Dim Sum: Light Clock Tree by Small Diameter Sum

Gengjie Chen and Evangeline F. Y. Young
Department of Computer Science and Engineering, The Chinese University of Hong Kong
{gjchen,fyyoung}@cse.cuhk.edu.hk

***Abstract*—By retrospecting the classical deferred-merge embedding (DME) algorithm, we found an intrinsic relationship between the zero-skew tree (ZST) problem and the hierarchical clustering (HC) problem. To be more specific, the wire length of a ZST is proved a linear function of the sum of diameters of its corresponding HC. With this new insight, an effective $O(n \log n)$-time $O(1)$-approximation algorithm and an optimal dynamic programming for ZST are designed. Using the ZST construction black box and a linear-time optimal tree decomposition algorithm, an improved algorithm for constructing the bounded-skew tree (BST) is derived. In the experiment, our approach shows superior wire length compared with previous methods for both ZST and BST.**

I. Introduction

Clock signal is distributed to synchronous elements in a VLSI circuit. In a clock tree, *skew*, the maximum difference in signal arrival time among all sinks, should be small to ensure timing correctness. There are in general two base formulations for clock tree construction, *zero-skew tree* (ZST) and *bounded-skew tree* (BST).

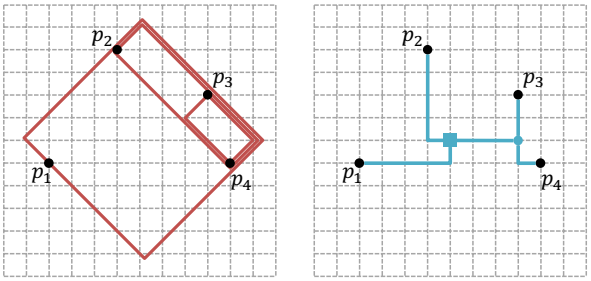
Many methods have been proposed for building ZSTs, e.g., [1], [2], [3], [4], [5], [6], [7]. Among them, deferred-merge embedding (ZST/DME) is a dynamic programming approach [3]. For a given topology, it outputs the locations of Steiner points achieving zero skew and optimal wire length. For determining the topology, Greedy-DME [5] is regarded as the best algorithm in practice (see [8], [9], [7]). Nowadays, ZST is not a good choice in practice due to two reasons. First, ZST is too expensive in wire length, which implies excessive power usage. Note that the clock net can consume 40% of the overall chip power due to its high frequency and large coverage [10]. Besides, longer wire length usually comes with larger path divergence and suffers from more on-chip variations. Second, ZST topology is not necessary, considering the large tolerance (due to buffer insertion and sizing [11], [12]) and the widely-used useful-skew optimization techniques [13], [14]. Nonetheless, ZST is still useful as it can serve as the backbone of a BST [6], [7].

Regarding BST, ZST/DME is extended to BST/DME by generalizing the merging segments to regions [9]. Due to the more complicated shapes of the merging regions, BST/DME prunes many possibilities (e.g., restrict the merging point to the boundary of a merging region or sample the interior points) and is not optimal for a given topology. Besides, skew variation is allocated over all levels relatively by chance.

With theoretical interest, approximation algorithms are also proposed [6], [7]. The best approximation ratios achieved are three and nine for ZST and BST respectively [7]. The algorithm is not as good as the best heuristics in practice, but provides several inspirations for this work. Besides, an integer linear programming (ILP) method is recently proposed for constructing optimal BST [15].

A problem similar to ZST is the *hierarchical clustering* (HC) [16]. In HC, each point starts as a cluster by itself, and pairs of clusters are merged when moving up the hierarchy. By retrospecting the classical DME algorithm, we found the equivalence between ZST and HC. To be more specific, the wire length of a ZST is a linear function of the sum of diameters of its corresponding HC (see Fig. 1 and Theorem 4 for a glance). With the help of the new insight as well as the "BST by ZST" idea from [6], [7], better algorithms for both ZST and BST construction are devised. Our contributions are summarized as follows.

- We proved the strict correspondence between the wire length of ZST and the diameter sum of HC.



(a) HC with diameter sum $= d(\{p_3, p_4\}) + d(\{p_2, p_3, p_4\}) + d(P) = 4 + 10 + 10 = 24$.

(b) ZST corresponding to (a) with wire length $= \frac{1}{2}(4 + 10) + 10 = 17$.

(c) HC with diameter sum $= d(\{p_3, p_4\}) + d(\{p_1, p_2\}) + d(P) = 4 + 8 + 10 = 22$.

(d) ZST corresponding to (c) with wire length $= \frac{1}{2}(4 + 8) + 10 = 16$.

Fig. 1: Equivalence between zero-skew tree (ZST) and hierarchical clustering (HC) on points $P = \{p_1, p_2, p_3, p_4\}$.

- With this new insight, we designed an effective $O(n \log n)$-time $O(1)$-approximation algorithm and an optimal dynamic programming for ZST construction.
- We proposed a linear-time optimal tree decomposition algorithm. Together with the black-box ZST construction, it leads to an improved BST generation method.

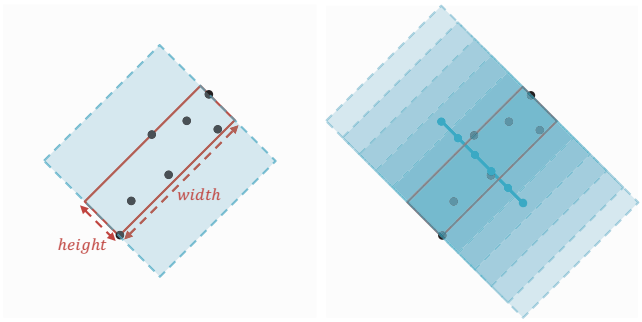
II. Zero-Skew Tree Properties

VLSI routing is in Manhattan space. Before formal illustration, we would like to define the notations for distance. For two points p_1 and p_2, $dist(p_1, p_2)$ represents their Manhattan (l_1) distance. For two sets of points P_a and P_b, their distance $dist(P_a, P_b)$ is defined as the smallest distance between a point in P_a and a point in P_b. That is, $dist(P_a, P_b) = \min_{p_1 \in P_a, p_2 \in P_b} dist(p_1, p_2)$. We use linear delay model in this paper.

