Computing Geometric Minimum Spanning Trees Using the Filter-Kruskal Method*

Samidh Chatterjee, Michael Connor, and Piyush Kumar

Department of Computer Science, Florida State University
Tallahassee, FL 32306-4530
{chatterj,miconnor,piyush}@cs.fsu.edu

Abstract. Let P be a set of points in $\mathbb{R}^d$. We propose GeoFilterKruskal, an algorithm that computes the minimum spanning tree of P using well separated pair decomposition in combination with a simple modification of Kruskal's algorithm. When P is sampled from uniform random distribution, we show that our algorithm runs in $\mathcal{O}(n \log^2 n)$ time with probability at least $1 - 1/n^c$ for a given $c > 1$. Although this is theoretically worse compared to known $\mathcal{O}(n)$ [31] or $\mathcal{O}(n \log n)$ [27, 11, 15] algorithms, experiments show that our algorithm works better in practice for most data distributions compared to the current state of the art [27]. Our algorithm is easy to parallelize and to our knowledge, is currently the best practical algorithm on multi-core machines for $d > 2$.

Key words: Computational Geometry, Experimental Algorithmics, Minimum spanning tree, Well separated pair decomposition, Morton ordering, multi-core.

1 Introduction

Computing the geometric minimum spanning tree (GMST) is a classic computational geometry problem which arises in many applications including clustering and pattern classification [34], surface reconstruction [25], cosmology [4, 6], TSP approximations [2] and computer graphics [23]. Given a set of n points P in $\mathbb{R}^d$, the GMST of P is defined as the minimum weight spanning tree of the complete undirected weighted graph of P, with edges weighted by the distance between their end points. In this paper, we present a practical deterministic algorithm to solve this problem efficiently, and in a manner that easily lends itself to parallelization.

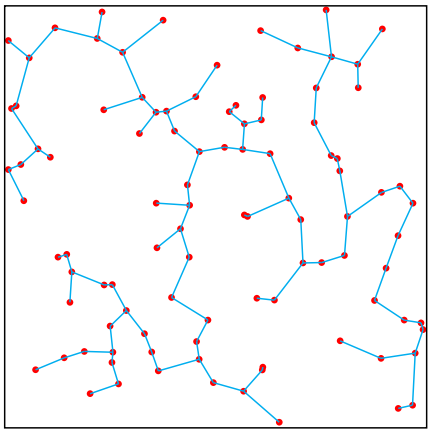

Fig. 1: A GMST of 100 points generated from the uniform distribution.

It is well established that the GMST is a subset of edges in the Delaunay triangulation of a point set [29] and similarly established that this method is inefficient for any dimension $d > 2$. It was shown by Agarwal et al. [1] that the GMST problem is related to solving bichromatic closest pairs for some subsets of the input set. The bichromatic closest pair problem is defined as follows: given two sets of points, one red and one green, find the red-green pair with minimum distance between them [22]. Callahan [8] used well separated pair decomposition and bichromatic closest pairs to solve the same problem in $\mathcal{O}(T_d(n, n) \log n)$, where $T_d(n, n)$ is the time required to solve the bichromatic closest pairs problem for n red and n green points. It is also known that the GMST problem is harder than bichromatic closest pair problem, and bichromatic closest pair is probably harder than computing the GMST [16].

Clarkson [11] gave an algorithm that is particularly efficient for points that are independently and uniformly distributed in a unit d-cube. His algorithm has an expected running time of $\mathcal{O}(n\alpha(cn, n))$, where c is a constant depending on the dimension and α is an extremely slow growing inverse Ackermann

* This research was partially supported by NSF through CAREER Grant CCF-0643593 and FSU COFRS. Advanced Micro Devices provided the 32-core workstation for experiments. Giri Narsimhan provided the source code to test many of the algorithms.

function. Bentley [5] also gave an expected nearly linear time algorithm for computing GMSTs in $\mathbb{R}^d$. Dwyer [15] proved that if a set of points is generated uniformly at random from the unit ball in $\mathbb{R}^d$, its Delaunay triangulation has linear expected complexity and can be computed in expected linear time. Since GMSTs are subsets of Euclidean Delaunay triangulations, one can combine this result with linear time MST algorithms [20] to get an expected $\mathcal{O}(n)$ time algorithm for GMSTs of uniformly distributed points in a unit ball. Rajasekaran [31] proposed a simple expected linear time algorithm to compute GMSTs for uniform distributions in $\mathbb{R}^d$. All these approaches use bucketing techniques to execute a spiral search procedure for finding a supergraph of the GMST with $\mathcal{O}(n)$ edges. Unfortunately, finding k-nearest neighbors for every point, even when $\mathsf{k} = \mathcal{O}(1)$ is quite an expensive operation in practice for non-uniform distributions [32], compared to practical GMST algorithms [27, 12].

Narasimhan et al. [27] gave a practical algorithm that solves the GMST problem. They prove that for uniformly distributed points, in fixed dimensions, an expected $\mathcal{O}(n \log n)$ steps suffice to compute the GMST using well separated pair decomposition. Their algorithm, GeoMST2, mimics Kruskal's algorithm [21] on well separated pairs and eliminates the need to compute bichromatic closest pairs for many well separated pairs. To our knowledge, this implementation is the state of the art, for practically computing GMSTs in low dimensions ($d > 2$). Although, improvements to GeoMST2 [27] have been announced [24], exhaustive experimental results are lacking in this short announcement. Another problem with this claim is that even for uniform distribution of points, there are no theoretical guarantees that the algorithm is indeed any better than $\mathcal{O}(n^2)$.

Brennan [7] presented a modification to Kruskal's classic minimum spanning tree (MST) algorithm [21] that operated similar in a manner to quicksort; splitting an edge set into "light" and "heavy" subsets. Recently, Osipov et al. [28] further expanded this idea by adding a multi-core friendly filtering step designed to eliminate edges that were obviously not in the MST (Filter-Kruskal). Currently, this algorithm seems to be the most practical algorithm for computing MSTs on multi-core machines.

The algorithm presented in this paper uses well separated pair decomposition in combination with a modified version of Filter-Kruskal for computing GMSTs. We use a compressed quad-tree to build a well separated pair decomposition, followed by a sorting based algorithm similar to Filter-Kruskal. By using a sort based approach, we eliminate the need to maintain a priority queue [27]. This opens up the possibility of filtering out well separated pairs with a large number of points, before it becomes necessary to calculate their bichromatic closest pair. Additionally, we can compute the bichromatic closest pair of well separated pairs of similar size in batches. This allows for parallel execution of large portions of the algorithm.

