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

Gengjie Chen, Peishan Tu, and Evangeline F. Y. Young

Department of Computer Science and Engineering, The Chinese University of Hong Kong, NT, Hong Kong

{gjchen,pstu,fyyoung}@cse.cuhk.edu.hk

***Abstract*—In a weighted undirected graph, a spanning/Steiner shallow-light tree (SLT) simultaneously approximates (i) shortest distances from a root to the other vertices, and (ii) the minimum tree weight. The Steiner SLT has been proved to be exponentially lighter than the spanning one [1], [2]. In this paper, we propose a novel Steiner SLT construction method called SALT (Steiner shAllow-Light Tree), which is efficient and has the tightest bound over all the state-of-the-art SLT algorithms. Applying SALT to Manhattan space offers a smooth trade-off between rectilinear Steiner minimum tree (RSMT) and rectilinear Steiner minimum arborescence (RSMA) for VLSI routing. In addition, the adaption further reduces the time complexity from $O(n^2)$ to $O(n \log n)$. 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.**

I. Introduction

Timing and power have been being crucial issues in chip design since more than two decades ago and become increasingly critical as technology scales. 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 8nm will be powered off and cannot be utilized due to the power constraint [4].

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 tree weight (i.e., wirelength) and path length are important. Essentially, tree weight implies routing resource usage (routability), power consumption, cell delay and wire delay, while path length implies wire delay.

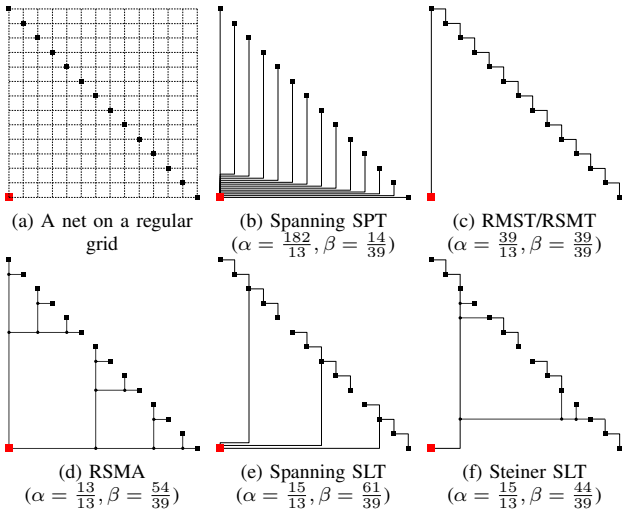
It is a well-studied problem if only one of the objectives between tree weight and path length should be minimized, whether the domain is the spanning tree or the rectilinear Steiner one. For spanning trees, the *minimum spanning tree* (MST) can be obtained by various classical algorithms like Prim's 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 [5]. For rectilinear Steiner trees, the one with minimum tree weight is called a *rectilinear Steiner minimum tree* (RSMT), while the one with all paths from root being shortest is a *rectilinear Steiner minimum arborescence* (RSMA).

Both RSMT and RSMA construction are NP-hard [6], [7], but there are efficient heuristics to solve them with near optimal quality. For RSMT, rectilinear MST (RMST) achieves an 1.5-approximation [8] and can be constructed in $O(n \log n)$ time [9]. Many fast algorithms (e.g., [10]–[12]) are also proposed in order to pursue a smaller tree weight. For RSMA, a 2-approximation can be obtained in $O(n \log n)$ time [13], [14], and some examples of efficient heuristics are [15], [16].

The spanning/Steiner *shallow-light tree* (SLT) combines the objectives of path length and 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 trade-off is in the order of $(1 + \epsilon, O(\frac{1}{\epsilon}))$. The ABP and BRBC algorithms [17], [18] are in fact identical and provide a bound of $(1 + 2\epsilon, 1 + \frac{2}{\epsilon})$. After them, the KRY algorithm [19] reduces the bound to $(1+\epsilon, 1+\frac{2}{\epsilon})$ and proves that it is tight. KRY also provides a smooth trade-off between SPT and MST controlled by ϵ, while ABP does not (e.g., a MST is not implied when $\epsilon = +\infty$). Besides, the PD algorithm [20] smoothly trades off between SPT and MST, but the resulted tree is not guaranteed to be SLT. Recently, Steiner SLTs are proved to be exponentially lighter than spanning ones by Elkin and Solomon [1], [2]. The ES algorithm can efficiently build a Steiner $(1+\epsilon, O(\log \frac{1}{\epsilon}))$-SLT with a time complexity of $O(n^2)$. The constants in the bound $(1+2\epsilon, 4+2\lceil \log \frac{2}{\epsilon} \rceil)$ are, however, quite large (log denotes $\log_2$ in this paper).

