PRACTICAL APPROXIMATION ALGORITHMS FOR ZERO- AND BOUNDED-SKEW TREES

ALEXANDER Z. ZELIKOVSKY† AND ION I. MĂNDOIU‡

Abstract. The skew of an edge-weighted rooted tree is the maximum difference between any two root-to-leaf path weights. Zero- or bounded-skew trees are needed for achieving synchronization in many applications, including network multicasting [G. N. Rouskas and I. Baldine, IEEE J. on Selected Areas in Communication, 15 (1997), pp. 346–356] and VLSI clock routing [H. Bakoglu, Circuits, Interconnections, and Packaging for VLSI, Addison-Wesley, Reading, MA, 1990, A. B. Kahng and G. Robins, On Optimal Interconnections for VLSI, Kluwer Academic Publishers, Norwell, MA, 1995]. In these applications edge weights represent propagation delays, and a signal generated at the root should be received by multiple recipients located at the leaves (almost) simultaneously. The objective is to find zero- or bounded-skew trees of minimum total weight, since the weight of the tree is directly proportional to the amount of resources (bandwidth and buffers for network multicasting, power and chip area for clock routing in VLSI) that must be allocated to the tree. Charikar et al. in [Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, ACM, New York, 1999, pp. 177–184] have recently proposed the first strongly polynomial algorithms with proven constant approximation factors, $2e \approx 5.44$ and $16.86$, for finding minimum weight zero- and bounded-skew trees, respectively.

In this paper we introduce a new approach to these problems, based on zero-skew “stretching” of spanning trees, and obtain algorithms with improved approximation factors of 4 and 14. For the case when tree nodes are points in the plane and edge weights are given by the rectilinear metric our algorithms find zero- and bounded-skew trees of length at most 3 and 9 times the optimum. This case is of special interest in VLSI clock routing. An important feature of our algorithms is their practical running time, which is asymptotically the same as the time needed for computing the minimum spanning tree.

Key words. Steiner trees, clock routing, VLSI physical design, approximation algorithms

AMS subject classifications. 05C05, 05C85, 68W25, 68W35, 68W40

PII. S0895480100378367

1. Introduction. The skew of an edge-weighted rooted tree is the maximum difference between any two root-to-leaf path weights. Zero- or bounded-skew trees are needed for achieving synchronization in many applications, including network multicasting [20] and VLSI clock routing [2, 17]. In these applications edge weights represent propagation delays, and a signal generated at the root should be received by multiple recipients, referred to as sinks, located at the leaves (almost) simultaneously. The goal is to find zero- or bounded-skew trees of minimum total weight, since the weight of the tree is directly proportional to the amount of resources (bandwidth and buffers for network multicasting, power and chip area for clock routing in VLSI) that must be allocated to the tree.
In order to meet the skew constraints in the above applications, one may increase edge weights of the underlying network or metric space. This corresponds to adding buffers to a network link, or wire wiggling, respectively. We will refer to this operation as stretching. Formally, let \((M, d)\) be an arbitrary metric space. A stretched tree \(T = (V, E, \pi, \text{cost})\) for a set of sinks \(S \subseteq M\) is a rooted tree with node set \(V\) and edge set \(E\), together with a pair of mappings, \(\pi : V \rightarrow M\) and \(\text{cost} : E \rightarrow \mathbb{R}_+\), such that

1. \(\pi\) is a one-to-one mapping between the leaves of \(T\) and \(S\), and
2. for every edge \((u, v) \in E\), \(\text{cost}(u, v) \geq d(\pi(u), \pi(v))\).

Informally, every edge \((u, v)\) of a stretched tree \(T\) embedded in \((M, d)\) can be stretched by wiggling such that its length increases from \(d(\pi(u), \pi(v))\) to \(\text{cost}(u, v)\).

A stretched tree \(T\) is a zero-skew tree (ZST) if all root-to-leaf paths in \(T\) have equal cost; \(T\) is a \(b\)-bounded-skew tree (\(b\)-BST, or just BST when the bound \(b\) is clear from the context) if the difference between the cost of any two root-to-leaf paths is at most \(b\).

The two problems that we study in this paper are the following:

Zero-skew tree problem. Given a set of sinks \(S\) in metric space \((M, d)\), find a minimum cost zero-skew tree for \(S\).

Bounded-skew tree problem. Given a set of sinks \(S\) in metric space \((M, d)\) and a bound \(b > 0\), find a minimum cost \(b\)-bounded-skew tree for \(S\).

The ZST and BST problems are NP-hard \([8]\). The restriction of the BST problem to the rectilinear plane is also known to be NP-hard, but the complexity of the rectilinear ZST problem is not known—for a fixed tree topology the problem can be solved in linear time by using the deferred-merge embedding (DME) algorithm independently introduced in \([5, 6, 10]\).

Although the rectilinear zero- and bounded-skew tree problems have received much attention in the VLSI CAD literature \([3, 5, 6, 7, 9, 10, 11, 15, 16, 19]\) (see Chapter 4 of \([17]\) for a detailed review), the first algorithms with constant approximation factors have been proposed only recently by Charikar et al. \([8]\). They give algorithms with approximation factors of \(2e \approx 5.44\) and 16.86 for the ZST and BST problems, respectively. The BST algorithm in \([8]\) relies on an approximation algorithm for the Steiner tree problem in graphs. Using the best current Steiner tree approximation of Robins and Zelikovsky \([21]\) and Arora’s PTAS for computing rectilinear Steiner trees \([1]\), the BST bounds in \([8]\) can be updated to 16.11 for arbitrary metric spaces and to 12.53 for the rectilinear plane (see Table 1).

In this paper we introduce a new approach to these problems, based on zero-skew “stretching” of spanning trees. Our contributions include the following:

- constructive lower bounds on the cost of the optimum ZST and BST in arbitrary metric spaces;
- improved approximation for the ZST problem in arbitrary metric spaces, based on a reduction to the zero-skew spanning tree problem;
- improved approximation for the ZST problem in metrically convex metric spaces,\(^1\) based on skew elimination using Steiner points;
- improved approximation for the BST problem in arbitrary and metrically convex metric spaces, based on combining an approximate ZST with a minimum spanning tree for the sinks.

