

Geometric Minimum Spanning Trees with GEOFILTERKRUSKAL*

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 takes one parallel sort plus a linear number of additional steps, with high probability, to compute the minimum spanning tree. Experiments show that our algorithm works better in practice for most data distributions compared to the current state of the art [31]. Our algorithm is easy to parallelize and to our knowledge, is currently the best practical algorithm on multi-core machines for $d > 2$.

Keywords: 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 [38], surface reconstruction [28], cosmology [4,6], TSP approximations [2] and computer graphics [26]. 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.

It is well established that the GMST is a subset of edges in the Delaunay triangulation of a point set [33]. It is well known that 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

* This research was partially supported by National Science Foundation through CAREER Grant CCF-0643593 and the AFOSR Young Investigator Program. Advanced Micro Devices (AMD) provided the 32-core workstation for experiments. Dr. Giri Narsimhan provided the source code for his Geometric Minimum Spanning tree algorithm [31], which was used as one of the benchmarks for the algorithm implemented in this paper. The source code associated with this paper is part of the STANN library and can be downloaded from www.compgeom.com/~stann

the red-green pair with minimum distance between them [25]. 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 [17].

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 function [14]. Bentley [5] also gave an expected nearly linear time algorithm for computing GMSTs in \mathbb{R}^d . Dwyer [16] 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 [23] to get an expected $\mathcal{O}(n)$ time algorithm for GMSTs of uniformly distributed points in a unit ball. Rajasekaran [35] 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, in our experiments, finding k -nearest neighbors for every point, even when $k = \mathcal{O}(1)$, proved to be as expensive as finding the actual GMST. We discuss this in Section 3, and show some experimental results in Section 5.

Narasimhan et al. [31] 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 [24] 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 [31] have been announced [27], 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 [24] that operated in a manner similar to quicksort; splitting an edge set into "light" and "heavy" subsets. Recently, Osipov et al. [32] 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 [31]. 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.

Since GeoMST2 is the only known practical algorithm for computing GMSTs, all our results, both experimental and theoretical, were compared against GeoMST2. The theoretical running time of GeoMST2 was analyzed only for uniform distributions. In this paper, when we talk about the theoretical running time of our algorithm, the underlying distribution should be assumed as uniform unless otherwise stated.

Our algorithm takes one parallel sort plus $\mathcal{O}(n)$ additional steps, with high probability, to compute the GMST. This result is an improvement over the original Filter-Kruskal algorithm [32]. The expected running time for constructing the MST for arbitrary graphs with random edge weights, using the original Filter-Kruskal algorithm [32] is $\mathcal{O}(m + n \log n \log m/n)$, where m and n are the number of edges and vertices of the graph respectively.

If the coordinates of the points in the input set are integers of size $\mathcal{O}(\log n)$, and the word size of the machine is greater than or equal to $\log n$, the running time of our algorithm is $\mathcal{O}(n)$ if we use radix sort [19,10]¹. The running time of GeoMST2 is on the contrary, $\mathcal{O}(n \log n)$, irrespective of the data-type of the point coordinates. Thus, for uniform distributions, we achieve the same runtime complexity as those algorithms described above, but without suffering from the drawback of the bucketing technique that makes them impractical to implement for other distributions. Assuming a CRCW PRAM model with $\mathcal{O}(\log n)$ processors, if sorting is allowed to be done in parallel (mergesort), the running time is $\mathcal{O}(n)$ for both integer and floating point coordinates.

The main contributions of this paper are: ① our algorithm shows significant runtime improvements over GeoMST2 for two and higher dimensions, ② the algorithm is easy to parallelize unlike GeoMST2, ③ in contrast with GeoMST2 which is inherently $\mathcal{O}(n \log n)$, our theoretical running time is upper bounded only by one sort which improves to $\mathcal{O}(n)$ for integers under the assumptions mentioned above, ④ our algorithm is faster compared to Filter-Kruskal on geometric instances and ⑤ a parallel implementation of the well separated pair decomposition on a compressed quadtree, which can be computed in $\mathcal{O}(n)$ time [10,20]. The code is now a part of the STANN library [12] and is available for download. At the time of submission of this paper we are not aware of any other open source implementation of the well separated pair decomposition using a compressed quadtree. For comparison purposes, we worked with the distributions on which GeoMST2 was run [31].