TABLE I: Spanning and Steiner Shallow-Light Tree Comparison

	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



(a) A net on a regular grid; (b) Spanning SPT ($\alpha = \frac{182}{13}, \beta = \frac{14}{39}$); (c) RMST/RSMT ($\alpha = \frac{39}{13}, \beta = \frac{39}{39}$); (d) RSMA ($\alpha = \frac{13}{13}, \beta = \frac{54}{39}$); (e) Spanning SLT ($\alpha = \frac{15}{13}, \beta = \frac{61}{39}$); (f) Steiner SLT ($\alpha = \frac{15}{13}, \beta = \frac{44}{39}$)

Fig. 1: Different routing topologies on the same net (the root is marked by red, α and β denote shallowness and lightness).

We propose an efficient algorithm called SALT for constructing a Steiner SLT and apply it to routing topology construction. Our contributions are summarized as follows.

1) We propose SALT for the Steiner SLT on general graphs, whose shallowness-lightness bound is $(1 + \epsilon, 2 + \lceil \log \frac{2}{\epsilon} \rceil)$. To the best

This work was partially supported by the Research Grants Council of Hong Kong SAR, China (Project No. CUHK14209214).

TABLE II: 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

of our knowledge, the bound is much tighter than all existing methods for constructing spanning/Steiner SLTs.

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.
3) We further decrease path lengths and tree weight in the Manhattan space by integrating SALT with the classical RSMA [13] and RSMT [12] algorithms. The method (rectilinear SALT) provides a smooth trade-off between RSMA and RSMT controlled by ϵ. This is similar to what KRY and PD algorithms do for MST and SPT, but with an exponentially tighter bound.
4) Several effective post-processing methods are also proposed to further improve the result.

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 not only valid but also desirable due to three reasons. First, SALT provides a bounded trade-off and has much stronger global view. It can generate high-quality initial solutions for later stage optimization. Second, the linear delay model is reasonable due to buffering [21], 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 [22], [23].

Last but not least, we want to highlight that even though the bound analysis of SALT is complex, it can be easily implemented with hundreds of lines of codes.

II. Steiner Shallow-Light Tree Algorithm

The exact problem formulation and the ES algorithm [2] for the Steiner SLT 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

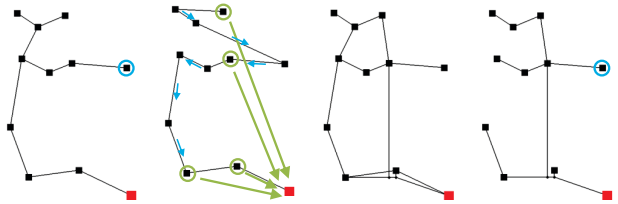
1) Problem Formulation

Our Steiner SLT 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 (i) it approximates all shortest-path distances $d_G(r, v)$ from r to $v \in V$, and (ii) its weight $w(T)$ is bounded by that of MST $w(MST(G))$. For a $(\bar{\alpha}, \bar{\beta})$-SLT, (i) the shallowness $\alpha = max\{\frac{d_T(r,v)}{d_G(r,v)} | v \in V \backslash \{r\}\} \leq \bar{\alpha}$, and (ii) lightness $\beta = \frac{w(T)}{w(MST(G))} \leq \bar{\beta}$. 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 $w(MST(G)) \leq 2 \cdot w(SMT(G))$ (i.e., $w(T) \leq \bar{\beta} \cdot w(MST(G)) \leq 2 \cdot \bar{\beta} \cdot w(SMT(G))$). For rectilinear Steiner trees, the gap is smaller with $w(RMST(G)) \leq 1.5 \cdot w(RSMT(G))$.

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 Steiner SLT cannot be embedded into some metric spaces (e.g., Euclidean space), 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.

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 $T_M \cup T_B$;



(a) MST T_M (b) Path P (c) Graph $T_M \cup T_B$ (d) ES T

Fig. 2: Sample run of Algorithm 1 on a net ($\epsilon = 1$). (a) Construct MST T_M (shallowness $\alpha = 3.14$ with worst sink circled by blue, lightness $\beta = 1$). (b) Identify breakpoints B (circled by green) on the Hamiltonian path P, where each blue arrow points from a non-breakpoint 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 Steiner SLT T ($\alpha = 1.90, \beta = 1.06$).