Manhattan arc, the line segment with slope +1 or -1, is widely needed in ZST construction. For convenience and efficiency, the computation in the l_1 space can be converted to that in the l_∞ space. More specifically, suppose the l_∞ distance between p_1 and p_2 in the coordinate system tilted by 45° is $dist'(p1, p2)$. The distance in the l_∞ space can be obtained by scaling $dist'(p1, p2) = \frac{\sqrt{2}}{2} dist(p1, p2)$, and the Manhattan arc will become axis-aligned.

A. Manhattan Bounding Circle

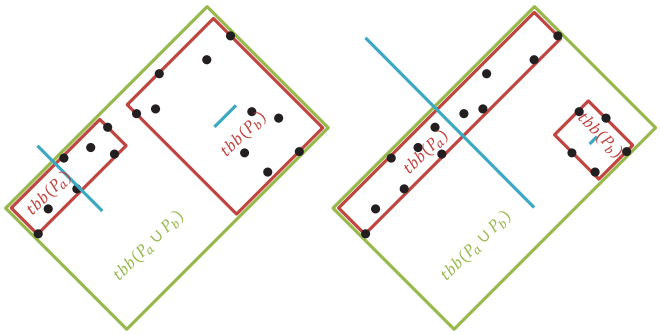
Manhattan circle $C(o, r)$ is a circle in the Manhattan space with *center* o and *radius* r. To be more specific, $C(o, r)$ is the set of all points that are



(a) The TBB (solid red) and an MBC (dash blue) for P

(b) Centers of all possible MBCs form center segment (solid blue)

Fig. 2: Manhattan bounding circle (MBC), tilted bounding box (TBB), and center segment for a set of points P.



(a) No domination

(b) P_a dominates P_b

Fig. 3: Relationship between two point sets P_a and P_b.

at a given Manhattan distance r from a given point o (i.e., $C(o,r) = \{p|dist(p,o) = r\}$). *Diameter* d of a circle C is the largest distance between any two points on the circle (i.e. $d = \max_{p_1,p_2 \in C} dist(p_1,p_2)$). Note that $d = 2r$.

For a set of points P, its *Manhattan bounding circle (MBC)* is a (minimum) Manhattan circle covering all the points (see Fig. 2(a)). Its *radius* $r(P)$ (resp. *diameter* $d(P)$) is the radius (resp. diameter) of the MBC. Note that for a point set P, there may be many MBCs, but the radius as well as the diameter of the MBCs are all the same (Fig. 2(b)). Regarding diameter $d(P)$, there is Lemma 1 (same as Property 11 of [17]), which provides another view to $d(P)$.

Lemma 1. *For a point set* P, $d(P) = \max_{p_1,p_2 \in P} dist(p_1,p_2)$.

For computational convenience, we can analyze and compute MBC in the 45°-tilted coordinate system. Refer the (minimum) bounding box for P in this tilted coordinate system as *tilted bounding box (TBB)* $tbb(P)$. A Manhattan circle is an MBC of points P iff it is an MBC of $tbb(P)$. In this way, it is easy to see that the centers of all possible MBCs of P form an Manhattan arc parallel to the shorter side of $tbb(P)$, which will be called the *center segment* $cs(P)$ hereafter.

For two point sets P_a and P_b, we say P_a *dominates* P_b if and only if $r(P_a) = r(P_a \cup P_b)$. The two kinds of relationships are illustrated by Fig. 3. We have Lemma 2 regarding domination. The proof is omitted due to the space limit.

Lemma 2. *If two point sets* P_a *and* P_b *do not dominate each other, then* $dist(cs(P_a), cs(P_b)) = 2r(P_a \cup P_b) - r(P_a) - r(P_b)$. *If* P_a *dominates* P_b, *then* $dist(cs(P_a), cs(P_b)) \leq r(P_a) - r(P_b)$.

B. ZST/DME by Manhattan Bounding Circle

ZST/DME algorithm [3] can find the optimal Steiner node placement for a given tree topology. For a tree T, we denote its vertex v (either leaf or Steiner node) by $v \in T$ and denote the leaves of the subtree rooted at v as $leaves(v)$. Path lengths from v to $leaves(v)$ are all the same and referred as $p(v)$. In ZST/DME, the *merging segment* $ms(v)$ of vertex $v \in T$ is a set of possible placements of v. There are two phases in ZST/DME. In the first bottom-up phase, a tree of merging segments is computed recursively. In the top-down phase, the merging point achieving the minimum wire length is picked according to the location of its parent Steiner node.

The reader may refer to [3] for algorithmic details, but in the bottom-up phase, $ms(v)$ for a vertex v with two children v_a and v_b is essentially computed as follows. Denote the nonnegative edge length cost from v to v_a (resp. v_b) as e_a (resp. e_b). The objective is to minimize $e_a + e_b$, which will be a part of the wire length of the final ZST. Suppose w.l.o.g. that $p(v_a) \leq p(v_b)$. Zero skew among $leaves(v) = leaves(v_a) \cup leaves(v_b)$ requires $e_a + p(v_a) = e_b + p(v_b)$. That is,

$$e_a - e_b = p(v_b) - p(v_a). \tag{1}$$

Meanwhile, by triangle inequality,

$$e_a + e_b \geq dist(ms(v_a), ms(v_b)), \tag{2}$$

Here, if $p(v_b) - p(v_a) \leq dist(ms(v_a), ms(v_b))$, then $e_a + e_b = dist(ms(v_a), ms(v_b))$. Otherwise, $e_a + e_b = p(v_b) - p(v_a)$, where $e_a = p(v_b) - p(v_a)$ and $e_b = 0$.

