenever the accumulated distance becomes too long. Breakpoints are then connected to the root r directly by a Steiner SPT (line 9). In this way, the distance $d_T(r, b)$ between a breakpoint b and r in the tree T becomes the shortest-path distance $d_G(r, b)$. For other vertices, the path length is bounded.

[2]In the Euclidean plane, a bound of $(1 + \epsilon, O(\sqrt{[1/\epsilon]}))$ is achievable and tight for SALTs [39], [40].

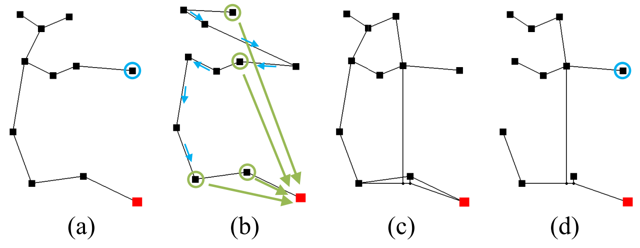

Fig. 2. Sample run of Algorithm 1 ($\epsilon = 1$). (a) Construct MST T_M (shallowness $\alpha = 3.14$ and lightness $\beta = 1$). (b) Identify breakpoints B (circled by green) on the Hamiltonian path P, where each blue arrow points from a nonbreakpoint v to its previous vertex for accumulating distance $d_P(b, v)$. (c) Obtain the Steiner SPT T_B on $G[B \cup \{r\}]$, and get graph $T_M \cup T_B$. (d) Construct the spanning SPT on $T_M \cup T_B$, which is the desired SALT T ($\alpha = 1.90, \beta = 1.06$).

The Steiner SPT for connecting breaking points is a dedicated design (refer to [2, Sec. 2] for details). Applying it to a graph G' leads to the lightness bound $\bar{\beta} = 1 + 2\lceil \log n \rceil$. The algorithm starts by building a skeleton of a full balanced binary tree, of which the leaves are the original vertices and the inner nodes are Steiner points. From bottom to top, the edge weights are assigned carefully, to make sure the tree will be an SPT that dominates G.

ES is not complicated, but surprisingly, it leads to an exponentially lighter SLT than ABP. Besides, it is reasonably fast. The exact bounds are shown by Theorem 1.

Theorem 1: The ES algorithm generates a Steiner $(1 + 2\epsilon, 4 + 2\lceil \log \lceil 2/\epsilon \rceil \rceil)$-SLT in $O(n^2)$ time.

Proof: See [2, Lemmas 3.4–3.6]. ■

B. Framework

SALT first identifies some breakpoints on an initial topology and then connect them to the root by a Steiner SPT, which is similar to ES. Inspired by the KRY algorithm [30], we propose to use: 1) a tighter criterion for identifying breakpoints and 2) a better initial topology (i.e., an MST instead of a Hamiltonian path) in the SALT construction. The framework with the two effective techniques is illustrated by Algorithm 2 and Fig. 3 with additional notations summarized in Table IV. As a subroutine, the light Steiner SPT construction method will be described by Algorithm 3 in the next section.

In SALT, the solution is initialized to an MST and gradually modified toward an SALT. The major routine is based on a depth-first search on the MST (function `DFS`). During DFS, if the shallowness constraint is violated at a vertex, the vertex will become a breakpoint (line 9). In the end, breakpoints will be connected to r via an SPT, so its distance estimate $d[v]$ is set to the shortest-path distance $d_G(r, v)$ for relaxing the distance estimates of the other vertices (line 10). Two relaxations are conducted on each edge, from parent to child and from child to parent (lines 12 and 14). After DFS, edges $(v, p[v])$ for nonbreakpoints v define a forest F, with tree roots being breakpoints. In the end, breakpoints are connected to r by a Steiner SPT T_B.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
ADDITIONAL NOTATIONS USED IN SALT

$p[v]$	Parent of vertex v
$d[v]$	Current distance estimate from r to vertex v

Algorithm 2 SALT

Require: Graph $G = (V, E, w)$, root r, trade-off parameter ϵ;
Ensure: Steiner SLT $T = (V', E', w')$ with $V' \supseteq V$ that dominates G;
1: Initialize ($B \leftarrow \emptyset, d[r] = 0, \forall v \in V, d[v] = +\infty, p[v] = null$);
2: $T_M \leftarrow MST(G)$;
3: DFS(r, T_M);
4: Forest $F \leftarrow \{(v, p[v]) | v \in V \backslash (B \cup \{r\})\}$;
5: $T_B \leftarrow$ Steiner SPT rooted at r for $G[B \cup \{r\}]$ by Algorithm 3;
6: $T \leftarrow F \cup T_B$;
7: **function** DFS(v, T_M)
8: **if** $d[v] > (1 + \epsilon) \cdot d_G(r, v)$ **then**
9: $B \leftarrow B \cup \{v\}$;
10: $d[v] \leftarrow d_G(r, v)$;
11: **for** each child u of v in T_M **do**
12: RELAX(v, u);
13: DFS(u, T_M);
14: RELAX(u, v);
15: **function** RELAX(u, v)
16: **if** $d[v] > d[u] + w(uv)$ **then**
17: $d[v] \leftarrow d[u] + w(uv)$;
18: $p[v] \leftarrow u$;
19: **else if** $d[v] = d[u] + w(uv)$ and $w(p[v]v) < w(uv)$ **then**
20: $p[v] \leftarrow u$;

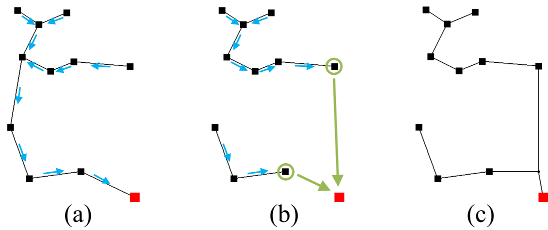

Fig. 3. Sample run of Algorithm 2 ($\epsilon = 1$). (a) Construct MST T_M, where each blue arrow points from a vertex v to its parent $p[v]$. (b) Update $p[v]$ and identify breakpoints B (circled by green) during the DFS on T_M, which results to a forest F with tree roots being B. (c) Obtain the Steiner SPT T_B on $G[B \cup \{r\}]$, and $T = F \cup T_B$ is the final SALT (shallowness $\alpha = 1.43$, lightness $\beta = 1.05$).

The relaxation (function `Relax`) from vertex u to v means updating distance estimate $d[v]$ if the path from r via u to v is shorter (line 16). Different from KRY, we also update the parent $p[v]$ of v even if $d[v]$ can not be shortened but its edge to the parent can becomes shorter (line 19). The latter situation actually frequently happens in Manhattan space and benefits the tree weight.

The two techniques mentioned above are detailed here. First, breakpoints are identified by checking distance estimate $d[v]$ instead of the accumulated distance $d_P(b, v)$ on the Hamiltonian cycle (in Algorithm 1 line 6). As a straightforward modification, $d[v]$ can be the sum of the shortest-path length $d_G(r, b)$ (from r to the previous breakpoint b) and the path length $d_P(b, v)$ (from b to v), which is an upper bound on $d_T(r, v)$ in the final T. More specifically, we can change the condition $d_P(b, v) > \epsilon \cdot d_G(r, v)$ to $d_G(r, b) + d_P(b, v) > (1 + \epsilon) \cdot d_G(r, v)$. Note that the value of $d[v]$ in Algorithm 2 is computed correctly by the relaxation steps before and after each recursive call (lines 12 and 14). Second, the initial topology is an MST instead of a Hamiltonian path. In this way, the distance estimate $d[v]$ is according to the MST, which is tighter than $d[v] = d_G(r, b) + d_P(b, v)$ based on the Hamiltonian path and can trigger fewer breakpoints. Note that in extreme cases, the second technique brings no benefit (e.g., MST is also a Hamiltonian path), but it does help in most practical cases.

TABLE V
ADDITIONAL NOTATIONS USED IN LIGHT STEINER SPT

$T(z)$	Subtree rooted at vertex z
$Leaves(z)$	Set of leaf vertices in T_z
$t(z)$	Distance from root r to vertex z in the SPT
$b(z_l, z_r)$	Disbalance between vertices z_l and z_r
$s(z_l, z_r)$	Distance surplus between vertices z_l and z_r
$c(z_l, z_r)$	Edge cost between vertices z_l and z_r
L	Vertex sequence
L_i	i-th vertex of L
$\|L\|$	Vertex number in $\|L\|$
v_i	i-th vertex along the traveling salesman circle
$f(z)$	First index of $Leaves(z) = \{v_{f(z)}, v_{f(z)+1}, ..., v_{l(z)}\}$
$l(z)$	Last index of $Leaves(z) = \{v_{f(z)}, v_{f(z)+1}, ..., v_{l(z)}\}$
$W(i, j)$	Length of path $(v_i, v_{i+1}, ..., v_j)$: $\sum_{k=i}^{j-1} d_G(v_k, v_{k+1})$
W'_i	Total weight of edges added in the i-th iteration

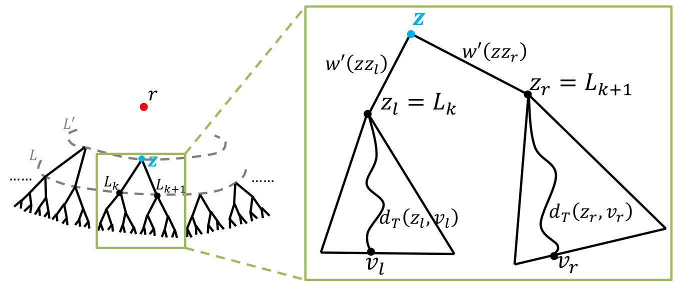

Fig. 4. During the Steiner SPT construction, neighboring vertices in L are merged pair by pair into Steiner vertices in L'. Shown by the enlarged the figure, vertices L_k (i.e., z_l) and L_{k+1} (i.e., z_r) are merged to a Steiner vertex z in L'.

C. Light Steiner Shortest-Path Tree

A light Steiner SPT can be constructed by Algorithm 3, which has smaller tree weight than that in the ES algorithm. Notations used are in Table V and Fig. 4.

Same as the Steiner SPT in ES, our SPT is also a full balanced binary tree, with leaves being the given vertices and inner nodes being Steiner vertices. Initially, the vertex sequence L contains all the given vertices. In each iteration of the main loop (lines 4–13), neighboring vertices are merged (i.e., connected to a parent Steiner vertex) pair by pair to form the vertex sequence L' for the next iteration. Note that the vertex number is reduced by half in each iteration and eventually becomes one.

