Experiments with Computing Geometric Minimum Spanning Trees

Giri Narasimhan and Jianlin Zhu
Department of Mathematical Sciences, The University of Memphis
and
Martin Zachariasen
Department of Computer Science, University of Copenhagen

Let S be a set of n points in $\Re^d$. We present an algorithm that uses the well-separated pair decomposition and computes the minimum spanning tree of S under any L_p or polyhedral metric. It has an expected running time of $O(n \log n)$ for uniform distributions. Experimental results show that this approach is practical. Under a variety of input distributions, the resulting implementation is robust and performs well for points in higher dimensional space.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms: Computational Geometry, Experimental Algorithmics

Additional Key Words and Phrases: geometric minimum spanning trees

1. INTRODUCTION

Given a set S of n points in $\Re^d$, we refer to the *Euclidean minimum spanning tree* (EMST) of S as the minimum spanning tree (MST) of the complete graph on S with edge weights equal to the Euclidean distance between the points. If the edge weights are from a Minkowski or polyhedral metric, then we refer to it as the *Geometric minimum spanning tree* (GMST). EMST for points in the plane can be computed efficiently in $O(n \log n)$ time using Delaunay triangulations. Unfortunately, it becomes inefficient to use the Delaunay triangulation for higher

Supported partially by the Danish Natural Science Research Council under contract 9701414 (project "Experimental Algorithmics").

Name: G. Narasimhan and J. Zhu
Address: Campus Box 526429, Memphis, TN 38152, USA; e-mail: `giri@msci.memphis.edu`
Name: M. Zachariasen
Address: Universitetsparken 1, Copenhagen, Denmark; e-mail: `martinz@diku.dk`

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works, requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept, ACM Inc., 1515 Broadway, New York, NY 10036 USA, fax +1 (212) 869-0481, or `permissions@acm.org`.

dimensions ($d > 2$), since it could have $\Theta(n^2)$ edges even for points in $\Re^3$.

Agarwal et al. [1991] showed that the EMST problem can be reduced to a set of instances of the *bichromatic closest pair* (BCP) problem, which can be stated as follows: given a set P of n red points and m blue points in $\Re^d$, find the closest red-blue pair. Given a set of n points in $\Re^d$, Agarwal et al. [1991] presented an $O(T_d(n,n)\log^d n)$-time algorithm to compute a EMST, where $T_d(n,m)$ is the time complexity of the BCP problem.

Callahan and Kosaraju [1993] used well-separated pair decompositions to design an algorithm with a running time of $O(T_d(n,n)\log n)$. They, too, used a reduction to a set of instances of the BCP problem. Krznaric et al. [1997] improved the running time to $O(T_d(n,n))$.

We develop efficient algorithms for the BCP and GMST problems. Our GMST algorithm (referred to as GeoMST2) has an expected time complexity of $O(n \log n)$ for uniformly random points. We present extensive experimental results to show that it performs well on a variety of input distributions.

For input points from $\Re^d$, we compared the performance of our program with existing ones, including LEDA [Mehlhorn and Näher 1995] and Triangle [Shewchuk 1996] for $d = 2$, and Qhull [Barber et al. 1996] for $d \geq 2$. After considering practical issues, several older algorithms were not selected for comparison. Among them, the best algorithm may be due to Bentley and Friedman [1978] (variant of Prim's using k-d trees). Their experiments suggest that it runs in $O(n \log n)$ time and that it performs poorly on clustered inputs.

It should be noted that algorithms having better expected running times do exist. Using the fact that geographic neighbor graphs can be computed in expected linear-time for quasi-uniform distributions [Bentley et al. 1980], Clarkson [1989] gave an (expected) nearly linear-time algorithm for computing EMSTs in fixed dimension. Furthermore, Dwyer [1991] gave an expected linear-time algorithm for computing (Euclidean) Delaunay triangulations for quasi-uniform distributions in fixed dimension. All these approaches used bucketing techniques, which tend to perform poorly on clustered distributions as argued by Shewchuk [1996]. Also, Bentley [1992] chose the algorithm in [Bentley and Friedman 1978] (over the one based on [Bentley et al. 1980]) when implementing the MST-heuristic for TSP.

We conclude that for computing GMSTs for points in $\Re^d$, $d \geq 3$, the well-separated pair decomposition is a practical and sophisticated tool and our resulting program outperforms available implementations for this problem. For 2-dimensional point sets, although our program compares favorably with existing programs, it may be more efficient to use methods that use Delaunay triangulations. We also conclude that methods that use the well-separated pair decomposition are likely to generalize smoothly to other metrics. Additionally, our implementation eliminates the use of square root function and is robust. It is significant that our algorithm shows no asymptotic deterioration in performance for inputs from a variety of distributions.

In Section 2, we briefly review the well-separated pair decomposition. In Section 3, we discuss our algorithms. In Section 4 and 5, we discuss the implementation and our computational experience.