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. $\text{Dist}(p, q)$ denotes the distance between the points p and q in L_2 metric. Upper-case Roman letters are reserved

¹ If the integer size is superlogarithmic, we can still get $o(n \log n)$ running time by applying known integer sorting algorithms on a word RAM model [10].

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. $\text{Vol}()$ denotes the volume of an object. For a given quadtree, $\text{Box}(p, q)$ denotes the smallest quadtree box containing points p and q ; Fraktur letters (\mathfrak{a}) denote a quadtree node. $\text{MinDist}(\mathfrak{a}, \mathfrak{b})$ denotes the minimum distance between the quadtree boxes of two nodes. $\text{Bccp}(\mathfrak{a}, \mathfrak{b})$ computes the bichromatic closest pair of two nodes, and returns $\{u, v, \delta\}$, where (u, v) is the edge defining the Bccp and δ is the edge length. $\text{Left}(\mathfrak{a})$ and $\text{Right}(\mathfrak{a})$ denotes the left and right child of a node. $|\cdot|$ 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 [14]. The Cartesian product of two sets X and Y , is denoted $X \times Y = \{(x, y) \mid x \in X \text{ and } y \in Y\}$.

2 Preliminaries

Before we present our algorithm in detail, we need to describe a few tools which our algorithm uses extensively. These include Morton ordering, quadtrees, well separated pair decomposition, a practical algorithm for bichromatic closest pair computation and the UnionFind data structure. We describe these tools below.

Morton ordering and Quadtrees: Morton order is a space filling curve with good locality preserving behavior [21]. 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 [13]. In the next paragraph, we briefly state the quadtree construction that we use in our algorithm [10,20].

Let p_1, p_2, \dots, p_n be a given set of points in Morton order. Let the index j be such that $\text{Vol}(\text{Box}\{p_{j-1}, p_j\})$ is the largest. Then the compressed quadtree Q for the point set consists of a root holding the $\text{Box}(p_{j-1}, p_j)$ and two subtrees recursively built for p_1, \dots, p_{j-1} and p_j, \dots, p_n . Note that this tree is simply the *Cartesian tree* [20] of $\text{Vol}(\text{Box}\{p_1, p_2\}), \text{Vol}(\text{Box}\{p_2, p_3\}), \dots, \text{Vol}(\text{Box}\{p_{n-1}, p_n\})$. 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 [20], given the ordering p_1, p_2, \dots, p_n [10]. Hence, for both integer as well as floating point coordinates, the compressed quadtree Q can be computed in $\mathcal{O}(n)$ excluding the time for sorting the points in Morton order which takes $\mathcal{O}(n \log n)$ [13] for floating

point coordinates. We use a compressed quadtree, as opposed to the more common fair split tree [8], because the later takes $\mathcal{O}(n \log n)$ time for construction. If parallel quicksort [14] is used for the Morton ordering, the construction is $\mathcal{O}(n)$ for both integer and floating point coordinates.

Well Separated Pair Decomposition [8]: We implemented well separated pair decomposition on a compressed quadtree, which can be computed in $\mathcal{O}(n)$ time [10,20]. Assume that we are given a compressed quadtree Q , built on a set of points P in \mathbb{R}^d . Two nodes \mathbf{a} and $\mathbf{b} \in Q$ are said to be an ϵ -well separated pair (ϵ -WSP) if the diameter of \mathbf{a} and the diameter of \mathbf{b} are both at most $\epsilon * \text{MinDist}(\mathbf{a}, \mathbf{b})$. An ϵ -well separated pair decomposition (ϵ -WSPD), of size m , for a quadtree Q , is a collection of well separated pairs of nodes $\{(\mathbf{a}_1, \mathbf{b}_1), \dots, (\mathbf{a}_m, \mathbf{b}_m)\}$, such that every pair of points $(p, q) \in P \times P$ ($p \neq q$) lies in exactly one pair $(\mathbf{a}_i, \mathbf{b}_i)$. 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 theoretical running time of $\mathcal{O}(n \log n)$ for WSPD construction. The number of well separated pairs produced by this approach is more than that produced by the fair-split tree approach, although only by a constant factor. For example, for uniform distributions, on an average, we produce 27% more WSPs but are 1.33 times faster.

