Fast Euclidean Minimum Spanning Tree: Algorithm, Analysis, and Applications

William B. March Parikshit Ram Alexander G. Gray
School of Computational Science & Engineering, Georgia Institute of Technology
266 Ferst Dr., Atlanta, GA 30332
{march@, p.ram@, agray@cc.} gatech.edu

ABSTRACT

The Euclidean Minimum Spanning Tree problem has applications in a wide range of fields, and many efficient algorithms have been developed to solve it. We present a new, fast, general EMST algorithm, motivated by the clustering and analysis of astronomical data. Large-scale astronomical surveys, including the Sloan Digital Sky Survey, and large simulations of the early universe, such as the Millennium Simulation, can contain millions of points and fill terabytes of storage. Traditional EMST methods scale quadratically, and more advanced methods lack rigorous runtime guarantees. We present a new dual-tree algorithm for efficiently computing the EMST, use adaptive algorithm analysis to prove the tightest (and possibly optimal) runtime bound for the EMST problem to-date, and demonstrate the scalability of our method on astronomical data sets.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; I.5.3 [**Clustering**]: Algorithms

General Terms

Algorithms, Theory

Keywords

Adaptive Algorithm Analysis, Euclidean Minimum Spanning Trees

1. INTRODUCTION

We present a new algorithm for the fundamental and widely applied Euclidean Minimum Spanning Tree (EMST) problem. Given a set of points S in $\mathbb{R}^d$, our goal is to find the lowest weight spanning tree in the complete graph on S with edge weights given by the Euclidean distances between points. With references in the literature as early as 1926, the MST problem is one of the oldest and most thoroughly studied problems in computational geometry [36]. In addition to this long-standing theoretical and algorithmic interest, the MST is useful for many practical data analysis problems. Many optimization problems can be posed as the search for the MST in a network [36]. The MST is also used as an approximation for the traveling salesman problem [24], in document clustering [48], analysis of gene expression data [15], wireless network connectivity [46], percolation analyses [8], and modeling of turbulent flows [44], among other areas. These problems are commonly solved in the Euclidean setting. In this case, the computational bottleneck in both traditional MST algorithms like Kruskal's [28] and Prim's [37] and more advanced methods is finding the nearest neighbor of components in a spanning forest. We propose a new method to overcome this obstacle and demonstrate its theoretical and experimental superiority.

In particular, we are interested in using the EMST to compute hierarchical clusterings [20, 53]. One such clustering is obtained by deleting all edges longer than a specified cutoff in the MST, generating a clustering through the remaining connected components. By varying the scale of the cutoff, this generates a hierarchical clustering. In the clustering literature, this is often referred to as a *single-linkage clustering* and is frequently represented by a *dendrogram.* While the single-linkage clustering is very simple and can be sub-optimal for many applications, it can form the basis of more insightful clusterings. The single linkage clustering can be pruned to obtain more useful astronomical results [4]. MST's also form the inner loop for methods to identify non-parametric clusters in noisy data [49]. Furthermore, theoretically optimal clusterings can be obtained efficiently from the single-linkage clustering [3].

In astronomy, EMST-based clustering is used to analyze deep-space surveys and simulations of the early universe. Each level of single-linkage clustering is known as a *friend-of-friends* clustering [40, 2]. The EMST is used to identify dark matter haloes in simulations, which are believed to be crucial to galaxy formation [29]. Clustering is also applied to sky surveys to identify the super-large scale structure of the universe, which sheds light on the conditions of the early universe and the mechanisms of galaxy formation [4].

The volume of data produced in the astronomy community has grown explosively in recent years. Recent large surveys include the Las Campanas Redshift Survey (26,418 objects) [42], the 6dF Galaxy Survey (125,071 galaxies) [25], the 2dF Galaxy Redshift Survey (382,323 objects) [13], and the Sloan Digital Sky Survey (over 230 million objects) [52]. In addition, our understanding of cosmology has benefitted from large-scale simulations of the formation of galaxies and conditions in the early universe. For example, the Millennium Simulation [43] contains over 1 billion points and produces terabytes of output. The analysis of the current structure of the universe as revealed in the large sky surveys and comparison to the predictions of theories in simulations are the keys to understanding the origins of the cosmos and validating new models, including verification of dark matter and dark energy. This in turn requires the ability to compute minimum spanning trees quickly and accurately for very large data from a variety of distributions.

Adaptive Analysis. Traditional approaches to algorithm analysis use the running time for the worst possible input as an upper bound for the running time of all instances. This often leads to overly pessimistic bounds due to a few pathological inputs. Adaptive analysis seeks to improve these results by considering properties of the inputs in the analysis. By bounding the runtime in terms of these properties, one can obtain tighter and more informative bounds. Adaptive analysis has been successfully applied to many fundamental problems including searching in lists [6], merging arrays [14], sorting [16], and the convex hull problem [27]. Despite these successes, the difficulty of characterizing the inputs in relation to the problem has limited the number of applications.

Our Contribution. We present a new Euclidean minimum spanning tree algorithm, DualTreeBoruvka. Using the dual-tree algorithmic framework [22], we can efficiently compute the shortest edge between components in a spanning forest, thus overcoming the bottleneck of most EMST methods. We show:

- The **first application of adaptive algorithm analysis to the EMST problem** in order to achieve tighter and more precise runtime bounds to-date.
- The **asymptotically fastest EMST runtime:**

  $$O(N \log N\, \alpha(N)) \approx O(N \log N)$$

  where $\alpha(N)$ is related to the functional inverse of Ackermann's function and $\alpha(10^{80}) \leq 4$. Our analysis, unlike some previous work, accounts for complexity of tracking connected components in a partial MST and reduces the difference between upper and lower bounds to a factor of $\alpha(N)$.
- The **fastest experimental results** compared to the previous state-of-the art, the GeoMST2 algorithm [31] and Bentley and Friedman's *kd*-tree-based method [5], on a range of synthetic and natural data sets, including both high- and low-dimensional data and data from astronomical surveys and simulations.

Paper Outline. We review MST algorithms, both general and Euclidean, and their runtime bounds in section 2. In section 3, we define certain inherent properties of a dataset that are necessary to characterize the difficulty of computing the MST. In section 4, we describe the DualTreeBoruvka algorithm and bound the running time of the cover-tree-based version in section 5. In section 6, we compare our algorithm to several competing methods with empirical timings for synthetic data and astronomical surveys, and we conclude in section 7.