2. WELL-SEPARATED PAIR DECOMPOSITION

For a set S of n points in $\Re^d$, a *well-separated pair decomposition* (henceforth abbreviated as WSPD) is a set of pairs of subsets of S, $(A_1, B_1), \ldots, (A_m, B_m)$, such that every pair of points in S is represented in exactly one of the pairs and every pair of sets is well-separated [Callahan and Kosaraju 1995]. A pair of point sets (A, B) is said to be *well-separated* if $\text{MARGDISTANCE}(A, B) \geq \max\{Diam(R_A), Diam(R_B)\}$. Here R_A denotes the bounding box of a point set A, $Diam(R_A)$ denotes the diameter of a bounding hypersphere for R_A (or the length of the diagonal of R_A). As used in [Callahan 1995], $\text{MARGDISTANCE}(A, B)$ is the minimum distance between the bounding hyperspheres for R_A and R_B. Note that a generalized definition of well-separatedness is given in Callahan and Kosaraju [1995] using a separation constant s as a parameter; the simpler definition given above is sufficient for our purposes and coincides with the generalized definition if $s = 2$. In this paper, we use a different MARGDISTANCE (see Section 3.2), which is provably a better choice.

Callahan and Kosaraju [1995] showed that for a set S of n points in $\Re^d$, a WSPD of S with $O(n)$ pairs can be constructed in $O(n \log n)$ time. The algorithm for computing a WSPD uses a fair split tree data structure. It is similar in spirit to the quad-trees and k-d trees. The two main advantages that the WSPD has over the Delaunay triangulation is that: (a) its size is smaller ($\Theta(n)$ vs. $\Theta(n^2)$), and is thus better suited for higher-dimensional geometric problems, and (b) it readily generalizes to L_p and polyhedral metrics.

3. THE MST ALGORITHMS

3.1 The Basic Algorithm

Let S be a given set of n points in $\Re^d$. Algorithm GEOMST (see Callahan and Kosaraju [1993] and Eppstein [1999]) shown in Figure 1 computes a GMST of S. In Section 3.3, we present GEOMST2, which improves on GEOMST.

```
algorithm GeoMST(S)
.   {(A_1,B_1),...,(A_m,B_m)} := WSPD(S)
.   E := ∅
.   for i := 1 to m do
.      {a_i,b_i,D} := BCP(A_i,B_i,∞)
.      E := E ∪ {(a_i,b_i)}
.   endfor
.   G := (S,E); return Kruskal(G)
end
```

Fig. 1. The basic GMST Algorithm.

GEOMST first constructs a WSPD of S, denoted by $\{(A_1, B_1), \ldots, (A_m, B_m)\}$. It then runs Kruskal's algorithm (or any efficient MST algorithm) on a sparse graph G defined as follows. The vertex set of graph G is S. For each pair (A_i, B_i) in the decomposition, G has one edge (a_i, b_i), where $\{a_i, b_i\}$ is the closest pair of points such that $a_i \in A_i$ and $b_i \in B_i$. Note that the call to BCP returns the closest pair $\{a_i, b_i\}$ and a value $D = |a_i b_i|$, which denotes the length of the edge (a_i, b_i). The graph G is sparse with only $O(n)$ edges because the number of pairs m in the WSPD is $O(n)$.

A WSPD with $O(n)$ pairs can be computed in $O(n \log n)$ time. Also, Kruskal's algorithm has a running time of $O(n \log n)$ for graphs with $O(n)$ edges. Thus, the worst-case time complexity of this algorithm is dominated by $O(n)$ BCP computations.

Our GMST algorithm starts with this basic framework. In this paper, we propose a new algorithm to address the BCP problem. Our idea is to use the WSPD itself (which has already been computed in the first step of GeoMST) in order to solve the BCP problem.

3.2 Computing BCPs

Our new algorithm to solve the BCP problem uses the fair split tree data structure as a "divide-and-conquer" tool. Figure 2 gives the BCP algorithm. The input to the BCP algorithm includes two well-separated point sets ($S(u)$ and $S(v)$) corresponding to two nodes (u and v) from the fair split tree. It outputs the closest pair of points $\{p, q\}$ such that $p \in S(u)$ and $q \in S(v)$ and $D = |pq|$. D is initialized to ∞.

```
algorithm BCP(S(u), S(v), D)
.  if (S(u) = {p} and S(v) = {q})
.    D = |pq|
.  else
.    if (Diam(S(u)) < Diam(S(v))) SWAP(u, v)
.    L = MARGDISTANCE(S(left(u)), S(v))
.    R = MARGDISTANCE(S(right(u)), S(v))
.    if (L < R)
.      if (L < D)
.        {p, q, D} := BCP(S(left(u)), S(v), D)
.      if (R < D)
.        {p, q, D} := BCP(S(right(u)), S(v), D)
.    else
.      if (R < D)
.        {p, q, D} := BCP(S(right(u)), S(v), D)
.      if (L < D)
.        {p, q, D} := BCP(S(left(u)), S(v), D)
.  return {p, q, D}
end
```

Fig. 2. The BCP Algorithm.