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 [25]. 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, $\text{MinDist}(\mathbf{a}, \mathbf{b})$ is the lower bound on the bichromatic closest pair distance of A and B (where A and B are the point sets contained in \mathbf{a} and \mathbf{b} respectively). We use a simple Bccp algorithm proposed by Narsimhan and Zachariasen [31].

The input to algorithm 1 are two nodes $\mathbf{a}, \mathbf{b} \in Q$, 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 $\delta = \text{Dist}(p, q)$. 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 , 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 $\text{Vol}(\mathbf{a})$ and $\text{Vol}(\mathbf{b})$ 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.

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: $\mathcal{G}.\text{Union}(u, v)$ updates the structure to indicate the partitions containing u and v belong to the same connected component; $\mathcal{G}.\text{Find}(u)$ returns

Algorithm 1. Bccp Algorithm [31]: Compute $\{p', q', \delta'\} = \text{BCCP}(a, b[, \{p, q, \delta\} = \eta])$

Require: $a, b \in Q, A, B \subseteq P, \delta \in \mathbb{R}^+$

Require: If $\{p, q, \delta\}$ is not specified, $\{p, q, \delta\} = \eta$, an upper bound on $\text{Bccp}(a, b)$.

```

1: procedure BCCP( $a, b[, \{p, q, \delta\} = \eta]$ )
2:   if ( $|A| = 1$  and  $|B| = 1$ ) then
3:     Let  $p' \in A, q' \in B$ 
4:     if  $\text{Dist}(p', q') < \delta$  then return  $\{p', q', \text{Dist}(p', q')\}$ 
5:   else
6:     if  $\text{Vol}(a) < \text{Vol}(b)$  then Swap( $a, b$ )
7:      $\gamma = \text{MinDist}(\text{Left}(a), b)$ 
8:      $\zeta = \text{MinDist}(\text{Right}(a), b)$ 
9:     if  $\gamma < \delta$  then  $\{p, q, \delta\} = \text{BCCP}(\text{Left}(a), b, \{p, q, \delta\})$ 
10:    if  $\zeta < \delta$  then  $\{p, q, \delta\} = \text{BCCP}(\text{Right}(a), b, \{p, q, \delta\})$ 
11:    return  $\{p, q, \delta\}$ 
12: end procedure

```

the node number of the *representative* of the partition containing u . Our implementation also does *union by rank* and *path compression*. A sequence of τ $\mathcal{G}.\text{Union}()$ and $\mathcal{G}.\text{Find}()$ operations on n elements takes $\mathcal{O}(\tau\alpha(n))$ time in the worst case. For all practical purposes, $\alpha(n) \leq 4$ (see [14]).

3 GEOFILTERKRUSKAL Algorithm

Our GEOFILTERKRUSKAL algorithm computes a GMST for $P \subseteq \mathbb{R}^d$. Kruskal's [24] 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, $\text{Dist}(u', v') < \text{Dist}(u, v)$. GeoMST2 [31] avoids the computation of Bccp for many well separated pairs that already belong 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 $\text{MinDist}(a, b)$ for all $(a, b) \in E_u$. 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: $S = \{(a_1, b_1), \dots, (a_m, b_m)\}$ is a WSPD, constructed from $P \subseteq \mathbb{R}^d$; $T = \{\}$.

Ensure: Bccp Threshold $\beta \geq 2$.