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 II. 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.

The Steiner SPT for connecting breaking points is a dedicated design (refer to Section 2 of [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 a 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* $(1 + 2\epsilon, 4 + 2\lceil \log \frac{2}{\epsilon} \rceil)$*-SLT in* $O(n^2)$ *time.*

Proof. See Lemmas 3.4, 3.5 and 3.6 of [2]. □

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 [19], we propose to use (i)

TABLE III: 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);

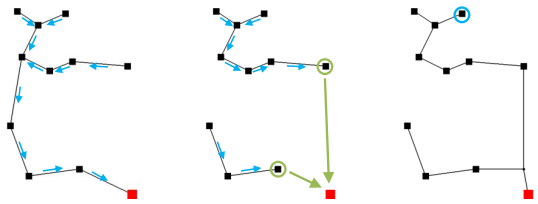
15: **function** Relax(u, v)
16: **if** $d[v] > d[u] + w(uv)$ **then**
17: $d[v] \leftarrow d[u] + w(uv)$;
18: $p[v] \leftarrow u$;
19: **else if** $d[v] = d[u] + w(uv)$ and $w(p[v]v) < w(uv)$ **then**
20: $p[v] \leftarrow u$;

a tighter criterion for identifying breakpoints and (ii) a better initial topology (i.e., a MST instead of a Hamiltonian path) in the Steiner SLT construction. The framework with the two effective techniques is illustrated by Algorithm 2 and Fig. 3. As a subroutine, the light Steiner SPT construction method will be described by Algorithm 3 in the next subsection.

In SALT, the solution is initialized to an MST and gradually modified towards a Steiner SLT. 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 a 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 non-breakpoints v define a forest F, with tree roots being breakpoints. In the last step, breakpoints are connected to r by a Steiner SPT T_B.

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 it 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 straight-forward 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 $d_G(r, b) + d_P(b, v) > (1 + \epsilon) \cdot d_G(r, v)$.



(a) MST T_M (b) Forest F (c) SALT T

Fig. 3: Sample run of Algorithm 2 on a net ($\epsilon = 1$). (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 Steiner SLT (shallowness $\alpha = 1.43$ with worst sink circled by blue, lightness $\beta = 1.05$).

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.

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 IV 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 (line 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:

$$b(z_l, z_r) = t(z_l) - t(z_r), \tag{1}$$

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

where $t(z)$ is the distance from root r to vertex z in the final SPT. It is straight-forward that $t(z) = d_G(r, z)$ if z is a leaf. t and b help maintain T to be a 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 Lemma 2.2 of [2]. □

TABLE IV: 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

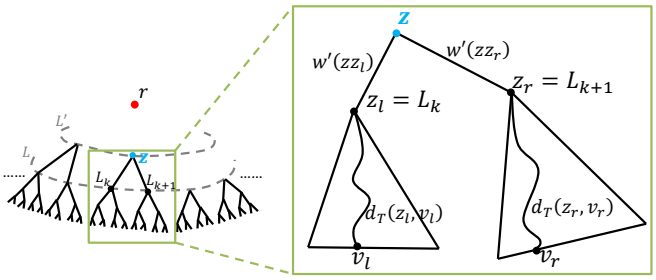


Fig. 4: During the Steiner SPT construction, neighboring vertices in L are merged pair by pair into Steiner vertices in L'. Shown by the enlarged the figure, vertices L_k (i.e., z_l) and L_{k+1} (i.e., z_r) are merged to a Steiner vertex z in L'.

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

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

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 a 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, $\sum_{k=1}^{|L|-1} c(L_k, L_{k+1}) \le W(1, n+1)$.*

Proof. See Appendix B. □

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;
Initialize ($V' \leftarrow V, E' \leftarrow \emptyset, \forall v \in V, t(v) \leftarrow d_G(r, v)$);
$L \leftarrow$ Hamiltonian circle based on $MST(G)$ ($L_{n+1} = L_1$);
while $|L| > 1$ **do**
  for $k = 1$ to n **do**
    Calculate $b(L_k, L_{k+1}), s(L_k, L_{k+1})$ by (1) (2);
    $c(L_k, L_{k+1}) \leftarrow max\{s(L_k, L_{k+1}), |b(L_k, L_{k+1})|\}$;
  $M_L \leftarrow$ a light perfect (or near perfect) matching on the circle defined by L and c;
  $L' \leftarrow$ empty vertex sequence;
  for $L_k L_{k+1} \in M_L$ **do**
    `AddSteiner`(L_k, L_{k+1});
  if $|L|$ is odd **then**
    Append the unmatched vertex to L';
  $L \leftarrow L'$;