When a Steiner vertex z is inserted as the parent for vertices z_l and z_r, the edge weights are assigned under the consideration of disbalance b and distance surplus s

$$b(z_l, z_r) = t(z_l) - t(z_r) \quad (1)$$

$$s(z_l, z_r) = \max\{d_G(v_l, v_r) - d_T(z_l, v_l) - d_T(z_r, v_r) \mid v_l \in \text{Leaves}(z_l), v_r \in \text{Leaves}(z_r)\} \quad (2)$$

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

Algorithm 3 Light Steiner SPT

Require: Graph $G = (V, E, w)$, root r;
Ensure: Steiner SPT $T = (V', E', w')$ with $V' \supseteq V$ that dominates G;
1: Initialize ($V' \leftarrow V, E' \leftarrow \emptyset, \forall v \in V, t(v) \leftarrow d_G(r, v)$);
2: $L \leftarrow$ Hamiltonian circle based on $MST(G)$ ($L_{n+1} = L_1$);
3: **while** $|L| > 1$ **do**
4: **for** $k = 1$ to n **do**
5: Calculate $b(L_k, L_{k+1}), s(L_k, L_{k+1})$ by (1) (2);
6: $c(L_k, L_{k+1}) \leftarrow \max\{s(L_k, L_{k+1}), |b(L_k, L_{k+1})|\}$;
7: $M_L \leftarrow$ a light perfect (or near perfect) matching on the circle defined by L and c;
8: $L' \leftarrow$ empty vertex sequence;
9: **for** $L_k L_{k+1} \in M_L$ **do**
10: ADDSTEINER(L_k, L_{k+1});
11: **if** $|L|$ is odd **then**
12: Append the unmatched vertex to L';
13: $L \leftarrow L'$;
14: **function** ADDSTEINER(z_l, z_r)
15: Add a Steiner vertex z into V';
16: Add edges zz_l and zz_r into E';
17: **if** $|b(z_l, z_r)| \leq s(z_l, z_r)$ **then**
18: $w'(zz_l) \leftarrow \frac{s(z_l,z_r)+b(z_l,z_r)}{2}$;
19: $w'(zz_r) \leftarrow \frac{s(z_l,z_r)-b(z_l,z_r)}{2}$;
20: **else**
21: $w'(zz_l) \leftarrow \max\{b(z_l, z_r), 0\}$;
22: $w'(zz_r) \leftarrow \max\{-b(z_l, z_r), 0\}$;
23: $t(z) \leftarrow d_G(r, v) - d_T(z, v)$ for an arbitrary $v \in Leaves(z)$;
24: Append z to L';

where $t(z)$ is the distance from root r to vertex z in the final SPT. It is obvious that $t(z) = d_G(r, z)$ if z is a leaf. t and b help maintain T to be an SPT and require the choice of edge weights $w'(zz_l)$ and $w'(zz_r)$ to satisfy

$$w'(zz_l) - w'(zz_r) = b(z_l, z_r). \tag{3}$$

In this way, $t(z_l) = t(z) + w'(zz_l)$ and $t(z_r) = t(z) + w'(zz_r)$ can be true at the same time. For distance surplus s, $w'(zz_l)$, and $w'(zz_r)$ should satisfy

$$w'(zz_l) + w'(zz_r) \geq s(z_l, z_r). \tag{4}$$

This guarantees $d_G(v_l, v_r) \leq d_T(z_l, v_l) + w'(zz_l) + w'(zz_r) + d_T(z_r, v_r) = d_T(v_l, v_r)$ (i.e., T dominates G). Algorithmic details are in function `AddSteiner`. Note that in line 23, arbitrary $v \in$ Leaves(z) can be picked to calculate $t(z)$ due to the following lemma.

Lemma 1: In Algorithm 3, for any vertex z in T and any vertex $v \in$ Leaves(z), $d_G(r, v) - d_T(z, v)$ is a constant.

Proof: See [2, Lemma 2.2]. ∎

Unlike the ES algorithm, which first determines the full-tree topology based on a Hamiltonian path and then assigns weight to the edges, our algorithm calculates the edge cost $c(L_k, L_{k+1})$ along L at each level and selects a good matching M_L to add Steiner vertices. According to the function `AddSteiner`, if a Steiner point z is inserted, the sum $c(z_l, z_r)$ of the weights of the two edges added will be

$$c(z_l, z_r) = w'(zz_l) + w'(zz_r) = \max\{|b(z_l, z_r)|, s(z_l, z_r)\}. \tag{5}$$

Since a cycle of even (resp. odd) number of edges can be decomposed into two perfect (resp. near perfect) matching, the weight of the lighter one will be no more than half of the cycle weight. In this way, the sum of the weights of the added edges is bounded.

Another technique that we use is to include the root r into the initial Hamiltonian circle. In this way, an edge between the final Steiner point and r is avoided and saved.

The resulted tree T is an SPT, of which the proof is simple and is similar to that in [2].

D. Bound Analysis

We first analyze the lightness β of the Steiner SPT generated by Algorithm 3.

Lemma 2: In Algorithm 3, for any vertex z in T, there exist $v_i, v_j \in$ Leaves(z), such that $d_T(z, v_i) + d_T(z, v_j) = d_G(v_i, v_j)$.

Proof: See Appendix A. ∎

The next lemma is the key to our weight analysis, which shows that the weight of the circle defined by L and c is bounded by the weight of the initial Hamiltonian cycle $W(1, n+1) = \sum_{k=1}^{n} d_G(v_k, v_{k+1})$.

Lemma 3: For the vertex sequence L in any iteration of Algorithm 3, $\sum_{k=1}^{|L|-1} c(L_k, L_{k+1}) \leq W(1, n+1)$.

Proof: See Appendix B. ∎

Lemma 4: In the ith iteration of Algorithm 3, the total weight of added edges $W'_i \leq w(\text{MST}(G))$.

Proof: Due to the perfect (or near perfect) matching used and Lemma 3, $W'_i \leq (1/2) \cdot \sum_{k=1}^{|L|-1} c(L_k, L_{k+1}) \leq (1/2) \cdot W(1, n+1)$. Because of triangle inequality, $W(1, n+1) \leq 2 \cdot w(\text{MST}(G))$. By combining them, $W'_i \leq w(\text{MST}(G))$. ∎

With the help of Lemma 4, the lightness bound of Algorithm 3 can be easily proved to be $\bar{\beta} = \lceil \log n \rceil$. Note that the Steiner SPT in ES has $\bar{\beta} = 1 + 2\lceil \log n \rceil$, which is more than twice of ours.

Theorem 2: The Steiner SPT T generated by Algorithm 3 has lightness bound $\bar{\beta} = \lceil \log n \rceil$.

Proof: With $\lceil \log n \rceil$ iterations, $|L|$ can be reduced from n to 1. Therefore, $w(T) = \sum_{i=1}^{\lceil \log n \rceil} W'_i \leq \lceil \log n \rceil \cdot w(\text{MST}(G))$. ∎

We then analyze the bounds on shallowness α and lightness β of SALT. Two lemmas are first needed.

Lemma 5: In Algorithm 3, if $\sum_{v \in V \setminus \{r\}} d_G(r, v) \leq \theta \cdot \eta$ ($\theta \geq 1, \eta > 0$), then $w(T) \leq \lceil \log \theta \rceil \cdot w(\text{MST}(G)) + \eta$.

Proof: See Appendix C. ∎

Lemma 6: In SALT, $\sum_{v \in B} d_G(r, v) \leq [2/\epsilon] \cdot w(\text{MST}(G))$.

Proof: See [30, Lemma 3.2]. ∎

According to Lemma 6, KRY, which connects breakpoints to root r by edges directly, leads to a spanning $(1+\epsilon, 1+[2/\epsilon])$-SLT. Introducing Steiner points by Algorithm 3 makes the bound tighter.

Theorem 3: SALT generates a Steiner $(1 + \epsilon, 2 + \lceil \log [2/\epsilon] \rceil)$-SLT.

Proof: Whenever $d[v]$ of a vertex v exceeds $(1+\epsilon)$ times its shortest-path length $d_G(r, v)$, $d[v]$ is set to $d_G(r, v)$ and fixed. Therefore, we have shallowness $\alpha \leq 1 + \epsilon$.

Since T_B is a Steiner SPT on graph $G[B \cup \{r\}]$, substituting $\theta = [2/\epsilon]$ and $\eta = w(\text{MST}(G))$ (by Lemma 6) into Lemma 5 makes $w(T_B) \leq (1 + \lceil \log [2/\epsilon] \rceil) \cdot w(\text{MST}(G))$. Besides, $w(F) \leq w(\text{MST}(G))$ because $F \subset \text{MST}(G)$. Hence, $w(T) = w(T_B) + w(\text{MST}(G)) \leq (2 + \lceil \log [2/\epsilon] \rceil) \cdot w(\text{MST}(G))$. ∎

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

Algorithm 4 Rectilinear SALT

Require: Points V on Manhattan plane, root r;
Ensure: Rectilinear Steiner SLT $T = (V', E')$ with $V' \supseteq V$;
1: Initialize ($B \leftarrow \emptyset, d[r] = 0, \forall v \in V, d[v] = +\infty, p[v] = null$);
2: $T_M \leftarrow$ RSMT on V by FLUTE;
3: DFS(r, T_M);
4: Forest $F \leftarrow \{(v, p[v])|v \in V\backslash(B \cup \{r\})\}$;
5: $T_B \leftarrow$ RSMA rooted at r on $B \cup \{r\}$ by CL;
6: $T \leftarrow F \cup T_B$;
7: **function** DFS(v, T_M)
8: **if** $v \in V$ and $d[v] > (1+\epsilon) \cdot d_G(r, v)$ **then**
9: $B \leftarrow B \cup \{v\}$;
10: $d[v] \leftarrow d_G(r, v)$;
11: **for** each child u of v in T_M **do**
12: RELAX(v, u);
13: DFS(u, T_M);
14: RELAX(u, v);

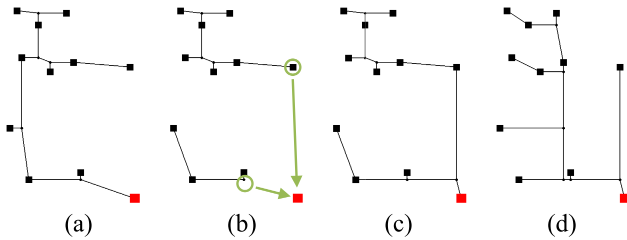

Fig. 5. Sample run of Algorithm 4 ($\epsilon = 1$). (a) Construct RSMT T_M by FLUTE (shallowness $\alpha = 2.66$, lightness $\beta = 0.91$). (b) Get breakpoints B (circled by green) and forest F. (c) Obtain the RSMA T_B on $G[B \cup \{r\}]$ by CL and $T = F \cup T_B$ is the rectilinear SALT ($\alpha = 1.22, \beta = 1.01$). (d) RSMA by CL on the net ($\alpha = 1, \beta = 1.11$).