We show that our algorithm has, with high probability, a running time of $\mathcal{O}(n \log^2 n)$ on uniformly random data in $\mathbb{R}^d$. Although this is a theoretically worse bound than is offered by the GeoMST2 algorithm, extensive experimentation indicates that our algorithm outperforms GeoMST2 on many distributions. Our theoretically worse bound comes from the fact that we have used a stricter definition of *high probability* than being used in GeoMST2 [27]. We also present experimental results for multi-core execution of the algorithm.

This paper is organized as follows: In the remainder of this section, we define our notation. In Section 2 we define and elaborate on tools that we use in our algorithm. Section 3 presents our algorithm, and a theoretical analysis of the running time. Section 4 describes our experimental setup, and Section 5 compares GeoMST2 with our algorithm experimentally. Finally, Section 6 concludes the paper and presents future research directions.

Notation: Points are denoted by lower-case Roman letters. $\mathsf{Dist}(p, q)$ denotes the distance between the points p and q in L_2 metric. Upper-case Roman letters are reserved for sets. Scalars except for c, d, m and n are represented by lower-case Greek letters. We reserve i, j, k for indexing purposes. $\mathsf{Vol}()$ denotes the volume of an object. For a given quadtree, $\mathsf{Box}\{p, q\}$ denotes the smallest quadtree box containing points p and q; Fraktur letters ($\mathfrak{a}$) denote a quadtree node. $\mathsf{MinDist}(\mathfrak{a}, \mathfrak{b})$ denotes the minimum distance between the quadtree boxes of two nodes. $\mathsf{Left}(\mathfrak{a})$ and $\mathsf{Right}(\mathfrak{a})$ denotes the left and right child of a node. $|.|$ denotes the cardinality of a set or the number of points in a quadtree node. $\alpha(n)$ is used to denote inverse of the Ackermann function [13]. The Cartesian product of two sets X and Y, is denoted $X \times Y = \{(x, y) | x \in X \text{and } y \in Y\}$.

2 Preliminaries

Before we present our algorithm in detail, we need to describe a few tools which our algorithm uses extensively. These include the well separated pair decomposition, Morton ordering and quadtree construction that uses this order, a practical algorithm for bichromatic closest pair computation and the union-find data structure. We describe these tools below.

Morton ordering and Quadtrees: Morton order is a space filling curve with good locality preserving behavior [18]. Due to this behavior, it is used in data structures for mapping multidimensional data into one dimension. The Morton order value of a point can be obtained by interleaving the binary representations of its coordinate values. If we recursively divide a d-dimensional hypercube into 2^d hypercubes until there is only one point in each, and then order those hypercubes using their Morton order value, the Morton order curve is obtained. In 2 dimensions we refer to this decomposition as a quadtree decomposition, since each square can be divided into four squares. We will explain the algorithm using quadtrees, but this can be easily extended to higher dimensions. Chan [9] showed that using a few binary operations for integer coordinates, the relative Morton ordering can be calculated by, determining the dimension in which corresponding coordinates have the first differing bit in the highest bit position. This technique can be extended to work for floating point coordinates, with only a constant amount of extra work [12]. In the next paragraph, we briefly state the quadtree construction that we use in our algorithm [10, 17].

Let $p_1, p_2, ..., p_n$ be a given set of points in Morton order. Let the index j be such that $\mathsf{Vol}(\mathsf{Box}\{p_{j-1}, p_j\})$ is the largest. Then the compressed quadtree Q for the point set consists of a root holding the $\mathsf{Box}\{p_{j-1}, p_j\}$ and two subtrees recursively built for $p_1, ..., p_{j-1}$ and $p_j, .., p_n$. Note that this tree is simply the *Cartesian tree* [17] of $\mathsf{Vol}(\mathsf{Box}\{p_1, p_2\})$, $\mathsf{Vol}(\mathsf{Box}\{p_2, p_3\})$, ..., $\mathsf{Vol}(\mathsf{Box}\{p_{n-1}, p_n\})$. A Cartesian tree is a binary tree constructed from a sequence of values which maintains three properties. First, there is one node in the tree for each value in the sequence. Second, a symmetric, in-order traversal of the tree returns the original sequence. Finally, the tree maintains a heap property, in which a parent node has a larger value than its child nodes. The Cartesian tree can be computed using a standard incremental algorithm in linear time [17], given the ordering $p_1, p_2, ..., p_n$ [10]. Hence, for both integer as well as floating point coordinates, the compressed quadtree Q can be computed in $\mathcal{O}(n \log n)$ [12]. We use a compressed quadtree, as opposed to the more common fair split tree [8], because of the ease in which construction can be parallelized.

Well Separated Pair Decomposition [8]: Assume that we are given a compressed quadtree Q, built on a set of points P in $\mathbb{R}^d$. Two nodes $\mathfrak{a}$ and $\mathfrak{b} \in Q$ are said to be an *ϵ-well separated pair* (ϵ-WSP) if the diameter of $\mathfrak{a}$ and the diameter of $\mathfrak{b}$ are both at most ϵ times the minimum distance between $\mathfrak{a}$ and $\mathfrak{b}$. An *ϵ-well separated pair decomposition (ϵ-WSPD)*, of size m for a quadtree Q, is a collection of well separated pairs of nodes $\{(\mathfrak{a}_1, \mathfrak{b}_1), ..., (\mathfrak{a}_m, \mathfrak{b}_m)\}$, such that every pair of points $(p, q) \in P \times P$ $(p \neq q)$ lies in exactly one pair $(\mathfrak{a}_i, \mathfrak{b}_i)$. In the following, we use WSP and WSPD to mean well separated pairs and decompositions with an ϵ value of 1.

On the compressed quadtree Q, one can execute Callahan's [8] WSPD algorithm [10]. It takes $\mathcal{O}(n)$ time to compute the WSPD, given the compressed quadtree, thus resulting in an overall running time of $\mathcal{O}(n \log n)$ for WSPD construction.