- 1: **procedure** GEOFILTERKRUSKAL(Sequence of WSPs : S , Sequence of Edges : T , UnionFind : \mathcal{G} , Integer : β)
- 2: $E_l = E_u = E_{l1} = E_{l2} = \emptyset$
- 3: **for all** $(a_i, b_i) \in S$ **do**
- 4: **if** $(|a_i| + |b_i|) \leq \beta$ **then** $E_l = E_l \cup \{(a_i, b_i)\}$ **else** $E_u = E_u \cup \{(a_i, b_i)\}$
- 5: **end for**
- 6: $\rho = \min\{\text{MinDist}\{a_i, b_i\} : (a_i, b_i) \in E_u, i = 1, 2, \dots, m\}$
- 7: **for all** $(a_i, b_i) \in E_l$ **do**
- 8: $\{u, v, \delta\} = \text{Bccp}(a_i, b_i)$
- 9: **if** $(\delta \leq \rho)$ **then** $E_{l1} = E_{l1} \cup \{(u, v)\}$ **else** $E_{l2} = E_{l2} \cup \{(u, v)\}$
- 10: **end for**
- 11: KRUSKAL(E_{l1}, T, \mathcal{G})
- 12: $E_{new} = E_{l2} \cup E_u$
- 13: FILTER(E_{new}, \mathcal{G})
- 14: **if** $(|T| < (n - 1))$ **then** GEOFILTERKRUSKAL($E_{new}, T, \mathcal{G}, \beta + 1$)
- 15: **end procedure**
- 16: **procedure** KRUSKAL(Sequence of WSPs : E , Sequence of Edges : T , UnionFind : \mathcal{G})
- 17: Sort(E): by increasing Bccp distance
- 18: **for all** $(u, v) \in E$ **do**
- 19: **if** $\mathcal{G}.\text{Find}(u) \neq \mathcal{G}.\text{Find}(v)$ **then** $T = T \cup \{(u, v)\}$; $\mathcal{G}.\text{Union}(u, v)$;
- 20: **end for**
- 21: **end procedure**
- 22: **procedure** FILTER(Sequence of WSPs : E , UnionFind : \mathcal{G})
- 23: **for all** $(a_i, b_i) \in E$ **do**
- 24: **if** $(\mathcal{G}.\text{Find}(u) = \mathcal{G}.\text{Find}(v) : u \in a_i, v \in b_i)$ **then** $E = E \setminus \{(a_i, b_i)\}$
- 25: **end for**
- 26: **end procedure**

3.1 Correctness

Given previous work by Kruskal [14] 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 [31], is the computation of the Bccp². If $|A| = |B| = \mathcal{O}(n)$, the Bccp algorithm stated in section 3 has a worst case time complexity of $\mathcal{O}(n^2)$. Since we have to process $\mathcal{O}(n)$ edges, naively, the computation time for GMST will be $\mathcal{O}(n^3)$ in the worst case.

² According to the algebraic decision tree model, the lower bound of the set intersection problem can be shown to be $\Omega(n \log n)$ [18]. We can solve the set intersection problem using Bccp. If the Bccp distance between two sets is zero, we can infer that the sets intersect, otherwise they do not. Since the set intersection problem is lower bounded by $\Omega(n \log n)$, the Bccp computation is also lower bounded by $\Omega(n \log n)$.

High Probability Bound Analysis: In this section we show that algorithm 2 takes one sort plus $\mathcal{O}(n)$ additional steps, with high probability (WHP) [30], to compute the GMST. Let $Pr(\mathcal{E})$ denote the probability of occurrence of an event \mathcal{E} , where \mathcal{E} is a function of n . An event \mathcal{E} is said to occur WHP if given $\beta > 1$, $Pr(\mathcal{E}) > 1 - 1/n^\beta$ [30]. Let P be a set of n points chosen uniformly from a unit hypercube H in \mathbb{R}^d . Given this, we state the following lemma from [31].