III. RECTILINEAR STEINER SHALLOW-LIGHT TREE ALGORITHM

SALT, which generates a Steiner $(1+\epsilon, 2+\lceil\log[2/\epsilon]\rceil)$-SLT for a general graph, can be directly applied in the Manhattan space. However, it can be enhanced with the help of some special properties as well as classical algorithms. The resulted algorithm, rectilinear SALT, is shown by Algorithm 4 and Fig. 5. W.l.o.g., we assume that the root r is at the origin of the space.

First of all, to build a rectilinear Steiner SPT, adding a Steiner point to merge two vertices (function `AddSteiner` in Algorithm 3) becomes easier on Manhattan plane. In the following discussion, we focus on the 2-D situation, but it can be extended to higher dimensions. For two vertices $z_l = (x_{z_l}, y_{z_l})$ and $z_r = (x_{z_r}, y_{z_r})$, the $\mathbf{x}$ coordinate of their parent Steiner point z is

$$x_z = \begin{cases} \min\{x_{z_l}, x_{z_r}\}, & x_{z_l}, x_{z_r} \geq 0 \\ \max\{x_{z_l}, x_{z_r}\}, & x_{z_l}, x_{z_r} \leq 0 \\ 0, & x_{z_l} \cdot x_{z_r} < 0. \end{cases} \tag{6}$$

y_z is computed similarly. This location assignment of z is determined by distances $w'(zz_l)$, $w'(zz_r)$, and $t(z)$. Note that the case $|b(z_l, z_r)| > s(z_l, z_r)$ (Algorithm 3 lines 20–22) never happens now. Intuitively, such Steiner point z maximizes the overlapping of the two shortest paths from r to vertices z_l and z_r. In this way, the coordinate of z can be directly obtained from locations of z_l and z_r, which avoids the checking of all leaves of z_l and z_r in (2). Therefore, the time complexity is now bounded by obtaining the MST and is improved to $O(n \log n)$.

Second, the Steiner SPT problem in Manhattan space is exactly the classical RSMA problem [20], [23]. The CL heuristics [24] is an approximation algorithm produces a tree of weight at most twice the optimal. In practice, it is mostly optimal or near optimal, and is very efficient with a time complexity of $O(n \log n)$. On the other hand, our light Steiner SPT algorithm with lightness $\beta \leq \lceil\log[2/\epsilon]\rceil$ may be far away from the optimal SPT in worst cases. For example, when all vertices locate on a straight line, the optimal SPT is a path and also the MST (i.e., $\beta = 1$). Hence, we use CL to construct the Steiner SPT to further reduce the tree weight in practice (Algorithm 4 line 5). Note that this modification maintain the proved complexity for both the quality (shallowness α and lightness β) and time of Algorithm 2. While the constant in the shallowness bound ($\bar{\alpha} = 1+\epsilon$) is also maintained, the constant in the lightness bound ($\bar{\beta} = 2+\lceil\log[2/\epsilon]\rceil$) may be slightly worsened in some corner cases but is better or much better in most cases.

Third, instead of starting from an MST in Algorithm 2, an initial tree with lighter weight is achievable by allowing Steiner points. In Manhattan space, RSMT is a well-investigated problem, and FLUTE [19] is adopted in our implementation (Algorithm 4 line 2). In this way, the bound on the tree weight $w(T)$ actually becomes tighter. There is still $w(T) \leq (2+\lceil\log[2/\epsilon]\rceil) \cdot w(T_M)$, where T_M is MST in Theorem 3 but now becomes RSMT. Note that different from Algorithm 2, the Steiner vertices in the RSMT do not need to be checked during the DFS (Algorithm 4 line 8).

By the above modifications, we reduce the lightness β of the SALT constructed and improve the time complexity to $O(n \log n)$. From another viewpoint, rectilinear SALT is a smooth tradeoff between RSMA and RSMT. The smaller the ϵ, the closer the rectilinear SALT is to an RSMA; the larger the ϵ, the closer it is to an RSMT. It is almost a CL RSMA when $\epsilon = 0$ and an FLUTE RSMT when $\epsilon = +\infty$. In the middle, it is a bounded tradeoff between them. To a certain extent, Fig. 5 illustrates the situation. The RSMT in Fig. 5(a) is the lightest but has some long paths, while the RSMA in Fig. 5(d) is the shallowest but is of a large tree weight. Combining the strengths of the both, the rectilinear SALT in Fig. 5(c) is not only light but also shallow.

IV. SAFE REFINEMENT

Three effective SR techniques are adopted to further improve rectilinear SALT, including intersected edge canceling (IEC), L-/Z-shape edge flipping (LEF), and U-shape edge shifting (UES). They are safe as they improve wirelength or path length or both without worsening any of them. For simplicity, rectilinear SALT will be referred as SALT hereafter.

A. Intersected Edge Canceling

In SALT, edges in RSMA T_B may intersect with edges in forest F, since T_B and F are constructed separately. Here, the *intersection* between two edges in the Manhattan space means that their bounding boxes intersect, which is illustrated by Fig. 6(a). For intersected edges v_3v_1 and v_4v_2, we can add a Steiner vertex z within the intersection box, connect

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

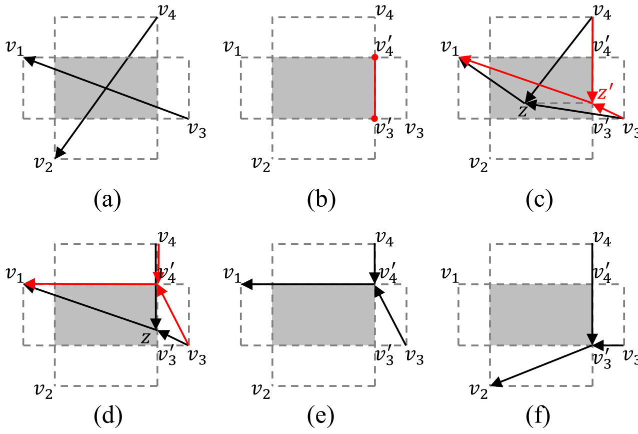

Fig. 6. IEC (arrows point to parents). (a) Intersection box (filled by gray). (b) Child corners v_3', v_4'. (c) z should be on edge v_3', v_4'. (d) z should be either v_3' or v_4'. (e) First solution. (f) Second solution.

child vertices v_3 and v_4 to it, and then connect it to either v_1 or v_2. By choosing the shorter path between $(z, v_1, \ldots, r)$ and $(z, v_2, \ldots, r)$, both path lengths and wirelength can be reduced. The question is where the best location for the Steiner vertex z is, and it can be answered by Theorem 4. Among the four corners of an intersection box, a *child corner* is the closest to a child vertex [e.g., v_3' and v_4' in Fig. 6(b)].

Theorem 4: For intersected edges, the optimal Steiner vertex z is a child corner of the intersection box.

Proof: First, z should be on a *child edge* (i.e., the edge between the two child corners). If not, its projected point z' on the child edge can improve the wirelength without impacting path lengths [Fig. 6(c)]. Supposing z is connected to v_1, there is $w(v_3z) + w(v_4z) + w(zv_1) = (w(v_3z') + w(z'z)) + (w(v_4z') + w(z'z)) + (w(z'v_1) - w(z'z)) \geq w(v_3z') + w(v_4z') + w(z'v_1)$.

When z is on the child edge but not a child corner, it can be improved by moving to a child corner [Fig. 6(d)]. Assume z is still connected to v_1. For wirelength, there is $w(v_3z) + w(v_4z) + w(zv_1) \geq w(v_3v_4') + w(v_4v_4') + w(v_4'v_1)$; for path lengths, there is $w(v_4z) + w(zv_1) \geq w(v_4v_4') + w(v_4'v_1)$, while $w(v_3z) + w(zv_1) = w(v_3v_4') + w(v_4'v_1)$.

The argument is similar if z is connected to v_2. In short, the optimal solution is a child corner (either v_3' or v_4') shown by Fig. 6(e) and (f). Note that, in some cases, the two child corners may merge into one, or the intersection box may even degenerates to a segment, but our discussion is generic. ∎

For a Steiner tree, we propose an iterative scheme for identifying and canceling all the intersected edges (Algorithm 5) based on R-tree [41]. Throughout the process, the major invariant is that the boxes in R-tree R do not intersect with each other. By iteratively examining boxes (lines 3–11), a new box r will be broken or shrank (due to the intersection canceling in Fig. 6) until all the intersections caused by it has been resolved. Regarding the running time of Algorithm 5, it is $O(n \log n)$ thanks to the $O(\log n)$-time query of R-tree and the $O(n)$ edges in total.

B. L-/Z-Shape Edge Flipping

Edges may be overlapped with each other by flipping (in L or Z shape) and thus improves wirelength and path lengths, as Figs. 7 and 8 show. Ho *et al.* [13] proposed a dynamic programming for edge flipping. The method is linear-time if the vertex degree is bounded, and generates optimal wirelength if only edge overlapping around a vertex is counted. We apply this technique. In SALT, the maximum vertex degree is the sum of that in FLUTE (four, according to [8]) and CL (four, considering the root), as an SALT T is the union of a FLUTE forest F and a CL RSMA T_B. Therefore, the vertex degree is bounded ($\leq 4 + 4 = 8$) and guarantees the $O(n)$ time. In our implementation, the optimal L-shape flipping is adopted, since the constant in the time complexity of the optimal Z-shape flipping is quite large. The Z-shape flipping can be achieved by iterative L-shape flipping, which is demonstrated by Fig. 8.

Algorithm 5 IEC

Require: Tree T;
Ensure: Tree T' without intersected edges;
1: Queue $Q \leftarrow$ bounding boxes of all edges in T;
2: R-tree $R \leftarrow \emptyset$;
3: **while** Q is not empty **do**
4: Box $r \leftarrow$ dequeue Q;
5: Search for a box r' in R that intersects with r;
6: **if** there is such r' **then**
7: Delete r' from R;
8: Cancel the intersection between r and r';
9: Enqueue newly generated edges to Q;
10: **else**
11: Insert r to R;

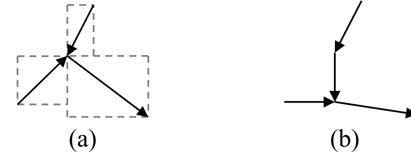

Fig. 7. L-shape edge flipping. (a) Input. (b) Output.