The algorithm works as follows. If both $S(u)$ and $S(v)$ are singleton sets, then the recursion is terminated. Otherwise, let u be the node for which the diameter of the bounding box of the corresponding point set $S(u)$ is the larger of the two (a swap is made, if this is not so). Now the algorithm goes on to recursively determine the BCP between two pairs of sets: $(S(left(u)), S(v))$ and $(S(right(u)), S(v))$, where $left(u)$ and $right(u)$ are the children of node u in the fair split tree. Before doing so, it does two things: it decides whether each of the recursive calls is necessary, and if both are deemed necessary, it decides the best order in which to perform them. The first task is achieved by determining a lower bound on the distance between the closest pair that will result from a recursive call. The call to MargDistance computes the lower bound; if this bound is not better than the distance between the closest pair of points found so far (stored in D), then that recursive call is not performed. The second task is achieved by comparing the two bounds. The one with the smaller bound is performed first. In many instances, the closest pair obtained from the first recursive call will eliminate the second recursive call.

Computing the Marginal Distance

For two point sets, we define MargDistance to be the minimum distance between the two corresponding bounding boxes (length d_{box} in Figure 3). As mentioned earlier, this is different from the one used in [Callahan 1995], where the length d_{ball} was used. Since $d_{\mathrm{box}} \geq d_{\mathrm{ball}}$, using d_{box} gives a tighter definition of MargDistance, and the resulting number of well-separated pairs decreases. In order to differentiate between the two measures of MargDistance, we refer to d_{box} as the *box distance* and to d_{ball} as the *ball distance*.

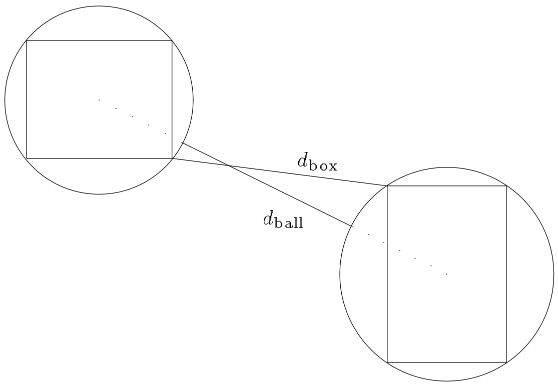

Fig. 3. Marginal distance measures: box distance (d_{box}) and ball distance (d_{ball}).

Time Complexity Analysis

If the BCP algorithm is called with two input sets of cardinalities n_u and n_v, then it has an obvious upper bound of $O(n_u \cdot n_v)$. This is because every pair of points can be inspected at most once. We conjecture that on uniformly random points, the expected running time of our BCP algorithm is only $O(\log^{d-1}(n_u + n_v))$. Our experiments seem to confirm this conjecture. This implies that on uniformly random points the expected running time for the $O(n)$ calls to BCP (and of algorithm GeoMST) is likely to be $O(n \log^{d-1} n)$. In Section 3.3 we propose a refined algorithm GeoMST2 for which the expected total running time for the $O(n)$ calls to BCP is only $O(n)$ for uniformly random points, thus giving an algorithm for MST that has an expected running time of $O(n \log n)$.

3.3 The Improved GMST Algorithm

Our algorithm GeoMST2 is given in Figure 4. While it follows the algorithm outlined in Figure 1, it contains several modifications that considerably improve the expected running time.

```
algorithm GeoMST2(S)
.  {(A_1,B_1),...,(A_m,B_m)} := WSPD(S)
.  E := ∅; PQ := ∅
.  for i := 1 to m do
.    PQ.Insert(i,MargDistance(A_i,B_i))
.    ComputedBCP[i] := false
.  endfor
.  while (not PQ.Empty()) do
.    i := PQ.ExtractMin()
.    if (A_i and B_i are not connected)
.      if (ComputedBCP[i])
.        E := E ∪ {(a_i,b_i)}
.      else
.        {a_i,b_i,D} := BCP(A_i,B_i,∞)
.        PQ.Insert(i,D)
.        ComputedBCP[i] := true
.  endwhile
.  T := (S,E); return (T)
end
```

Fig. 4. The improved GMST Algorithm.