Based on Lemma 2, we can prove an intrinsic correspondence between DME and MBC (Theorem 3). It says that for a vertex v, the path length $p(v)$ equals the radius $r(leaves(v))$ of its leaves, while its merging segment $ms(v)$ is exactly the center segment $cs(leaves(v))$ of its leaves. For example, in Fig. 1, the path length from the ZST root is always $r(P) = 5$, regardless of the topology. Similarly, the merging segment of the ZST root is uniquely determined by $tbb(P)$. The key point of the proof is that whether $p(v_b) - p(v_a) \leq dist(ms(v_a), ms(v_b))$ depends on the domination relationship between $leaves(v_a)$ and $leaves(v_b)$.

Lemma 3 (DME-MBC Correspondence)**.** *In ZST/DME, for a vertex* v, *(i)* $p(v) = r(leaves(v))$; *(ii)* $ms(v) = cs(leaves(v))$.

Proof. See Appendix A. □

In this way, the merging segments in DME can be more efficiently computed. Moreover, the new insight unlocks several better clock tree construction methods, which will be introduced later.

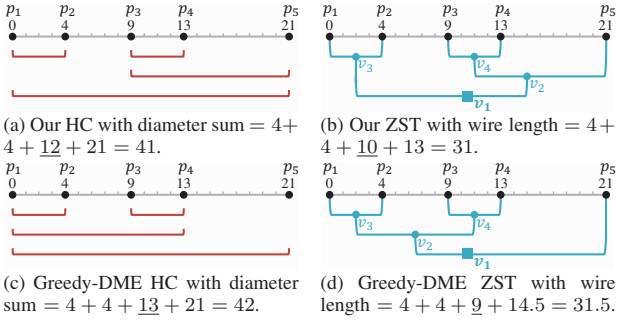
C. ZST by Hierachical Clustering

On a set of point P, a *hierachical clustering* (HC) [16] is a hierachy of clusters with each cluster composed of two sub-clusters or points. There are generally two approaches for creating it. In the agglomerative approach, each point starts as a cluster by itself, and pairs of clusters are merged and move up the hierarchy. By the divisive one, all points start in one cluster, and splits are performed recursively as one moves down the hierarchy.

W.l.o.g, assume that any ZST is binary (i.e., each Steiner node has two children). If a Steiner node has three (or more) children, overlapped Steiner node(s) can be inserted to make it binary without affecting the actual structure. In this way, a ZST topology T implies a HC, where $leaves(v)$ of every $v \in T$ is treated as a cluster, and vice versa (see Fig. 1 for two ZST/HC instances on the same set of points). Furthermore, we prove Theorem 4, which shows the equivalence between ZST and HC. To be more specific, the wire length of a ZST $length(T)$ is a linear function of the sum of diameters $\sum_{v \in T} d(leaves(v))$ of its corresponding HC. Besides, note that Theorem 4 is stronger than Lemma 2.1 of [7] and implies the later.

Algorithm 1 ZST by Iterative Merging

Require: Sinks P
Ensure: ZST T
1: Initialize clusters with each sink being a cluster by itself
2: **for** $i = 1, 2, ..., |P| - 1$ **do**
3: Merge the two clusters with the smallest diameter of their union among all pairs
4: Run DME top-down phase on the HC to realize the ZST T



(a) Our HC with diameter sum $= 4 + 4 + \underline{12} + 21 = 41$.
(b) Our ZST with wire length $= 4 + 4 + \underline{10} + 13 = 31$.
(c) Greedy-DME HC with diameter sum $= 4 + 4 + \underline{13} + 21 = 42$.
(d) Greedy-DME ZST with wire length $= 4 + 4 + \underline{9} + 14.5 = 31.5$.

Fig. 4: 1D example showing the impact of different preferences in iterative merging. Ours prefers small diameter of merged cluster, while Greedy-DME prefers small distance between two merging segments.

Theorem 4 (ZST-HC Equivalence)**.** *For points/sinks P in the Manhatten space, a ZST T has a wire length cost $length(T)$ equivalent to the sum of diameters $\sum_{v \in T} d(leaves(v))$ of its corresponding HC: $length(T) = \frac{1}{2}(\sum_{v \in T} d(leaves(v)) + d(P))$.*

Proof. See Appendix B. □

III. Zero-Skew Tree Construction

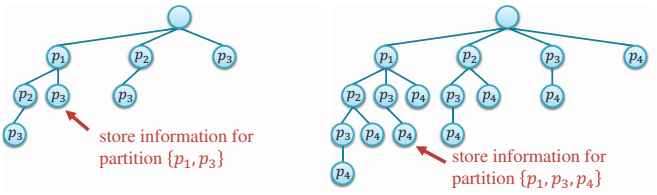
The zero-skew tree (ZST) problem has been converted to a hierarchical clustering (HC) problem with the diameter sum as objective, then how good can HC be solved? Two algorithms are proposed in this section.

A. Efficient and Effective Iterative Merging

With Theorem 4, a simple algorithm that comes up immediately is to iteratively merge two clusters with the smallest diameter resulted (Algorithm 1). *Complete-linkage clustering (c-link)* [16] is a popular agglomerative HC method where the merging in each iteration minimizes the maximum intra-cluster distance. It is noticeable that our algorithm is actually c-link under the Manhattan metric since the diameter of a cluster equals the the maximum intra-cluster distance in the Manhattan metric (Lemma 1).