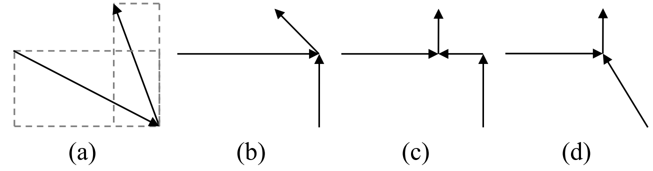

Fig. 8. Z-shape edge flipping by iterative L-shape flipping. (a) Input. (b) First L-shape flipping. (c) Second L-shape flipping (i.e., a Z-shape flipping). (d) Removing redundant Steiner vertex.

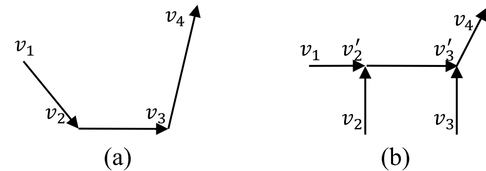

Fig. 9. (Canonical) UES. (a) Input. (b) Output.

C. U-Shape Edge Shifting

The UES is proposed by Boese *et al.* [42]. It is beneficial not only to wirelength and path lengths but also to Elmore delay. An example is in Fig. 9, where the edge v_2v_3 is shifted

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

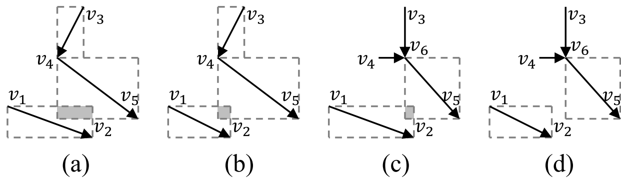

Fig. 10. LEF may make the edge intersection shrank or even disappeared. (a) Case where edge v_1v_2 intersects with edge v_3v_4. (b) Another case where edge v_1v_2 intersects with edge v_3v_4. (c) After L-shape edge flipping on (a), the edge intersection shrinks. (d) After L-shape edge flipping on (b), the edge intersection disappears.

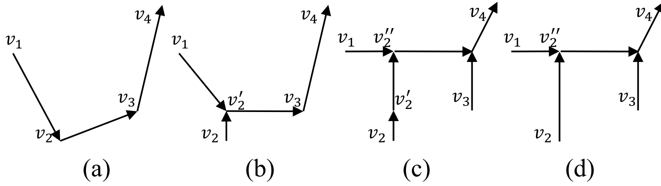

Fig. 11. General UES. (a) Input. (b) LEF. (c) Canonical UES. (d) Removing redundant Steiner vertex.

to $v'_2v'_3$. UES can be performed during a tree traversal. It takes $O(n)$ time due to the bounded vertex degree in SALT.

D. Order of Safe Refinement Techniques

In our implementation, the three SR techniques are performed in the following order: 1) IEC; 2) LEF; and 3) UES. It is based on two considerations.

First, among the three SR methods, IEC changes topologies more globally and significantly, while the other two methods work on topologically neighbored edges only. Meanwhile, the other two methods may influence the solution space of IEC. For example in Fig. 10, after LEF, the previous edge intersection may shrink [Fig. 10(b)] or disappear [Fig. 10(d)], which means less or even missed improvement.

Second, a general UES can be decomposed into LEF and a canonical UES (Fig. 11). In a canonical U-shape path, the middle edge (e.g., edge v_2v_3 of path $v_1v_2v_3v_4$ in Fig. 9) is strictly horizontal or vertical. Therefore, conducting LEF first avoids handling the many corner cases and thus eases the implementation of general UES.

V. Shallowness-Constrained Edge Substitution

Compared with SR, shallowness-constrained edge substitution (SCES) is more aggressive in wirelength minimization. It allows slight path length degradation but make it under control by constraining the shallowness.

Edge substitution is an effective technique for constructing RSMT [16], [17], where a *target vertex* (e.g., v_i in Fig. 12) is considered for connecting to a nearby *candidate edge* ($v_jv'_j$ in Fig. 12). The idea can be brought to SLTs, but the consideration in shallowness besides lightness poses a great limit on the solution space. To minimize wirelength in RSMT construction, it simply requires removing the longest edge along the circle formed by the new edge. But this could incur huge influence on the path lengths. Because not only can the path lengths of many vertices be degraded (if a vertex has longer path to the root, all its descendants suffer), but also the directions of edges may be reversed. SCES is thus proposed. Here,

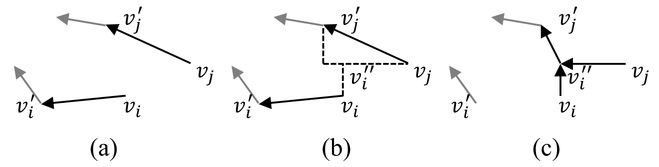

Fig. 12. SCES. (a) Before SCES, v_i is the target vertex with v'_i being its parent, while $v_jv'_j$ is the candidate edge. (b) v''_i is the closest point to v_i within the bounding box of edge $v_jv'_j$. (c) After SCES, edge $v_iv'_i$ is substituted by edges $v_iv''_i$, $v_jv''_i$, and $v''_iv'_j$.

Algorithm 6 SCES

Require: Tree $T(V, E)$;
Ensure: Refined tree T';
1: Compute $slack(T(v))$ for $v \in V$ (by two tree traversals);
2: Query candidate edges for V (by nearest neighbors or R-tree);
3: **for** $v_i \in V$ **do**
4: Best edge index $k \leftarrow null$
5: Best wirelength change $\Delta WL^* \leftarrow 0$
6: **for** each candidate edge $v_jv'_j$ of v_i **do**
7: Continue if $v_j \in T(v_i)$;
8: $v''_i \leftarrow$ closest point to v_i in the bounding box of $v_jv'_j$;
9: Wirelength change $\Delta WL \leftarrow d_G(v_i, v''_i) - d_G(v_i, v'_i)$;
10: Path length change $\Delta PL \leftarrow d_T(r, v'_j) + d_G(v'_j, v_i) - d_T(r, v'_i)$;
11: **if** $\Delta WL < \Delta WL^*$ and $\Delta PL < slack(T(v_i))$ **then**
12: $\Delta WL^* \leftarrow \Delta WL$;
13: $k \leftarrow j$;
14: **if** $\Delta WL^* < 0$ **then**
15: Disconnect $v_iv'_i$, connect $v_iv''_i$, $v_kv''_i$ and $v''_iv'_k$;
16: Update $slack(T(u))$ for vertex u in $T(v_i)$ and path to r;

for a tree resulted by running SALT with ϵ, its shallowness α after SCES will be still under $1 + \epsilon$.

In SCES, the *substituted edge* is restricted to be the parent edge of the target vertex only [e.g., edge $v_iv'_i$ in Fig. 12(b) and (c)] due to two reasons. First, reversing edges tends to cause detour and thus shallowness violation. Second, for each of the $O(n)$ possible substituted edges along the circle, $O(n)$ vertices may have path lengths affected, leading to high computation cost for a single pair of target vertex and candidate edge.

Algorithm 6 shows the details of the proposed SCES. In order to efficiently check whether an edge substitution violates shallowness constraint, path length $d_T(r, v)$ for each vertex v is precomputed by a preorder traversal. Slack slack(v) of each vertex v and slack$(T(v))$ of the subtree $T(v)$ rooted at v are then computed by a post-order traversal followed (line 1):

$$\text{slack}(v) = (1+\epsilon)\cdot d_G(r,v) - d_T(r,v) \quad (7)$$

$$\text{slack}(T(v)) = \min_{u\in T(v)} \text{slack}(u). \quad (8)$$

In this way, for a target vertex v_i, if a candidate substitution increases its path length by ΔPL, its legality means ΔPL $<$ slack$(T(v_i))$ (line 11). Among all the candidate edges of a vertex v_i, the one that legally saves most wirelength will be connected to v_i by a Steiner point [v''_i in Fig. 12(c)]. Note that a legal candidate edge $v_jv'_j$ cannot be in $T(v_i)$ (line 7), which will otherwise make the tree disconnected. For a good order of visiting vertices (line 3), Algorithm 6 can be run twice in different modes. The first run calculates the wirelength

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

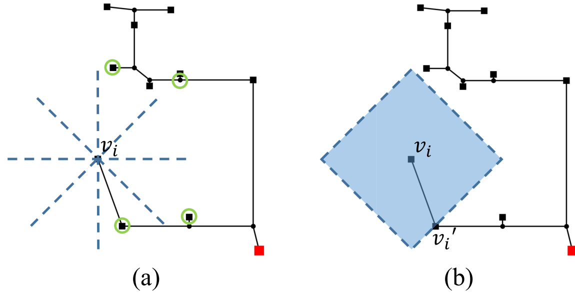

Fig. 13. Two ways to find the candidate edges for SCES: (a) NN in each octant (marked by green) and (b) R-tree query by an Manhattan circle.

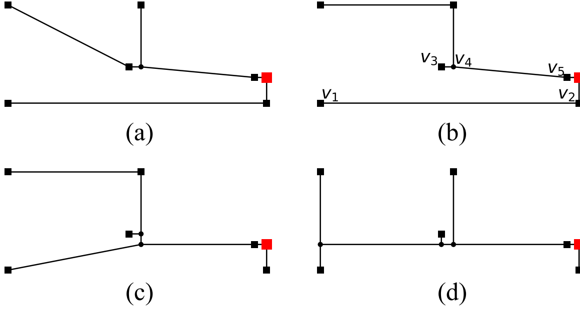

Fig. 14. Insufficiency of SCES based on nearest neighbors. Assume that a small ϵ (e.g., 0.05) is used. (a) Input tree. (b) Only SCES that can be achieved by considering nearest neighbors. (c) SCES achieved by using R-tree. (d) Final result of iterative SCES.

improvement under the input topology T, while the second processes the edge substitutions in the order of descending improvement and commits those that are still legal.

Now the only problem left is how to efficiently identify candidate edges (line 2). The first way is to exploit the geometrical proximity information embedded in the *spanning graph* [12], similar to what Zhou [17] does for RSMT. That is, consider the edges connected to the nearest neighbor (NN) vertex of the target vertex in each octant [Fig. 13(a)]. Therefore, candidate edges are in a total number of $O(n)$ and can be obtained in $O(n \log n)$ time [17].