An important feature of our algorithms is their practical running time, which is asymptotically the same as the time needed for computing a minimum spanning tree. Thus,

\(^{1}\)A metric space \((M, d)\) is called metrically convex if, for every \(u, v \in M\) and \(0 \leq \lambda \leq 1\), there exists a point \(w \in M\) such that \(d(u, w) = \lambda d(u, v)\) and \(d(w, v) = (1 - \lambda)d(u, v)\).
Table 1
Summary of results and comparison to results of Charikar et al. [8]. Values marked with asterisks update those reported in [8] by taking into account the best current Steiner tree approximation of Robins and Zelikovsky [21] and Arora’s PTAS for computing rectilinear Steiner trees [1].

<table>
<thead>
<tr>
<th>Problem</th>
<th>Zero-skew tree</th>
<th>Bounded-skew tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>General</td>
<td>M. convex</td>
</tr>
<tr>
<td>Approximation factor in [8]</td>
<td>$2e \approx 5.44$</td>
<td>16.11*</td>
</tr>
<tr>
<td>Approximation factor in this paper</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Runtime in [8]</td>
<td>strongly polynomial</td>
<td>strongly polynomial</td>
</tr>
<tr>
<td>Runtime in this paper</td>
<td>$O(n^2)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

our algorithms can easily handle the clock nets with hundreds of thousands of sinks that occur in large cell based or multichip module designs. For a summary of our results and a comparison to the results of Charikar et al. [8] we refer the reader to Table 1.

The rest of the paper is organized as follows. In the next section we prove new lower bounds on the cost of the optimal ZST and BST. Then, in section 3, we show how to convert (or “stretch”) a rooted tree $T$ spanning the set $S$ of sinks into a ZST for $S$. We show that such “stretching” increases the cost by the sum of sink delays, where the delay in $T$ of a sink $s$ is the length of the path connecting $s$ to its furthest descendant. We also show that, for metrically convex metric spaces such as the Euclidean or rectilinear planes, it is possible to reduce the cost increase to half the sum of delays.

In section 4 we give a Kruskal-like algorithm that builds a rooted spanning tree $T$ whose total delay does not exceed its length and whose length is at most twice the cost of an optimal ZST. These two facts yield an approximation factor of 4 for the ZST problem in arbitrary metric spaces and an approximation factor of 3 for metrically convex metric spaces. In section 5 we discuss the implications of combining our ZST heuristics with the DME algorithm when solving rectilinear instances.

Finally, in section 6, we describe how to construct approximate BSTs by combining an approximate ZST for a subset of the sinks with subtrees of a minimum spanning tree (MST) or approximate minimum Steiner tree for the sinks. In combination with the MST, this gives a 14-approximation algorithm for the BST problem in arbitrary metric spaces; the factor is reduced to 11 for arbitrary metrically convex metric spaces and to 9 for the rectilinear plane.

2. Constructive lower bounds. In this section, we establish new lower bounds for the ZST and BST problems in an arbitrary metric space. In contrast to the lower bounds of Charikar et al. [8] these bounds are constructive. A practical advantage of constructive lower bounds is that they can give tighter bounds on the quality of the computed solution on an instance by instance basis.

The minimum cost of a ZST (BST) for $S$ will be denoted by $ZST^*(S)$ (respectively, $BST^*(S)$). In our analysis we will use the following constructive lower bound on $ZST^*(S)$.

---

2 The running time in [8] is not explicitly estimated.
Lemma 2.1. Let $S$ be a set of $n$ sinks. Then, for any enumeration $s_1, s_2, \ldots, s_n$ of the sinks in $S$,

\[ ZST^*(S) \geq \text{MinDist}\{s_1, s_2\} + \frac{1}{2} \sum_{i=2}^{n-1} \text{MinDist}\{s_1, \ldots, s_{i+1}\}, \]

where \( \text{MinDist}\{A\} = \min_{u, v \in A, u \neq v} d(u, v) \).

Proof. For any $r \geq 0$, let $N(r)$ denote the minimum number of closed balls of radius $r$ of $(M, d)$ needed to cover all sinks in $S$. Charikar et al. [8] established that

\[ ZST^*(S) \geq \int_{0}^{R} N(r) \, dr, \]

where $R$ is the smallest radius $r$ for which $N(r) = 1$.

Let $r_i = \text{MinDist}\{s_1, \ldots, s_{i+1}\}/2$ for every $i = 1, \ldots, n-1$, and let $r_n = 0$. Clearly, $R \geq r_1 \geq r_2 \cdots \geq r_{n-1} \geq r_n$. Note that $N(r) \geq i + 1$ for every $r < r_i$, since no two points in the set $\{s_1, \ldots, s_{i+1}\}$ can be covered by the same ball of radius $r$. Hence,

\[ \int_{0}^{R} N(r) \, dr \geq \sum_{i=1}^{n-1} \int_{r_{i+1}}^{r_i} (i+1) \, dr = \sum_{i=1}^{n-1} (i+1)(r_i - r_{i+1}) = 2r_1 + \sum_{i=2}^{n-1} r_i, \]

and the lemma follows. \( \Box \)

It can be shown that natural greedy enumerations (e.g., start from a diametrical pair of points and add each time the point maximizing minimum distance to previously enumerated points) do not always deliver the maximum to the lower bound established in Lemma 2.1. The complexity of finding the best enumeration is an open question.

Below we bound the cost of the optimum BST by comparing it with the cost of the optimum ZST for a subset of the sinks.