2. RELATED WORK

Many MST algorithms rely on Tarjan's blue rule [45], which says the minimum weight edge across any edge cut is in the minimum spanning tree. This allows us to greedily form cuts in the graph and add the minimum weight edge across each. Algorithms using this rule include Kruskal's [28] and Prim's [37], which require $O(m \log n)$ and $O(m+n \log n)$ time, respectively, on a graph with n points and m edges. Both algorithms maintain one or more components in spanning forest and use the cut between one component of the forest and the rest of the graph, adding the edges found in this way one at a time.

Boruvka's Algorithm. In this work, we focus on the earliest known minimum spanning tree algorithm, Borůvka's algorithm, which dates from 1926. See [32] for a translation and commentary on Boruvka's original papers. As in Kruskal's algorithm, a minimum spanning forest is maintained throughout the algorithm. Kruskal's algorithm adds the minimum weight edge between any two components of the forest at each step, thus requiring $N-1$ steps to complete. Borůvka's algorithm finds the minimum weight edge incident with each component, and adds all such edges, thus requiring at most $\log N$ steps and a total running time of $O(m \log n)$. We define the *nearest neighbor pair* of a component C as the pair of points $q \in C, r \notin C$ that minimizes $d(q,r)$. Finding the nearest neighbor pair for each component and adding the edges (p,q) to the forest is called a *Boruvka step.* Boruvka's algorithm then consists of forming an initial spanning forest with each point as a component and iteratively applying Boruvka steps until all components are joined.

General MST Algorithms. More recently, sophisticated algorithms have been developed for the MST problem on general graphs. Fredman & Tarjan [17] showed a bound of $O(m \log^* n)$, which was soon improved to $O(m \log \log^* m)$ [19]. Yao further improved the bound to $O(m \log \log n)$ [50]. Chazelle showed $O(m\alpha \log \alpha)$, where $\alpha(m,n)$ is a functional inverse of Ackermann's function [11]. Chazelle [12] and Pettie [34] improved this to $O(m\alpha)$. The current tightest bound, due to Pettie & Ramachandran [35] in 2002, is $O(\mathcal{T}^*(m,n))$, where $\mathcal{T}^*$ is bounded from below by $\Omega(m)$ and above by $O(m \cdot \alpha)$. All these general algorithms are insufficient for large, metric problems because they depend linearly on the number of edges. In the Euclidean case, the edge set consists of all pairs of points. Therefore, linear scaling in m corresponds to quadratic scaling in the number of points, and thus we need to consider other approaches.

Euclidean MST Algorithms. Shamos & Hoey [41] applied the Voronoi diagram to constructing the MST in the Euclidean plane. The Voronoi diagram can be constructed in $O(N \log N)$ time for N points and contains $O(N)$ edges. Since the MST is a subset of the edges in the dual of the Voronoi diagram, the MST can be found in $O(N \log N)$ time using one of the algorithms above. This bound worsens to $O(N^2 \log N)$ in three or more dimensions, fundamentally limiting this method to two dimensional cases. Preparata and Shamos [36] give a lower bound for the EMST problem of $\Omega(N \log N)$, which is the tightest known lower bound.

Bentley and Friedman [5] developed an EMST algorithm using *kd*-tree-based nearest neighbor searches to find the next edge to add in Prim's algorithm. While their method lacks a formally rigorous bound, they estimate that it requires $O(N \log N)$ time for most distributions of points. An alternate implementation of this approach is given in [33]. In 1982, Yao gave a bound of $O(N^{2-a(k)}(\log N)^{1-a(k)})$ where $a(k) = 2^{-(k+1)}$ for points in a k-dimensional metric space, along with a $O((N \log N)^{1.8})$ bound for points in three dimensions [51]. Agarwal *et al.* (1991) related the running time to the *bichromatic closest pair (BCP)* problem. Given a set of red and a set of blue points, the bichromatic closest pair is the red point r and blue point b such that $d(r, b)$ is minimized. They showed a bound of $O(F_d(N, N) \log^d(N))$, where $F_d(N, M)$ is the time to solve the BCP problem with N blue and M red points in d dimensions [1].

WSPD-based Methods. Callahan & Kosaraju's *Well-Separated Pair Decomposition (WSPD)* [10] forms the basis of the most recent EMST algorithms. The WSPD is defined as a set of pairs of nodes in a space-partitioning tree such that for each pair of points (p, q), we have $p \in P, q \in Q$ for exactly one pair of nodes (P, Q), and the the nodes in any pair are farther apart than the diameter of either node. It can be shown that the WSPD has $O(N)$ pairs of nodes, and that the MST is a subset of the edges formed between the closest pair of points in each pair of nodes. In [9], the authors use the WSPD to improve Agarwal and coworker's 1991 bound to $O(F_d(N, N) \log N)$. Their algorithm uses the WSPD-based nearest neighbor algorithm to compute neighbors of components for Boruvka's algorithm. The method identifies a list of pairs in the WSPD, for which bichromatic closest pair computations are performed to find edges of the MST. This algorithm is superficially similar to our method, but only locates neighbors for small components in each iteration. It also requires bookkeeping and connectedness queries which are not factored into the analysis, and no experimental results are shown.

Narasimhan *et al.* [31] implement a variant of this method, which they attribute to [9]. In this algorithm, GeoMST, they compute the BCP for each pair in the WSPD, then apply Kruskal's algorithm to the resulting edge set. They improve this method by postponing and avoiding some BCP computations and refer to the resulting algorithm as GeoMST2. This method can be successfully applied to point sets of any dimensionality; however, the constant in the $O(N)$ size of the WSPD grows exponentially in the dimension and is often very large in practice. The authors argue that the algorithm has an expected $O(N \log N)$ running time, but do not prove this rigorously. They also demonstrate favorable running times on several data sets.

These algorithms are the most sophisticated methods for the EMST problem in terms of both theoretical analysis and practical performance. The runtime bound in terms of the bichromatic closest pairs problem is the tightest available given optimistic runtimes for bichromatic closest pairs, but it is incomplete without bounding F_d. Bentley and Friedman's *kd*-tree-based method and the tree- and WSPD-based GeoMST2 are the most practically viable algorithms. We return to these methods in our experimental analysis.