The second way adopts an R-tree [41], which stores the bounding boxes of all the edges. For a target vertex v_i with parent v_i', we query candidate edges by the Manhattan circle centered at v_i and with a radius of $d_G(v_i, v_i')$ [Fig. 13(b)]. Here, an edge outside the Manhattan circle is unable to save wirelength. Compared with using nearest neighbors, querying by R-tree has two strengths. First, it never misses any candidate that can reduce wirelength. Meanwhile, a "good" candidate edge may be blocked by other vertices in the spanning graph. For example, for the tree in Fig. 14(b), SCES that connects target vertex v_1 to candidate edge v_4v_5 can lead to an improved tree [Fig. 14(c)]. Note that connecting v_1 to edge v_3v_4 also reduces wirelength, but it causes detour and may violate the shallowness constraint for the path from v_1 to the root. Here, the method of nearest neighbors cannot identify edge v_4v_5 as a candidate for v_1 because v_4 is blocked by v_3 in the spanning graph. With the help of R-tree query, the candidate edge v_4v_5, however, can be easily obtained. Second, R-tree usually results fewer candidate edges and saves runtime, especially for vertex with shorter parent edge (recall Fig. 13). Therefore, R-tree-based SCES is used in our default flow. Besides, it can be iterated to accumulate wirelength improvement [Fig. 14(d)].

Besides [16] and [17], SCES also recalls the detour-aware Steinerization (DAS) in PD-II [5]. DAS also restricts the substituted edge to the parent edge of the target vertex. However, SCES and DAS have twofold differences. First, DAS uses nearest neighbors to identify candidate edges. Even though the NN graph in DAS is not exactly the spanning graph, it also suffers from the two aforementioned problems. Second, it only constrains the path length degradation on the target vertex v_i, without considering its impact to the downstream vertices [i.e., $d_T(r, v_i) \leq 0.5 \cdot \max_{u \in V} d_T(r, u)$ instead of our $\Delta\text{PL} \leq \text{slack}(T(v_i))$]. That is, the path length of a vertex may be degraded several times due to its ancestors without constraint on such accumulation.

SCES is performed after SR due to two reasons. First, SR mostly reduces path length and never degrades path length, which slacks the shallowness constraint for SCES. Second, SCES by R-tree is a generalization of IEC and L-/Z-shape edge substitution. It explores a larger solution space but also requires more runtime. Conducting SR first will trigger SCES fewer times and save the total runtime.

TABLE VI
ICCAD 2015 BENCHMARK STATISTICS

Design	# cells ($\times 10^3$)	# nets classified by pin number ($\times 10^3$)						
		2	3	4-7	8–15	16–31	≥ 32	≥ 4
superblue1	1932	893	146	121	30	19	5.6	176
superblue3	1876	952	113	87	38	28	6.0	160
superblue4	796	610	88	63	20	18	3.3	104
superblue5	982	824	119	115	20	15	4.5	154
superblue7	768	1493	184	134	59	53	10.4	257
superblue10	1087	1457	238	129	40	26	9.0	204
superblue16	1213	756	99	103	23	13	4.5	144
superblue18	1210	575	101	47	22	22	4.8	96
Total	9863	7559	1087	800	253	194	48	**1295**

VI. EXPERIMENTAL RESULTS

We implement SALT as well as ES [2], CL [24], ABP/BRBC [28], [29], KRY [30], PD [32], and Bonn [37] algorithms in C++, while the source code of FLUTE [19] is obtained from the authors. For a low-degree net, the idea of FLUTE has been extended to generate all the RSMTs instead of just one [43]. Among all the RMSTs, the shallowest one can be selected to serve as a better reference. We obtain the look-up table files from the authors. Moreover, the results of PD-II [5][3] are provided by the authors.

Benchmarks of the ICCAD 2015 Contest [44] are used for a comprehensive evaluation and comparison. The benchmark statistics are shown in Table VI. By ignoring 2-pin and 3-pin nets, which are trivial, the batch test covers around 1.3 million nets in total. Experiments are performed on a 64-bit Linux workstation with Intel Xeon 3.4 GHz CPU and 32 GB memory. A single thread is used for simplicity, in spite that different nets can be routed with SALT in parallel.

[3]Here, PD-II denotes the complete flow in [5]. It is the PD construction followed by the spanning tree refinement, Steinerization, the Steiner tree refinement, and a meta-heuristic. The meta-heuristic runs FLUTE in parallel. If FLUTE is better in both wirelength and path lengths, it is output. In the original paper, PD-II stands for PD with spanning tree refinement.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

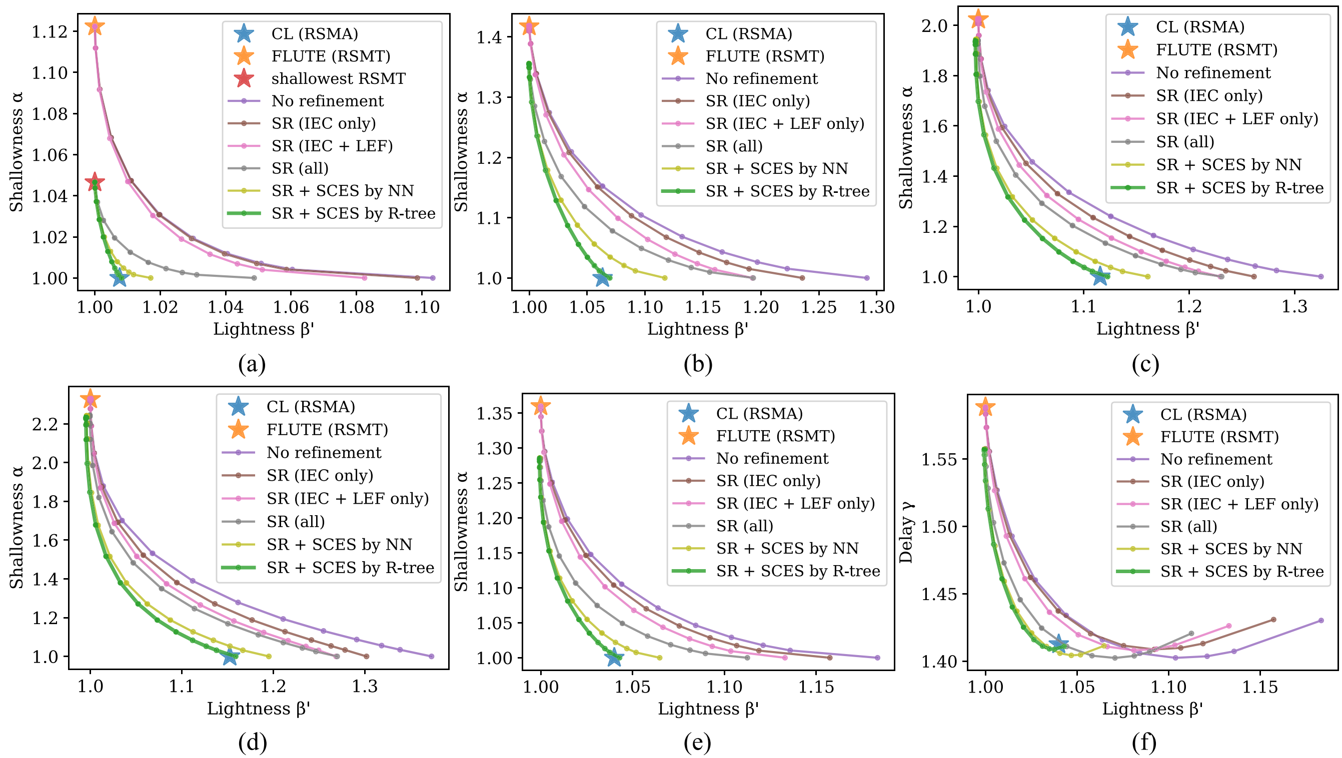

Fig. 15. Effectiveness of post processing shown by shallowness–lightness and delay-lightness tradeoff on nets of various scales. (a) Shallowness–lightness on nets with 4–7 pins. (b) Shallowness–lightness on nets with 8–15 pins. (c) Shallowness–lightness on nets with 16–31 pins. (d) Shallowness–lightness on nets with 32+ pins. (e) Shallowness–lightness on all nets. (f) Delay-lightness on all nets.

In the batch test, ϵ is set to 20 values ranging from 0 to 73.895 (mainly a geometric sequence 0.05×1.5^i) to cover the variation of different methods. The lightness metric is changed to $\beta' = [w(T)/w(\text{FLUTE})]$ (instead of $\beta = [w(T)/w(\text{MST})]$), where FLUTE serves as a tighter baseline than MST. Besides, a normalized Elmore delay metric γ, which assumes uniform unit-length capacitance and resistance, is also used. For each routing tree, *delay* γ is the longest Elmore delay among all source-sink paths, which is then normalized by a delay lower bound using the method in [37]. For each method and each ϵ, we average the scores over all the nets.

A. Effectivenss of Post Processing

Table VII and Fig. 15 show the effectiveness of our post-processing techniques, SR and SCES. The contributions of the three SR techniques, IEC, LEF, and UES, are all shown. Performance of both implementation of SCES, by NN and by R-tree, is also presented.

As Table VII shows, SR simultaneously improves α, β', and γ for every ϵ. Meanwhile, for a given tree, SCES gives large improvement on lightness by possibly slightly sacrificing shallowness (and delay). For example, when $\epsilon = 0.05$, "SR + SCES by R-tree" reduces the lightness of SR by 4.8% (from 1.0897 to 1.0376), with shallowness only increased by 0.14% (from 1.0062 to 1.0076). In general, it is obvious from Fig. 15 that the Pareto frontiers are pushed toward the origin by both SR and SCES. Regarding the two kinds of implementation of SCES, R-tree achieves larger wirelength savings than NN due to its more comprehensive scope.

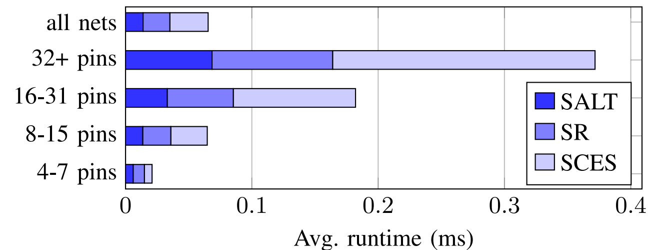

Fig. 16. Runtime breakdown of SALT with post processing.

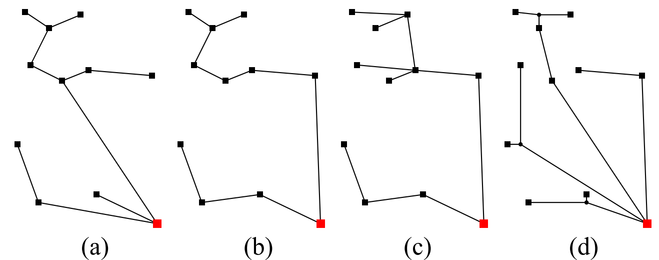

Fig. 17. Sample runs of various algorithms ($\epsilon = 1$). (a) ABP/BRBC (shallowness $\alpha = 2.24$ and lightness $\beta' = 1.51$). (b) KRY ($\alpha = 1.43$, $\beta' = 1.22$). (c) PD ($\alpha = 1.11$, $\beta' = 1.30$). (d) Bonn ($\alpha = 1.22$, $\beta' = 1.87$).