Algorithm 1 is similar to Greedy-DME [5] except that the preference in selecting pairs is changed. The preference is changed from the distance between two merging segments to the diameter of the merged cluster. The diameter objective in our iterative merging has more global view and tends to generate better results. In Section V, the experimental results (TABLE I and Fig. 6) will evidence the difference. Here, an 1D example in Fig. 4 shows the idea. In the first two iterations, both methods get clusters $\{p1, p2\}$ and $\{p3, p4\}$. In the third iteration, there are two choices (ignoring the bad choice of merging $\{p1, p2\}$ and $\{p_5\}$). The first is merging $\{p3, p4\}$ with $\{p_5\}$, which gives a merged diameter of 12 and a merging segment distance of 10. The second choice is merging $\{p1, p2\}$ and $\{p3, p4\}$, which gives a merged diameter of 13 and a distance of 9. It turns out that the first choice with a smaller diameter is better with an eventual wire length of $31 < 31.5$. The detailed structural insight is in the proof of Theorem 4.

Moreover, Algorithm 1 is an $O(1)$-approximation for ZST/HC. Before showing it, we first introduce the non-hierarchical clustering problem, which serves for the HC lower bound. The *diameter k-clustering* [18], [19], [20] on points P is to partition P into k clusters and minimize the maximum diameter of the k clusters. A lower bound of HC on P is thus the diameter sum of a series of optimal k-clustering on P with $k = 1, 2, ..., |P| - 1$. Essentially, in ZST/HC, every Steiner node v can correspond a unique k-clustering with the maximum diameter being $d(leaves(v))$. The mapping can be obtained as follows. First, sort the Steiner nodes by $d(leaves(v))$ in descending order into $v_1, v_2, ..., v_{|P|-1}$. To get k clusters, we split at $v_1, v_2, ..., v_{k-1}$. Take the ZST/HC in Fig. 4(a)(b) as an example. Splitting at v_1 and v_2 leads to the 3-clustering of $\{\{p_1, p_2\}, \{p_3, p_4\}, \{p_5\}\}$, where the maximum diameter is $d(leaves(v_3))$.



(a) Trie for three sinks $\{p_1, p_2, p_3\}$ (b) Trie for four sinks $\{p_1, p_2, p_3, p_4\}$

Fig. 5: Trie for storing and retrieving the optimal sub-ZST/HC.

According to Theorem 24 of [21], in the l_p space, any k-clustering induced by c-link is an $O(1)$-approximation of the optimal k-clustering. Together with Theorem 4 and the above lower bound for ZST/HC, we have the following theorem.

Theorem 5. *Algorithm 1 is an $O(1)$-approximation for ZST.* [1]

Regarding the efficiency of Algorithm 1, it can be done in $O(n \log n)$ time and $O(n)$ space [23] with the help of the data structure for dynamic closest pair queries [24].

B. Optimal Dynamic Programming

In the ZST/HC problem, it is easy to show that an optimal ZST consists of two optimal sub-ZSTs (the two subtrees at the root), which enables a dynamic programming approach. The general idea is to expand the optimal solutions sink after sink, where the optimal ZST on a set P' is obtained by checking every bipartition (P_A, P_B) of P' and looking up the optimal ZSTs on subsets P_A and P_B.

Trie (prefix tree) [25] is used for efficiently storing and retrieving the optimal sub-ZST/HCs (see Fig. 5). Each node in the trie represents a partition/subset $P' \subseteq P$ of the input sinks P. Suppose $P' = \{p_{\pi_1}, p_{\pi_2}, ..., p_{\pi_i}\}$ and $\pi_1 < \pi_2 < ... < \pi_i$. The optimal cost and bipartition of P' will be stored at the corresponding trie node, which is at the end of the path $(p_{\pi_1}, p_{\pi_2}, ..., p_{\pi_i})$ (starting from the root of the trie). For example in Fig. 5(a), the optimal cost of partition $\{p_1, p_3\}$ is stored at the end of path (p_1, p_3). In such a trie data structure, a partition P' takes $O(1)$ memory only. Retrieving it takes $O(|P'|)$ time, which can be amortized to $O(1)$ if conducted incrementally.

The optimal trie-based dynamic programming (Algorithm 2) is as follows. Sinks are added one after another (line 3). For each new sink p_k, every existing trie node (i.e., partition) P' is expanded with it (line 4). To obtain the optimal ZST/HC on the expanded partition $P'' = P' \cup \{p_k\}$, all the bipartitions of P'' are enumerated (line 7). For each of the bipartitions, the costs of the two partitions need to be retrieved from the trie in order to compute the cost after merging (line 8). Note that one of the two partitions can have its cost retrieved incrementally by following a traversal on the trie to save runtime. An important issue for the dynamic programming is to order the generation of the trie nodes

[1]The hidden constant here is forwarded from Theorem 24 of [21] and is not small. We conjecture that a much tighter approximation ratio exists. In the experiment on realistic and random cases, the gap between Algorithm 1 and the optimal solution observed almost never exceeds 10%. The largest gap encountered on artificial cases is 22% (Fig. 11 of [22]).

Algorithm 2 Optimal ZST by Dynamic Programming

Require: Sinks P with $|P| = n$
Ensure: Optimal ZST T
1: Get the furthest-first traversal $(p_1, p_2, ..., p_n)$ of P
2: Initialize an empty trie
3: **for all** $k = 1, 2, ..., n$ **do**
4: **for all** existing trie node P' **do** ▷ By pre-order traversal
5: Skip if P' is marked as inferior
6: Initialize new trie node $P'' = P' \cup \{p_k\}$
7: **for all** bipartition of P'' **do**
8: Retrieve sub-partition cost from the trie
9: Update the optimal cost & solution of P''
10: Add P'' to be a child of P'
11: Obtain the optimal HC by backtracking from trie node P
12: Run DME top-down phase on the HC to realize the ZST T

Algorithm 3 ZST Refinement by Dynamic Programming

Require: Size of enumeration k, ZST vertex v ($|leaves(v)| \geq k$)
Ensure: Refined ZST beneath v
1: Vertices $U \leftarrow v.children$
2: **while** $|U| < k$ **do**
3: $u^* \leftarrow \arg\min_{u \in U} d(leaves(u))$
4: $U \leftarrow U - u^* + u^*.children$
5: Get the optimal ZST/HC above U by Algorithm 2

properly. The order of expanding the trie nodes should guarantee that a partition will have all of its sub-partitions expanded before its own expansion and with their costs available for looking up. The pre-order traversal that iterates children with larger indexes first will suffice.