Lemma 2. *Let C_1 and C_2 be convex regions in H such that $\alpha \leq \text{volume}(C_1)/\text{volume}(C_2) \leq 1/\alpha$ for some constant $0 < \alpha < 1$. If $|C_1 \cap P|$ is bounded by a constant, then with high probability $|C_2 \cap P|$ is also bounded by a constant.*

We will use the above lemma to prove the following claims.

Lemma 3. *Given a constant $\gamma > 1$, WHP, algorithm 2 filters out WSPs that have more than γ points.*

Proof. The proof of this lemma is similar to the one for GeoMST2. Consider a WSP (a, b) . If both $|a|$ and $|b|$ are less than or equal to γ then the time to compute their Bccp distance is $\mathcal{O}(1)$. Let us now assume, w.lo.g., that $|a| > \gamma$. We will show that, in this case, we do not need to compute the Bccp distance of (a, b) WHP. Let \overline{pq} be a line segment joining a and b such that the length of \overline{pq} (let us denote this by $|\overline{pq}|$) is $\text{MinDist}(a, b)$. Let C_1 be a hypersphere centered at the midpoint of \overline{pq} and radius $|\overline{pq}|/4$. Let C_2 be another hypersphere with the same center but radius $3|\overline{pq}|/2$. Since a and b are well separated, C_2 will contain both a and b . Now, $\text{volume}(C_1)/\text{volume}(C_2) = 6^{-d}$. Since C_1 is a convex region, if $|C_1|$ is empty, then by Lemma 2, $|C_2|$ is bounded by a constant WHP. But C_2 contains a which has more than γ points. Hence C_1 cannot be empty WHP. Let $a \in a, b \in b$ and $c \in C_1$. Also, let the pair (a, c) and (b, c) belong to WSPs (u_1, v_1) and (u_2, v_2) respectively. Note that $\text{Bccp}(a, b)$ must be greater than $\text{Bccp}(u_1, v_1)$ and $\text{Bccp}(u_2, v_2)$. Since our algorithm adds the Bccp edges by order of their increasing distance, c and the points in a will be connected before the Bccp edge between a and b is examined. The same is true for c and the points in b . This causes a and b to belong to the same connected component WHP, and thus, our filtering step will get rid of the well separated pair (a, b) before we need to compute its Bccp edge. \square

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

Proof. Lemma 3 shows that, WHP, we only need to compute Bccp distances of WSPs of constant size. Since we compute Bccp distances incrementally, WHP, the number of calls to the GEOFILTERKRUSKAL procedure is also bounded above by $\mathcal{O}(1)$. 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 $\mathcal{O}(n)$ Find(u) operations done WHP. Thus the overall running time of the Union() and Find() operations is $\mathcal{O}(\alpha(n)n)$ WHP (see the paragraph on UnionFind in Section 2). \square

Theorem 1. *Algorithm 2 takes one sort plus $\mathcal{O}(n)$ additional steps, WHP, to compute the GMST.*

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. From Lemma 3, we can infer that the recursive call on GEOFILTERKRUSKAL is performed $\mathcal{O}(1)$ times WHP. Thus the total time spent in partitioning is $\mathcal{O}(n)$ WHP. Since the total number of Bccp edges required to compute the GMST is $\mathcal{O}(n)$, by Lemma 3, the time spent in computing all such edges is $\mathcal{O}(n)$ WHP. Total time spent in sorting the edges in the base case is $\mathcal{O}(n \log n)$. Including the time to compute the Morton order sort for the WSPD, the total running time of the algorithm is one sort plus $\mathcal{O}(n)$ additional steps WHP. \square

Remark 1. As explained in Section 2, the Morton order sort required to construct the compressed quadtree for the WSPD is $\mathcal{O}(n)$ if points in P have integer coordinates. This is also applicable in case of sorting the edges inside the KRUSKAL procedure of algorithm 2. Thus the whole algorithm runs in $\mathcal{O}(n)$ time in this case, WHP. On the contrary, GeoMST2 will always take $\mathcal{O}(n \log n)$ steps even if the points in the input set have integer coordinates and the WSPD is constructed using a quadtree. This is because it adds edges to the GMST one at a time and before each addition it has to invoke an insertion and/or deletion procedure in a priority queue of WSPs [31]. Each such operation is $\mathcal{O}(\log n)$ [14]. Since there are $\mathcal{O}(n)$ WSPs, GeoMST2 will run in $\mathcal{O}(n \log n)$ steps.