For nets with various pin numbers, the shallowness and lightness gaps between RSMA and RSMT are enlarged as net scales increase [see Fig. 15(a)–(d)]. SALT with "SR + SCES by R-tree," however, can always deliver a smooth tradeoff between RSMA and RSMT. For low-degree nets, the shallowest RSMT achieves much better α than FLUTE

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

TABLE VII
EFFECTIVENESS OF POST PROCESSING

Trade-off parameter	No refinement			SR (IEC only)			SR (IEC + LEF only)			SR (all)			SR + SCES by NNs			SR + SCES by R-tree		
ϵ	Lightness β'	Shallowness α	Delay γ	Lightness β'	Shallowness α	Delay γ	Lightness β'	Shallowness α	Delay γ	Lightness β'	Shallowness α	Delay γ	Lightness β'	Shallowness α	Delay γ	Lightness β'	Shallowness α	Delay γ
0.000	1.1834	1.0000	1.4302	1.1574	1.0000	1.4309	1.1330	1.0000	1.4262	1.1125	1.0000	1.4206	1.0647	1.0000	1.4114	**1.0429**	**1.0000**	**1.4110**
0.050	1.1358	1.0105	1.4073	1.1188	1.0102	1.4131	1.1035	1.0095	1.4119	1.0897	**1.0062**	1.4070	1.0519	1.0076	**1.4047**	**1.0376**	1.0076	1.4088
0.075	1.1211	1.0178	**1.4036**	1.1067	1.0174	1.4099	1.0934	1.0163	1.4090	1.0812	**1.0110**	1.4039	1.0469	1.0132	1.4043	**1.0349**	1.0131	1.4091
0.113	1.1038	1.0291	1.4025	1.0922	1.0286	1.4087	1.0810	1.0271	1.4080	1.0707	**1.0187**	**1.4024**	1.0407	1.0220	1.4058	**1.0312**	1.0219	1.4111
0.169	1.0844	1.0463	1.4058	1.0754	1.0455	1.4115	1.0665	1.0436	1.4108	1.0582	**1.0309**	**1.4040**	1.0333	1.0354	1.4109	**1.0265**	1.0353	1.4160
0.253	1.0638	1.0711	1.4159	1.0574	1.0701	1.4204	1.0507	1.0678	1.4196	1.0444	**1.0492**	**1.4109**	1.0252	1.0549	1.4209	**1.0208**	1.0547	1.4253
0.380	1.0441	1.1053	1.4340	1.0397	1.1040	1.4372	1.0349	1.1016	1.4363	1.0307	**1.0748**	**1.4246**	1.0172	1.0814	1.4371	**1.0147**	1.0812	1.4402
0.570	1.0272	1.1478	1.4602	1.0246	1.1464	1.4621	1.0215	1.1441	1.4612	1.0190	**1.1068**	**1.4456**	1.0103	1.1140	1.4593	**1.0090**	1.1141	1.4611
0.854	1.0145	1.1985	1.4927	1.0131	1.1971	1.4937	1.0114	1.1951	1.4928	1.0101	**1.1456**	**1.4729**	1.0052	1.1526	1.4855	**1.0045**	1.1530	1.4868
1.281	1.0064	1.2509	1.5269	1.0058	1.2499	1.5272	1.0050	1.2485	1.5266	1.0044	**1.1873**	**1.5028**	1.0019	1.1934	1.5121	**1.0015**	1.1938	1.5131
1.922	1.0022	1.2952	1.5556	1.0020	1.2945	1.5556	1.0017	1.2938	1.5553	1.0014	**1.2252**	**1.5283**	1.0002	1.2295	1.5334	**1.0000**	1.2298	1.5340
2.883	1.0006	1.3244	1.5735	1.0006	1.3242	1.5735	1.0005	1.3239	1.5734	1.0003	**1.2530**	**1.5445**	0.9996	1.2549	1.5460	**0.9995**	1.2545	1.5459
4.325	1.0001	1.3448	1.5834	1.0001	1.3448	1.5833	1.0001	1.3447	1.5833	1.0000	**1.2721**	1.5532	0.9995	1.2731	1.5534	**0.9994**	1.2722	**1.5530**
6.487	1.0000	1.3554	1.5873	1.0000	1.3554	1.5873	1.0000	1.3553	1.5872	0.9999	1.2815	1.5566	0.9994	1.2822	1.5565	**0.9994**	**1.2808**	**1.5559**
9.731	1.0000	1.3592	1.5884	1.0000	1.3591	1.5884	1.0000	1.3591	1.5883	0.9999	1.2848	1.5575	0.9994	1.2854	1.5573	**0.9994**	**1.2837**	**1.5568**
14.597	1.0000	1.3598	1.5885	1.0000	1.3598	1.5885	1.0000	1.3598	1.5885	0.9999	1.2853	1.5576	0.9994	1.2858	1.5574	**0.9994**	**1.2841**	**1.5569**
...	1.0000	1.3598	1.5885	1.0000	1.3598	1.5885	1.0000	1.3598	1.5885	0.9999	1.2853	1.5577	0.9994	1.2858	1.5574	**0.9994**	**1.2841**	**1.5569**

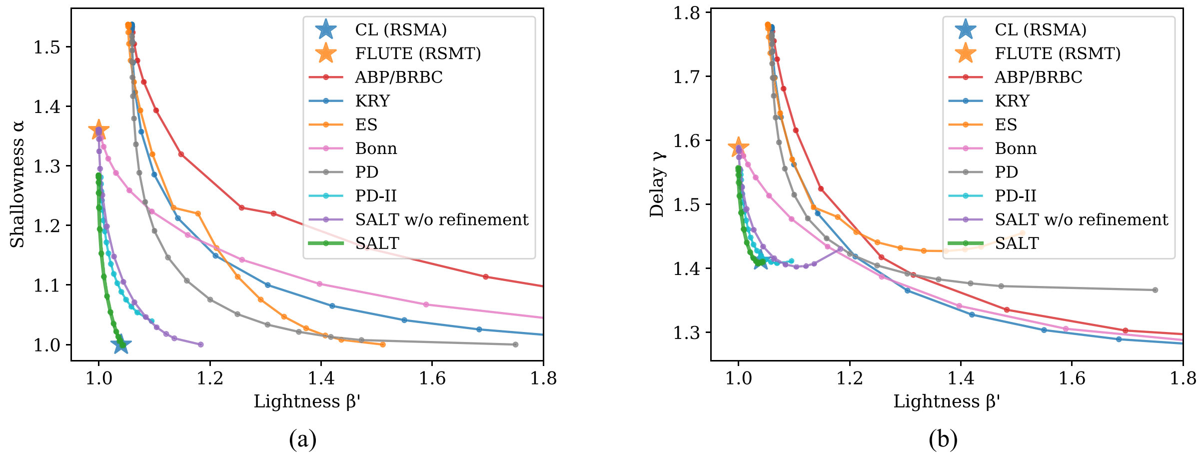

Fig. 18. Comparing SALT with other routing tree construction methods. (a) Tradeoff between shallowness and lightness. (b) Tradeoff between delay and lightness.

by enumerating all RSMTs [Fig. 15(a)]. However, after post processing, SALT obtains almost the same α even when β' is the minimum. Note that this is achieved without the time-consuming enumeration.

Fig. 15(e) and (f) summarize the shallowness–lightness and delay-lightness tradeoff for all nets. Note that as lightness β increases, delay γ first decreases and then slightly goes up. The reason is that larger β causes higher load capacitance for the driving cell and thus more cell delay.

SALT is also very efficient. The runtime breakdown is shown in Fig. 16. An $O(n \log n)$ runtime growth (with respect to net scales) can be observed. Moreover, for the 1.3 million nets in the ICCAD 2015 benchmark, SALT with post processing spends 0.0654 ms for each net on average. That is, it finishes routing all the eight benchmarks in 1.42 min with a single thread under an ϵ.

B. Superiority Over Other Methods

First of all, to give the readers some understanding of other routing tree construction methods, sample runs on the example net are shown in Fig. 17. Tradeoff parameter ϵ is set to 1. Recall that it implies a shallowness–lightness bound of

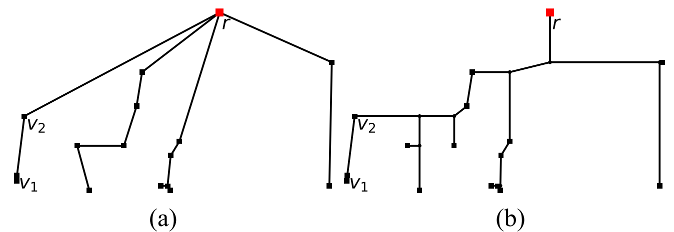

Fig. 19. Comparing SALT with KRY on a 16-pin net of superblue1. The tradeoff parameter ϵ leading to the smallest delay γ is picked. (a) KRY ($\beta' = 1.572, \gamma = 1.292$). (b) SALT ($\beta' = 1.098, \gamma = 1.994$).

$(1+2\epsilon, 1+\lceil 2/\epsilon \rceil)$ for ABP and $(1+\epsilon, 1+\lceil 2/\epsilon \rceil)$ for KRY. In PD, it means shallowness $\alpha \leq 1+\epsilon$. In Bonn, which targets the Elmore delay, the total tree capacitance is at most $1+\lceil 2/\epsilon \rceil$ times the minimum (i.e., lightness bound $\bar{\beta} = 1+\lceil 2/\epsilon \rceil$ if pin capacitances are ignorable), while wire delay is at most a factor of $(1+\epsilon)^2$ compared to a lower bound.