Lemma 2.2. Let $S$ be a set of sinks. Then, for any $W \subseteq S$ and skew bound $b > 0$,

\[ \text{BST}^*(S) \geq ZST^*(W) - b \cdot (|W| - 1). \]

Proof. Let $T$ be a $b$-BST for $S$. We use $T$ to construct a ZST for $W$ of cost no larger than $\text{cost}(T) + b \cdot (|W| - 1)$ as follows. First, notice that $T$ contains a $b$-BST for $W$, say $T'$, as a subtree. Let $P_u$ denote the unique path in $T'$ connecting $u$ to the root, and let $u_0$ be a leaf of $T'$ for which $\text{cost}(P_{u_0})$ is maximum. We get a ZST for $W$ by adding to $T'$ a loop of cost $\text{cost}(P_{u_0}) - \text{cost}(P_u)$ for each leaf $u \neq u_0$. Since $T'$ has skew at most $b$, each of the $|W| - 1$ added loops has cost at most $b$. Thus, the resulting ZST has cost at most $\text{cost}(T') + b \cdot (|W| - 1) \leq \text{BST}^*(S) + b \cdot (|W| - 1)$. \( \Box \)

3. Zero-skew stretching of spanning trees. Let $T = (S, E)$ be a rooted tree spanning a set $S$ of sinks from metric space $(M, d)$. For any sink $u$, let $T_u$ denote the subtree of $T$ rooted at $u$. The delay in $T$ of $u$ is defined by

\[ \text{delay}_T(u) = \max\{\text{length}(P_{uv}) \mid v \text{ leaf in } T_u\}, \]

where $P_{uv}$ denotes the unique path in $T$ connecting $u$ and $v$, and $\text{length}(P_{uv}) = \sum_{e \in P_{uv}} d(e)$. 


Let \( \text{length}(T) = \sum_{e \in E} d(e) \) and \( \text{delay}(T) = \sum_{u \in S} \text{delay}_T(u) \). In this section we show that, for any metric space \((M, d)\), \(T\) can be stretched to a ZST of cost \( \text{length}(T) + \text{delay}(T) \). The stretched ZST uses no Steiner points, i.e., has all nodes embedded at the sinks. We also show that, by using Steiner points, the amount of stretching can be reduced to half the delay of \(T\) in case the underlying space is metrically convex.

### 3.1. Zero-skew stretching in arbitrary metric spaces.

The stretching algorithm for arbitrary metric spaces (Figure 2) constructs a ZST \(T_1\) from a given rooted tree \(T\) spanning \(S\). The construction proceeds in two phases. In the first phase (Steps 1–3) the following transformation is applied to each sink \(u\) (see Figure 1). First, the children \(v_1, \ldots, v_k\) of \(u\) are sorted in nondecreasing order of \(d(u, v_i) + \text{delay}_T(v_i)\). Then \(k\) new nodes \(u_1, \ldots, u_k\) are embedded at \(u\) and connected to \(u\) by a path of total cost \(\text{delay}_T(u)\). Finally, each \(v_i\) is disconnected from \(u\) and reattached to \(u_i\) by an edge of cost \(d(u, v_i)\). The result of the first phase is a tree \(T_1\) in which every sink either is a leaf or has a single child.

In the second phase (Steps 4–5) we convert \(T_1\) into a ZST for \(S\) as follows. First, we change the root of \(T_1\) to \(r' = r_t\), where \(r\) is the root of \(T\) and \(t = \text{deg}_T(r)\). Notice that every sink \(u\) that is not yet a leaf in \(T_1\) is incident to its parent, say \(v\), and to \(u_1\). For every such sink \(u\) the edge \((u, v)\) is replaced in \(T_1\) with \((u_k, v)\), where \(k = \text{deg}_T(u)\). After this transformation all sinks become leaves in \(T_1\).

**Lemma 3.1.** The stretched tree \(T_1\) produced by the algorithm in Figure 2 is a ZST with total cost \(\text{length}(T) + \text{delay}(T)\).

**Proof.** We will prove that every path in \(T_1\) from \(u_k\), \(u \in S\), \(k = \text{deg}_T(u)\), to a descendant sink has cost equal to \(\text{delay}_T(u)\); this immediately implies that \(T_1\) is a ZST. Let \(v_1, \ldots, v_h\) be the sorted children of \(u\) in \(T\), and let \(u_1, \ldots, u_k\) be the copies of \(u\) added to \(T_1\) in Step 3. Consider a path \(P\) from \(u_k\) to a descendant sink \(s\) going through edge \((u_i, w)\), where \(w = \text{deg}_T(v_i)\)th copy of \(v_i\). Inductively, we can assume that the cost of the path from \(w\) to \(s\) is equal to \(\text{delay}_P(v_i)\). Hence, it suffices to show that the cost of the path from \(u_k\) to \(w\) is equal to \(\text{delay}_T(u) - \text{delay}_T(v_i)\). Indeed,

---

3For clarity, in Figure 2 we omit curly braces for single element sets and use “−” and “∪” instead of “\(\setminus\)" and “\(\cup\)”, respectively.
the cost of this path is
\[ \text{cost}(u, v) \leq d(u, v) \]