Bichromatic Closest Pair Algorithm: Given a set of points in $\mathbb{R}^d$, the *Bichromatic Closest Pair* (BCCP) problem asks to find the closest red-green pair [22]. The computation of the bichromatic closest pairs is necessary due to the following Lemma from [8]:

Lemma 1. *The set of all the BCCP edges of the WSPD contain the edges of the GMST.*

Clearly, $\mathsf{MinDist}(\mathfrak{a}, \mathfrak{b})$ is the lower bound on the bichromatic closest pair distance of A and B (where A and B are the point sets contained in $\mathfrak{a}$ and $\mathfrak{b}$ respectively). Next, we present the BCCP algorithm:

Algorithm 1 for computing the bichromatic closest pair was proposed by Narsimhan and Zachariasen [27]. The input to the algorithm are two nodes $\mathfrak{a}, \mathfrak{b} \in Q$, that contain sets $A, B \subseteq P$ and a positive real number δ, which is used by the recursive calls in the algorithm to keep track of the last closest pair distance found. The output of the algorithm is the closest pair of points (p, q), such that $p \in A$ and $q \in B$ with minimum distance $\delta = \mathsf{Dist}(p, q)$. Initially, the BCCP is equal to η, where η represents the WSP containing only the last point in Morton order of A and the first point in the Morton order of B,

Algorithm 1 BCCP Algorithm [27]: Compute $\{p', q', \delta'\}$ =Bccp($\mathfrak{a}, \mathfrak{b}[, \{p, q, \delta\} = \eta]$)

Require: $\mathfrak{a}, \mathfrak{b} \in Q, A, B \subseteq P, \delta \in \mathbb{R}^+$
Require: If $\{p, q, \delta\}$ is not specified, $\{p, q, \delta\} = \eta$, an upper bound on BCCP($\mathfrak{a}, \mathfrak{b}$).
1: **procedure** Bccp($\mathfrak{a}$,$\mathfrak{b}$[, $\{p, q, \delta\} = \eta$])
2: **if** ($|A| = 1$ and $|B| = 1$) **then**
3: Let $p' \in A$, $q' \in B$
4: **if** Dist(p', q') $< \delta$ **then return** $\{p', q',$ Dist(p', q')$\}$
5: **else**
6: **if** Vol($\mathfrak{a}$) $<$ Vol($\mathfrak{b}$) **then** Swap($\mathfrak{a}$,$\mathfrak{b}$)
7: γ =MinDist(Left($\mathfrak{a}$), $\mathfrak{b}$)
8: ζ =MinDist(Right($\mathfrak{a}$), $\mathfrak{b}$)
9: **if** $\gamma < \delta$ **then** $\{p, q, \delta\}$ =Bccp(Left($\mathfrak{a}$), $\mathfrak{b}$, $\{p, q, \delta\}$)
10: **if** $\zeta < \delta$ **then** $\{p, q, \delta\}$ =Bccp(Right($\mathfrak{a}$), $\mathfrak{b}$, $\{p, q, \delta\}$)
11: **return** $\{p, q, \delta\}$
12: **end procedure**

assuming without loss of generality, all points in A are smaller than all points in B, in Morton order. If both A and B are singleton sets, then the distance between the two points is trivially the BCCP distance. Otherwise, we compare Vol($\mathfrak{a}$) and Vol($\mathfrak{b}$) and compute the distance between the lighter node and each of the children of the heavier node. If either of these distances is less than the closest pair distance computed so far, we recurse on the corresponding pair. If both of the distances are less, we recurse on both of the pairs. With a slight abuse of notation, we will denote the length of the actual bichromatic closest pair edge, between $\mathfrak{a}$ and $\mathfrak{b}$, by BCCP($\mathfrak{a}$,$\mathfrak{b}$).

UnionFind Data Structure: The UnionFind data structure maintains a set of partitions indicating the connectivity of points based on the edges already inserted into the GMST. Given a UnionFind data structure $\mathcal{G}$, and $u, v \in P \subseteq \mathbb{R}^d$, $\mathcal{G}$ supports the following two operations: $\mathcal{G}$.Union(u, v) updates the structure to indicate the partitions containing u and v are now joined; $\mathcal{G}$.Find(u) returns the node number of the *representative* of the partition containing u. Our implementation also does *union by rank* and *path compression*. A sequence of m $\mathcal{G}$.Union() and $\mathcal{G}$.Find() operations on n elements takes $\mathcal{O}(m\alpha(n))$ time in the worst case. For all practical purposes, $\alpha(n) \leq 4$ (see [13]).

3 GeoFilterKruskal algorithm

Our GeoFilterKruskal algorithm computes a GMST for $P \subseteq \mathbb{R}^d$. Kruskal's [21] algorithm shows that given a set of edges, the MST can be constructed by considering edges in increasing order of weight. Using Lemma 1, the GMST can be computed by running Kruskal's algorithm on the BCCP edges of the WSPD of P. When Kruskal's algorithm adds a BCCP edge (u, v) to the GMST, where $u, v \in P$, it uses the UnionFind data structure to check whether u and v belong to the same connected component. If they do, that edge is discarded. Otherwise, it is added to the GMST. Hence, before testing for an edge (u, v) for inclusion into the GMST, it should always attempt to add all BCCP edges (u', v') such that Dist(u', v') $<$ Dist(u, v). GeoMST2 [27] avoids the computation of BCCP for many well separated pairs that already belonged to the same connected component. Our algorithm uses this crucial observation as well. Algorithm 2 describes the GeoFilterKruskal algorithm in more detail.

The input to the algorithm, is a WSPD of the point set $P \subseteq \mathbb{R}^d$. All procedural variables are assumed to be passed by reference. The set of WSPs S is partitioned into set E_l that contains WSPs with cardinality less than β (initially 2), and set $E_u = S \setminus E_l$. We then compute the BCCP of all elements of set E_l, and compute ρ equal to the minimum MinDist($\mathfrak{a}, \mathfrak{b}$) for all $(\mathfrak{a}, \mathfrak{b}) \in E_u$. E_l is further partitioned into E_{l1}, containing all elements with a BCCP distance less than ρ, and $E_{l2} = E_l \setminus E_{l1}$. E_{l1} is passed to the Kruskal procedure, and $E_{l2} \cup E_u$ is passed to the Filter procedure. The Kruskal procedure adds the edges to the GMST or discards them if they are connected. Filter removes all connected WSPs. The GeoFilterKruskal procedure is recursively called, increasing the threshold value (β) by one each time, on the WSPs that survive the Filter procedure, until we have constructed the minimum spanning tree.