Remark 2. Using a k -nearest neighbor graph, we can modify algorithm 2 such that its running time is $\mathcal{O}(n)$ WHP, if points in P have integer coordinates. 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 [22]. In this forest, let T' be the tree with the largest number of points. Rajasekaran [35] 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 $\mathcal{O}(n^\beta)$ additional computations will be necessary to compute the GMST.

Remark 3. 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 $\mathcal{O}(1)$ 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 $\mathcal{O}(kn)$ processors. Hence, using $\mathcal{O}(n)$ processors, the minimum spanning tree of P can then be computed in $\mathcal{O}(\log n)$ time [37].

Remark 4. We did not pursue implementing the algorithms described in remarks two and three because they are inherently Monte Carlo algorithms [29]. 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 [13,3] showed construction times that were slower than the GEOFILTERKRUSKAL algorithm.

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 parallelize this loop using OpenMP [15] 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 $\mathcal{O}(\lceil (n \log n)/p \rceil + \log n)$ 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 [34] 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 [34]. 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 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 [31], excluding some based on the availability of source code and the clear advantage shown by some algorithms in the aforementioned work. Table 1 lists the algorithms that will be referred to in the experimental section.

GEOFILTERKRUSKAL was written in C++ and compiled with g++ 4.3.2 with -O3 optimization. Parallel code was written using OpenMP [15] and the parallel mode extension to the STL [34]. C++ source code for GeoMST, GeoMST2, and Triangle were provided by Narsimhan. In addition, Triangle used Shewchuk's triangle library for Delaunay triangulation [36]. 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 [31]. 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 Kruskal’s algorithm. Triangle only works with two dimensional data.

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 $\frac{1}{2}$

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 [31] (given n d -dimensional points with coordinates $c_1...c_d$), shown in Table 2.

5 Experimental Results

As shown in Figure 1, 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.

Our algorithm tends to slowdown as the dimension increases, primarily because of the increase in the number of well separated pairs [31]. 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 1, column 2, 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