1. \( V_1 \leftarrow S; \pi(v) \leftarrow v \) for each \( v \in V_1 \).
2. \( E_1 \leftarrow E; \text{cost}(u, v) \leftarrow d(u, v) \) for each \( (u, v) \in E_1 \).
3. For each sink \( u \in S \), do:
   \[ k \leftarrow \deg_T(u) \]
   Sort \( u \)'s children in \( T \), say \( v_1, v_2, \ldots, v_k \), such that
   \[ d(u, v_1) + \text{delay}_T(v_1) \leq d(u, v_2) + \text{delay}_T(v_2) \leq \cdots \leq d(u, v_k) + \text{delay}_T(v_k) \]
   // Add \( k \) new nodes embedded at \( u \)
   \[ V_1 \leftarrow V_1 + \{ u_1, \ldots, u_k \}; \quad \pi(u_1) = \cdots = \pi(u_k) \leftarrow u \]
   // Connect the \( k \) new nodes and \( u \) with a path
   \[ E_1 \leftarrow E_1 + (u, u_1); \quad \text{cost}(u, u_1) \leftarrow d(u, v_1) + \text{delay}_T(v_1) \]
   For \( i = 1, \ldots, k-1 \) do
   \[ E_1 \leftarrow E_1 + (u_i, u_{i+1}); \quad \text{cost}(u_i, u_{i+1}) \leftarrow \left[ d(u, v_{i+1}) + \text{delay}_T(v_{i+1}) \right] - \left[ d(u, v_i) + \text{delay}_T(v_i) \right] \]
   // Reattach children \( v_i \) to the corresponding copies of \( u \)
   For \( i = 1, \ldots, k \) do
   \[ E_1 \leftarrow E_1 - (u, v_i) + (u_i, v_i); \quad \text{cost}(u_i, v_i) \leftarrow \text{cost}(u, v_i) \].
4. Change the root of \( T_1 = (V_1, E_1) \) from \( r \) to \( r_1 \), where \( r = \deg_T(r) \).
5. For each sink \( u \in S - r, \deg_T(u) > 0 \), do:
   \[ v \leftarrow \text{parent}_{T_1}(u); \quad k \leftarrow \deg_T(u) \]
   \[ E_1 \leftarrow E_1 - (u, v) + (u_k, v); \quad \text{cost}(u_k, v) \leftarrow \text{cost}(u, v) \].
6. Output \( T_1 = (V_1, E_1, \pi, \text{cost}) \).

Fig. 2. The zero-skew stretching algorithm for arbitrary metric spaces.

the cost of this path is
\[
\text{cost}(w, u) + \text{cost}(u_i, u_{i+1}) + \cdots + \text{cost}(u_{k-1}, u_k)
\]
\[
= d(v_i, u) + \sum_{j=i}^{k-1} \{ [d(u, v_{j+1}) + \text{delay}_T(v_{j+1})] - [d(u, v_j) + \text{delay}_T(v_j)] \}
\]
\[
= [d(u, v_k) + \text{delay}_T(v_k)] - \text{delay}_T(v_i)
\]
\[
= \text{delay}_T(u) - \text{delay}_T(v_i).
\]

A similar computation shows that the cost of the path from \( u_k \) to \( u \) is \( d(u, v_k) + \text{delay}_T(v_k) = \text{delay}_T(u) \).

The cost of \( T_1 \) is equal to \( \text{length}(T) \) after Step 2 of the algorithm. In Step 3 it increases for each sink \( u \in S \) by the cost of the path \( (u, u_1, u_2, \ldots, u_k) \), i.e., by \( \text{delay}_T(u) \). Hence, the total cost of \( T_1 \) is
\[
\text{length}(T) + \sum_{u \in S} \text{delay}_T(u) = \text{length}(T) + \text{delay}(T) \quad \Box
\]

3.2. Zero-skew stretching in metrically convex metric spaces. Before stating the algorithm, we need to introduce some more notation. A path \( P = (p_1, p_2, \ldots, p_k) \) in \( T_1 \) is called critical if it ends at a leaf node \( p_k \) and contains no loops. By construction, it follows that the tree \( T_1 \) produced by the algorithm in Figure 2 has at least one critical path starting from each node. Let \( P = (p_1, p_2, \ldots, p_k) \)
Input: Rooted spanning tree \( T = (S, E) \) in a metric space \( (M, d) \).
Output: ZST \( T_2 = (V_2, E_2, \pi, \text{cost}) \) for \( S \).

1. Find \( T_1 = (V_1, E_1, \pi, \text{cost}) \) using the algorithm in Figure 2.
2. \( (V_2, E_2, \pi, \text{cost}) \leftarrow (V_1, E_1, \pi, \text{cost}) \).
3. For each sink \( u \in S \) and \( i = 0, 1, \ldots, \deg_S(u) \), do:
   - // Add attachment node \( w_i \) on the critical path from \( u_{i+1} \)
     - Find edge \( (x, y) = e(P, \delta/2) \) on the critical path \( P \) from \( u_{i+1} \), where \( \delta = \text{cost}(u_i, u_i+1) \).
     - \( V_2 \leftarrow V_2 + w_i; \quad \pi(w_i) \leftarrow v(P, \delta/2) \)
     - \( E_2 \leftarrow E_2 - (x, y) + (x, w_i) + (w_i, y) \)
     - \( \text{cost}(x, w_i) \leftarrow d(x, \pi(w_i)); \quad \text{cost}(w_i, y) \leftarrow d(\pi(w_i), y) \)
     - // Replace the loop \( (u_i, u_{i+1}), \) where \( u_0 = u \), with the edge \( (u_i, w_i) \)
     - \( E_2 \leftarrow E_2 - (u_i, u_{i+1}) + (u_i, w_i); \quad \text{cost}(u_i, w_i) \leftarrow \delta/2. \)
4. Output \( T_2 = (V_2, E_2, \pi, \text{cost}) \).

Fig. 3. Loop folding in metrically convex metric spaces.

Fig. 4. The zero-skew stretching algorithm for metrically convex metric spaces.

be a critical path in \( T_1 \), and let \( \text{length}(P) = \text{length}(\pi(p_1), \pi(p_2), \ldots, \pi(p_k)) \). For every \( 0 \leq \delta \leq \text{length}(P) \), there exist \( i \) such that \( \text{length}(\pi(p_1), \pi(p_2), \ldots, \pi(p_i)) \leq \delta < \text{length}(\pi(p_1), \pi(p_2), \ldots, \pi(p_{i+1})) \). We denote the edge \( (p_i, p_{i+1}) \) by \( e(P, \delta) \).

The improved stretching algorithm for metrically convex metric spaces (Figure 4) first computes a ZST \( T_1 \) using the algorithm in Figure 2. Then it “folds” half of each loop along a critical path of \( T_1 \) (see Figure 3). Folding can be applied to each loop \( (u_i, u_{i+1}) \), since \( \text{cost}(u_i, u_{i+1}) \) is at most the length of the critical path \( P \) from \( u_{i+1} \). Indeed, by Lemma 3.1, every path from \( u_{i+1} \) to a descendant leaf has the same cost. Hence, \( \text{cost}(u_i, u_{i+1}) \leq \text{cost}(P) \). Finally, since \( P \) does not contain loops, each edge of \( P \) has cost equal to the distance between the embedding of its ends, and thus \( \text{cost}(P) = \text{length}(P) \).