function `AddSteiner`(z_l, z_r)
  Add a Steiner vertex z into V';
  Add edges zz_l and zz_r into E';
  if $|b(z_l, z_r)| \le s(z_l, z_r)$ **then**
    $w'(zz_l) \leftarrow \frac{s(z_l,z_r)+b(z_l,z_r)}{2}$;
    $w'(zz_r) \leftarrow \frac{s(z_l,z_r)-b(z_l,z_r)}{2}$;
  else
    $w'(zz_l) \leftarrow max\{b(z_l, z_r), 0\}$;
    $w'(zz_r) \leftarrow max\{-b(z_l, z_r), 0\}$;
  $t(z) \leftarrow d_G(r, v) - d_T(z, v)$ for an arbitrary $v \in Leaves(z)$;
  Append z to L';

Lemma 4. *In the i-th iteration of Algorithm 3, the total weight of added edges $W'_i \le w(MST(G))$.*

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

With the help of Lamma 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, $w(T) = \sum_{i=1}^{\lceil \log n \rceil} W'_i \le \lceil \log n \rceil \cdot w(MST(G))$. □

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

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

Proof. See Appendix C. □

Lemma 6. *In SALT, $\sum_{v \in B} d_G(r, v) \le \frac{2}{\epsilon} \cdot w(MST(G))$.*

Proof. See Lemma 3.2 of [19]. □

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

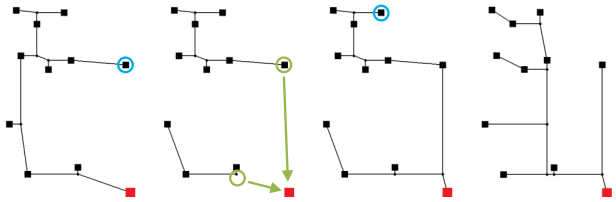
Theorem 3. *SALT generates a Steiner $(1+\epsilon, 2+\lceil \log \frac{2}{\epsilon} \rceil)$-SLT.*

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

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);



(a) RSMT T_M by FLUTE (b) Forest F (c) Rectilinear SALT T (d) RSMA by CL

Fig. 5: Sample run of Algorithm 4 on a net ($\epsilon = 1$). (a) Construct RSMT T_M by FLUTE (shallowness $\alpha = 2.66$ with worst sink circled by blue, 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 Steiner SLT. ($\alpha = 1.22, \beta = 1.01$). (d) RSMA by CL on the net ($\alpha = 1, \beta = 1.11$).

have shallowness $\alpha \leq 1 + \epsilon$.

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

III. Rectilinear Steiner Shallow-light Tree Algorithm

SALT, which generates a Steiner $(1+\epsilon, 2+\lceil \log \frac{2}{\epsilon} \rceil)$-SLT for a general graph, can be directly applied in the Manhattan space. However, it can be further improved 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 two-dimensional 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 x coordinate of their parent Steiner point z is

$$x_z = \begin{cases} min\{x_{z_l}, x_{z_r}\}, & x_{z_l}, x_{z_r} \geq 0, \\ max\{x_{z_l}, x_{z_r}\}, & x_{z_l}, x_{z_r} \leq 0, \\ 0, & x_{z_l} \leq 0 \leq x_{z_r} \text{ or } x_{z_r} \leq 0 \leq x_{z_l}. \end{cases} \quad (6)$$

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 on 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 on Manhattan plane is exactly the classical RSMA problem [7], [13]. The CL heuristics [14] 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 $\beta \leq \lceil \log \frac{2}{\epsilon} \rceil$ may be far away from the optimal SPT in worst cases. For example, when all vertices locates 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 ($\bar{\beta} = 2 + \lceil \log \frac{2}{\epsilon} \rceil$) 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 [12] 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 $w(T) \leq (2 + \lceil \log \frac{2}{\epsilon} \rceil) \cdot w(T_M)$, 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 further reduce the lightness β of the Steiner SLT constructed and improve the time complexity to $O(n \log n)$. From another viewpoint, rectilinear SALT is a smooth trade-off between RSMA and RSMT. The smaller the ϵ, the closer the rectilinear SALT is to a RSMA; the larger the ϵ, the closer it is to a RSMT. It is almost a CL RSMA when $\epsilon = 0$ and a FLUTE RSMT when $\epsilon = +\infty$. In the middle, it is a bounded trade-off between them. To a certain extent, Fig. 5 illustrates the situation. The RSMT in (a) is the lightest but has some long paths, while the RSMA in (d) is the shallowest but is of a large tree weight. Combining the strengths of the both, the Rectilinear SALT in (c) is not only light but also shallow.