Algorithm 2 GeoFilterKruskal Algorithm

Require: $S = \{(\mathfrak{a}_1, \mathfrak{b}_1), ..., (\mathfrak{a}_m, \mathfrak{b}_m)\}$ is a WSPD, constructed from $P \subseteq \mathbb{R}^d$; $T = \{\}$.
Ensure: BCCP Threshold $\beta \geq 2$.

procedure GeoFilterKruskal(Sequence **of** WSPs : S, Sequence **of** Edges : T, UnionFind : $\mathcal{G}$, Integer : β)
  $E_l = E_u = E_{l1} = E_{l2} = \emptyset$
  for all $(\mathfrak{a}_i, \mathfrak{b}_i) \in S$ **do**
    if $(|\mathfrak{a}_i| + |\mathfrak{b}_i|) \leq \beta$ **then** $E_l = E_l \cup \{(\mathfrak{a}_i, \mathfrak{b}_i)\}$ **else** $E_u = E_u \cup \{(\mathfrak{a}_i, \mathfrak{b}_i)\}$
  end for
  $\rho = \min\{\text{MinDist}\{\mathfrak{a}_i, \mathfrak{b}_i\} : (\mathfrak{a}_i, \mathfrak{b}_i) \in E_u, i = 1, 2, ..., m\}$
  for all $(\mathfrak{a}_i, \mathfrak{b}_i) \in E_l$ **do**
    $\{u, v, \delta\} = \text{Bccp}(\mathfrak{a}_i, \mathfrak{b}_i)$
    if $(\delta \leq \rho)$ **then** $E_{l1} = E_{l1} \cup \{(u, v)\}$ **else** $E_{l2} = E_{l2} \cup \{(u, v)\}$
  end for
  Kruskal$(E_{l1}, T, \mathcal{G})$
  $E_{new} = E_{l2} \cup E_u$
  Filter$(E_{new}, \mathcal{G})$
  if ($|T| < (n-1)$) **then** GeoFilterKruskal$(E_{new}, T, \mathcal{G}, \beta + 1)$
end procedure
procedure Kruskal(Sequence **of** WSPs : E, Sequence **of** Edges : T, UnionFind : $\mathcal{G}$)
  Sort(E): by increasing BCCP distance
  for all $(u, v) \in E$ **do**
    if $\mathcal{G}$.Find$(u) \neq \mathcal{G}$.Find(v) **then** $T = T \cup \{(u, v)\}$; $\mathcal{G}$.Union(u, v);
  end for
end procedure
procedure Filter(Sequence **of** WSPs : E, UnionFind : $\mathcal{G}$)
  for all $(\mathfrak{a}_i, \mathfrak{b}_i) \in E$ **do**
    if $(\mathcal{G}$.Find$(u) = \mathcal{G}$.Find$(v) : u \in \mathfrak{a}_i, v \in \mathfrak{b}_i)$ **then** $E = E \setminus \{(\mathfrak{a}_i, \mathfrak{b}_i)\}$
  end for
end procedure

3.1 Correctness

Given previous work by Kruskal [13] as well as Callahan [8], it is sufficient to show two facts to ensure the correctness of our algorithm. First, we are considering WSPs to be added to the GMST in the order of their BCCP distance. This is obviously true considering WSPs are only passed to the Kruskal procedure if their BCCP distance is less than the lower bound on the BCCP distance of the remaining WSPs. Second, we must show that the Filter procedure does not remove WSPs that should be added to the GMST. Once again, it is clear that any edge removed by the Filter procedure would have been removed by the Kruskal procedure eventually, as they both use the UnionFind structure to determine connectivity.

3.2 Analysis of the running time

The real bottleneck of this algorithm, as well as the one proposed by Narasimhan [27], is the computation of the BCCP. If $|A| = |B| = \mathcal{O}(n)$, the BCCP algorithm stated in section 3 has a worst case time complexity of $\mathcal{O}(n^2)$. In this section we improve this bound for uniform distributions.

Let $Pr(\mathcal{E})$ denote the probability of occurrence of an event $\mathcal{E}$. In what follows, we define an event $\mathcal{E}$ to occur with high probability (WHP) if given $c > 1$, $Pr(\mathcal{E}) > 1 - 1/n^c$. Our analysis below shows that WHP, for uniform distribution, each BCCP computation takes $\mathcal{O}(\log^2 n)$ time. Thus our whole algorithm takes $\mathcal{O}(n \log^2 n)$ time WHP.

Let P be a set of n points chosen uniformly from a unit hypercube H in $\mathbb{R}^d$. We draw a grid of n *cells* in this hypercube, with the grid length being n^{-d}, and the cell volume equal to $1/n$.

Lemma 2. *A quadtree box containing $c \log n$ cells contains,* WHP, *$\mathcal{O}(\log n)$ points.*

Proof. Let $\mathfrak{a}$ be a quadtree box containing $c \log n$ cells. The volume of $\mathfrak{a}$ is $c \log n/n$. Let $\mathsf{X_i}$ be a random variable which takes the value 1 if the *ith* point of P is in $\mathfrak{a}$; and 0 otherwise, where $i = 1, 2, ..., n$. Given $Pr(\mathsf{X_i} = 1) = c \log n/n$ and $\mathsf{X_1}, ..., \mathsf{X_n}$ are *independent and identically* distributed, let $\mathsf{X} = \sum_{i=0}^{n} \mathsf{X_i}$, which

is binomially distributed [26] with parameters n and $c \log n/n$. Thus the expected number of points in a box of volume $c \log n/n$ is $c \log n$ [26]. Applying Chernoff's inequality [26], where $c' > e$ is a given constant, we see that:

$$Pr(\mathsf{X} > c' \log n) < e^{\log n^{c'-1}}/c'^{c' \log n} = n^{c'-1}/n^{c'/\log_{c'} e}.$$

Now $c'/\log_{c'} e - (c'-1) > 1$. Therefore, given $c' > e$, $Pr(\mathsf{X} > c' \log n) < 1/n^{\kappa(c')}$ where $\kappa(c') > 1$ is a function of c'. Thus WHP, a quadtree containing $c \log n/n$ cells will contain $\mathcal{O}(\log n)$ points. □

Corollary 1. WHP*, a quadtree box with fewer than* $c \log n$ *cells contains* $\mathcal{O}(\log n)$ *points.*

Lemma 3. *Each BCCP computation takes* $\mathcal{O}(\log^2 n)$ *time* WHP*.*

Proof. Let $(\mathfrak{a}, \mathfrak{b})$ be a WSP. Without loss of generality, assume the quadtree box $\mathfrak{a}$ has larger volume. If $\mathfrak{a}$ has χ cells, then the volume of $\mathfrak{a}$ is χ/n. Since $\mathfrak{a}$ and $\mathfrak{b}$ are well separated, there exists a quadtree box $\mathfrak{c}$ with at least the same volume as $\mathfrak{a}$, that lies between $\mathfrak{a}$ and $\mathfrak{b}$. Let $\mathcal{E}$ denote the event that this region is empty. Then $Pr(\mathcal{E}) = (1 - \chi/n)^n \leq 1/e^{\chi}$. Replacing χ by $c \log n$ we have $Pr(\mathcal{E}) \leq 1/n^c$. Hence if $\mathfrak{a}$ is a region containing $\Omega(\log n)$ cells, $\mathfrak{c}$ contains one or more points with probability at least $1 - 1/n^c$.