Lemma 3.2. The stretched tree \( T_2 \) produced by the algorithm in Figure 4 has zero-skew and total cost equal to \( \text{length}(T) + \text{delay}(T)/2 \).

Proof. The total cost of the loops in the stretched tree \( T_1 \) is equal to \( \text{delay}(T) \). Step 3 of the algorithm replaces each loop by an edge with half its cost. Therefore, \( \text{cost}(T_2) = \text{length}(T) + \text{delay}(T)/2 \). The tree \( T_2 \) has zero-skew, since \( T_1 \) has zero-skew,
**Input:** Finite set $S \subseteq M$.

**Output:** Rooted spanning tree $T$ on $S$.

1. Initialization:
   
   $ROOTS \leftarrow S$; $E \leftarrow \emptyset$
   
   For each $v \in S$, $h(v) \leftarrow 0$.

2. While $|ROOTS| > 1$ do:
   
   Find the closest two sinks $r, r' \in ROOTS$ with respect to metric $d$
   
   If $h(r) < h(r')$, then swap $r$ and $r'$
   
   $E \leftarrow E + (r, r')$
   
   $h(r) \leftarrow \max\{h(r), d(r, r') + h(r')\}$
   
   $ROOTS \leftarrow ROOTS - r'$.

3. Output the tree $T = (S, E)$, rooted at the only remaining sink in $ROOTS$.

**Fig. 5.** The Rooted-Kruskal algorithm.

and loop folding preserves the cost of all root-to-leaf paths.

4. **ZST approximation via spanning trees.** In the previous section we have shown that any rooted spanning tree can be stretched into a ZST whose cost is equal to the length of the spanning tree plus its delay (half the delay, for metrically convex metric spaces). This motivates the following:

   **Zero-skew spanning tree problem.** Given a set of points $S$ in a (metrically convex) metric space $(M, d)$, find a rooted spanning tree $T$ on $S$ such that $\text{cost}(T) = \text{length}(T) + \text{delay}(T)$ (respectively, $\text{length}(T) + \text{delay}(T)/2$) is minimized.

   Note that the MST on $S$ has the shortest possible length but may have very large delay—if the MST is a simple path, then its delay may be as much as $O(n)$ times larger than its length. On the other hand, a star having the least delay may be $O(n)$ times longer than the MST.

   In this section we give an algorithm for finding a rooted spanning tree which has both delay and length at most two times the minimum ZST cost. Therefore, our algorithm gives factor 4 and 3 approximations for the ZST problem in general and metrically convex metric spaces, respectively. Simultaneously, our algorithm gives factor 4 and 3 approximations for the zero-skew spanning tree problem in the respective metric spaces, since $\text{cost}(T)$ cannot be smaller than the cost of the minimum ZST.

   The algorithm (Figure 5) can be thought of as a rooted version of the well-known Kruskal MST algorithm. At all times, the algorithm maintains a collection of rooted trees spanning the sinks; initially, each sink is a tree by itself. In each step, the algorithm chooses two trees that have the smallest distance between their roots and merges them by linking the root of one tree as the child of the other. In order to keep the delay of the resulting tree small, the child root is always chosen to be the root with smaller delay.

   **Lemma 4.1.** $\text{delay}(T) \leq \text{length}(T)$.

   **Proof.** Note that, at the end of the Rooted-Kruskal algorithm, $h(u)$ represents exactly the delay of node $u$ in $T$. Every iteration of the algorithm adds the edge $(r, r')$ to $E(T)$, thus increasing $\text{length}(T)$ by $d(r, r')$. On the other hand, since $h(r) \geq h(r')$ when $h(r)$ is updated, the iteration contributes at most $d(r, r') + h(r') - h(r) \leq d(r, r')$ to $\sum_{u \in S} h(u)$, i.e., to the total delay of $T$. □
Let $n$ be the number of sinks in $S$.

**Lemma 4.2.** \( \text{length}(T) \leq 2(1 - 1/n)\text{ZST}^*(S) \).

**Proof.** Let \( s_1 \) be the root of \( T \), and let \( s_2, \ldots, s_n \) be the remaining \( n - 1 \) nodes of \( T \), indexed in reverse order of their deletion from \( \text{ROOTS} \). Since in each iteration the algorithm adds to \( T \) the edge joining a closest pair of points in \( \text{ROOTS} \),
\[
\text{length}(T) = \sum_{i=1}^{n-1} \text{MinDist}\{s_1, \ldots, s_{i+1}\}.
\]
Thus, by Lemma 2.1,
\[
\text{length}(T) \leq 2 \text{ZST}^*(S) - \text{MinDist}\{s_1, s_2\} = 2 \text{ZST}^*(S) - d(s_1, s_2).
\]
Since \( (s_1, s_2) \) is the longest edge in \( T \), \( d(s_1, s_2) \geq \text{length}(T)/(n - 1) \), and the lemma follows. \( \square \)

Lemmas 3.1, 4.1, and 4.2 give the following theorem.

**Theorem 4.3.** For any metric space and any set of \( n \) sinks, running the algorithm in Figure 2 on the tree \( T \) produced by the Rooted-Kruskal algorithm gives a ZST whose cost is at most \( 4(1 - 1/n) \) times larger than \( \text{ZST}^*(S) \).

**Proof.** By Lemma 3.1, the cost of the embedding is equal to \( \text{length}(T) + \text{delay}(T) \). However, \( \text{delay}(T) \leq \text{length}(T) \) by Lemma 4.1, and the approximation factor follows from Lemma 4.2. \( \square \)

Similarly, Lemmas 3.2, 4.1, and 4.2 give the following theorem.