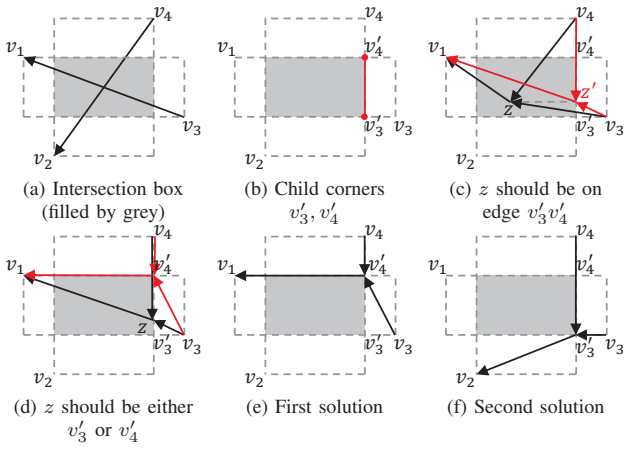
Algorithm 5 Refinement of Rectilinear SALT

Require: Rectilinear SALT T;
Ensure: Refined rectilinear SALT T';
1: Cancel intersected edges;
2: Do L-shape flipping until no improvement;
3: Do U-shape shifting;

IV. Post Processing

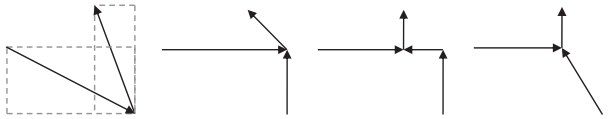
Three effective post processing techniques are adopted to further improve the rectilinear SALT, which is shown in Algorithm 5. For simplicity, rectilinear SALT will be referred as SALT hereafter.

First of all, 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 overlap, 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 *child vertices* v_3 and v_4 to it, and also connect it to either v_1 or v_2. By choosing the shorter path between $(z, v_1, ..., r)$ and $(z, v_2, ..., r)$, both path length and wirelength can be reduced. The question is where the best location of the Steiner vertex z is, which can be answered by the following theorem. Among the four



(a) Intersection box (filled by grey) (b) Child corners v'_3, v'_4 (c) z should be on edge $v'_3v'_4$ (d) z should be either v'_3 or v'_4 (e) First solution (f) Second solution

Fig. 6: Intersection cancelling (arrows point to the root).



(a) (b) (c) (d)

Fig. 7: Z-shape flipping by iterative L-shape flipping. (a) The input. (b) The first L-shape flipping. (c) The second L-shape flipping (i.e., a Z-shape flipping). (d) Removing the redundant Steiner vertex.

corners of the intersection box, a *child corner* is the one 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.*

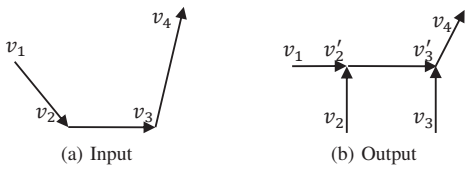
Proof. First, z should be on a *child edge* (i.e., the edge between the two child corners). If not, its projected point z' on the child edge can improve the wirelength without impacting path lengths (Fig. 6(c)). Supposing z is connected to v_1, there is $w(v_3z)+w(v_4z)+w(zv_1) = (w(v_3z')+w(z'z))+(w(v_4z')+w(z'z))+(w(z'v_1)-w(z'z))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

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 $w(v_3z)+w(v_4z)+w(zv_1) \geq w(v_3v'_4)+w(v_4v'_4)+w(v'_4v_1)$; for path lengths, there is $w(v_4z)+w(zv_1) \geq w(v_4v'_4)+w(v'_4v_1)$, while $w(v_3z)+w(zv_1) = w(v_3v'_4)+w(v'_4v_1)$.