Besides constructing an optimal ZST, Algorithm 2 can be generalized for improving a ZST/HC T (Algorithm 3). For a vertex $v \in T$, the ZST/HC beneath it can be refined by local reconstruction. Essentially, the hierarchy below v is first recursively broken to obtain k descendants U with their corresponding clusters $\{leaves(u)|u \in U\}$ having small diameters (lines 1–4). The optimal local HC above U is then obtained by Algorithm 3. The original Algorithm 3 runs on points/sinks, but it can be trivially extended for working on several clusters. Note that the local change here has no impact to the topologies either beneath U or above v according to Theorem 3. By setting k as a constant (eight in our implementation) and running Algorithm 3 on every $v \in T$ (with $|leaves(v)| \geq k$ to make sure k descendants exist), the refinement takes $O(n)$ time.

IV. Bounded-Skew Tree Construction

Inspired by [6], [7], our bounded-skew tree (BST) construction is to graft light topologies on a backbone of ZST. Compared with BST/DME [9], there are two significant differences. First, the flexibility of skew tolerance is put on lower levels (closer to leaves) instead of allocating over all levels by chance. Second, the algorithm is a simple combination of the efficient black-box constructions for light topologies and ZSTs, which means ease of implementation and runtime efficiency.

The algorithmic flow is described by Algorithm 4. The routing topology on the input sinks P is initialized to be a very light one T_M, e.g., generated by the rectilinear Steiner minimal tree (RMST) heuristics like FLUTE [26] (line 1). In order to satisfy the skew bound b, T_M is decomposed into several subtrees, where the distance between any two vertices in the same subtree along tree edges does not exceed $d_{max} = 2b$ (line 2). For each of the obtained subtrees, there thus exists a *center* that can reach all the vertices of the subtree within a distance of b. The centers P_M are then used as taps connecting to the clock source by a ZST T (line 4). In this way, distances from the source to sinks P are between $p(T)$ and $p(T) + b$, where $p(T)$ is the path length of T (from the clock source to P_M). A BST is thus resulted.

Algorithm 4 BST by Combining Light Topology with ZST

Require: Sinks P, skew bound b
Ensure: BST T_B
1: Construct light topology T_M on P (e.g., by RSMT heuristics)
2: Forests $F_M \leftarrow$ decomposition of T_M with maximum distance being $2b$ by Algorithm 5
3: Taps $P_M \leftarrow$ centers of F_M
4: $T \leftarrow$ ZST on P_M by Algorithm 1
5: **return** $T_B \leftarrow T \cup F_M$

Algorithm 5 Optimal Tree Decomposition

Require: Tree $T_M(V_M, E_M, w_M)$, maximum distance d_{max} allowed within each subtree
Ensure: Forest F_M with number of trees minimized
1: $F_M \leftarrow T_M$
2: $v.depth \leftarrow 0, \forall v \in V_M$
3: Make T_M rooted by specifying an arbitrary vertex as $root$
4: PostOrderTraversal($root$)
5: **function** PostOrderTraversal(v)
6: $v.depth \leftarrow w_M(v, v.parent)$ and return if v is a leaf
7: **for all** $u \in v.children$ **do**
8: PostOrderTraversal(u)
9: Sort $v.children$ by descending $depth$ to $u_1, u_2, ...$
10: **for** $i = 1, 2, ..., |v.children| - 1$ **do**
11: **if** $u_i.depth + u_{i+1}.depth > d_{max}$ **then**
12: Remove edge (u_i, v) from F_M
13: $u_i.depth \leftarrow 0$
14: $v.depth \leftarrow \max_{u \in v.children} u.depth + w_M(v, v.parent)$

The core algorithm that bridges RSMT and ZST is the optimal tree decomposition (Algorithm 5). It decomposes an input tree T_M into a minimum possible number of subtrees under the distance constraint d_{max}. A post-order traversal on T_M is run after a $root$ is specified (line 3). The information that a vertex u passes to its parent v is $u.depth$, which is the longest path from v to the current subtree of u (after possible decomposition of its original subtree). There may be multiple optimal decompositions for the original subtree of u. During the algorithm, $u.depth$ is made as small as possible among them. Lemma 6 explains how the invariant on $u.depth$ is kept by the greedy condition (line 11). The proof is omitted. By induction, we can prove Theorem 7 on the optimality of the whole algorithm.

Lemma 6. *In Algorithm 5, for children u_i and u_j of vertex v, if $u_i.depth + u_j.depth > d_{max}$, at least one of them should be decomposed from v.*

Theorem 7. *For a tree T, Algorithm 5 generates a decomposition with the minimum number of subtrees.*

The time of Algorithm 5 is $O(n)$, because each vertex is processed once and in $O(1)$ time. Note that the sorting in line 9 is in constant time due to the bounded degree of a vertex in RSMT. Moreover, it can be avoided by scanning $v.children$ a few passes. Essentially, all children with $depth \leq \frac{1}{2}d_{max}$ can always be kept, while all children with $depth > \frac{1}{2}d_{max}$ should be split except at most one.

V. Experimental Results

We implement our ZST and BST construction methods by C++ on a 3.8 GHz Linux machine. There are two kinds of benchmarks tested. The first is the realistic benchmark (prim1-2 from [1], r1-r5 from [4], and ISPD 2019 and 2010 contest benchmarks [27], [28]). The second is a large number of random nets with different number of sinks. We generate 100 nets for each number of sinks under a uniform distribution.