The significant feature of algorithm GeoMST2 is that it merges two of the steps from algorithm GeoMST: the step that makes the BCP calls and the step from Kruskal's algorithm that considers edges for inclusion into the MST. The consequence is that it does not compute the BCP for every well-separated pair in the decomposition. When Kruskal's algorithm considers an edge $e = (a, b)$, it needs to test whether a and b belong to the same connected component. Kruskal's algorithm would consider edge $e = (a, b)$ for inclusion in the MST only if $\{a, b\}$ is the closest pair of points for some pair (A_i, B_i) of the WSPD. We make the following crucial observation: if, for *some* $a \in A_i$ and $b \in B_i$, a and b belong to the same connected component of the MST, then for *every* $a \in A_i$ and $b \in B_i$, a and b must belong to the same connected component of the MST built so far. This is an easy consequence of the properties of the WSPD and the fact that Kruskal's algorithm considers edges in increasing order of their length. If a pair (a, b) is being inspected by Kruskal's algorithm, then every pair (c, d) such that $|cd| < |ab|$ must belong to the same component of the MST. This implies that, in particular, when Kruskal's algorithm is considering the closest pair between a well-separated pair (A_i, B_i) for inclusion in the MST, it has already connected all points within A_i and all points within B_i. Thus, if the two sets A_i and B_i belong to the same connected component of the MST, then there is no need to compute the BCP for this pair.

In order to further reduce the number of BCP computations, we make yet another modification to the algorithm. Observe that if Kruskal's algorithm needs to consider the edges between the well-separated pairs in sorted order, they need to be precomputed. But then this implies making calls to BCP for every well-separated pair. The sorted order is simulated by using a priority queue PQ of well-separated pairs. Instead of using the length of the shortest edge between them as the priority, the idea is to use a lower bound for the length of the shortest edge connecting the well-separated pair. We use the box distance as the bound.

The ordering of the well-separated pairs according to the box distances is not necessarily the same as the order according to the length of the shortest edge between them. Therefore, when a well-separated pair that is not already connected in the MST is considered, and the shortest edge connecting them is computed using BCP, this edge cannot be directly added to the MST (as would be done by the greedy Kruskal's algorithm). Instead, the well-separated pair is returned to the priority queue with the priority value updated to the length of the computed shortest edge. Thus, the priority queue may consist of well-separated pairs for which the shortest edge has been computed as well as those for which the shortest edge has not been computed.

When a well-separated pair that is not already connected in the MST is considered, and if its shortest edge has already been computed, then it is safe to add the edge to the MST. Thus, an edge is added to the MST only after all well-separated pairs with box distances smaller than the length of this edge have been processed, guaranteeing the correctness of GeoMST2. Furthermore, the expected running time for the BCP calls actually made turns out to be significantly better as shown both in the analysis in Section 3.4 as well as in the experimental data in Section 5.

3.4 Expected Time Analysis

A WSPD with $O(n)$ pairs can be computed in $O(n \log n)$ time in the worst-case. So at most $O(n)$ edges are dealt with by the priority queue, whose operations (Insert and ExtractMin) require a worst-case running time of $O(n \log n)$. What remains to be analyzed is the time taken by the $O(n)$ calls to BCP. Theorem 1 below states that the total time spent by all the recursive calls to BCP is only $O(n)$ and not $O(n \log^{d-1} n)$ as one might guess, thus proving that the expected running time of GeoMST2 is $O(n \log n)$.

In what follows, we assume that S is a set of n points chosen uniformly at random from the unit hypercube H in $\Re^d$. Theorem 1 requires the following two technical lemmas, which are stated here without proof. We use $|A|$ to denote the cardinality of a point set A.

Lemma 1. *Let C' and C'' be convex regions in H, such that*

$$\alpha \leq volume(C')/volume(C'') \leq 1/\alpha$$

for some constant $0 < \alpha < 1$. If $|C' \cap S|$ is bounded by a constant, then with high probability $|C'' \cap S|$ is also bounded by a constant.

Lemma 2. *Let (A, B) be a well-separated pair in the decomposition. With high probability, $Diam(R_A)$, $Diam(R_B)$ and MargDistance(A, B) are within a constant factor of each other; also, $|A|$ and $|B|$ are within a constant factor of each other.*

THEOREM 1. *Let S be a set of n points chosen uniformly at random from the unit hypercube H in $\Re^d$. The expected running time of algorithm* GEOMST2 *is $O(n \log n)$. Each of the calls to BCP is expected to run in $O(1)$ time.*

PROOF. If algorithm GEOMST2 calls algorithm BCP for a well-separated pair (A, B), then the points in set A must belong to one connected component in the MST (since these points are closer to each other than the points in the pair (A, B)), as must the points in set B. Since BCP is called for (A, B), the points in A and the points in B must belong to different components.

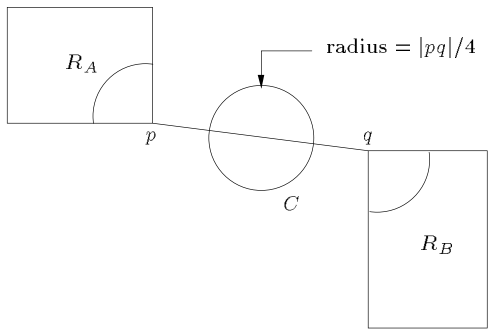

Fig. 5. Average-case analysis for GEOMST2.