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 Figures 6(e) and 6(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 and sufficient. □

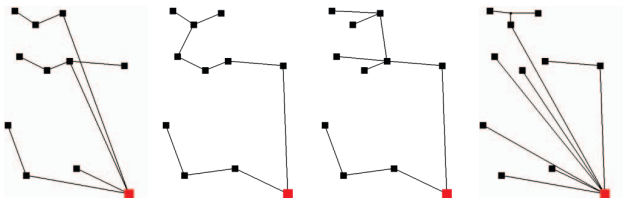
Second, edges may be overlapped with each other by flipping (L-shape or Z-shape), which improves wirelength and path lengths, as Fig. 7 shows. A linear time dynamic programming [10] can construct a flipping solution with optimal wirelength if the vertex degree is bounded and only edge overlapping around a vertex is counted. In Rectilinear SALT, the maximum vertex degree is the sum of that in FLUTE (four, according [24]) and CL (four, considering the root), as a SALT T is the union of a FLUTE forest F and a CL RSMA T_B. Therefore, the vertex degree is bounded ($\leq 4+4=8$), which guarantees $O(n)$ 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. 7.

Third, the U-shape shifting is beneficial not only to wirelength and path lengths but also to Elmore delay [25], which should be conducted if possible. An example is in Fig. 8, where the edge v_2v_3 is shifted to $v'_2v'_3$. Note that after the L-shape flipping, the middle edge of the U-shape path should be strictly horizontal or vertical (e.g., v_2v_3 in $v_1v_2v_3v_4$ is strictly horizontal), which makes identifying U easier.



(a) Input (b) Output

Fig. 8: U-shape shifting.



(a) ABP (b) KRY (c) PD (d) Bonn

Fig. 9: Sample runs of various algorithms on the example net ($\epsilon = 1$). (a) ABP/BRBC (shallowness $\alpha = 1.90$, lightness $\beta = 1.35$). (b) KRY ($\alpha = 1.43, \beta = 1.10$). (c) PD ($\alpha = 1.11, \beta = 1.15$) (d) Bonn ($\alpha = 1.22, \beta = 2.25$).

The above methods are safe because they improve wirelength or path length or both without worsening any of them. They can also be evaluated and guided by a more accurate delay model.

V. Experimental Results

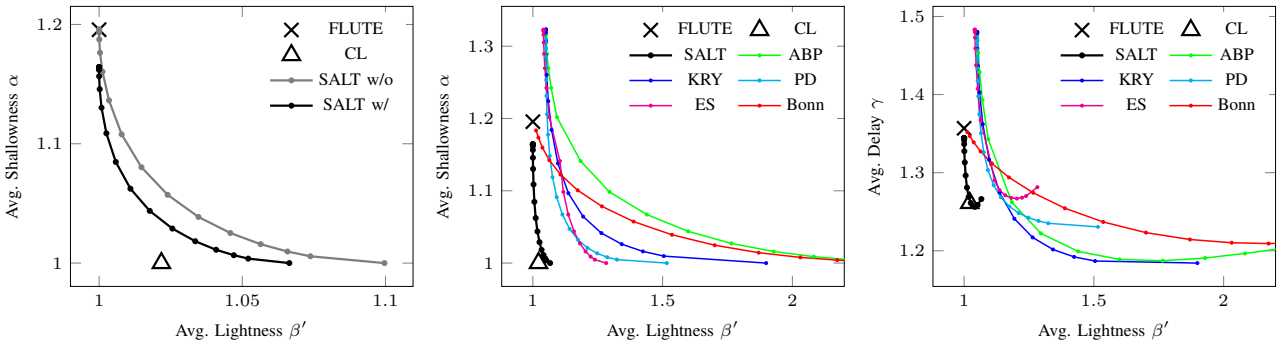
We implement SALT as well as ES [2], CL [14], ABP/BRBC [17], [18], KRY [19], PD [20] and Bonn [22] algorithms in C++, while the source code of FLUTE [12] is obtained from the authors. Benchmarks of ICCAD 2015 Contest [26] are used for a comprehensive evaluation and comparison. The benchmark statistics is shown in TABLE V. By ignoring the 2-pin nets, which are trivial, the batch test covers around 2.4 million nets in total.

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

In the batch test, ϵ is set to 20 values ranging from 0 to 73.895 (mainly a geometric sequence 0.05×1.5^n) to cover the variation of different methods. The lightness metric is changed to $\beta' = \frac{w(T)}{w(FLUTE)}$

TABLE V: ICCAD 2015 Benchmark Statistics

Design	#cells	#nets classified by pin number ($\times 10^3$)						
	($\times 10^3$)	2	3–9	10–19	20–29	30–39	≥ 40	≥ 3
superblue1	1932	893	281	23	11	6	0.9	323
superblue3	1876	952	215	35	15	6	1.1	273
superblue4	796	610	162	17	9	4	0.5	192
superblue5	982	824	242	18	8	5	0.7	273
superblue7	768	1493	338	63	27	11	1.7	441
superblue10	1087	1457	385	31	14	9	1.2	441
superblue16	1213	756	213	17	7	5	0.3	243
superblue18	1210	575	156	24	11	5	0.6	197
Total	9863	7559	1992	229	103	51	7.0	**2382**



(a) SALT w/ and w/o post processing. (b) Shallowness-lightness trade-off. (c) Delay-lightness trade-off.