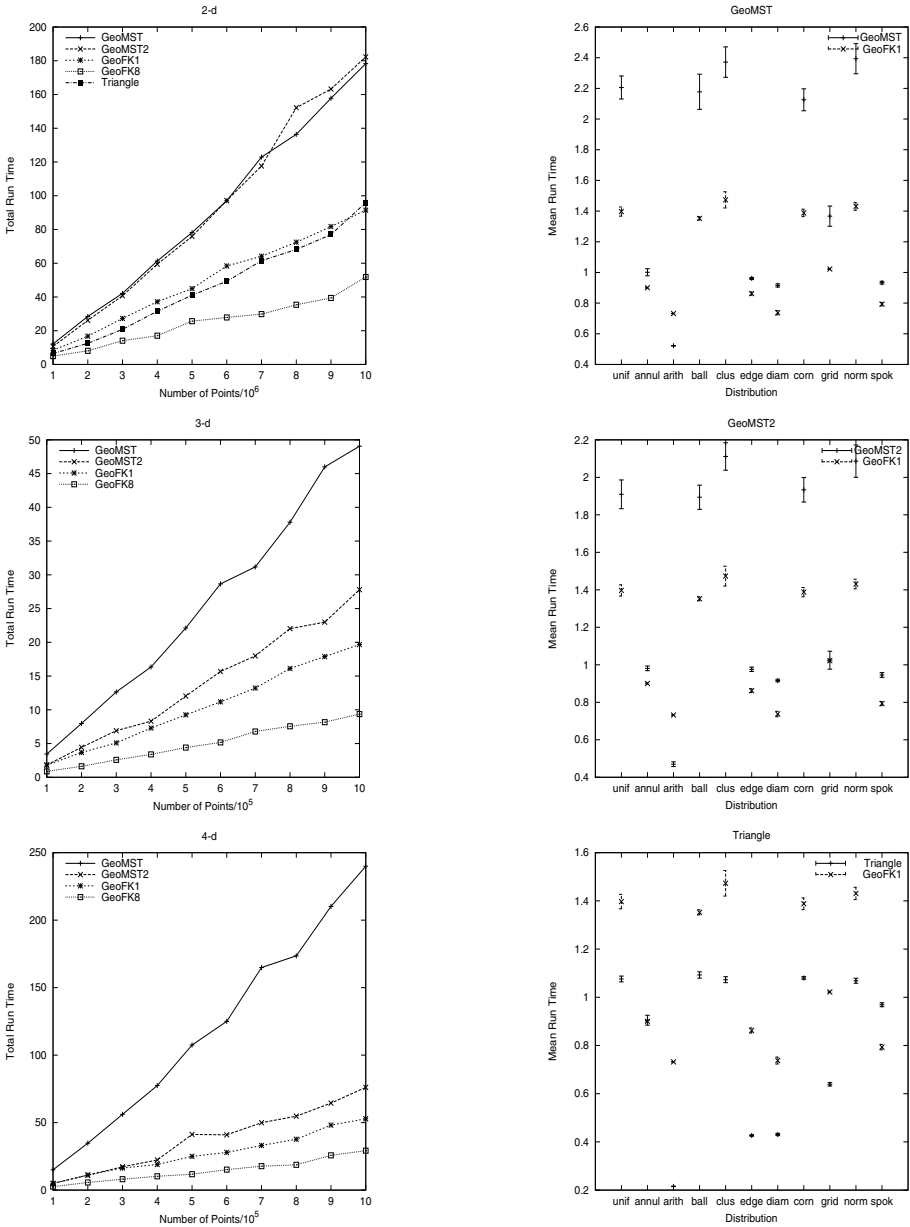


Fig. 1. The series of graphs in column 1 shows the total running time for each algorithm for varying sized data sets of uniformly random points, as the dimension increases. Data sets ranged from 10^6 to 10^7 points for 2-d data, and 10^5 to 10^6 for other dimensions. The graphs in column 2 show mean run time and standard deviation to compute the GMST on data sets of various distributions (see Table 2). The data set size was 10^6 points. For each data set size 5 tests were done and the results averaged.

as based on the **Bccp** distance. Hence the second partitioning step in **GeoFK1** acts as an overhead. The results from Figure 1, column 2 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.

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 $\mathcal{O}(n \log n)$, as well as provided 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. Agarwal, P.K., Edelsbrunner, H., Schwarzkopf, O., Welzl, E.: Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.* 6(5), 407–422 (1991)
2. Arora, S.: Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM* 45(5), 753–782 (1998)
3. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM* 45(6), 891–923 (1998)
4. Barrow, J.D., Bhavsar, S.P., Sonoda, D.H.: Minimal spanning trees, filaments and galaxy clustering. *MNRAS* 216, 17–35 (1985)
5. Bentley, J.L., Weide, B.W., Yao, A.C.: Optimal expected-time algorithms for closest point problems. *ACM Trans. Math. Softw.* 6(4), 563–580 (1980)
6. Bhavsar, S.P., Splinter, R.J.: The superiority of the minimal spanning tree in percolation analyses of cosmological data sets. *MNRAS* 282, 1461–1466 (1996)
7. Brennan, J.J.: 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. Chan, T.M.: Manuscript: A minimalist’s implementation of an approximate nearest neighbor algorithm in fixed dimensions (2006)
10. Chan, T.M.: Well-separated pair decomposition in linear time? *Inf. Process. Lett.* 107(5), 138–141 (2008)
11. Clarkson, K.L.: An algorithm for geometric minimum spanning trees requiring nearly linear expected time. *Algorithmica* 4, 461–469 (1989); Included in PhD Thesis