Note that BCCP$(\mathfrak{a}, \mathfrak{b})$ must be greater than BCCP$(\mathfrak{a}, \mathfrak{c})$ and BCCP$(\mathfrak{b}, \mathfrak{c})$. Since our algorithm adds the BCCP edges by order of their increasing BCCP distance, the BCCP edge between $\mathfrak{c}$ and $\mathfrak{a}$, as well as the one between $\mathfrak{b}$ and $\mathfrak{c}$, will be examined before the BCCP edge between $\mathfrak{a}$ and $\mathfrak{b}$. This causes $\mathfrak{a}$ and $\mathfrak{b}$ to belong to the same connected component, and thus our filtering step will get rid of the well separated pair $(\mathfrak{a}, \mathfrak{b})$ before we need to compute its BCCP edge. From Lemma 2 we see that a quadtree box containing $c \log n$ cells contains $\mathcal{O}(\log n)$ points WHP. This fact along with Corollary 1 shows that WHP we do not have to compute the BCCP for a quadtree box containing more than $\mathcal{O}(\log n)$ points, limiting the run time for a BCCP calculation to $\mathcal{O}(\log^2 n)$ WHP. □

Lemma 4. WHP*, the total running time of the* UnionFind *operation is* $\mathcal{O}(\alpha(n) n \log n)$*.*

Proof. From Lemma 3, WHP, all the edges in the minimum spanning tree are BCCP edges of WSPs which are $\mathcal{O}(\log n)$ in size. Since we increment β in steps of one, WHP, we make $\mathcal{O}(\log n)$ calls to GeoFilterKruskal. In each of such calls, the Filter function is called once, which in turn calls the Find(u) function of the UnionFind data structure $\mathcal{O}(n)$ times. Hence, there are in total $\mathcal{O}(n \log n)$ Find(u) operations done WHP. Thus the overall running time of the Union() and Find() operations is $\mathcal{O}(\alpha(n) n \log n)$ WHP. □

Theorem 1. *Total running time of the algorithm is* $\mathcal{O}(n \log^2 n)$*.*

Proof. We partition the list of well separated pairs twice in the GeoFilterKruskal method. The first time we do it based on the need to compute the BCCP of the well separated pair. We have the sets E_l and E_u in the process. This takes $\mathcal{O}(n)$ time except for the BCCP computation. In $\mathcal{O}(n)$ time we can find the pivot element of E_u for the next partition. This partitioning also takes linear time. As explained in Lemma 4, we perform the recursive call on GeoFilterKruskal $\mathcal{O}(\log n)$ times WHP. Thus the total time spent in partitioning is $\mathcal{O}(n \log n)$ WHP. Since the total number of WSPs is $\mathcal{O}(n)$, the time spent in computing all such edges is $\mathcal{O}(n \log^2 n)$. Hence, the total running time of the algorithm is $\mathcal{O}(n \log^2 n)$ WHP. □

Remark 1: Under the assumption that the points in P have integer coordinates, we can modify algorithm 2 such that its running time is $\mathcal{O}(n)$. WSPD can be computed in $\mathcal{O}(n)$ time if the coordinates of the points are integers [10]. One can first compute the minimum spanning forest of the k-nearest neighbor graph of the point set, for some given constant k, using a randomized linear time algorithm [19]. In this forest, let T' to be the tree with the largest number of points. Rajasekaran [31] showed that there are only n^β points left to be added to T' to get the GMST, where $\beta \in (0, 1)$. In our algorithm, if the set T is initialized to T', then our analysis shows that WHP, only $\mathcal{O}(n^\beta \log^2 n) = \mathcal{O}(n)$ additional computations will be necessary to compute the GMST.

Remark 2: Computation of GMST from k-nearest neighbor graph can be parallelized efficiently in a CRCW PRAM. From our analysis we can infer that in case of a uniformly randomly distributed set of points P, if we extract the $\mathcal{O}(\log n)$ nearest neighbors for each point in the set, then these edges will contain the GMST of P WHP. Callahan [8] showed that it is possible to compute the k-nearest neighbors of all points in P in $\mathcal{O}(\log n)$ time using $\mathcal{O}(\mathsf{k}n)$ processors. Hence, using $\mathcal{O}(n \log n)$ processors, the minimum spanning tree of P can then be computed in $\mathcal{O}(\log n)$ time [33].

Remark 3: We did not pursue implementing the algorithms described in remarks one and two because they are inherently Monte Carlo algorithms [26]. While they can achieve a solution that is correct with high probability, they do not guarantee a correct solution. One can design an exact GMST algorithm using k-nearest neighbor graphs; however preliminary experiments using the best practical parallel nearest neighbor codes [12, 3] showed construction times that were slower than the GeoFilterKruskal algorithm. For example, on a two dimensional data set containing one million uniformly distributed points, GeoFilterKruskal computed the GMST in 5.5 seconds, while 8-nearest neighbor graph construction took 5.8 seconds [12].

3.3 Parallelization

Parallelization of the WSPD algorithm: In our sequential version of the algorithm, each node of the compressed quadtree computes whether its children are well separated or not. If the children are not well separated, we divide the larger child node, and recurse. We can parallelize this loop using OpenMP [14] with a dynamic load balancing scheme. Since for each node there are $\mathcal{O}(1)$ candidates to be well separated [10], and we are using dynamic load balancing, the total time taken in CRCW PRAM, given p processors, to execute this algorithm is $\mathcal{O}(\lceil (n \log n)/\mathsf{p} \rceil + \log n)$ including the preprocessing time for the Morton order sort.

Parallelization of the GeoFilterKruskal algorithm: Although the whole of algorithm 2 is not parallelizable, we can parallelize most portions of the algorithm. The parallel partition algorithm [30] is used in order to divide the set S into subsets E_l and E_u (See Algorithm 2). ρ can be computed using parallel prefix computation. In our actual implementation, we found it to be more efficient to wrap it inside the parallel partition in the previous step, using the atomic compare-and-swap instruction. The further subdivision of E_l, as well as the FILTER procedure, are just instances of parallel partition. The sorting step used in the KRUSKAL procedure as well as the Morton order sort used in the construction of the compressed quadtree can also be parallelized [30]. We use OpenMP to do this in our implementation. Our efforts to parallelize the linear time quadtree construction showed that one can not use more number of processors on multi-core machines to speed up this construction, because it is memory bound.

4 Experimental Setup

The GeoFilterKruskal algorithm was tested in practice against several other implementations of geometric minimum spanning tree algorithms. We chose a subset of the algorithms compared in [27], excluding some based on the availability of source code and the clear advantage shown by some algorithms in the aforementioned work. In the experimental results section the following algorithms will be referred to:

GEOFILTERKRUSKAL was written in C++ and compiled with g++ 4.3.2 with -O3 optimization. Parallel code was written using OpenMP [14] and the parallel mode extension to the STL [30]. C++ source code for GeoMST, GeoMST2, and Triangle were provided by Narsimhan. In addition, Triangle used Shewchuk's triangle library for Delaunay triangulation [32]. The machine used has 8 Quad-Core AMD Opteron(tm) Processor 8378 with hyperthreading enabled. Each core has a L1 cache size of 512 KB, L2 of 2MB and L3 of 6MB with 128 GB total memory. The operating system was CentOS 5.3. All data was generated and stored as 64 bit C++ doubles.