For ZST construction on realistic cases (TABLE I), our iterative merging (Algorithm 1, Dim Sum) is compared with other topology

generation methods including MMM [1], Rooted Kruskal [7], geometric merging algorithm (GMA) [2], balance bipartition (BB) [3], and Greedy-DME [5]. The results of GMA and BB are cited from [3]. The others are implemented by ourselves. The only implementation difference between Greedy-DME and Dim Sum is the score of merging a pair (the distance between two merging segments v.s. the diameter of the merged cluster), where the simple change leads to 1.5% improvement on average. Together with the dynamic-programming-based refinement, the wire length is furthered improved by 1.6%.

To systematically know the optimality of the methods, a batch test is conducted with random nets. A solid reference is the optimal ZST generated by the dynamic programming (Algorithm 2, Optimal Dim Sum). But it does not scale and is thus used on the small nets with up to 20 sinks. Even though Optimal Dim Sum is not polynomial-time, it is significantly more efficient than the ILP in [15]. The optimal BST by this ILP needs more time as skew bound decreases; for a relative skew bound of 0.3 and on 16 sinks, it already takes 3477.73 s [15]. Meanwhile, Optimal Dim Sum takes only 1.08 s for 16 sinks. In Fig. 6(a), each data point represents the average of the normalized wire length on 100 nets. As the number of sinks increases, the average gap from Dim Sum to Optimal Dim Sum becomes larger. However, for 20 sinks, Dim Sum (with refinement) still gets a small gap of 0.22%, while that of Greedy-DME is as large as 3.88%. Besides, note that Optimal Dim Sum on average takes 149.7 s, but Dim Sum takes 0.7 ms. With more sinks (up to 65536 in Fig. 6(b)), Dim Sum with refinement is used as the reference for normalization. The improvement over Greedy-DME reaches the peak of around 4% at 32 sinks. TABLE II shows the scalablity of Dim Sum. For 65536 sinks, even with refinement, it still needs 6.71 s only.

We implement our BST method (Algorithm 4) with RSMT heuristics FLUTE [26] for initial topologies. Fig. 7 shows the comparison with BST/DME [29], [9]. The implementation of BST/DME is obtained from its authors [30]. We use the wire length and skew of FLUTE to normalize the result on each net. The input skew bound is set to 0, 0.05, 0.1, ..., 1 of the skew of FLUTE. Due to the space limit, we only show two skew-wirelength trade-off curves. One is for r5, the largest net among those publicly-available cases (Fig. 7(a)). The other is for the 100 random nets with 16384 sinks (Fig. 7(b)). Here, our method shows better Pareto frontiers compared with BST/DME. In general, as the skew bound becomes larger, a smooth decrease of wire length from ZST to RSMT can also be observed. Besides, our method is significantly more efficient than BST/DME. For each method and each number of sinks, Fig. 7(c) shows an average runtime of all 100 nets and all 21 skew bounds. For relatively large nets, a 10× speed-up stably exists.

VI. Conclusion

Based on the intrinsic equivalence between ZST and HC, an efficient $O(1)$-approximation algorithm, Dim Sum, and an optimal dynamic programming, Optimal Dim Sum, are proposed for ZST construction. Besides, the optimal tree decomposition method enables a simple yet effective integration between ZST and RSMT. The directions of future work may include proving better approximation ratio for Dim Sum or designing even better algorithms based on the understanding of ZST-HC equivalence. For "BST by ZST", other clustering methods instead of the tree decomposition may worth trying. Besides, another line of research is to extend the idea to consider more practical factors (e.g, layer assignment, Elmore delay, etc).

References

[1] M. A. Jackson *et al.*, "Clock routing for high-performance ics," in *Proc. DAC*, 1990, pp. 573–579.

[2] J. Cong *et al.*, "Matching-based methods for high-performance clock routing," *IEEE TCAD*, vol. 12, no. 8, pp. 1157–1169, 1993.

[3] T.-H. Chao *et al.*, "Zero skew clock routing with minimum wirelength," *IEEE TCAS II*, vol. 39, no. 11, pp. 799–814, 1992.

[4] R.-S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE TCAD*, vol. 12, no. 2, pp. 242–249, 1993.

[5] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proc. DAC*, 1993, pp. 612–616.

[6] M. Charikar *et al.*, "Minimizing wirelength in zero and bounded skew clock trees," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 4, pp. 582–595, 2004.

[7] A. Z. Zelikovsky and I. I. Mandoiu, "Practical approximation algorithms for zero-and bounded-skew trees," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 97–111, 2002.

[8] A. B. Kahng and G. Robins, *On optimal interconnections for VLSI*. Springer Science & Business Media, 1994, vol. 301.

[9] J. Cong *et al.*, "Bounded-skew clock and steiner routing," *ACM TODAES*, vol. 3, no. 3, pp. 341–388, 1998.

[10] D. Papa *et al.*, "Physical synthesis with clock-network optimization for large systems on chips," *Proc. MICRO*, vol. 31, no. 4, pp. 51–62, 2011.

[11] C. J. Alpert *et al.*, "Buffered steiner trees for difficult instances," *IEEE TCAD*, vol. 21, no. 1, pp. 3–14, 2002.

[12] M. M. Ozdal *et al.*, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proc. ICCAD*, 2011, pp. 724–731.

[13] S. Held *et al.*, "Clock scheduling and clocktree construction for high performance asics," in *Proc. ICCAD*, 2003, p. 232.

[14] C. Y. Tan *et al.*, "Clustering of flip-flops for useful-skew clock tree synthesis," in *Proc. ASPDAC*, 2018, pp. 507–512.

[15] K. Han *et al.*, "A study of optimal cost-skew tradeoff and remaining suboptimality in interconnect tree constructions," in *Proc. SLIP*, 2018, p. 2.

[16] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[17] M. Edahiro, "Equispreading tree in manhattan distance," *Algorithmica*, vol. 16, no. 3, pp. 316–338, 1996.