Fig. 10: Comparing SALT with other routing tree construction methods.

TABLE VI: SALT with and without Post Processing

trade-off parameter ϵ	SALT w/o			SALT w/		
	Light -ness β'	Shallow -ness α	Delay γ	Light -ness β'	Shallow -ness α	Delay γ
0.000	1.100	1.000	1.271	**1.066**	**1.000**	**1.266**
0.050	1.074	1.006	1.258	**1.052**	**1.004**	**1.259**
0.075	1.066	1.010	1.256	**1.047**	**1.007**	**1.257**
0.113	1.056	1.016	1.256	**1.041**	**1.011**	**1.256**
0.169	1.046	1.025	1.258	**1.034**	**1.018**	**1.257**
0.253	1.035	1.039	1.263	**1.026**	**1.029**	**1.261**
0.380	1.024	1.057	1.273	**1.018**	**1.044**	**1.269**
0.570	1.015	1.080	1.287	**1.011**	**1.062**	**1.281**
0.854	1.008	1.108	1.305	**1.006**	**1.085**	**1.296**
1.281	1.003	1.136	1.323	1.003	**1.109**	**1.313**
1.922	1.001	1.160	1.339	1.001	**1.130**	**1.328**
2.883	1.000	1.176	1.349	1.000	**1.146**	**1.337**
4.325	1.000	1.187	1.354	1.000	**1.157**	**1.342**
6.487	1.000	1.193	1.356	1.000	**1.162**	**1.344**
9.731	1.000	1.195	1.357	1.000	**1.164**	**1.344**
...	1.000	1.196	1.357	1.000	**1.164**	**1.344**

(instead of $\beta = \frac{w(T)}{w(MST)}$), where FLUTE serves as a better 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 [22]. For each method and each ϵ, we average the scores over all nets.

TABLE VI and Fig. 10(a) show the effectiveness of our post processing techniques, which simultaneously improve α , β' and γ. Note that SALT is very efficient. It completes the routing as well as post processing on 2.4 million nets for 20 times (due to different ϵ) in just 22.5 minutes.

Compared to other algorithms targeting the shallowness-lightness trade-off of a tree (including ABP, KRY, PD, ES and Bonn), SALT has superior performance, which mostly leads to both smaller wirelength and shorter path lengths. The average situation is illustrated by Fig. 10(b). It can be clearly observed that our method has the best Pareto frontier between RSMT and RSMA, while ES, PD and Bonn compete for the second best.

Fig. 10(c) illustrates the delay and lightness of different methods, where SALT still achieves a good trade-off. Though KRY may obtain a slightly smaller delay, the wirelength cost is actually significant (note that the scales of x and y axes are different). Besides, the smaller Elmore delay there is sometimes achieved by parallel edges, which is much less preferable than assigning the edge to a higher metal layer in practice.

VI. Conclusion

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

References

[1] M. Elkin *et al.*, "Steiner shallow-light trees are exponentially lighter than spanning ones," in *Proc. FOCS*, 2011, pp. 373–382.

[2] ——, "Steiner shallow-light trees are exponentially lighter than spanning ones," *SIAM Journal on Computing*, 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 *et al.*, "Power challenges may end the multicore era," *Communications of the ACM*, vol. 56, no. 2, pp. 93–102, 2013.

[5] T. H. Cormen *et al.*, *Introduction to algorithms*. MIT press, 2009.

[6] M. R. o. Garey, "The rectilinear steiner tree problem is NP-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[7] W. Shi *et al.*, "The rectilinear steiner arborescence problem is NP-complete," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 729–740, 2005.

[8] F. K. Hwang, "On steiner minimal trees with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 30, no. 1, pp. 104–114, 1976.

[9] H. Zhou *et al.*, "Efficient minimum spanning tree construction without delaunay triangulation," *Information Processing Letters*, vol. 81, no. 5, pp. 271–276, 2002.

[10] J.-M. Ho *et al.*, "New algorithms for the rectilinear steiner tree problem," *IEEE TCAD*, vol. 9, no. 2, pp. 185–193, 1990.