In the next section there are two distinct series of graphs. The first set displays graphs of total running time versus the number of input points, for two to five dimensional points, with uniform random distribution in a unit hypercube. The L_2 metric was used for distances in all cases, and all algorithms were run on the same random data set. Each algorithm was run on five data sets, and the results were averaged. As noted above, Triangle was not used in dimensions greater than two.

Table 1: Algorithm Descriptions

Name	Description
GeoFK#	Our implementation of Algorithm 2. There are two important differences between the implementation and the theoretical version. First, the BCCP Threshold β in section 3 is incremented in steps of size $\mathcal{O}(1)$ instead of size 1, because this change does not affect our analysis but helps in practice. Second, for small well separated pairs (less than 32 total points) the BCCP is computed by a brute force algorithm. In the experimental results, GeoFK1 refers to the algorithm running with 1 thread. GeoFK8 refers to the algorithm using 8 threads.
GeoMST	Described by Callahan and Kosaraju [8]. This algorithm computes a WSPD of the input data followed by the BCCP of every pair. It then runs Kruskal's algorithm to find the MST.
GeoMST2	Described in [27]. This algorithm improves on GeoMST by using marginal distances and a priority queue to avoid many BCCP computations.
Triangle	This algorithm first computes the Delaunay Triangulation of the input data, then applies Kruskals algorithm. Triangle was only used with two dimensional data.

The second set of graphs shows the mean total running times for two dimensional data of various distributions, as well as the standard deviation. The distributions were taken from [27] (given n d-dimensional points with coordinates $c_1...c_d$), shown in Table 2.

Table 2: Point Distribution Info

Name	Description
unif	c_1 to c_d chosen from unit hypercube with uniform distribution (U^d)
annul	c_1 to c_2 chosen from unit circle with uniform distribution, $c_3...c_d$ chosen from U^d
arith	c_1 = 0, 1, 4, 9, 16... c_2 to c_d are 0
ball	c_1 to c_d chosen from unit hypersphere with uniform distribution
clus	c_1 to c_d chosen from 10 clusters of normal distribution centered at 10 points chosen from U^d
edge	c_1 chosen from U^d, c_2 to c_d equal to c_1
diam	c_1 chosen from U^d, c_2 to c_d are 0
corn	c_1 to c_d chosen from 2^d unit hypercubes, each one centered at one of the 2^d corners of a (0,2) hypercube
grid	n points chosen uniformly at random from a grid with $1.3n$ points, the grid is housed in a unit hypercube
norm	c_1 to c_d chosen from $(-1, 1)$ with normal distribution
spok	For each dimension d' in d $\frac{n}{d}$ points chosen with $c_{d'}$ chosen from U^1 and all others equal to $\frac{1}{2}$

5 Experimental Results

As shown in Figure 2, GeoFK1 performs favorably in practice for almost all cases compared to other algorithms (see Table 1). In two dimensions, only Triangle outperforms GeoFK1. In higher dimensions, GeoFK1 is the clear winner when only one thread is used. Figure 4 shows how the algorithm scales as multiple threads are used. Eight threads were chosen as a representative for multi-core execution. We ran experiments for points drawn from uniform random distribution, ranging from two to five dimensions. In each of these experiments, we also show how the running time of GeoFK1 varies when run on 8 processors. We performed the same set of experiments for point sets from other distributions. Since the graphs were similar to the ones for uniform distribution, they are not included.

Our algorithm tends to slowdown as the dimension increases, primarily because of the increase in the number of well separated pairs [27]. For example, the number of well separated pairs generated for a two dimensional uniformly distributed data set of size 10^6 was approximately 10^7, whereas a five dimensional data set of the same size had 10^8 WSPs.

Figure 3 shows that in most cases, GeoFK1 performs better regardless of the distribution of the input point set. Apart from the fact that Triangle maintains its superiority in two dimensions, GeoFK1 performs better in all the distributions that we have considered, except when the points are drawn from

arith distribution. In the data set from *arith*, the ordering of the WSPs based on the minimum distance is the same as based on the BCCP distance. Hence the second partitioning step in GeoFK1 acts as an overhead. The results from Figure 3 are for two dimensional data. The same experiments for data sets of other dimensions did not give significantly different results, and so were not included.