[18] R. J. Fowler *et al.*, "Optimal packing and covering in the plane are NP-complete," *Information Processing Letters*, vol. 12, no. 3, pp. 133–137, 1981.

[19] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985.

[20] T. Feder and D. Greene, "Optimal algorithms for approximate clustering," in *Proc. STOC*, 1988, pp. 434–444.

[21] A. Großwendt and H. Röglin, "Improved analysis of complete-linkage clustering," *Algorithmica*, vol. 78, no. 4, pp. 1131–1150, 2017.

[22] M. R. Ackermann *et al.*, "Analysis of agglomerative clustering," *Algorithmica*, vol. 69, no. 1, pp. 184–215, 2014.

[23] D. Krznaric and C. Levcopoulos, "Optimal algorithms for complete linkage clustering in d dimensions," *Theoretical Computer Science*, vol. 286, no. 1, pp. 139–149, 2002.

[24] S. N. Bespamyatnikh, "An optimal algorithm for closest-pair maintenance," *Discrete & Computational Geometry*, vol. 19, no. 2, pp. 175–195, 1998.

[25] D. E. Knuth, *The art of computer programming*. Pearson Education, 1997, vol. 3.

[26] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE TCAD*, vol. 27, no. 1, pp. 70–83, 2008.

[27] C. N. Sze *et al.*, "ISPD 2009 clock network synthesis contest," in *Proc. ISPD*, 2009, pp. 149–150.

[28] C. N. Sze, "ISPD 2010 high performance clock network synthesis contest: benchmark suite and results," in *Proc. ISPD*, 2010, pp. 143–143.

[29] D. J. Huang *et al.*, "On the bounded-skew clock and steiner routing problems," in *Proc. DAC*, 1995, pp. 508–513.

[30] VLSI CAD software bookshelf: Bounded-skew clock tree routing. [Online]. Available: https://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/

Appendix

A. Proof of Lemma 3

By induction. Denote $r(leaves(v))$ as $r(v)$ and $cs(leaves(v))$ as $cs(v)$ for short. It is definitely true for a leaf v, since (i) $p(v) = r(v) = 0$, and (ii) $ms(v) = cs(v)$ degenerates from a segment to a fixed point. Suppose it is true for children v_a and v_b of vertex v.

(a) We will first prove the inductive part of $p(v) = r(v)$.

Assume w.l.o.g that $r(v_a) \geq r(v_b)$. Note that $e_b - e_a = p(v_a) - p(v_b) = r(v_a) - r(v_b) \geq 0$. There are two cases.

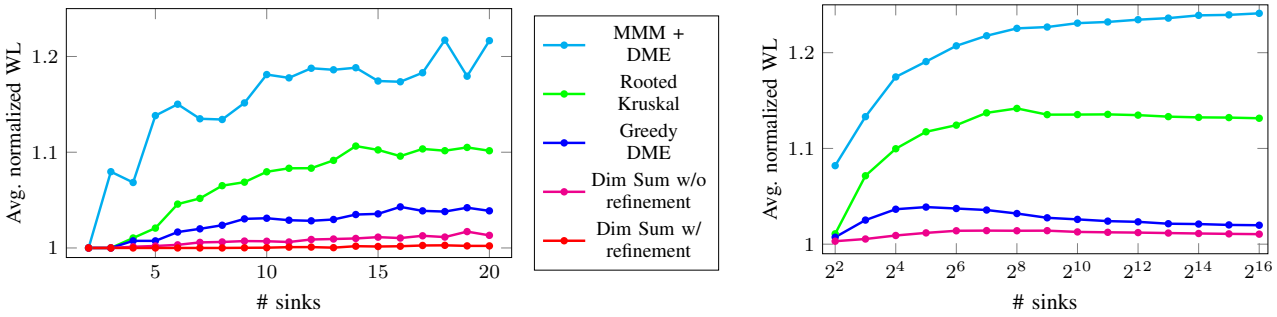
Case 1: No domination, i.e., $r(v) > r(v_a) \geq r(v_b)$. According to Lemma 2, $dist(ms(v_a), ms(v_b)) = dist(cs(v_a), cs(v_b)) = 2r(v) - r(v_a) -$

TABLE I: Wire Length Comparison of ZST Methods on Realistic Benchmarks (unit: μm)

Benchmarks	[1]		[4]					ISPD 2009 [27]										ISPD 2010 [28]								Avg.
	p1	p2	r1	r2	r3	r4	r5	f11	f12	f21	f22	f31	f32	f33	f34	f35	fnb1	fnb2	1	2	3	4	5	6	7	8	ratio
# sinks	269	594	267	598	862	1903	3101	121	117	117	91	273	190	209	157	193	330	440	1107	2249	1200	1845	1016	981	1915	1134	
MMM + DME	149	373	1601	3240	4165	8218	12274	186	171	205	113	439	321	315	270	324	45.2	120	355	636	57.8	126	66.3	49.6	93.3	61.9	1.310
Rooted Kruskal + DME	142	341	1461	2837	3711	7656	11052	190	171	202	116	420	314	317	257	290	41.5	110	306	564	31.2	99.8	45.6	38.5	71.1	44.2	1.138
GMA + DME	140	350	1497	3013	3902	7782	11665	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.173
BB + DME	141	361	1500	3010	3908	8000	11757	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1.185
Greedy DME	133	314	1313	2566	3339	6707	9943	174	156	192	105	380	291	292	241	273	36.6	96.6	277	510	28.2	87.4	40.4	34.0	62.6	40.2	1.031
Dim Sum w/o refinement	131	309	1297	2568	3285	6631	9801	176	157	184	106	375	283	287	238	268	35.0	95.4	273	504	27.8	86.6	40.0	33.3	61.5	39.2	1.016
Dim Sum w/ refinement	**128**	**306**	**1272**	**2506**	**3248**	**6545**	**9711**	**171**	**153**	**182**	**103**	**369**	**281**	**282**	**236**	**264**	**34.7**	**93.6**	**269**	**496**	**27.6**	**85.5**	**39.0**	**32.7**	**60.6**	**38.8**	**1.000**