3. PROPERTIES OF THE DATA

We now consider the problem of computing the EMST in relation to properties of the data. We present three objective parameters, independent of any algorithm and argue that these parameters capture difficulties in computing the MST.

Expansion Constant. The *expansion constant*, due to Karger and Ruhl [26], bounds the maximum increase in the density of points as a function of the distance from any point, and was used in adaptive analysis of nearest neighbors in previous work [7, 38].

Definition 1. Let S be a set of points in a metric space (X, d). Let $B_S(p, r) = \{q \in S : d(p, q) \leq r\}$. Then, the *expansion constant* c of S is defined as the smallest $c \geq 2$ such that for all $p \in X$ and all $r > 0$

$$|B_S(p, 2r)| \leq c|B_S(p, r)| \tag{1}$$

While the expansion constant depends only on the pairwise distances between points, the MST has a "higher-order" structure. In other words, the MST depends on distances between clusters of points in addition to distances between the individual points. Since the expansion constant does not capture this structure, we define two new parameters: the *cluster expansion constant* and *linkage expansion constant.*

Boruvka Clustering. We first require a definition of clusters. Independently of how they are computed, successive Boruvka steps define a hierarchical clustering of the data. We can therefore define and use the *Boruvka clustering* without reference to any method for computing it.

Definition 2. Given a point set S, the *Boruvka clustering* at level i, D_i, is the clustering obtained from applying a single Boruvka step to the clustering D_{i-1}. D_1 consists of each point as its own cluster.

Cluster Expansion Constant. Given the Boruvka clustering, we define the new expansion constants. Let $B_i^c(q, r)$ be the set of all components C_p with a point $p \in C_p$ such that $d(q, p) \leq r$. Using this *component-wise ball*, we define the *cluster expansion constant.*

Definition 3. The *cluster expansion constant* is the smallest real number c_p such that

$$|B_i^c(q, 2r)| \leq c_p|B_i^c(q, r)| \tag{2}$$

for all points $q \in S$, distances $r > 0$, and each level of the Boruvka clustering D_i.

Linkage Expansion Constant. Let C_1 and C_2 be two clusters in the Boruvka clustering at level i and let $S_1 \subseteq C_1$ and $S_2 \subseteq C_2$. Let $B_i^l(S_1, S_2, r)$ be the set of all pairs (p, q) such that $p \in S_1$, $q \in S_2$, and $d(p, q) \leq r$.

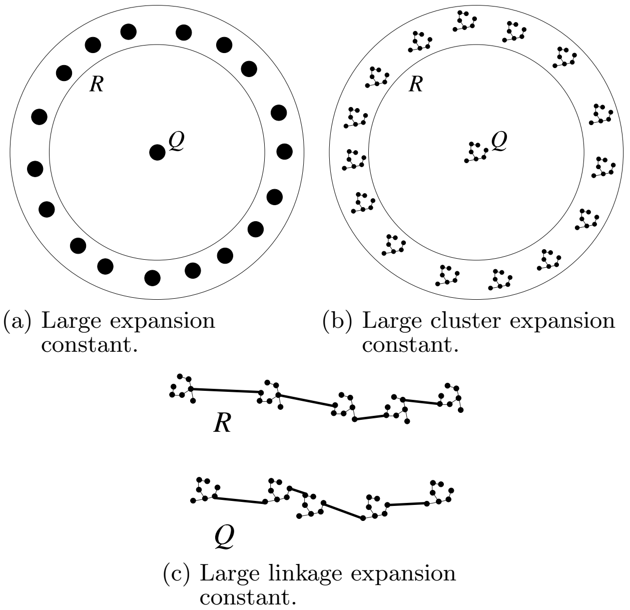
Definition 4. The *linkage expansion constant* is the smallest real number c_l such that

$$|B_i^l(S_1, S_2, 2r)| \leq c_l|B_i^l(S_1, S_2, r)| \tag{3}$$

for all levels of the Boruvka clustering D_i, clusters C_1 and C_2 at level i, subsets $S_1 \subseteq C_1, S_2 \subseteq C_2$, and distances $r > 0$.

Intuition for New Parameters. The time to compute the EMST for any algorithm is ultimately governed by the number of distance computations needed to distinguish edges that belong in the MST from those that do not. If the incorrect edges can be excluded from consideration with few computations, then it may be possible to compute the MST efficiently. On the other hand, if there are very many possible nearest neighbors of a given component, it may be impossible to avoid computing the distances to all such neighbors to find the nearest.

One possible case where the correct MST edge may be difficult to identify is given in Figure 1(a). Since Q is nearly



(a) Large expansion constant. (b) Large cluster expansion constant.

(c) Large linkage expansion constant.

Figure 1: Cases illustrating expansion constants.

equidistant from the points in R, there are many edges with nearly the same length as the edge in the MST. In this case, the expansion constant must be at least as large as the number of points in R.

Figure 1(b) shows an analogous possibility. Here, the MST must have an edge from the component Q to one of the components in the ring R. As before, the possible edges have nearly the same length, and it may be difficult to find the shortest edge. However, the expansion constant may still be small relative to the number of points. If Q contains $|R|/c$ points, then it is possible for the expansion constant to be bounded by c, even if R has $O(N)$ points. The cluster expansion constant is large in this case, so it captures this possible source of difficulty in computing the MST.

Figure 1(c) shows another case where finding the correct edges of the MST is difficult. There are many possible nearest neighbor pairs, and therefore many possible nearest neighbor edges, between Q and R. As in Figure 1(b), the expansion constant may still be small. Since this set has only two components, the cluster expansion constant is small as well. Here, the linkage expansion constant is large, capturing the difficulty of identifying the MST edge.

As we contended above, the EMST depends on distances between groups of points as well as the pairwise distances. We also argued that the expansion constant alone is insufficient to quantify the amount of computation needed to identify edges in the MST, thus motivating the need for further parameters to understand the behavior of any MST algorithm. We now turn to the definition and adaptive analysis of our new algorithm, DualTreeBoruvka, and show that the combination of all three types of expansion constant does contain enough information about the data to obtain a more precise analysis of the EMST problem.