**Theorem 4.4.** For any metrically convex metric space and any set of \( n \) sinks, running the algorithm in Figure 4 on the tree \( T \) produced by the Rooted-Kruskal algorithm gives a ZST whose cost is at most \( 3(1 - 1/n) \) times larger than \( \text{ZST}^*(S) \).

**Proof.** By Lemma 3.2, the cost of the embedding is now equal to \( \text{length}(T) + (1/2) \cdot \text{delay}(T) \), and the theorem follows again from Lemmas 4.1 and 4.2. \( \square \)

The following example shows that the algorithm in Theorem 4.3 can produce ZSTs which are \( 4(1 - 1/n) \) times larger than optimal. A similar example shows that the algorithm in Theorem 4.4 has a tight approximation factor of \( 3(1 - 1/n) \).

**Example 1.** Consider a discrete metric space on \( 2^k + 1 \) points, \( n = 2^k \) of which are sinks. We label the sinks with 0-1 sequences of length \( k \), i.e., \( S = \{ \alpha = b_{k-1}b_{k-2} \ldots b_0 \mid b_i \in \{0, 1\} \} \). All sink-to-sink distances are equal to 1 and the distance from the single Steiner point to each of the sinks is 1/2. In this space, the optimal ZST is a star rooted at the Steiner point and has cost equal to \( n/2 \). The Rooted-Kruskal algorithm may construct the spanning tree \( T \) with root \((11 \ldots 1)\) and edges \((\alpha, \alpha')\) such that \( \alpha' \) is identical to \( \alpha \) except that the rightmost 0 in \( \alpha' \) is replaced with 1 in \( \alpha \). Indeed, at each iteration of Step 2, the algorithm may choose to merge trees rooted at \( \alpha \) and \( \alpha' \) as above. It may choose \( \alpha \) to be the root of the merged tree, since \( h(\alpha) = h(\alpha') \).

Clearly, \( \text{length}(T) = n - 1 \). On the other hand, since we always merge two roots with the same \( h \)-value, each merge contributes exactly 1 to the total delay of \( T \). Thus, \( \text{delay}(T) = n - 1 \). By Lemma 3.1, the cost of the ZST produced by the algorithm is
\[
\text{length}(T) + \text{delay}(T) = 2(n - 1) = 4(1 - 1/n) \cdot \frac{n}{2}.
\]

**Running time.** The running time of the stretching algorithms given in section 3 is dominated by the time needed to sort the children of each node; this can be done in \( O(n \log n) \) overall. For arbitrary metrics the Rooted-Kruskal algorithm can
Fig. 6. When Figure 4 is applied to the Rooted-Kruskal spanning tree, the topology of the stretched tree remains the same, since each attachment node \( w_i \) belongs to the edge \((u_{i+1}, v_{i+1})\).

be implemented in \( O(n^2) \) time using Eppstein’s dynamic closest-pair data structure [12]. In the rectilinear plane (in fact, in any fixed dimensional \( L_p \) space), the running time can be reduced to \( O(n \log n) \) time by using the dynamic closest-pair data structure of Bespamyatnikh [4]. These implementations of the Rooted-Kruskal algorithm are asymptotically optimal, since the running times match known lower bounds for computing the first closest pair.

Thus, the total time for running the Rooted-Kruskal algorithm followed by one of the stretching algorithms given in section 3 is \( O(n^2) \) in arbitrary metric spaces (respectively, \( O(n \log n) \) in the rectilinear plane). Notice that this matches asymptotically the time needed for computing an MST for the sinks.

5. Practical considerations for approximating the rectilinear ZST. In the previous two sections it has been shown that the minimum cost ZST can be approximated in metrically convex metric spaces within a factor of 3. In order to obtain better ZSTs in the rectilinear plane, we may combine the stretched spanning tree with the DME algorithm [5, 6, 10]. The DME algorithm gives the optimal rectilinear ZST for any given topology, which is an unweighted binary tree with the leaves labeled by the sinks. Therefore, we may shorten only the rectilinear ZST if we feed the topology of the stretched spanning tree into the DME algorithm.

In section 3 we suggested two different ways of stretching a spanning tree. One may expect that the topology produced by the algorithm in Figure 4 (the loop folding algorithm) is superior to the topology produced by the algorithm in Figure 2. Surprisingly, when stretching the spanning tree produced by the Rooted-Kruskal algorithm, both algorithms lead to the same topology. As proven below, every attachment node \( w_i \) inserted by the algorithm in Figure 4 belongs to the edge \((u_{i+1}, v_{i+1})\). Hence, loop folding does not change the topology of the stretched tree (see Figure 6).

**Theorem 5.1.** Let \( T \) be the rooted spanning tree constructed by the Rooted-Kruskal algorithm. In any metrically convex metric space, the topologies produced by running Figures 2 and 4 on \( T \) are identical.

**Proof.** Let the children \( \{v_1, \ldots, v_k\} \) of a node \( u \) be sorted as in the algorithm in Figure 2, i.e., in nondecreasing order of \( d(u, v_i) + \text{delay}_T(v_i) \). For brevity, denote \( d_i = d(u, v_i) \) and \( D_i = \text{delay}_T(v_i) \). We will show that \( \delta = \text{cost}(u_i, u_{i+1}) \) is no greater than \( d_{i+1} \). This will ensure that the attachment node \( w_i \) lies on the edge \((u_{i+1}, v_{i+1})\) and, therefore, the tree topologies produced by the two stretching algorithms are the
same (see Figure 6). Since \( \delta = (d_{i+1} + D_{i+1}) - (d_i + D_i) \), it suffices to prove that

\[
D_{i+1} \leq d_i + D_i. \tag{5.1}
\]

We say that index \( k \) precedes index \( l \) if the node \( v_k \) has been attached to \( u \) before \( v_l \) in the Rooted-Kruskal algorithm. Let \( p_1 \) be the maximum index preceding \( i + 1 \), \( p_2 \) be the maximum index preceding \( p_1 \), and so on, until we arrive at an index \( p_m \) with \( D_{p_m} = 0.4 \) Then \( d_{p_1} + D_{p_1} \) represents the length of the critical path from \( u \) at the time when \( v_{i+1} \) is linked to \( u \) by the Rooted-Kruskal algorithm, and \( d_{p_{i+1}} + D_{p_{i+1}} \) is the length of the critical path from \( u \) at the time when \( v_{p_i} \) is linked to \( u \).