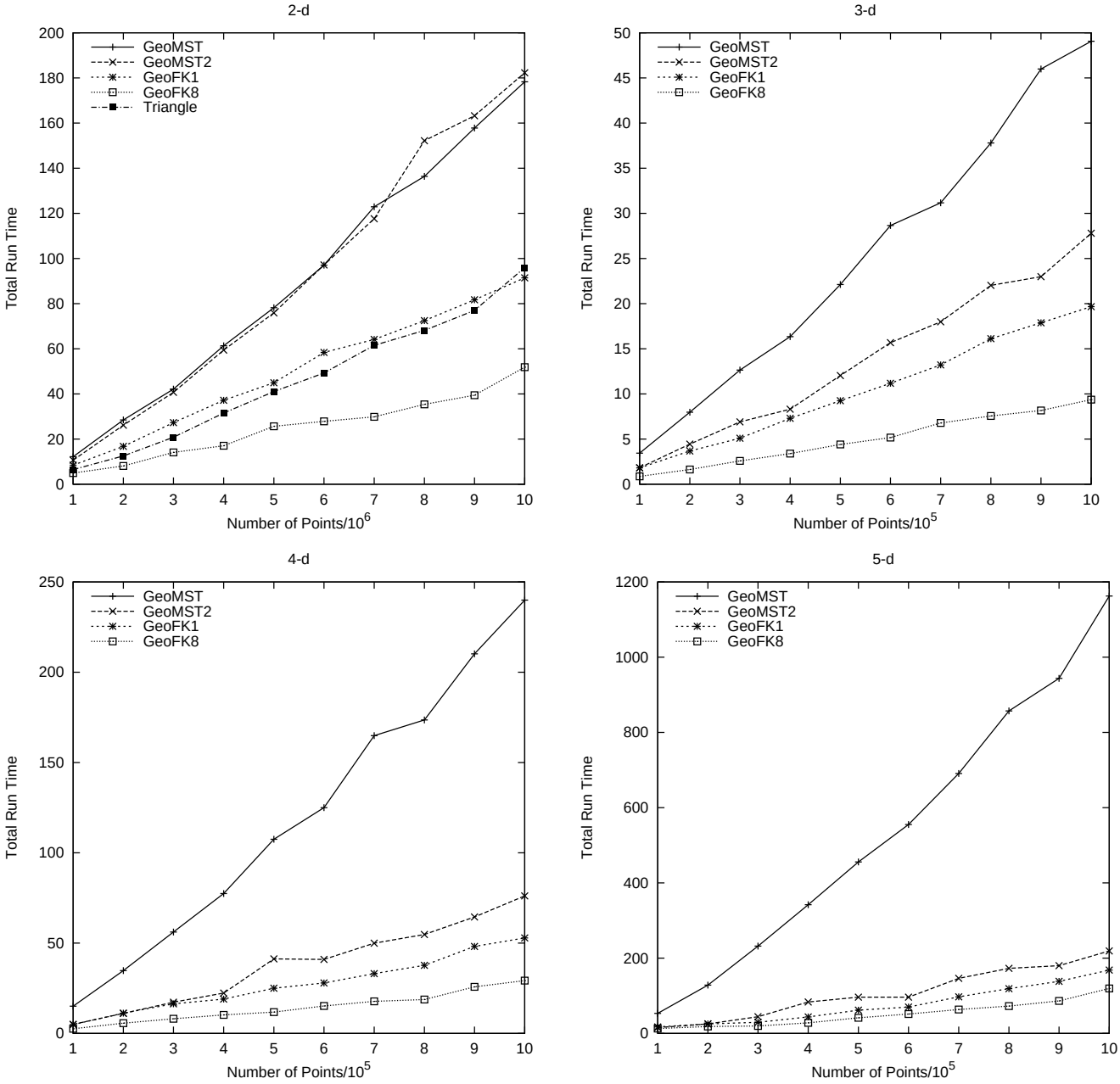

Fig. 2: This series of graphs shows the total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from 10^6 to 10^7 points for 2-d data, and 10^5 to 10^6 for other dimensions. For each data set size 5 tests were done and the results averaged.

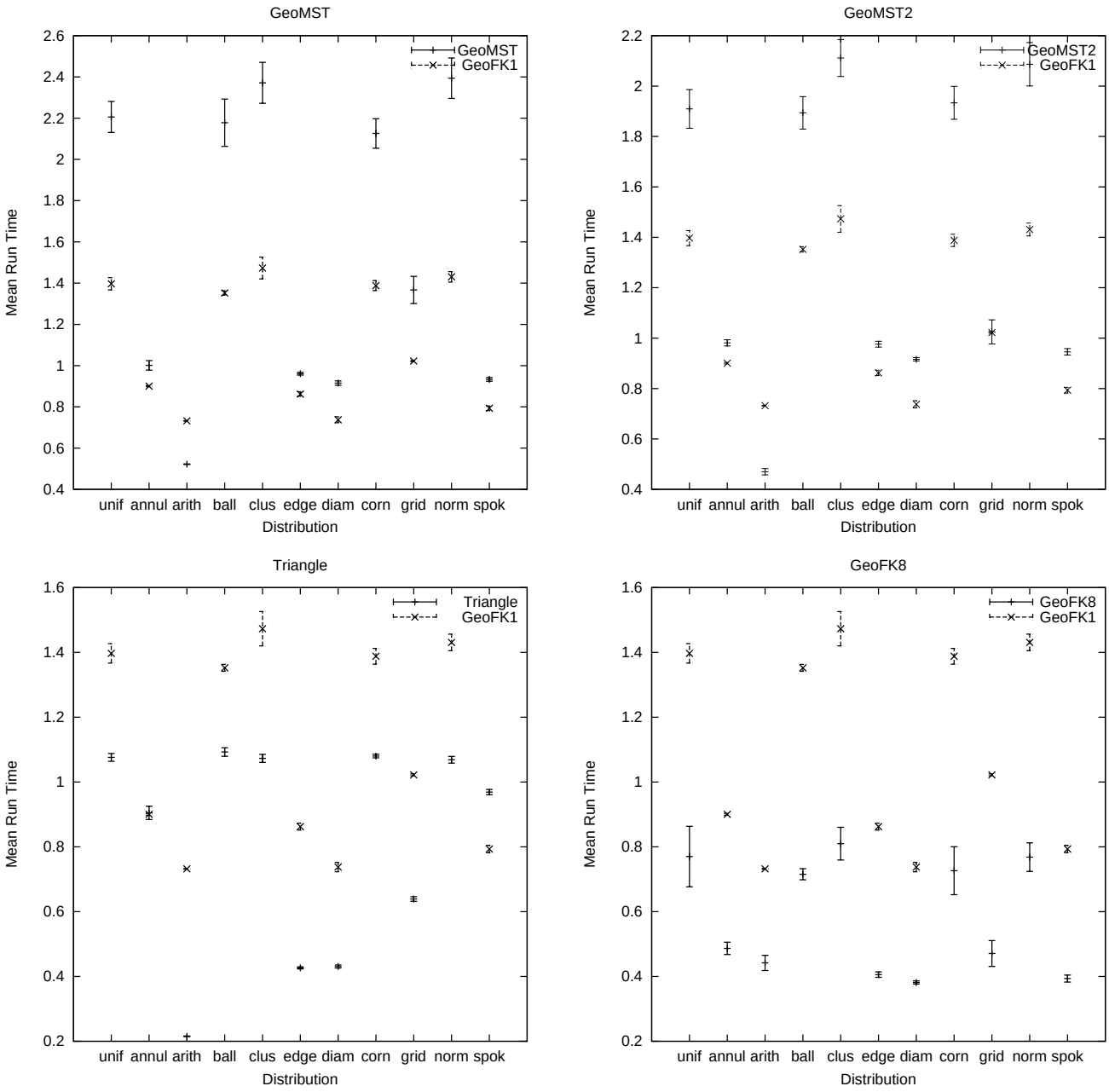

Fig. 3: For four different algorithms that compute GMST in two dimensions, this series of graphs show the mean run time and standard deviation of various distributions (see Table 2). In all cases, algorithms were run on a series of ten random data sets from a particular distribution. The data set size was 10^6 points.