4. DUAL-TREE BORUVKA ALGORITHM

The running time of Borůvka's algorithm depends on an efficient method to find the nearest neighbor pair of each component. Here, we describe a method to compute all nearest neighbor pairs simultaneously by amortizing some computations across different points. This allows us to implement Boruvka's algorithm more efficiently than previous methods.

Dual-Tree Algorithms. Dual-tree algorithms are a computational framework that has been applied to many problems in computational statistics, physics, and machine learning. These algorithms are the overall fastest known methods for many problems, including all nearest neighbors [22], kernel density estimation [23], mean shift [47], kernel discriminant analysis [39], and n-point correlation functions [22, 30].

For concreteness, consider the all nearest neighbors problem: we are given a query set Q and a reference set R of points in Euclidean space, each of $O(N)$ size. The goal is to find, for each point $q \in Q$, the point $r \in R$ such that $d(q, r)$ is minimized. A brute-force solution to this problem consists of two nested "for" loops which compute all pairwise distances and requires $O(N^2)$ time. We can improve on this algorithm with a space partitioning tree (e.g. a *kd*-tree) built on the set of references. For each query, we descend the tree, expanding reference nodes closer to the query first. We store the smallest distance $d(q, r)$ found so far at each stage of the algorithm, which provides an upper bound on the true nearest neighbor distance. Given this upper bound, we can *prune* nodes in the tree that are far enough away to not contain the true nearest neighbor, thus reducing the total number of distance computations. This *single-tree* method, described in [18], requires roughly $O(N \log N)$ time.

A *dual-tree* method further improves this by constructing another tree on Q. We consider both a query and reference node, and expand both. When the upper bounds allow, we can then prune a distant reference node for an entire node of queries with a single distance computation. This delivers tremendous speedups in practice and has been shown to require $O(N)$ time after tree construction with a cover tree [7]. We extend this idea to efficiently find the nearest neighbor pair for each component in a spanning forest.

New Algorithm. Our new algorithm, DualTreeBoruvka, uses a dual-tree method to find the nearest neighbor pair for each component. Algorithm 1 gives the description of the outer loop. The subroutine UpdateTree handles the propagation of any bounds up and down the tree and resets the upper bounds $d(C_q)$ to infinity. We also make use of a disjoint set data structure [45] to store the connected components at each stage of the algorithm. Our algorithm is independent of the particular space partitioning tree used. In this paper, we present experimental results on two instantiations of the algorithm. Algorithm 2 uses a *kd*-tree, and algorithm 3 uses the cover tree[7].

For the remainder of this work, we assume that we are given a set S of N points in $\mathbb{R}^d$. Furthermore, we make the standard assumption that all pairwise distances between points are unique. We make use of the following notation:

- $q \sim r$: q and r belong to the same component of the spanning forest.

- $R \bowtie Q$: all points in node R are in the same component as all points in node Q. Similarly, $r \bowtie q$ in a cover tree

Algorithm 1 Dual-Tree Borůvka (Tree root q)

```
    $E = \emptyset$
    while $|E| < N - 1$ do
3:    FindComponentNeighbors($q, q, e$)
      $E \leftarrow E \cup e$
      UpdateTree($q$)
6:  end while
```

denotes that all descendants of r are connected to all descendants of q.

- C_q : the component of the forest containing q
- $d(C_q)$: distance to current nearest neighbor of component C_q (initialized to ∞).
- $e(C_q)$: edge from C_q to its candidate nearest neighbor
- $d(Q, R)$: the minimum distance between the bounding boxes of nodes Q and R

DualTreeBoruvka on a *kd*-tree. The *kd*-tree [36] is a binary space-partitioning tree which maintains a bounding box for all the points in each node. The root consists of the entire set. Children are formed recursively by splitting the parent's bounding box along the midpoint of its largest dimension and partitioning the points on either side.

In the *kd*-tree version of DUALTREEBORUVKA, each node Q maintains an upper bound $d(Q) = \max_{q \in Q} d(C_q)$ and records whether all the points belong to the same component of the spanning forest. A node where all points belong to the same component is referred to as *fully connected*. With these records, we can prune when the distance between the query and reference is larger than $d(Q)$ or when all the points in Q and R belong to the same component.

THEOREM 4.1. *The* FINDCOMPONENTNEIGHBORS *routine in Algorithm 2 returns the correct nearest neighbor pairs.*

PROOF. The algorithm can only prune in two ways. If Q and R are fully connected, then no edges (q, r) with $q \in Q$ and $r \in R$ can be nearest neighbor pairs. The distance-based prune only occurs when for all $q \in Q$, $d(C_q) < d(Q, R)$. Therefore, all components with points in Q must have a candidate neighbor closer than any point in R, which again implies that no edge (q, r) can be a nearest neighbor pair. So, for each $q \in Q$, the correct Boruvka neighbor r of the component C_q cannot be pruned and must be found in the base case. □

DualTreeBoruvka on a Cover Tree. A cover tree is a data structure introduced by Beygelzimer *et al.* [7] for practically and theoretically efficient nearest-neighbor computations. In previous work, a proof of linear running time for the dual-tree all nearest neighbor algorithm used cover trees [38]. Here, we give a brief overview of the properties of a cover tree used in this paper.

A cover tree consists of a set of nested sets C_i, each at a scale i. A node in the cover tree consists of a single point and links to the node's children. The root is a single point at level ∞. As we descend the tree, the scale decreases, until $C_{-\infty}$ contains the entire set of points. For convenience, we index nodes in the cover tree with the node's point and use p_i to denote the node indexed by p at level i of the tree.

Algorithm 2 FindComponentNeighbors(*kd*-tree node Q, *kd*-tree node R, Edge set e)

```
    if $Q \bowtie R$ then
      return
3:  else if $d(Q, R) > d(Q)$ then
      return
    else if $Q$ and $R$ are leaves then
6:    for all $q \in Q, r \in R, r \not\sim q$ do
        if $d(q, r) < d(C_q)$ then
          $d(C_q) = d(q, r),\ e(C_q) = (q, r)$
9:      end if
      end for
      $d(Q) = \max_{q \in Q} d(C_q)$
12: else
      FindComponentNeighbors($Q.left, R.left, e$)
      FindComponentNeighbors($Q.right, R.left, e$)
15:   FindComponentNeighbors($Q.left, R.right, e$)
      FindComponentNeighbors($Q.right, R.right, e$)
      $d(Q) = \max\{d(Q.left), d(Q.right)\}$
18: end if
```

As in [7], we consider two representations of a cover tree. Conceptually, an algorithm descends from the root C_∞ to the set of all points at level $C_{-\infty}$, touching every level in between. Each point in level C_i has itself as a child in level C_{i-1} along with any other children. We refer to this idea of the tree as the *implicit* representation and make use of it in the algorithm description and proof. The *explicit representation* allows us to use the cover tree in practice. In the implicit representation, there are many levels where a node has only itself as a child. To create the explicit representation, we combine all such nodes. Therefore, a node is a single point, and contains pointers to all its children. The explicit representation has $O(N)$ nodes [7].