Notice that, since the distance between the closest two sinks in ROOTS does not decrease during the Rooted-Kruskal algorithm,

\[
d_{i+1} \geq d_{p_1} \geq \cdots \geq d_{p_m}. \tag{5.2}
\]

Moreover,

\[
D_{i+1} \leq d_{p_1} + D_{p_1} \tag{5.3}
\]

and

\[
D_{p_j-1} \leq d_{p_j} + D_{p_j} \tag{5.4}
\]

for every \( j = 2, \ldots, m-1 \), since through all attachments node \( u \) remains the root.

Assume, for a contradiction, that (5.1) does not hold. We will show by induction on \( j \) that \( p_j > i + 1 \) and \( D_{i+1} \leq D_{p_j} \) for every \( j = 1, \ldots, m \). Since \( D_{p_m} = 0 \), the above claim implies that \( D_{i+1} = 0 \), making (5.1) trivially true.

To prove the claim, first consider \( j = 1 \). If \( p_1 \leq i \), then \( d_{p_1} + D_{p_1} \leq d_i + D_i \), and (5.3) implies (5.1). So, it must be the case that \( i + 1 < p_1 \). Then \( d_{i+1} + D_{i+1} \leq d_{p_1} + D_{p_1} \), and (5.2) implies that \( D_{i+1} \leq D_{p_1} \).

Now assume that \( D_{i+1} \leq D_{p_j-1} \) for some \( j \geq 2 \). If \( p_j \leq i \), using (5.4) we get

\[
D_{i+1} \leq D_{p_{j-1}} \leq d_{p_j} + D_{p_j} \leq d_i + D_i.
\]

So, it must be the case that \( i + 1 < p_j \). Then \( d_{i+1} + D_{i+1} \leq d_{p_j} + D_{p_j} \), and, since \( d_{i+1} \geq d_{p_j} \) by (5.2), this implies that \( D_{i+1} \leq D_{p_j} \).

**Corollary 5.2.** Combination of the Rooted-Kruskal algorithm with the stretching algorithm for arbitrary metric spaces (Figure 2) and with the DME algorithm gives a 3-approximation for the rectilinear ZST problem.

### 6. Approximate BSTs

In this section we give two approximation algorithms for the BST problem, both built around a black-box ZST approximation algorithm. In both cases we construct a ZST for an appropriately chosen subset of the sinks, then extend this ZST to a \( b \)-BST for all sinks. In the first algorithm (Figure 7) the extension is done by adding subtrees of an MST on the sinks; in the second (Figure 8) subtrees are extracted from an approximate Steiner tree.

---

4We will always arrive at an index \( p_m \) with \( D_{p_m} = 0 \), since at least one child of \( u \) has zero delay. Indeed, let \( v \) be the child first connected to \( u \). At the moment when the edge \((u, v)\) is added by the Rooted-Kruskal algorithm, \( u \) has zero delay and thus \( v \) must also have zero delay. The delay of \( v \) never changes after its removal from ROOTS.
**Input:** Finite set $S \subseteq M$, bound $b > 0$.

**Output:** $b$-BST for $S$.

1. Find an MST $T_0$ on $S$, with respect to the metric $d$, and choose an arbitrary sink $r$ as root.
2. Find a set $W$ of sinks and a collection of subtrees of $T_0$, $(B_u)_{u \in W}$, as follows:
   - $W \leftarrow \emptyset$; $T \leftarrow T_0$
   - While $T \neq \emptyset$ do:
     - Find a sink $v$ of $T$ which is furthest from the root
     - Find the highest ancestor, say $u$, of $v$ that still has $\text{delay}_T(u) \leq b$
     - $W \leftarrow W + u$;
     - $B_u \leftarrow T_u$;
     - $T \leftarrow T - (u, \text{parent}(u)) - B_u$.
3. Find an approximate ZST, $T_1$, for $W$.
4. Output the tree $T' = T_1 \cup \bigcup_{u \in W} B_u$ rooted at the root of $T_1$.

**Fig. 7.** The MST based BST algorithm.

### 6.1. The MST based algorithm.

The first algorithm (Figure 7) uses a simple iterative construction to cover the sinks by disjoint $b$-skew subtrees of an MST $T_0$ of $S$. The algorithm then outputs the union of these subtrees with a ZST $T_1$ on their roots. Clearly, the resulting tree $T'$ is a $b$-BST for $S$. Moreover, $\text{cost}(T') \leq \text{cost}(T_1) + \text{length}(T_0)$, since the subtrees are disjoint pieces of $T_0$. Hence, if the ZST algorithm used in Step 3 has an approximation factor of $r_{ZST}$, by Lemma 2.2 we get that

$$\text{cost}(T') \leq r_{ZST} \text{ZST}^*(W) + \text{length}(T_0)$$

$$\leq r_{ZST} (\text{BST}^*(S) + b \cdot (|W| - 1)) + \text{length}(T_0).$$

For each node $u \neq r$ added to $W$ in Step 2 of the algorithm in Figure 7, the path from the parent of $u$ to the sink $v$ is deleted from the tree. Since $v$ is a furthest sink, the length of this path is equal to $\text{delay}_T(\text{parent}(u))$. By the choice of $u$, $\text{delay}_T(\text{parent}(u)) > b$. Thus, $b \cdot (|W| - 1) \leq \text{length}(T_0)$, and so

$$\text{cost}(T') \leq r_{ZST} \text{BST}^*(S) + (r_{ZST} + 1)\text{length}(T_0).$$

Let $r_{MST}$ be the Steiner ratio for the metric space $(M, d)$, i.e., the supremum, over all sets of points $S$ in $(M, d)$, of the ratio between the length of an MST and the length of a minimum Steiner tree for $S$. Since the length of the minimum Steiner tree for $S$ is a lower bound on $\text{BST}^*(S)$, we get that $\text{length}(T_0) \leq r_{MST} \text{BST}^*(S)$. Hence, we have the following theorem.