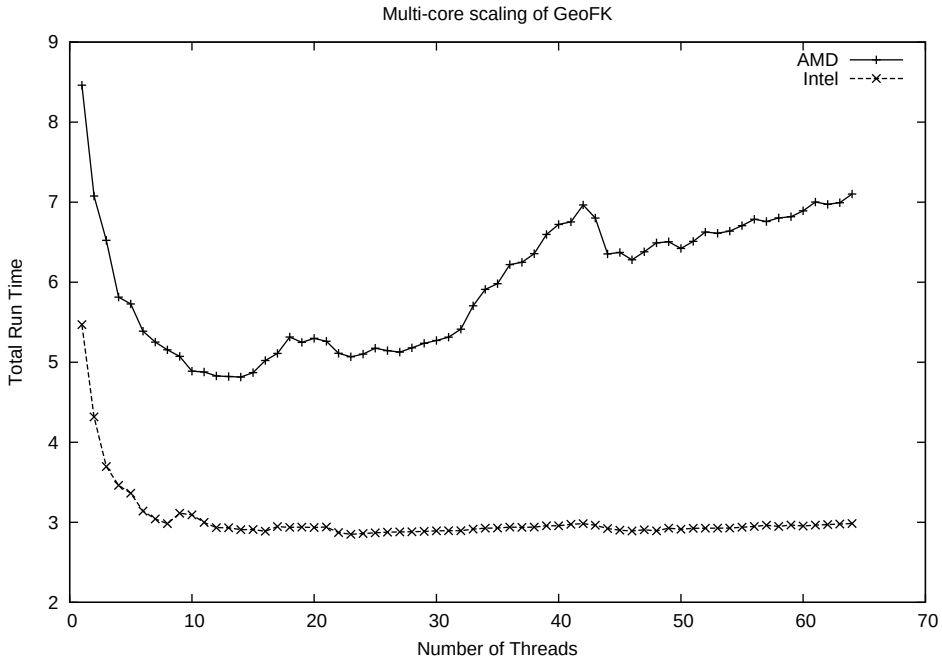

Fig. 4: This figure demonstrates the run time gains of the algorithm as more threads are used. We present scaling for two architectures. The AMD line was computed on the machine described in Section 4. The Intel line used a machine with four 2.66GHz Intel(R) Xeon(R) CPU X5355, 4096 KB of L1 cache, and 8GB total memory. All cases were run on 20 data sets from uniform random distribution of size 10^6 points, final total run time is an average of the results.

6 Conclusions and Future Work

This paper strives to demonstrate a practical GMST algorithm, that is theoretically efficient on uniformly distributed point sets, works well on most distributions and is multi-core friendly. To that effect we introduced the GeoFilterKruskal algorithm, an efficient, parallelizable GMST algorithm that in both theory and practice accomplished our goals. We proved a running time of $\mathcal{O}(n \log^2 n)$ with a strict definition of high probability, as well as showed extensive experimental results.

This work raises many interesting open problems. Since the main parallel portions of the algorithm rely on partitioning and sorting, the practical impact of other parallel sort and partition algorithms should be explored. In addition, since the particular well separated pair decomposition algorithm used is not relevant to the correctness of our algorithm, the use of a tree that offers better clustering might make the algorithm more efficient. Experiments were conducted only for L_2 metric in this paper. As a part of our future work, we plan to perform experiments on other metrics. We also plan to do more extensive experiments on the k-nearest neighbor approach in higher dimensions, for example $d > 10$.

References

1. Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(5):407–422, 1991.
2. Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.
3. Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.
4. J.D. Barrow, S.P. Bhavsar, and D.H. Sonoda. Min-imal spanning trees, filaments and galaxy clustering. *MNRAS*, 216:17–35, Sept 1985.
5. Jon Louis Bentley, Bruce W. Weide, and Andrew C. Yao. Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw.*, 6(4):563–580, 1980.

6. 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, Oct 1996.
7. J.J. Brennan. Minimal spanning trees and partial sorting. *Operations Research Letters*, 1(3):138–141, 1982.
8. Paul B. Callahan. *Dealing with higher dimensions: the well-separated pair decomposition and its applications.* PhD thesis, Johns Hopkins University, Baltimore, MD, USA, 1995.
9. Timothy M. Chan. Manuscript: A minimalist's implementation of an approximate nearest n eighbor algorithm in fixed dimensions, 2006.
10. Timothy M. Chan. Well-separated pair decomposition in linear time? *Inf. Process. Lett.*, 107(5):138–141, 2008.
11. Kenneth L. Clarkson. An algorithm for geometric minimum spanning trees requiring nearly linear expected time. *Algorithmica*, 4:461–469, 1989. Included in PhD Thesis.
12. Michael Connor and Piyush Kumar. Parallel construction of k-nearest neighbor graphs for point clouds. In *Proceedings of Volume and Point-Based Graphics.*, pages 25–32. IEEE VGTC, August 2008.
13. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2001.
14. L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.
15. R. A. Dwyer. Higher-dimensional voronoi diagrams in linear expected time. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 326–333, New York, NY, USA, 1989. ACM.
16. Jeff Erickson. On the relative complexities of some geometric problems. In *In Proc. 7th Canad. Conf. Comput. Geom*, pages 85–90, 1995.
17. Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 135–143, New York, NY, USA, 1984. ACM.
18. G.M.Morton. A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada, 1966.
19. David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42:321–328, 1995.
20. Philip N. Klein and Robert E. Tarjan. A randomized linear-time algorithm for finding minimum spanning trees. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 9–15, New York, NY, USA, 1994. ACM.
21. J.B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. American Math. Society*, pages 7–48, 1956.
22. Drago Krznaric, Christos Levcopoulos, and Bengt J. Nilsson. Minimum spanning trees in d dimensions. *Nordic J. of Computing*, 6(4):446–461, 1999.
23. Elmar Langetepe and Gabriel Zachmann. *Geometric Data Structures for Computer Graphics.* A. K. Peters, Ltd., Natick, MA, USA, 2006.
24. W. March and A. Gray. Large-scale euclidean mst and hierarchical clustering. *Workshop on Efficient Machine Learning*, 2007.
25. Robert Mencl. A graph based approach to surface reconstruction. *Computer Graphics Forum*, 14:445 – 456, 2008.
26. Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.
27. Giri Narasimhan and Martin Zachariasen. Geometric minimum spanning trees via well-separated pair decompositions. *J. Exp. Algorithmics*, 6:6, 2001.
28. Vitaly Osipov, Peter Sanders, and Johannes Singler. The filter-kruskal minimum spanning tree algorithm. In Irene Finocchi and John Hershberger, editors, *ALENEX*, pages 52–61. SIAM, 2009.
29. Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1985.
30. Felix Putze, Peter Sanders, and Johannes Singler. Mcstl: the multi-core standard template library. In *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 144–145, New York, NY, USA, 2007. ACM.
31. Sanguthevar Rajasekaran. On the euclidean minimum spanning tree problem. *Computing Letters*, 1(1), 2004.
32. Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.
33. Francis Suraweera and Prabir Bhattacharya. An o(log m) parallel algorithm for the minimum spanning tree problem. *Inf. Process. Lett.*, 45(3):159–163, 1993.
34. C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *Transactions on Computers*, C-20(1):68–86, 1971.