Note that if $|A|$ and $|B|$ are both bounded by a constant, then the call to BCP takes $O(1)$ time. So we may assume that $|A|$ is not bounded by a constant. Lemma 2 states that for any well-separated pair (A, B), the number of points contained in them should be within a constant factor of each other. Thus the event that $|B|$ is bounded by a constant and $|A|$ is not bounded by any constant has arbitrarily small probability.

We are thus left with the case when both $|A|$ and $|B|$ are not bounded by a constant and are within a constant factor of each other. Let R_A and R_B be the bounding boxes for sets A and B respectively. Consider a shortest line segment with endpoints p and q that touches the boundaries of R_A and R_B without touching their interiors. Since (A, B) is a well-separated pair, we know that $|pq|$ is at least as large as the length of a diagonal for both R_A and R_B. Let C be a hypersphere whose center is located at the midpoint of segment pq and whose radius is $|pq|/4$ (see Figure 5). First we show that C cannot be empty of points from S (with high probability). The proof is by contradiction. Consider another hypersphere C' with the same center as C and with radius $3|pq|/2$. Thus, $volume(C)/volume(C') = 6^{-d}$. Since C is a large empty convex region, by Lemma 1, $|C'|$ must be bounded by a constant with high probability. But then, since R_A, R_B and C lie inside C', if $|C'|$ is bounded by a constant, then so are $|A|$ and $|B|$, contradicting our assumption.

Therefore C must be nonempty. Let c be a point in C. We will now show that c and the points in A are in the same connected component of the MST. Let a be the closest point to c from among the points in A. Let C_A be the intersection of

R_A and a hypersphere centered at p with radius $|pq|/8$. Since the diagonal of R_A is of length at most $|pq|$, we know that $volume(C_A)$ is a sufficiently large fraction of $volume(R_A)$. By Lemma 1 we know that with high probability C_A cannot be empty (since $|A|$ is not bounded by a constant). Any point in C_A is at distance at most $7|pq|/8$ from c, implying that $|ac| < |pq|$. Since the algorithm is considering a well-separated pair with marginal distance $|pq|$, any pair of points at distance less than $|pq|$ from each other must already be in one connected component. Hence a and c must be in the same connected component, implying that c and all points in A are in the same connected component. Using a similar argument, it can be shown that c and all points in B are in the same connected component. Thus A and B are already connected and the call to BCP for the pair (A, B) could not have been made. Hence the expected total time spent on the $O(n)$ calls to BCP is bounded by $O(n)$ and the expected running time of GeoMST2 is $O(n \log n)$. □

4. IMPLEMENTATION

All experiments were made on a HP9000 workstation using the C++ programming language[1]. In this section we present the algorithms that were compared (Section 4.1) and give some details on their implementation (Section 4.2).

4.1 Algorithms Implemented

The following GMST algorithms were compared: (a) **GeoMST**: an implementation of the GeoMST algorithm (Figure 1). (b) **GeoMST2**: an implementation of the GeoMST2 algorithm (Figure 4). (c) **LEDA**: we used the MIN_SPANNING_TREE procedure from LEDA [Mehlhorn and Näher 1995] (version 3.8); this procedure uses LEDA's own Delaunay triangulation algorithm and graph MST algorithm and is applicable only for 2-d. (d) **Triangle**: we used Shewchuk's fast and robust two-dimensional Delaunay triangulator Triangle [Shewchuk 1996] (version 1.3) in combination with our own implementation of Kruskal's algorithm; we used the default divide-and-conquer Delaunay Triangulation algorithm based on exact arithmetic. (e) **Qhull**: we used Qhull [Barber et al. 1996] (version 2.5) to compute the Delaunay triangulation and combined it with our own implementation of Kruskal's algorithm. The Delaunay triangulation is computed by lifting the input points to a paraboloid and computing the convex hull for the lifted points. (f) **Kruskal**: our own implementation of Kruskal's algorithm applied to the complete graph.

4.2 Implementation Details

As pointed out by Callahan [1995] the theoretically optimal $O(n \log n)$ algorithm for constructing a fair split tree is less attractive from a practical viewpoint when compared to a simpler sub-optimal alternative, which we chose to use. This alternative algorithm always divides the dimension with maximum spread exactly in half and recurses. The running time of this algorithm depends on the ratio d_{max}/d_{min} between the largest and smallest interpoint-distance. As long as $\log(d_{max}/d_{min}) = O(\log n)$, a worst-case running time of $O(n \log n)$ is guaranteed.