[11] A. B. Kahng *et al.*, "A new class of iterative steiner tree heuristics with good performance," *IEEE TCAD*, vol. 11, no. 7, pp. 893–902, 1992.

[12] C. Chu *et al.*, "FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE TCAD*, vol. 27, no. 1, pp. 70–83, 2008.

[13] S. K. Rao *et al.*, "The rectilinear steiner arborescence problem," *Algorithmica*, vol. 7, no. 1-6, pp. 277–288, 1992.

[14] J. Córdova *et al.*, "A heuristic algorithm for the rectilinear steiner arborescence problem," Tech. Rep., 1994.

[15] J. Cong *et al.*, "Performance-driven interconnect design based on distributed RC delay model," in *Proc. DAC*. IEEE, 1993, pp. 606–611.

[16] M. Pan *et al.*, "A novel performance-driven topology design algorithm," in *Proc. ASPDAC*. IEEE, 2007, pp. 244–249.

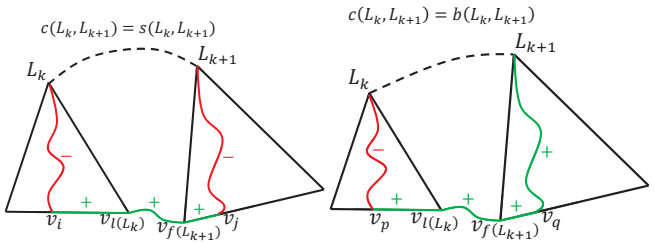
[17] B. Awerbuch *et al.*, "Effcient broadcast and light-weight spanners," Tech. Rep., 1991.

[18] J. Cong *et al.*, "Provably good performance-driven global routing," *IEEE TCAD*, vol. 11, no. 6, pp. 739–752, 1992.

[19] S. Khuller *et al.*, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, pp. 305–321, 1995.

[20] C. J. Alpert *et al.*, "Prim-Dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE TCAD*, vol. 14, no. 7, pp. 890–896, 1995.

[21] C. Bartoschek *et al.*, "The repeater tree construction problem," *Information Processing Letters*, vol. 110, no. 24, pp. 1079–1083, 2010.



(a) Balanced case (b) Unbalanced case

Fig. 11: Decomposed edge cost $c(L_k, L_{k+1})$.

[22] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, no. 1, pp. 86–109, 2017.

[23] ——, "RC-aware global routing," in *Proc. ICCAD*. ACM, 2016, pp. 21:1–21:8.

[24] M. Hanan, "On steiners problem with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[25] K. D. Boese *et al.*, "High-performance routing trees with identified critical sinks," in *Proc. DAC*. IEEE, 1993, pp. 182–187.

[26] M.-C. Kim *et al.*, "ICCAD-2015 CAD Contest in incremental timing-driven placement and benchmark suite," in *Proc. ICCAD*. ACM, 2015, pp. 921–926.

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 $v_i \in Leaves(z_l)$ and $v_j \in Leaves(z_r)$ 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,

$$\begin{aligned} & d_T(z, v_i) + d_T(z, v_j) \\ = \; & w'(zz_l) + d_T(z_l, v_i) + w'(zz_r) + d_T(z_r, v_j) \\ = \; & s(z_l, z_r) + d_T(z_l, v_i) + d_T(z_r, v_j) \\ = \; & d_G(v_i, v_j). \end{aligned} \tag{7}$$

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 $v_i, v_j \in Leaves(z_l)$ such that $d_T(z_l, v_i) + d_T(z_l, v_j) = d_G(v_i, v_j)$. Hence,

$$d_T(z, v_i) + d_T(z, v_j) = d_T(z_l, v_i) + d_T(z_l, v_j) = d_G(v_i, v_j). \tag{8}$$

Note that $v_i, v_j \in Leaves(z_l) \subset Leaves(z)$.

B. Proof of Lemma 3

We start by decomposing $c(L_k, L_{k+1})$. There are two cases, as Fig. 11 shows. First, suppose $c(L_k, L_{k+1}) = s(L_k, L_{k+1})$. Let $v_i \in Leaves(L_k)$ and $v_j \in Leaves(L_{k+1})$ be two vertices that achieve $s(L_k, L_{k+1})$. Therefore,

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

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, $\forall v_p \in Leaves(L_k), \forall v_q \in Leaves(L_{k+1})$,

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

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, 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.

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 i-th 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 $|Leaves(e)| = 2^{i-1}$ for $e \in E'_i$,

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

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,

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