A cover tree has the following invariant properties:

1. Nesting: $C_i \subseteq C_{i-1}$
2. Covering: For every $p \in C_{i-1}$, there exists a $q \in C_i$ such that $d(p, q) \leq 2^i$ and exactly one such q is a parent of p. Note that this implies that if p' is any descendant of a point $p \in C_i$, then $d(p, p') \leq 2^{i+1}$.
3. Separation: For all $p, q \in C_i$, $d(p, q) > 2^i$.

The cover tree version of FINDCOMPONENTNEIGHBORS (Algorithm 3) follows the all nearest neighbor pseudocode given in [38]. The reference set R_i contains all points at level i that may have a nearest neighbor of a descendant of q_j as one of their descendants. Therefore, points are pruned from R_{i-1} in line 12 only when they are too distant to provide a neighbor. All descendants of q_j are within 2^{j+1} of q_j and all descendants of points in R are within 2^i of a point in R by the covering invariant. Therefore, any point outside the bound in line 12 cannot be a nearest neighbor for descendants of q_j.

THEOREM 4.2. *The* FINDCOMPONENTNEIGHBORS *routine in Algorithm 3 returns the correct nearest neighbor pair.*

PROOF. For a query q_j being considered at level j, the algorithm must guarantee that it finds the nearest neighbor pair both for the component C_q and for all components $C_{q'}$, where q' is a descendant of q_j. Pruning a fully-connected node can never delete the true nearest neighbor pair.

Algorithm 3 FindComponentNeighbors(Cover tree node q_j, Reference Set R_i, Edge set e)

```
    if $i = -\infty$ then
        // base case
 3:     for all $q$ that are descendants of $q_j$ and $r \in R_i$ with $r \not\sim q$ do
            if $d(q,r) < d(C_q)$ then
                $d(C_q) = d(q,r),\ e(C_q) = (q,r)$
 6:         end if
        end for
    else if $j < i$ then
 9:     // reference descend
        $R = \{r \in \text{Children}(r') : r' \in R_i \text{ and } r \not\bowtie q_j\}$
        $d = \min \left\{ d(C_q),\ \min_{\substack{r \in R \\ r \sim q_j}} \{d(q_j,r) + 2^i\},\ \min_{\substack{r \in R \\ r \not\sim q_j}} \{d(q_j,r)\} \right\}$
12:     $R_{i-1} = \{r \in R : d(q_j,r) \leq d + 2^i + 2^{j+2}\}$
        $d(C_q) = d$
        FindComponentNeighbors($q_j, R_{i-1}, e$)
15: else
        // query descend
        for all $p_{j-1} \in \text{Children}(q_j)$ do
18:         FindComponentNeighbors($p_{j-1}, R_i, e$)
        end for
    end if
```

We then consider distance-based pruning. As before, we use the nearest neighbor of the component C_q that the algorithm has seen up to this point in the execution. This candidate neighbor can be either a previously found nearest neighbor of another point in C_q (in which case $d = d(C_q)$), a point $r \in R$ ($d = d(q_j, r)$), or an inferred descendant of a connected point r ($d = d(q_j, r) + 2^i$). If $q_j \sim r$ but $q_j \not\bowtie r$, then r must have a descendant r' that is not connected to q_j. By the covering invariant, $d(q_j, r') \leq d(q_j, r) + 2^i$. Therefore, d is a valid upper bound for C_q. Since the distance between any point in R and any descendant is bounded by 2^i, any ancestor of the true nearest neighbor of q_j must be within $d + 2^i$, so the algorithm can never prune the ancestor of this neighbor.

We must also show that d is a valid bound for any descendant q' of q. If q and q' are in the same component, then this is clearly true, since bounds are shared across components. Otherwise, q is a candidate neighbor for q' and $d(q, q') \leq 2^{j+1}$. Therefore, we can be sure that $d(C_{q'}) \leq 2^{j+1}$. Let r' be the correct neighbor for q', and let r be the ancestor of r' in R. Then, $d(q_j, r) \leq d(q_j, q') + d(q', r') + d(r', r) \leq 2^{j+1} + 2^{j+1} + 2^i = 2^{j+2} + 2^i$. Therefore, the distance prune cannot remove the neighbor of any descendant of q. □

5. RUNTIME ANALYSIS

In this section, we prove our main theoretical result:

THEOREM 5.1. *For a set S of N points in a metric space with expansion constant c, cluster expansion constant c_p, and linkage expansion constant c_l, the* DualTreeBoruvka *algorithm using a cover tree requires*

$$O(\max\{c^6, c_p^2 c_l^2\} \cdot c^{10} N \log N \, \alpha(N))$$

time (where $\alpha(N)$ is defined below).

PROOF. Since each Borůvka step reduces the number of components in the spanning forest by a factor of at least two, the entire algorithm requires at most $\log N$ iterations. The construction of the cover tree takes $O(N \log N)$ time (proved in [7]) and only needs to be done once as a preprocessing step. Bookkeeping and cleanup in the tree in between calls to FindComponentNeighbors requires a single depth-first traversal, which takes $O(N)$ time.

Adding edges requires at most $O(N)$ Union operations on the disjoint-set structure, each of which requires $O(\alpha(N))$ time, with $\alpha(N)$ defined as follows. Let $A_k(j) = A_{k-1}^{(j+1)}(j)$ and let $A_0(j) = j + 1$. Then, define

$$\alpha(N) = \min\{k : A_k(1) \geq N\} \tag{4}$$

Therefore, in order to complete the proof, we only need to show that the FindComponentNeighbors subroutine on a cover tree requires $O(N\,\alpha(N))$ time. □

We first require two lemmas about cover trees, proven in [7]. Both lemmas assume the cover tree is built on a set of N points in a metric space with expansion constant c.