[1]Machine: HP Visualize Model C200. Processor: 200 MHz PA-8200. MM: 1 GB. Performance: 14.3 SPECint95 and 21.4 SPECfp95. OS: HP-UX 10.20. Compiler: GNU C/C++ 2.8.1.

The sorting step of Kruskal's algorithm can be sped up siginficantly in practice by using the following technique (which is also employed by the implementation of Kruskal's algorithm in LEDA). Instead of sorting all edges by non-decreasing length we first select the $N = 1.5nd$ smallest edges; the functional dependency on n (the number of points) and d (the dimension) was found by experimenting with various alternatives.

Now, the small edges are sorted and considered for insertion into the MST. All remaining edges that connect points in two different components are then sorted and considered for insertion into the MST. When the number of edges is significantly larger than N, this does (in practice) reduce the running time to being linear in the number of edges since the smallest edges contain almost all edges of the MST; therefore very few edges are selected among the long edges.

A similar technique has been used in the GeoMST2 algorithm. Firstly, we do not use a priority queue for storing all well-separated pairs. Instead we select the N closest well-separated pairs (based on their marginal distances), sort them by marginal distance and use the algorithm described in Figure 4. Whenever BCP is invoked the resulting edge is inserted into an initially empty priority queue[2]. The minimum distance is then selected either from the sorted list or from the priority queue. The advantage is that the priority queue is typically very small compared to the number of well-separated pairs. Therefore, the overhead of using a priority queue is minimized. After the N well-separated pairs have been processed, the procedure described above is repeated for the remaining pairs.

5. EXPERIMENTAL RESULTS

First we describe the problem instances considered (Section 5.1). Then we give some computational results on the WSPD (Section 5.2). Finally, in Sections 5.3, 5.4 and 5.5 we present comprehensive experimental results on the performance of the implemented GMST algorithms.

5.1 Input Distributions

The selection of input distributions were inspired by Bentley's paper [Bentley 1990] on k-d trees. He considered 11 different point distributions, which included the uniform distribution (`uni`), the normal distribution (`normal`), several 1-d distributions and some clustered distributions; all distributions were generalized to higher dimensions.

In addition to these 11 distributions, we added two classes of problem instances. The first class, denoted `ladder`, is a "bad" instance for our algorithm. Place $n/2$ evenly spaced points on the line segment from $(-n, -1)$ to $(-n, 1)$; place another $n/2$ evenly spaced points on the line segment from $(n, -1)$ to $(n, 1)$. Now rotate all the points 45° around $(0, 0)$. It is easy to verify that both GeoMST and GeoMST2 — using either ball or box marginal distances — require quadratic time for this particular instance (actually, quadratic time is spent on only a *single* BCP computation).

[2] After experimenting with several possibilities, we used the skip-list [Pugh 1990] data structure to implement the priority queue. It turned out to be significantly faster than both Fibonacci and pairing heaps.

The last class of instances, denoted `tsplib`, is a selection of 33 instances from TSPLIB [Reinelt 1991]. All these instances have at least 1000 cities and are given by 2-dimensional point coordinates (e.g., of type `CEIL_2D` or `EUC_2D`). The smallest was `dsj1000` and the largest was `pla85900`.

Experiments with all these point distributions under the Euclidean metric in dimensions 2, 3, 4 and 5 have been made (whenever applicable). For dimension 2 the largest instances had 10^6 points; for dimensions 3, 4 and 5 the largest instances had 10^5, $10^{4.5}$ and $10^{4.5}$ points, respectively. For problem instance classes given by random distributions the results were averaged over 50 runs, each with a different seed for the random number generator.

The experiments were also repeated for the L_1 and L_∞ metrics.

5.2 Computing the WSPD

In Section 3.2 we defined the *ball* and *box* (marginal) distances. The reduction in the number of pairs using the box distance is approximately 15%, 27%, 35% and 40% for dimensions 2, 3, 4 and 5, respectively (for uniform distributions). The smaller number of pairs is obtained at the cost of increased running time for computing the well-separated pairs, since the box distance computation is more than twice as expensive as the ball distance computation. However, for the MST algorithm the reduced number of pairs speeds up the processing of the pairs resulting in a decrease in the total running time. Also, the box distance function has the advantage that the square-root function can be avoided completely when computing the decomposition; this again makes it easier to achieve numerical robustness (if the input coordinates are rationals, only rational numbers will be compared).

Of the 13 point distributions, the uniform distribution produced among the largest number of pairs (as conjectured by Callahan [1995]). The running time for constructing the WSPD becomes increasingly dependent on the number of pairs for higher dimensions. Recall that while the construction of the fair split tree takes $O(n \log n)$, the generation of the pairs is proportional to the number of pairs — which is $O(n)$. In 2-d the fair split tree construction and pair generation take approximately the same time. In 5-d the fair split tree construction takes less than 1/40 of the time that the generation requires. Therefore, the overall time is essentially proportional to the number of pairs in higher dimensions.