Compared with all the other methods (including ABP, KRY, ES, Bonn, PD, and PD-II), SALT shows superior performance, which mostly leads to both smaller wirelength and shorter path lengths. The average situation is illustrated by Fig. 18(a). It can

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

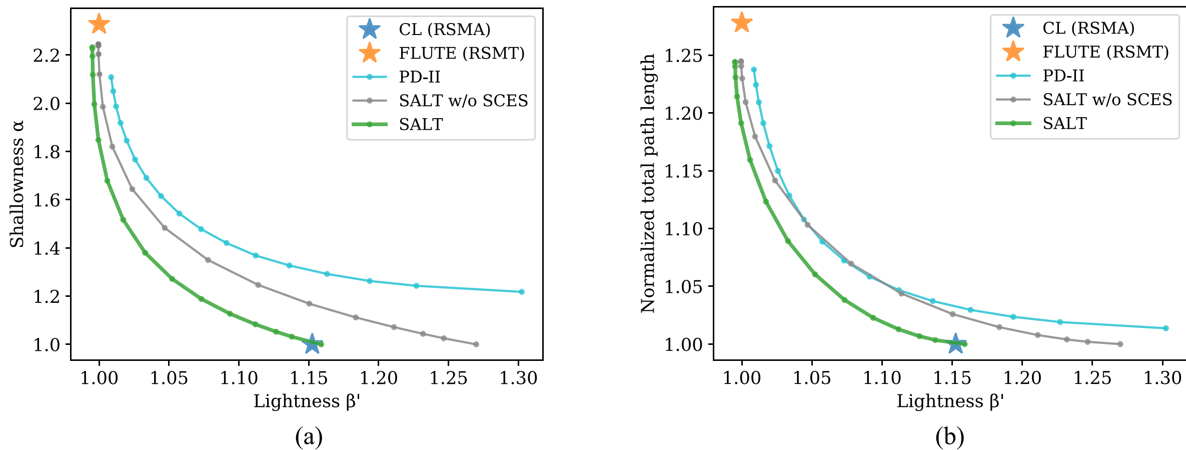

Fig. 20. Comparing SALT with PD-II on high-pin nets (# pins ≥ 32). (a) Tradeoff between shallowness and lightness. (b) Tradeoff between total path length and lightness.

be clearly observed that our method has the best Pareto frontier between RSMT and RSMA.

Fig. 18(b) illustrates the delay and lightness of different methods, where SALT still achieves a good tradeoff. Though KRY may obtain a slightly smaller delay, the wirelength cost is actually significant. Besides, the smaller Elmore delay there is usually achieved by unnecessary long edges, which is much less preferable than assigning the edge to a higher metal layer or buffering. For example in Fig. 19, the longest path of SALT (measured by Elmore delay) is the path from root r to pin v_1. Comparing with that in KRY, $r - v_1$ path in SALT has the same path length but drives more capacitance load before vertex v_2. KRY reduces the delay by connecting v_2 to r directly at the cost of wirelength. However, appropriate layer assignment or buffering on $r - v_2$ path will be more economical in practice.

Lastly, we conduct a detailed comparison between SALT and PD-II, a closest competitor among all other methods. Alpert *et al.* [5] compare PD-II with the preliminary version of SALT [33], which is without SCES. They use two metrics to measure the path lengths. The first is our shallowness metric $\alpha = \max\{[d_T(r, v)/d_G(r, v)] | v \in V\}$; the second is the total path length normalized by the total shortest-path distance, i.e., $([\sum_{v \in V} d_T(r, v)]/[\sum_{v \in V} d_G(r, v)])$. There, PD-II wins SALT in some cases, especially for nets with 32+ pins. The experimental results of PD-II and SALT (with and without SCES) on nets with 32+ pins are shown in Fig. 20. As we can see, if without SCES, SALT loses PD-II in total path length (when lightness β' is around 1.05–1.10). However, with the help of SCES, SALT dominates PD-II. Regarding the shallowness–lightness tradeoff, SALT is always superior to PD-II, even without SCES.

VII. Conclusion

We describe a novel SALT construction method called SALT, which is efficient and has the tightest bound over all the state-of-the-art general-graph SLT algorithms. Applying SALT to Manhattan space leads a smooth tradeoff between RSMT and RSMA for VLSI routing. Cooperating with some post-processing techniques, it achieves superior tradeoff between path length (or delay) and wirelength, compared to both classical and recent routing tree construction algorithms. A promising further work may be to integrate SALT into a complete routing optimization flow. Another line of research is to consider congestion when building the tree.

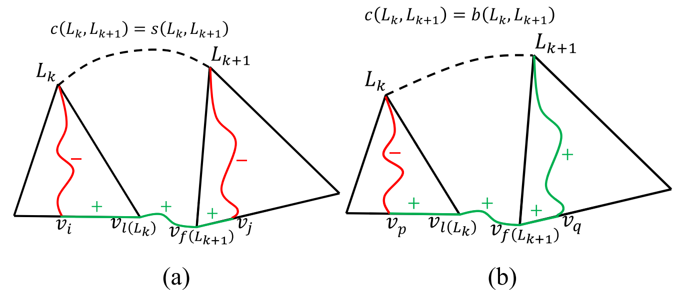

Fig. 21. Decomposed edge cost $c(L_k, L_{k+1})$. (a) Balanced case. (b) Unbalanced case.

Appendix A
Proof of Lemma 2

The proof is by induction. If z is a leaf, it is trivial by making $v_i = v_j = z$. We then assume that the statement holds for the two children z_l and z_r of z, and prove it for z.

Suppose first that $|b(z_l, z_r)| \le s(z_l, z_r)$, i.e., $w'(zz_l) + w'(zz_r) = s(z_l, z_r)$. Let $v_i \in \text{Leaves}(z_l)$ and $v_j \in \text{Leaves}(z_r)$ be two vertices that achieve $s(z_l, z_r) = d_G(v_i, v_j) - (d_T(z_l, v_i) + d_T(z_r, v_j))$. Therefore, $d_T(z, v_i) + d_T(z, v_j) = w'(zz_l) + d_T(z_l, v_i) + w'(zz_r) + d_T(z_r, v_j) = s(z_l, z_r) + d_T(z_l, v_i) + d_T(z_r, v_j) = d_G(v_i, v_j)$.

Otherwise, $|b(z_l, z_r)| > s(z_l, z_r)$. Suppose w.l.o.g. that $w'(zz_l) = 0$. By the induction hypothesis, there are $v_i, v_j \in \text{Leaves}(z_l)$ such that $d_T(z_l, v_i) + d_T(z_l, v_j) = d_G(v_i, v_j)$. Hence, $d_T(z, v_i) + d_T(z, v_j) = d_T(z_l, v_i) + d_T(z_l, v_j) = d_G(v_i, v_j)$. Note that $v_i, v_j \in \text{Leaves}(z_l) \subset \text{Leaves}(z)$.

Appendix B
Proof of Lemma 3

We start by decomposing $c(L_k, L_{k+1})$. There are two cases, as Fig. 21 shows. First, suppose $c(L_k, L_{k+1}) = s(L_k, L_{k+1})$.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

Let $v_i \in \text{Leaves}(L_k)$ and $v_j \in \text{Leaves}(L_{k+1})$ be two vertices that achieve $s(L_k, L_{k+1})$. Therefore,

$$\begin{aligned} c(L_k, L_{k+1}) &= s(L_k, L_{k+1}) \\ &= d_G(v_i, v_j) - \big(d_T(L_k, v_i) + d_T(L_{k+1}, v_j)\big) \\ &\le \underbrace{d_G(v_i, v_{l(L_k)}) - d_T(L_k, v_i)}_{\text{within } T(L_k)} + \underbrace{W(l(L_k), f(L_{k+1}))}_{\text{between } T(L_k),\ T(L_{k+1})} \\ &\quad + \underbrace{d_G\Big(v_{f(L_{k+1})}, v_j\Big) - d_T\big(L_{k+1}, v_j\big)}_{\text{within } T(L_{k+1})} \end{aligned} \tag{9}$$

where the last inequality holds due to triangle inequality.

Second, $c(L_k, L_{k+1}) = |b(L_k, L_{k+1})|$. If $b(L_k, L_{k+1}) \ge 0$, by Lemma 1, $\forall v_p \in \text{Leaves}(L_k), \forall v_q \in \text{Leaves}(L_{k+1})$

$$\begin{aligned} c(L_k, L_{k+1}) &= b(L_k, L_{k+1}) = t(L_k) - t(L_{k+1}) \\ &= \big(d_G(r, v_p) - d_T(L_k, v_p)\big) - \big(d_G(r, v_q) - d_T(L_{k+1}, v_q)\big) \\ &\le d_G(v_p, v_q) - d_T(L_k, v_p) + d_T(L_{k+1}, v_q) \\ &\le \underbrace{d_G(v_p, v_{l(L_k)}) - d_T(L_k, v_p)}_{\text{within } T(L_k)} + \underbrace{W(l(L_k), f(L_{k+1}))}_{\text{between } T(L_k),\ T(L_{k+1})} \\ &\quad + \underbrace{d_G\Big(v_{f(L_{k+1})}, v_q\Big) + d_T\big(L_{k+1}, v_q\big)}_{\text{within } T(L_{k+1})}. \end{aligned} \tag{10}$$

If $b(L_k, L_{k+1}) < 0$, the result is symmetric. Therefore, the part decomposed from $c(L_k, L_{k+1})$ into $T(L_k)$ is

$$C_r(L_k) = \begin{cases} d_G(v_i, v_{l(L_k)}) - d_T(L_k, v_i), & c(L_k, L_{k+1}) = s(L_k, L_{k+1}) \\ d_G(v_p, v_{l(L_k)}) - d_T(L_k, v_p), & c(L_k, L_{k+1}) = b(L_k, L_{k+1}) \\ d_G(v_q, v_{l(L_k)}) + d_T(L_k, v_q), & c(L_k, L_{k+1}) = -b(L_k, L_{k+1}) \end{cases} \tag{11}$$

where indices i is fixed while p and q are flexible. Meanwhile, there is $C_l(L_k)$, which is decomposed from $c(L_{k-1}, L_k)$ and can be calculated similarly. The weight sum within $T(L_k)$ is then $C(L_k) = C_l(L_k) + C_r(L_k)$.

We will prove $C(L_k) \le W(f(L_k), l(L_k))$, which has three cases.