LEMMA 5.2. *(Width Bound) The number of children of any node in the cover tree is bounded by c^4.*

LEMMA 5.3. *(Depth Bound) The maximum depth in the tree of any point in the explicit representation is $O(c^2 \log N)$.*

We now apply our adaptive analysis to bounding the runtime of FindComponentNeighbors.

THEOREM 5.4. *Under the assumptions of Thm. 5.1, the* FindComponentNeighbors *algorithm on a cover tree (Algorithm 3) finds the nearest neighbor of each component in time bounded by:*

$$O\big(N + c^4 N + \max\{c^6, c_p^2 c_l^2\} \cdot N\,\alpha(N) + \max\{c^6, c_p^2 c_l^2\} \cdot c^{10} \log N\,\alpha(N)\big)$$

PROOF. We show that the amount of work done in each line of the algorithm during the entire execution is at most $O(\max_i |R_i| N\,\alpha(N))$. We complete the proof by showing $\max_i |R_i|$ depends only on c, c_p, and c_l.

Base Case. The base case (lines 1 through 7) is executed at most once for each explicit query node. Each base case requires $\max_i |R_i|$ Find operations, each of which requires $O(\alpha(N))$ time, so this step takes $O(\max_i |R_i| \cdot N\,\alpha(N))$ time.

Query Descends. Each query node in the explicit representation is expanded at most once (line 18), so this step requires $O(N)$ time overall.

Reference Descends. On the other hand, a reference node may be expanded more than once. When a query node is expanded, its reference cover set R_i needs to be duplicated for each child of the query. By the width bound, this creates at most c^4 duplications. Therefore, the total number of reference nodes considered in Line 12 is $O(c^4 N)$.

At each level, $|R| \leq c^4 \max_i |R_i|$. Since the maximum depth of a node is $O(c^2 \log N)$ (depth bound), the number of nodes considered in Line 14 is $O(c^6 \max_i |R_i| \log N)$. Considering possible duplication across queries, the total number of calls to Line 14 is at most $O(c^{10} \max_i |R_i| \log N)$. Computing d in each reference descend involves checking the connectedness of q_j and r, which requires $O(\alpha(N))$ time, for a total running time of $O(c^{10} \max_i |R_i| \log N\,\alpha(N))$.

Bounding $|R_i|$. For a given query q_j and reference cover set R_i, we compute the upper bound distance d. Then, $R_{i-1} = \{r \in R : d(q_j, r) \leq d + 2^i + 2^{j+2}\}$. Since $j < i$ in this part of the algorithm, and since the query and reference trees are identical, $j = i-1$. Therefore, $B(q_j, d+2^i+2^{j+2}) = B(q_j, d+2^{i+1}+2^i)$.

Consider two cases: first let $d \leq 2^{i+2}$. Then, as in [7], we bound number of balls of radius 2^{i-2} that can be packed into $B(q_j, d+2^{i+1})$ by:

$$\begin{aligned}
&|B(q_j, d+2^{i+1}+2^i+2^{i-2})| \\
&\leq \ |B(p, 2(d+2^{i+1}+2^i)+2^{i-2})| \\
&\leq \ |B(p, 2^{i+4})| \\
&\leq \ c^6|B(p, 2^{i-2})|
\end{aligned}$$

Each ball of radius 2^{i-2} can contain at most one point in C_{i-1} by the separation invariant. Therefore, the number of points in $B(q_j, d+2^{i+1}+2^i) \cap C_{i-1} \subseteq R_{i-1}$ is at most c^6.

Consider the other case where $d > 2^{i+2}$. Without loss of generality, assume that we have computed k previous iterations. First note that all points within $B(q_j, d-2^{i+1})$ must be connected to q_j. Otherwise, let q' be a point in $B(q_j, d-2^{i+1})$ that is not connected to q_j. Then, q' has a grandparent q'' at level C_{i-1} such that $d(q', q'') \leq 2^i$. Therefore,

$$d(q_j, q'') \leq d(q_j, q') + d(q', q'') < d - 2^{i+1} + 2^i = d - 2^i$$

Therefore, $d(q_j, q'') + 2^i < d$ and $q_j \not\bowtie q''$, which contradicts the definition of d in line 11.

The number of components that q_j may have to search is bounded by

$$\begin{aligned}
|B_p^c(q_j, d+2^{i+1}+2^i)| &\leq |B_k^c(q_j, 2d)| \\
&\leq c_p^2|B_k^c(q_j, d/2)| \\
&\leq c_p^2|B_k^c(q_j, d-2^{i+1})|
\end{aligned}$$

As noted above, all points within $d-2^{i+1}$ of q_j are connected to q_j, so the only component in $B_k^c(q_j, d-2^{i+1})$ is C_q.

We now bound the number of points within a component that q_j may have to consider. Let C_r be a component distinct from C_q. Let $L(q_j)$ denote the set of all leaves that are descendants of q_j. Let $d' = \min_{q \in L(q_j), r \in C_r} d(q, r)$. Then,

$$\begin{aligned}
&|B_k^l(C_q \cap L(q_j), C_r, d+2^{i+1}+2^i)| \\
&\leq \ |B_k^l(C_q \cap L(q_j), C_r, 4(d-2^{i+1}))| \\
&\leq \ c_l^2|B_k^l(C_q \cap L(q_j), C_r, (d-2^{i+1})| \\
&\leq \ c_l^2|B_k^l(C_q \cap L(q_j), C_r, d')|
\end{aligned}$$

By the above argument, there can be at most one pair in $B_k^l(C_q \cap L(q_j), C_r, d')$. Therefore, there are at most c_l^2 points in C_r contained in $B(q_j, d+2^{i+1}+2^i)$. In the worst case, each of these points is at level C_{i-1} of the tree and must be considered in R_{i-1}. There are at most c_p^2 components C_r that can contribute points, so the maximum number of points in R_{i-1} is $c_p^2 c_l^2$.

Combining these cases, we have $\max_i |R_i| \leq \max\{c^6, c_p^2 c_l^2\}$.