5.3 Effect of Dimensions

Figure 6 presents total normalized CPU-times for all the implemented MST algorithms for dimensions 2, 3, 4, and 5. Normalized here means that the actual running times (in seconds) were divided by $10^{-6} n \log_{10} n$. This gives a clearer picture of any deviation from $O(n \log n)$ behavior. Results for **Kruskal** are omitted for the 2-d and 3-d cases since they were not competitive.

In 2-d all algorithms show $O(n \log n)$ running time behavior. The (normalized) running time curves have a small positive slope; we conjecture that this can be attributed to an increasing number of cache misses due to larger memory requirements. Both **GeoMST** and **GeoMST2** performed well when compared with the Delaunay triangulation algorithms. **Triangle** is the fastest algorithm while **Qhull** and **LEDA** perform worse than both **GeoMST** and **GeoMST2**. The results show that using the Delaunay triangulation is only faster if a well-engineered Delaunay

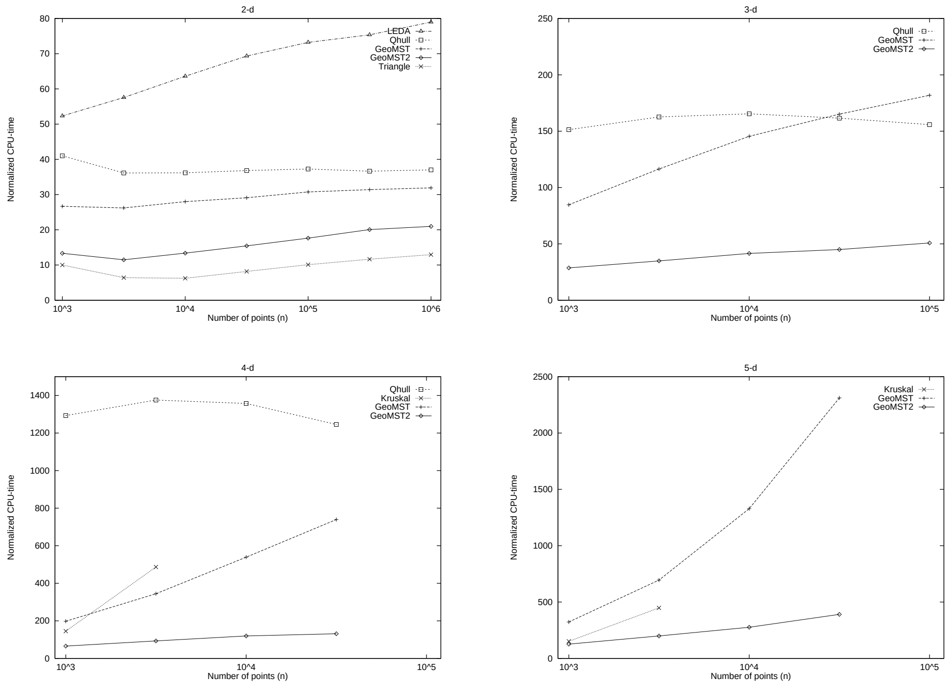

Fig. 6. Normalized CPU-times for all algorithms in 2-d (top-left), 3-d (top-right), 4-d (bottom-left) and 5-d (bottom-right); normalized here means that the actual running times (in seconds) were divided by $10^{-6} n \log_{10} n$. All point sets drawn from the uniform distribution in a (hyper)cube. CPU-times for all other distributions (except `ladder`) had similar asymptotic behavior.

triangulation algorithm is used. In fact, using the so called *triangular* data structure instead of the *quad-edge* data structure in the Delaunay triangulation algorithm — as is done in TRIANGLE — speeds up the algorithm by a factor of two [Shewchuk 1996]; without this improvement, the performance of **GeoMST2** and **Triangle** would be on par with each other for the uniform distribution.

In 3-d, the new algorithms were only compared to **Qhull**. The running time for **GeoMST** is clearly $\Omega(n \log n)$, while both **Qhull** and **GeoMST2** exhibit $O(n \log n)$ behavior. **GeoMST2** is more than three times faster than **Qhull**. In 4-d, both **GeoMST** and **GeoMST2** are clearly faster than **Qhull**. For $n \leq 10^{4.5}$, **Kruskal** is faster than **Qhull** and **GeoMST**; however, its huge memory requirements made it impossible for us to run it on larger instances. In 5-d the **Qhull** triangulation algorithm became very unstable and extremely time consuming, and we have therefore not included any results for it. Otherwise, the results are similar to 4-d. Again, **GeoMST2** shows close to $O(n \log n)$ behavior.