TABLE II: Runtime of Dim Sum on Random Nets (unit: s)

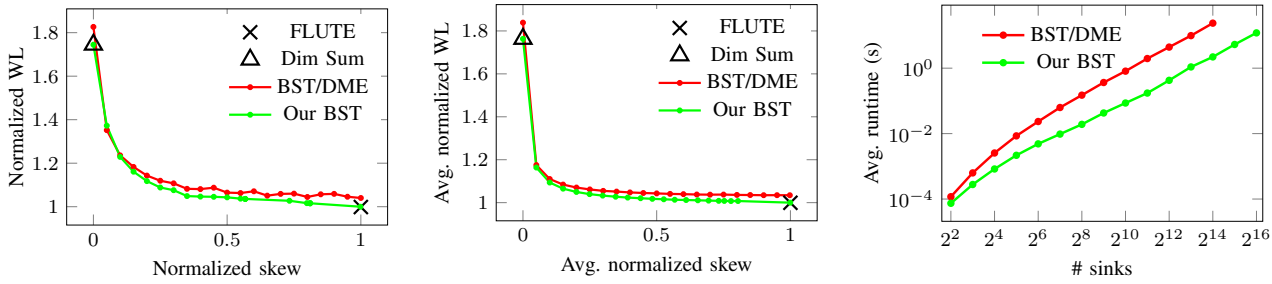
# sinks	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536
w/o refinement	0.00004	0.00005	0.00021	0.00048	0.00116	0.00256	0.00546	0.01209	0.02622	0.05805	0.17352	0.41776	0.84875	1.62305	3.37029
w/ refinement	0.00005	0.00032	0.00074	0.00186	0.00426	0.00853	0.01637	0.03444	0.07067	0.14736	0.37476	0.87238	1.75207	3.28965	6.70760



(a) On small nets. WL normalized by Optimal Dim Sum.

(b) On large nets. WL normalized by Dim Sum w/ refinement.

Fig. 6: Wire length (WL) comparison of ZST methods on random nets.



(a) WL on case r5 with 3101 sinks.

(b) WL on 100 random nets with 16384 sinks.

(c) Runtime on random nets with various sizes.

Fig. 7: Comparison between BST/DME and our BST construction method.

$r(v_b)$. Consider,

$$\begin{aligned} e_b - e_a &= r(v_a) - r(v_b) \\ &< 2(r(v) - r(v_a)) + r(v_a) - r(v_b) \\ &= 2r(v) - r(v_a) - r(v_b) \\ &= dist(ms(v_a), ms(v_b)), \end{aligned} \tag{3}$$

where the DME algorithm will let $e_b + e_a$ achieve its lower bound $dist(ms(v_a), ms(v_b))$. Then, $p(v) = \frac{1}{2}(p(v_a) + p(v_b) + e_a + e_b) = \frac{1}{2}(r(v_a) + r(v_b) + dist(ms(v_a), ms(v_b))) = \frac{1}{2}(2r(v)) = r(v)$.

Case 2: Having domination, i.e., $r(v) = r(v_a) \geq r(v_b)$. According to Lemma 2,

$$\begin{aligned} e_b - e_a &= r(v_a) - r(v_b) \\ &\geq dist(cs(v_a), cs(v_b)) \\ &= dist(ms(v_a), ms(v_b)). \end{aligned} \tag{4}$$

In this situation, DME lets $e_a = 0$ and $e_b = dist(ms(v_a), ms(v_b))$. Therefore, $p(v) = p(v_a) + e_a = p(v_a) = r(v_a) = r(v)$.

(b) For proving the induction on $ms(v) = cs(v)$, we will first show that $ms(v) \subseteq cs(v)$. For arbitrary $p \in v_a$, there is $dist(p, v) = dist(p, v_a) + dist(v, v_a) \leq r(v_a) + p(v) - p(v_a) = r(v)$. Similar for $p \in v_b$. That is, for $p \in v$, there is $dist(p, v) \leq r(v)$. Therefore, v should be an MBC center of v.

We then prove that $ms(v) \supseteq cs(v)$. That is, for arbitrary $o \in cs(v)$, $dist(o, ms(v_a)) = p(v) - p(v_a)$ and $dist(o, ms(v_b)) = p(v) - p(v_b)$. By $v_a \subset r(v)$ and Lemma 2, we have $dist(o, cs(v_a)) = dist(o, ms(v_a)) = r(v) - r(v_a) = p(v) - p(v_a)$. Similarly $dist(o, ms(v_b)) = p(v) - p(v_b)$ also holds and completes the proof.

B. Proof of Theorem 4

Simplify notation $r(leaves(v))$ as $r(v)$. We will prove $length(T) = \frac{1}{2}(\sum_{v \in T} d(leaves(v)) + d(P)) = \sum_{v \in T} r(v) + r(P)$ by induction.

Denote the subtree rooted at v as T_v. The claim then becomes $length(T_v) = \sum_{u \in T_v} r(u) + r(v)$. It is trivial for a leaf v, since (i) $length(T_v) = 0$, and (ii) $v = \{v\}$.

Suppose it is true for children v_a and v_b of vertex v. Note that $T_v = T_{v_a} + T_{v_b} + \{v\}$. Then it is sufficient to prove that $length(T) - length(T_{v_a}) - length(T_{v_b}) = \sum_{u \in \{v\}} r(u) + r(v) - r(v_a) - r(v_b) = 2r(v) - r(v_a) - r(v_b)$. Meanwhile, according to Theorem 3, there is $length(T) - length(T_{v_a}) - length(T_{v_b}) = e_a + e_b = (p(v) - p(v_a)) + (p(v) - p(v_b)) = 2p(v) - p(v_a) - p(v_b) = 2r(v) - r(v_a) - r(v_b)$.