**Theorem 6.1.** The algorithm in Figure 7 has an approximation factor of $r_{ZST} + r_{MST} + r_{ZST} r_{MST}$.

Since the Steiner ratio is at most 2 for any metric space [18], and 3/2 for the rectilinear plane [13], by using the results in Theorems 4.3 and 4.4 we get the following corollary.

**Corollary 6.2.** The approximation factor of the algorithm in Figure 7 is 14 in arbitrary metric spaces, 11 in arbitrary metrically convex metric spaces, and 9 in the rectilinear plane.

Notice that the running time of the algorithm in Figure 7 is still $O(n \log n)$ for the rectilinear plane and $O(n^2)$ for arbitrary metric spaces. The MST in Step 1 can
Input: Finite set $S \subseteq M$, bound $b > 0$.
Output: $b$-BST for $S$.

1. Find an approximate Steiner tree $T_0$ on $S$, with respect to the metric $d$.
2. Find a set $W$ of sinks and a collection of subtrees of $T_0$, $(B_u)_{u \in W}$, as follows:
   \begin{itemize}
   \item $W \leftarrow \emptyset$; $T \leftarrow T_0$
   \item While $T \neq \emptyset$ do:
   \begin{itemize}
   \item Pick an arbitrary sink $u$ in $T$, and let $B_u$ be the subtree of $T$ induced by vertices within tree distance of at most $b$ from $u$
   \item $W \leftarrow W \cup \{u\}$; $T \leftarrow T \setminus B_u$
   \end{itemize}
   \end{itemize}
3. Find an approximate ZST, $T_1$, for $W$.
4. Output the tree $T' = T_1 \cup (\bigcup_{u \in W} B_u)$.

Fig. 8. The approximate Steiner tree based BST algorithm.

be computed within these time bounds using Hwang’s [14] rectilinear MST algorithm and Kruskal’s algorithm, respectively, while Step 2 can be implemented in linear time.

6.2. The approximate Steiner tree based algorithm. The second BST algorithm combines a ZST for a subset $W$ of the sinks with $b$-skew subtrees of an approximate Steiner tree $T_0$ (Figure 8).

Theorem 6.3. The BST problem can be approximated within a factor of $r_{ZST} + r_{SMT} + 2r_{ZST}r_{SMT}$, given $r_{ZST}$ (respectively, $r_{SMT}$), approximation algorithms for the ZST, and minimum Steiner tree problems.

Proof. By construction, the distance in $T_0$ between any two sinks in $W$ is at least $b$. Consider the set of open balls of radius $b/2$ centered at the sinks in $W$, with the balls considered in the metric space induced by $T_0$. Since any two such balls are disjoint, and each of them must cover at least $b/2$ worth of edges of $T_0$, we get that

$$b|W| \leq 2 \text{length}(T_0). \quad (6.1)$$

To estimate the cost of the BST produced by the algorithm, notice that $\bigcup_{u \in W} B_u$ has total cost of at most $\text{length}(T_0)$. By Lemma 2.2 and (6.1), we get

$$\text{cost}(T') \leq r_{ZST}ZST^*(W) + \text{length}(T_0)$$
$$\leq r_{ZST}(BST^*(S) + b \cdot (|W| - 1)) + \text{length}(T_0)$$
$$\leq r_{ZST}(BST^*(S) + 2 \text{length}(T_0)) + \text{length}(T_0),$$

and the theorem follows by observing that $\text{length}(T_0) \leq r_{SMT}BST^*(S)$, since, as noted above, the length of the minimum Steiner tree for $S$ is a lower bound on $BST^*(S)$.

With the currently known approximation factors for Steiner trees and ZST, Theorem 6.1 gives better BST approximations than Theorem 6.3 for the rectilinear plane, as well as arbitrary (metrically convex) metric spaces. However, Theorem 6.3 may improve upon Theorem 6.1 for metric spaces with good Steiner tree approximation ($r_{SMT}$ close to 1) and large Steiner ratio ($r_{MST}$ close to 2), e.g., for high dimensional $L_p$ spaces.

7. Conclusions and open problems. We have given approximation algorithms for the ZST and BST problems with improved approximation factors for
general and metrically convex metric spaces, as well as the rectilinear plane. Our algorithms have a practical running time: $O(n \log n)$ in the rectilinear plane and $O(n^2)$ in general metric spaces. Preliminary experiments also show that, when combined with the linear time DME algorithm of [5, 6, 10], our rectilinear ZST algorithm gives results competitive to those obtained by the Greedy DME heuristic of Edahiro [11], which is regarded in the VLSI CAD community as the best ZST heuristic to date (see [17]).

An interesting open question is to determine the limitations of the spanning-tree based ZST construction introduced in this paper. One can define the zero-skew Steiner ratio of a metric space as the supremum, over all sets of sinks, of the ratio between the minimum zero-skew cost (i.e., length $+$ delay) of a spanning tree and the minimum ZST cost. The results in section 4 imply that the zero-skew Steiner ratio is at most 4 in arbitrary metric spaces, and at most 3 in metrically convex metric spaces. On the other hand, we have constructed instances showing that the zero-skew Steiner ratio can be as large as 3 for arbitrary metric spaces; we conjecture that the ratio is never larger than 3. Determining the complexity of the zero-skew spanning tree problem is another interesting open question.

In the planar versions of the rectilinear ZST and BST problems, one seeks zero and bounded-skew trees in the rectilinear plane with no self-intersecting edges. Charikar et al. [8] have given the first constant approximation factors for these versions; it would be interesting to find algorithms with improved approximation factors.

REFERENCES