12. Connor, M., Kumar, P.: Stann library, <http://www.compgeom.com/~stann/>
13. Connor, M., Kumar, P.: Parallel construction of k -nearest neighbor graphs for point clouds. In: *Proceedings of Volume and Point-Based Graphics*, August 2008, pp. 25–32. IEEE VGTC (2008); Accepted to *IEEE Transactions on Visualization and Computer Graphics* (2009)
14. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*. MIT Press/McGraw-Hill (2001)
15. Dagum, L., Menon, R.: Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering* 5(1), 46–55 (1998)
16. Dwyer, R.A.: Higher-dimensional voronoi diagrams in linear expected time. In: *SCG 1989: Proceedings of the fifth annual symposium on Computational geometry*, pp. 326–333. ACM, New York (1989)
17. Erickson, J.: On the relative complexities of some geometric problems. In: *Proc. 7th Canad. Conf. Comput. Geom.*, pp. 85–90 (1995)
18. Erickson, J.G.: Lower bounds for fundamental geometric problems. PhD thesis, University of California, Berkeley, Chair-Seidel, Raimund (1996)
19. Franceschini, G., Muthukrishnan, S.M., Pătraşcu, M.: Radix sorting with no extra space. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) *ESA 2007. LNCS*, vol. 4698, pp. 194–205. Springer, Heidelberg (2007)
20. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: *STOC 1984: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pp. 135–143. ACM, New York (1984)
21. Morton, G.M.: A computer oriented geodetic data base; and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada (1966)
22. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM* 42, 321–328 (1995)
23. Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm for finding minimum spanning trees. In: *STOC 1994: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pp. 9–15. ACM, New York (1994)
24. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. In: *Proc. American Math. Society*, pp. 7–48 (1956)
25. Krznaric, D., Levkopoulos, C., Nilsson, B.J.: Minimum spanning trees in d dimensions. *Nordic J. of Computing* 6(4), 446–461 (1999)
26. Langetepe, E., Zachmann, G.: *Geometric Data Structures for Computer Graphics*. A. K. Peters, Ltd., Natick (2006)
27. March, W., Gray, A.: Large-scale euclidean mst and hierarchical clustering. In: *Workshop on Efficient Machine Learning* (2007)
28. Mencl, R.: A graph based approach to surface reconstruction. *Computer Graphics Forum* 14, 445–456 (2008)
29. Mitzenmacher, M., Upfal, E.: *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York (2005)
30. Motwani, R., Raghavan, P.: *Randomized algorithms*. *ACM Comput. Surv.* 28(1), 33–37 (1996)
31. Narasimhan, G., Zachariassen, M.: Geometric minimum spanning trees via well-separated pair decompositions. *J. Exp. Algorithmics* 6, 6 (2001)
32. Osipov, V., Sanders, P., Singler, J.: The filter-kruskal minimum spanning tree algorithm. In: Finocchi, I., Hershberger, J. (eds.) *ALLENEX*, pp. 52–61. SIAM, Philadelphia (2009)
33. Preparata, F.P., Shamos, M.I.: *Computational geometry: an introduction*. Springer, New York (1985)
34. Putze, F., Sanders, P., Singler, J.: Mcstl: the multi-core standard template library. In: *PPoPP 2007: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 144–145. ACM, New York (2007)

35. Rajasekaran, S.: On the euclidean minimum spanning tree problem. *Computing Letters* 1(1) (2004)
36. Shewchuk, J.R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: Lin, M.C., Manocha, D. (eds.) FCRC-WS 1996 and WACG 1996. LNCS, vol. 1148, pp. 203–222. Springer, Heidelberg (1996); From the First ACM Workshop on Applied Computational Geometry
37. Suraweera, F., Bhattacharya, P.: An $o(\log m)$ parallel algorithm for the minimum spanning tree problem. *Inf. Process. Lett.* 45(3), 159–163 (1993)
38. Zahn, C.T.: Graph-theoretical methods for detecting and describing gestalt clusters. *Transactions on Computers* C-20(1), 68–86 (1971)