Therefore, the running time is:

$$\begin{aligned}
O\big(N + c^4 N + \max\{c^6, c_p^2 c_l^2\} \cdot N\,\alpha(N) \\
+ \max\{c^6, c_p^2 c_l^2\} \cdot c^{10} \log N\,\alpha(N)\big)
\end{aligned}$$

which completes the proof. □

Theorem 5.4 shows that each call to FindComponentNeighbors requires at most $O(N\,\alpha(N))$ time. By combining this with the observation above that DualTreeBoruvka requires at most $\log N$ calls to FindComponentNeighbors, we arrive at the runtime stated in Thm. 5.1, namely

$$O(N \log N\,\alpha(N)) \approx O(N \log N)$$

6. IMPLEMENTATION & EXPERIMENTS

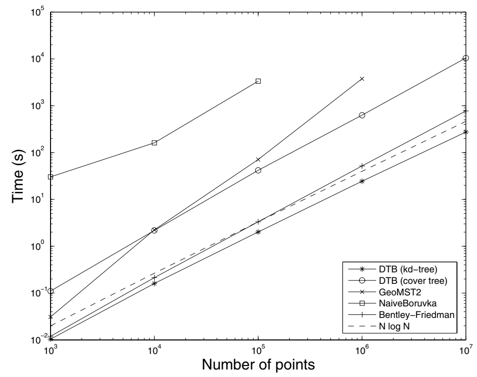
Algorithms Implemented. We present results for *kd*-tree-based and cover-tree-based DualTreeBoruvka. For comparsion, we implemented the other fast EMST methods mentioned in section 2. Specifically, we compare against the single-fragment EMST algorithm from Bentley and Friedman [5], which is an implementation of Prim's algorithm. The algorithm uses a single-tree algorithm on a *kd*-tree to find the next edge to add at each step. We also show results for the WSPD-based algorithm GeoMST2 [31], described above. Finally, we compare against a naïve implementation of Boruvka's algorithm in which nearest neighbor pairs are computed by iterating over all pairs of points.

Datasets. The experiments here are on four datasets: one synthetic and three sets of astronomy data. The synthetic data are drawn from a mixture of ten evenly weighted Gaussians placed uniformly at random in the unit cube in three dimensions. Figure 2(a) compares timing results on these data. Figure 2(b) shows runtimes on four dimensional samples of spectral data from the Sloan Digital Sky Survey. Table 1 has results for two other astronomy datasets: a 40,000 point, 3,840-dimensional set of color spectra from the SDSS, and a million point, 3 dimensional set of (x, y, z) coordinates from a galaxy-formation simulation.

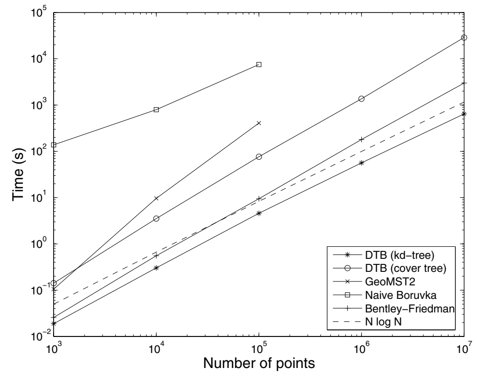
Implementation Details. We implemented all algorithms in the FASTlib C++ library [21]. The code was compiled with gcc version 4.1.2 with the -O2 flag. All experiments were performed on a 3.0 GHz Intel Xeon processor with 8GB of RAM running Linux.

Results. We attempted to run all the algorithms on all the sets of data. However, the naïve experiments were limited by time, since the brute-force algorithm scales quadratically. Thus results are missing for the larger sets. The GeoMST2 algorithm is limited by available memory. Although the WSPD contains $O(N)$ pairs of nodes, the constant factor can be very large. The constant in the $O(N)$ analysis scales exponentially with the dimension [10], so the storage bottleneck becomes tighter with higher-dimensional data. Missing timings for GeoMST2 indicate that the available memory was exceeded. In our experiments, the Bentley-Friedman algorithm is more efficient than either of these.

In both the synthetic data (Figure 2(a)) and the SDSS data (Figure 2(b)), DualTreeBoruvka on a *kd*-tree is the fastest method, by a factor of 2.8 and 4.6 over the Bentley-Freidman method, respectively. On both figures, we plot the slope of the predicted $N \log N$ performance, scaled to align with the timings for our method.



(a) Three-dimensional data generated from a mixture of gaussians.



(b) Four-dimensional SDSS spectra.

Figure 2: EMST computation times on log-log scale. All timings are in seconds.

N	dim	DTB *kd*	DTB cover	BF [5]
40,000	3840	45825.18	**15791.37**	45780.43
1,000,000	3	**17.39**	333.45	42.54

Table 1: Comparison of DualTreeBoruvka and Bentley-Friedman timings. Timings are in seconds.

Our results also consider dimensionality of the data. In the three- and four-dimensional data given in figure 2, the *kd*-tree based DualTreeBoruvka is fastest. Unlike most EMST algorithms, our method can also efficiently handle high-dimensional data, as shown in table 1. For the high-dimensional SDSS data, the two methods using *kd*-trees require roughly the same time. DualTreeBoruvka on a cover tree, however, is faster by a factor of 2.9.

7. CONCLUSION

We presented a new algorithm for the EMST problem, DualTreeBoruvka. We also present the first adaptive analysis of this long-standing problem. Combining these, we obtain the tightest runtime bound to-date for computing the EMST - $O(N \log N \, \alpha(N)) \approx O(N \log N)$ - which is separated from the best lower bound only by the overwhelmingly slowly-growing function $\alpha(N) \approx O(1)$. We leave for future work whether our bound is optimal or if a rigorous $O(N \log N)$ algorithm can be shown. Our analysis is also the first to avoid explicit (and exponential) dependence on the dimension of the input. We demonstrate the practical utility of our method for astronomical problems with experiments on data from the Sloan Digital Sky Survey and simulations of galaxy formulation. Comparison against algorithms in the literature shows our method to be the considerably faster on these sets. These experiments also support our theoretical analysis and demonstrate the applicability of our algorithm to both low- and high-dimensional data.

8. ACKNOWLEDGMENTS

The authors would like to thank Dongryeol Lee and Ryan Riegel for helpful comments on the proof.

9. REFERENCES

[1] P. K. Agarwal et al. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.

[2] G. J. Babu and E. D. Feigelson. *Astrostatistics.* Chapman & Hall/CRC, 1996.