Case 1: $C_l(L_k)$ and $C_r(L_k)$ both contain minus. Then $C(L_k) = d_G(v_{f(L_k)}, v_j) - d_T(L_k, v_j) + d_G(v_i, v_{l(L_k)}) - d_T(L_k, v_i)$. When $j \le i$, it is obvious. Otherwise, since $d_T(L_k, v_j) + d_T(L_k, v_i) \ge d_T(v_i, v_j) \ge d_G(v_i, v_j)$

$$\begin{aligned} C(L_k) &\le d_G(v_{f(L_k)}, v_j) + d_G(v_i, v_{l(L_k)}) - d_G(v_i, v_j) \\ &\le d_G(v_{f(L_k)}, v_i) + d_G(v_i, v_j)) + d_G(v_i, v_{l(L_k)}) \\ &\le W(f(L_k), l(L_k)). \end{aligned} \tag{12}$$

Case 2: Only one of $C_l(L_k)$ and $C_r(L_k)$ contains minus. Suppose w.l.o.g. that $C_l(L_k)$ does, then $C(L_k) = d_G(v_{f(L_k)}, v_q) + d_T(L_k, v_q) + d_G(v_i, v_{l(L_k)}) - d_T(L_k, v_i)$. By setting $q = i$

$$C(L_k) = d_G(v_{f(L_k)}, v_i) + d_G(v_i, v_{l(L_k)}) \le W(f(L_k), l(L_k)). \tag{13}$$

Case 3: Neither of $C_l(L_k)$ and $C_r(L_k)$ contains minus. That is, $C(L_k) = d_G(v_{f(L_k)}, v_q) + d_T(L_k, v_q) + d_G(v_p, v_{l(L_k)}) + d_T(L_k, v_p)$. By Lemma 2, there exist $f(L_k) \le q \le p \le l(L_k)$ such that

$$\begin{aligned} C(L_k) &= d_G(v_{f(L_k)}, v_q) + d_G(v_q, v_p) + d_G(v_p, v_{l(L_k)}) \\ &\le W(f(L_k), l(L_k)). \end{aligned} \tag{14}$$

By (9), (10), and $C(L_k) \le W(f(L_k), l(L_k))$, the proof is done.

Appendix C
Proof of Lemma 5

First, n is assumed to be the power of 2. Indeed, we can duplicate r into $2^{\lceil \log n \rceil} - n$ new vertices if it is not. Besides, if $\lceil \log \theta \rceil \ge \log n$, it is trivial by Theorem 2. Hence, we assume that $\lceil \log \theta \rceil < \log n$.

Let $E'_i \subseteq E'$ denote the set of edges added during the ith iteration ($1 \le i \le \log n$), W'_i denote $\sum_{e \in E'_i} w'(e)$, Leaves(e) denote the set of leaf vertices in the downstream from an edge e. Since T is SPT and $|\text{Leaves}(e)| = 2^{i-1}$ for $e \in E'_i$

$$\begin{aligned} \sum_{v \in V \setminus \{r\}} d_G(r, v) &= \sum_{v \in V \setminus \{r\}} d_T(r, v) = \sum_{i=1}^{\log n} \sum_{e \in E_i} |\text{Leaves}(e)| w'(e) \\ &= \sum_{i=1}^{\log n} 2^{i-1} \cdot W'_i \ge \sum_{i=\lceil \log \theta \rceil + 1}^{\log n} 2^{i-1} \cdot W'_i \ge \theta \sum_{i=\lceil \log \theta \rceil + 1}^{\log n} W'_i. \end{aligned} \tag{15}$$

Therefore, $\sum_{i=\lceil \log \theta \rceil + 1}^{\log n} W'_i \le \eta$ since $\sum_{v \in V \setminus \{r\}} d_G(r, v) \le \theta \cdot \eta$. Together with Lemma 4

$$\begin{aligned} w(T) = \sum_{i=1}^{\log n} W'_i &= \sum_{i=1}^{\lceil \log \theta \rceil} W'_i + \sum_{i=\lceil \log \theta \rceil + 1}^{\log n} W'_i \\ &\le \lceil \log \theta \rceil \cdot w(\text{MST}(G)) + \eta. \end{aligned} \tag{16}$$

Acknowledgment

The authors would like to thank Dr. K. Han, Dr. W.-K. Chow, and the other authors of PD-II [5] for providing their detailed results and helpful discussions. They would also like to thank Prof. S. Held for his constructive comments on the preliminary version of this paper.

References

[1] M. Elkin and S. Solomon, "Steiner shallow-light trees are exponentially lighter than spanning ones," in *Proc. IEEE Symp. Found. Comput. Sci.*, 2011, pp. 373–382.
[2] 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.
[3] I. L. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, no. 7513, pp. 147–154, 2014.
[4] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Power challenges may end the multicore era," *Commun. ACM*, vol. 56, no. 2, pp. 93–102, 2013.
[5] C. J. Alpert *et al.*, "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *Proc. ACM Int. Symp. Phys. Design*, 2018, pp. 10–17.
[6] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integr. VLSI J.*, vol. 21, nos. 1–2, pp. 1–94, 1996.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.
[8] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.

[9] 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.
[10] D. M. Warme, P. Winter, and M. Zachariasen, "Exact algorithms for plane Steiner tree problems: A computational study," in *Advances in Steiner Trees*. Boston, MA, USA: Springer, 2000, pp. 81–116.
[11] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM J. Appl. Math.*, vol. 30, no. 1, pp. 104–114, 1976.
[12] H. Zhou, N. Shenoy, and W. Nicholls, "Efficient minimum spanning tree construction without delaunay triangulation," *Inf. Process. Lett.*, vol. 81, no. 5, pp. 271–276, 2002.
[13] J.-M. Ho, G. Vijayan, and C.-K. Wong, "New algorithms for the rectilinear Steiner tree problem," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 2, pp. 185–193, Feb. 1990.
[14] A. B. Kahng and G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 7, pp. 893–902, Jul. 1992.
[15] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, "Closing the gap: Near-optimal Steiner trees in polynomial time," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 11, pp. 1351–1365, Nov. 1994.
[16] M. Borah, R. M. Owens, and M. J. Irwin, "An edge-based heuristic for Steiner routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 13, no. 12, pp. 1563–1568, Dec. 1994.
[17] H. Zhou, "Efficient Steiner tree construction based on spanning graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 704–710, May 2004.
[18] A. B. Kahng, I. I. Măndoiu, and A. Z. Zelikovsky, "Highly scalable algorithms for rectilinear and octilinear Steiner trees," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf.*, 2003, pp. 827–833.
[19] 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.
[20] W. Shi and C. Su, "The rectilinear Steiner arborescence problem is NP-complete," *SIAM J. Comput.*, vol. 35, no. 3, pp. 729–740, 2005.
[21] L. Nastansky, S. M. Selkow, and N. F. Stewart, "Cost-minimal trees in directed acyclic graphs," *Math. Methods Oper. Res.*, vol. 18, no. 1, pp. 59–67, 1974.
[22] J. Cong, A. B. Kahng, and K.-S. Leung, "Efficient algorithms for the minimum shortest path Steiner arborescence problem with applications to VLSI physical design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 17, no. 1, pp. 24–39, Jan. 1998.
[23] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, nos. 1–6, pp. 277–288, 1992.
[24] J. Córdova and Y.-H. Lee, "A heuristic algorithm for the rectilinear Steiner arborescence problem," Dept. Comput. Inf. Sci., Univ. Florida, Gainesville, FL, USA, Rep. TR-94-025, 1994.
[25] 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.*, 1993, pp. 606–611.
[26] M. J. Alexander and G. Robins, "New performance-driven FPGA routing algorithms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1505–1517, Dec. 1996.
[27] M. Pan, C. Chu, and P. Patra, "A novel performance-driven topology design algorithm," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf.*, 2007, pp. 244–249.
[28] B. Awerbuch, A. Baratz, and D. Peleg, "Efficient broadcast and lightweight spanners," Dept. Appl. Math. Comput. Sci., Weizmann Inst. Sci., Rehovot, Israel, Rep. CS92-22, 1992.
[29] 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.
[30] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, pp. 305–321, 1995.
[31] S. Held and D. Rotter, "Shallow-light Steiner arborescences with vertex delays," in *Proc. Int. Conf. Integer Program. Comb. Optim.*, 2013, pp. 229–241.
[32] C. J. Alpert, T. Hu, J. 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.
[33] G. Chen, P. Tu, and E. F. Young, "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2017, pp. 569–576.
[34] C. Bartoschek, S. Held, J. Maßberg, D. Rautenbach, and J. Vygen, "The repeater tree construction problem," *Inf. Process. Lett.*, vol. 110, no. 24, pp. 1079–1083, 2010.
[35] S. Held and B. Rockel, "Exact algorithms for delay-bounded Steiner arborescences," in *Proc. ACM/IEEE Design Autom. Conf.*, 2018, p. 44.
[36] R. Scheifele, "RC-aware global routing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2016, p. 21.
[37] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, no. 1, pp. 86–109, 2017.
[38] S. Held *et al.*, "Global routing with timing constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 406–419, Feb. 2018.
[39] S. Solomon, "Euclidean Steiner shallow-light trees," in *Proc. Int. Symp. Comput. Geometry*, 2014, p. 454.
[40] S. Solomon, "Euclidean Steiner shallow-light trees," *J. Comput. Geometry*, vol. 6, no. 2, pp. 113–139, 2015.
[41] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Conf.*, 1984, pp. 47–57.
[42] K. D. Boese, A. B. Kahng, and G. Robins, "High-performance routing trees with identified critical sinks," in *Proc. ACM/IEEE Design Autom. Conf.*, 1993, pp. 182–187.
[43] S.-E. D. Lin and D. H. Kim, "Construction of all rectilinear Steiner minimum trees on the Hanan grid," in *Proc. ACM Int. Symp. Phys. Design*, 2018, pp. 18–25.
[44] 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*, 2015, pp. 921–926.

Gengjie Chen received the B.Sc. degree from the Department of Electronic and Communication Engineering, Sun Yat-sen University, Guangzhou, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong.

His current research interests include combinatorial optimization, numerical optimization, and electronic design automation.

Mr. Chen was a recipient of First Place at the ACM SIGDA Student Research Competition in 2018, the Best Paper Award at ICCAD 2017, the Hong Kong Ph.D. Fellowship in 2015, and six ICCAD/ISPD Contest Awards.

Evangeline F. Y. Young received the B.Sc. degree in computer science from the Chinese University of Hong Kong (CUHK), Hong Kong, and the Ph.D. degree from the University of Texas at Austin, Austin, TX, USA, in 1999.

She is currently a Professor with the Department of Computer Science and Engineering, CUHK. She researches actively on floorplanning, placement, routing, DFM, and EDA on physical design in general. Her current research interests include optimization, algorithms, and very large scale integration CAD.

Dr. Young was a recipient of the Best Paper Awards from ICCAD 2017, ISPD 2017, SLIP 2017, and FCCM 2018, and several championships and prizes in renowned EDA contests, including the 2018, 2016, 2015, 2013, and 2012 CAD Contests at ICCAD, DAC 2012, and ISPD 2018, 2017, 2016, 2015, 2011, and 2010 with her research group. She has served on the organization committees of ISPD, ARC, and FPT and on the program committees of conferences, including DAC, ICCAD, ISPD, ASP-DAC, SLIP, DATE, and GLSVLSI. She also served on the editorial boards of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, ACM TODAES, and *Integration, the VLSI Journal*.

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on May 24,2020 at 02:12:28 UTC from IEEE Xplore. Restrictions apply.