[3] M. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *Symposium on Theory of Computing*, pages 671–680. ACM, 2008.

[4] J. D. Barrow, S. P. Bhavsar, and D. H. Sonoda. Minimal spanning trees, filaments and galaxy clustering. *MNRAS*, 216:17–35, Sept. 1985.

[5] J. Bentley and J. Friedman. Fast Algorithms for Constructing Minimal Spanning Trees in Coordinate Spaces. *IEEE T. Comput.*, 27:97–105, 1978.

[6] J. Bentley and A. Yao. An almost optimal algorithm for unbounded searching. *Inform. Process. Lett.*, 5(3):82–87, 1976.

[7] A. Beygelzimer, S. Kakade, and J. Langford. Cover Trees for Nearest Neighbor. *23rd International Conference on Machine learning*, pages 97–104, 2006.

[8] S. P. Bhavsar and R. J. Splinter. The superiority of the minimal spanning tree in percolation analyses of cosmological data sets. *MNRAS*, 282:1461–1466, 1996.

[9] P. Callahan and S. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Fourth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1993.

[10] P. B. Callahan and S. R. Kosaraju. A Decomposition of Multidimensional Point Sets with Applications to k-Nearest-Neighbors and n-body Potential Fields. *J. ACM*, 62(1):67–90, 1995.

[11] B. Chazelle. A faster deterministic algorithm for minimum spanning trees. In *Symposium on Foundations of Computer Science*, pages 22–31, 1997.

[12] B. Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000.

[13] M. Colless et al. The 2dF Galaxy Redshift Survey: spectra and redshifts. *MNRAS*, 328:1039–1063, 2001.

[14] E. Demaine, A. López-Ortiz, and J. Munro. Adaptive set intersections, unions, and differences. In *SODA*, pages 743–752, 2000.

[15] M. B. Eisen et al. Cluster analysis and display of genome-wide expression patterns. *PNAS*, 95(25):14863–14868, 1998.

[16] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Comput. Surv.*, 24(4):441–476, 1992.

[17] M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[18] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[19] H. Gabow et al. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[20] J. C. Gower and G. J. S. Ross. Minimum spanning trees and single linkage cluster analysis. *Applied Statistics*, 18(1):54–64, 1969.

[21] A. Gray et al. Mlpack, 2008. http://mloss.org/software/view/152/.

[22] A. Gray and A. W. Moore. N-body problems in statistical learning. In *Advances in Neural Information Processing Systems 13*, 2001.

[23] A. G. Gray and A. W. Moore. Rapid Evaluation of Multiple Density Models. In *The Ninth Conference on Artificial Intelligence and Statistics*, 2003.

[24] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[25] D. H. Jones et al. The 6dF Galaxy Survey: samples, observational techniques and the first data release. *MNRAS*, 355:747–763, 2004.

[26] D. R. Karger and M. Ruhl. Finding Nearest Neighbors in Growth-Restricted Metrics. *ACM Symposium on Theory of Computing*, pages 741–750, 2002.

[27] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull alorithm? *SIAM J. Comput.*, 15(1):287–299, 1986.

[28] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Am. Math. Soc.*, 7:48–50, 1956.

[29] C. Lacey and S. Cole. Merger rates in hierarchical models of galaxy formation. II- Comparison with N-body simulations. *MNRAS*, 271(3):676–692, 1994.

[30] A. Moore et al. Fast Algorithms and Efficient Statistics: n-point Correlation Functions. In *Proceedings of MPA/MPE/ESO Conference Mining the Sky*, 2000.

[31] G. Narasimhan, M. Zachariasen, and J. Zhu. Experiments with computing geometric minimum spanning trees. In *Proceedings of ALENEX'00*, pages 183–196, 2000.

[32] J. Nesetril. Otakar Boruvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history. *Discrete Math.*, 233:3–36, 2001.

[33] O. Nevalainen, J. Ernvall, and J. Katajainen. Finding minimal spanning trees in a Euclidean coordinate space. *BIT Numerical Mathematics*, 21(1):46–54, 1981.

[34] S. Pettie. Finding Minimum Spanning Trees in $O(m\alpha(m,n))$ Time. Technical report, University of Texas at Austin, Austin, TX, USA, 1999.

[35] S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

[36] F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

[37] R. C. Prim. Shortest connection networks and some generalizations. *Bell Sys. Tech. J.*, 36:1389–1401, 1957.

[38] P. Ram et al. Linear time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems 23*, 2009.

[39] R. Riegel, A. Gray, and G. Richards. Massive-Scale Kernel Discriminant Analysis: Mining for Quasars. In *SIAM International Conference on Data Mining*, 2008.

[40] S. Schmeja and R. S. Klessen. Evolving structures of star-forming clusters. *AAP*, 449:151–159, 2006.

[41] M. Shamos and D. Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science*, pages 151–162, 1975.

[42] S. Shectman et al. The Las Campanas Redshift Survey. *Astrophys. J.*, 470, 1996.

[43] V. Springel et al. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435(7042):629–636, 2005.

[44] S. Subramaniam and S. B. Pope. A mixing model for turbulent reactive flows based on euclidean minimum spanning trees. *Combust. Flame*, 115(4):487–514, 1998.

[45] R. Tarjan. *Data Structures and Network Algorithms.* Society for industrial and Applied Mathematics, 1988.

[46] P. J. Wan et al. Minimum-energy broadcast routing in static ad hoc wireless networks. In *IEEE Infocom*, 2001.

[47] P. Wang et al. Fast Mean Shift with Accurate and Stable Convergence. In *The Eleventh Workshop on Artificial Intelligence and Statistics*, 2007.

[48] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Inf. Process. Manage.*, 24(5):577–597, 1988.

[49] W.-K. Wong and A. Moore. Efficient algorithms for non-parametric clustering with clutter. In *Proceedings of the 34th Interface Symposium*, 2002.

[50] A. Yao. An $O(|E| \log\log |V|)$ algorithm for finding minimum spanning trees. *Inf. Process. Lett.*, 4:21–23, 1975.

[51] A. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

[52] D. York et al. The Sloan Digital Sky Survey: Technical Summary. *Astronomical Journal*, 120:1579–1587, 2000.

[53] C. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput.*, 20(1):68–86, 1971.