5.4 Effect of Input Distributions and Metrics

With the exception of the `ladder` instance, **GeoMST** and **GeoMST2** performed well on all point distributions considered (the running time increase over the uniform distribution was at most 20%). In Figure 7 we present normalized CPU-times for $n = 10^4$. **GeoMST2** consistently outperformed **GeoMST** for all point distributions. Finally, it should be noted that the normalized CPU-time for the `ladder` instance was approximately 1000, or about 30 times the normalized CPU-times for all other distributions.

The results for the L_1 and L_∞ metrics were qualitatively similar to those obtained for the L_2 metric; they are omitted from this version.

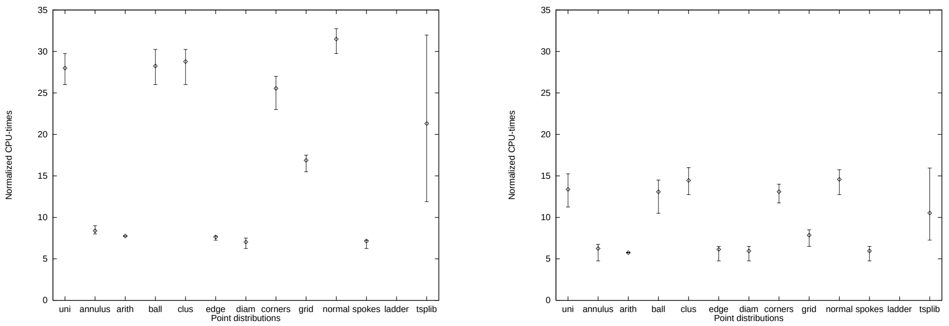

Fig. 7. Normalized CPU-times, including spread, for **GeoMST** (left) and **GeoMST2** (right) in 2-d ($n = 10^4$) for all point distributions.

5.5 **GeoMST** vs. **GeoMST2**

From a worst-case point of view, the bottleneck of the MST algorithms based on WSPDs is the BCP computations for each well-separated pair. **GeoMST2** avoids most of these computations by merging the BCP calls with the Kruskal step of the MST algorithm. The performance gains are shown in Figure 8, which gives the total number of BCP calls (including recursive calls) divided by n for **GeoMST** and **GeoMST2**. This number is proportional to the CPU-time spent per point on BCP calls.

In the **GeoMST** algorithm the number of BCP calls increases both with the number of points and with the dimension. As predicted by Theorem 1 the number of BCP calls for **GeoMST2** is linear in n. Our experiments indicate that not only is the number of calls linear in n for a given dimension, but the number of calls is also independent of dimension. However, the running time does increase with the dimension since each BCP call becomes more time consuming (distances are more expensive to compute).

6. CONCLUSIONS

We propose an improved GMST algorithm based on WSPDs. Using expected-time analysis and extensive experimental results, we argue that the resulting implementation performs well for points in higher dimensional space. We conclude that the WSPD is a practical and sophisticated tool for higher dimensional space.

REFERENCES

Agarwal, P. K., Edelsbrunner, H., Schwarzkopf, O., and Welzl, E. 1991. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom. 6*, 5, 407–422.

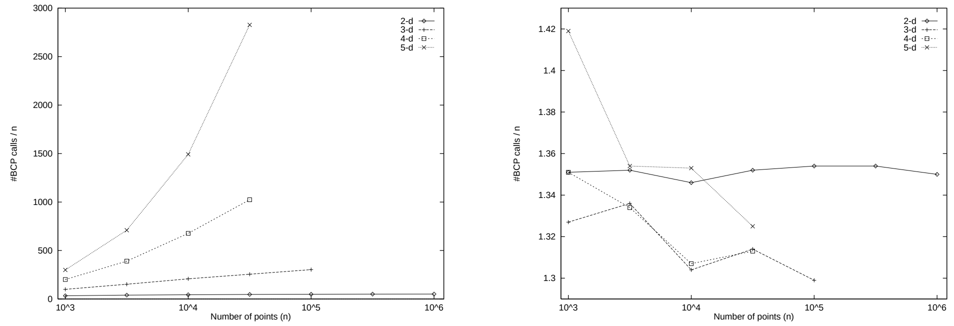

Fig. 8. Number of BCP calls (including recursive calls) divided by n in **GeoMST** (left) and **GeoMST2** (right) for the uniform distribution. Note the significantly different scales on the y-axis.

Barber, B. C., Dobkin, D. P., and Huhdanpaa, H. 1996. The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software 22*, 4 (Dec.), 469–483.

Bentley, J. L. 1990. *K*-d trees for semidynamic point sets. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.* (1990), pp. 187–197.

Bentley, J. L. 1992. Fast algorithms for geometric traveling salesman problems. *ORSA J. Comput. 4*, 4, 387–411.

Bentley, J. L. and Friedman, J. H. 1978. Fast algorithms for constructing minimal spanning trees in coordinate spaces. *IEEE Trans. Comput. C-27*, 97–105.

Bentley, J. L., Weide, B. W., and Yao, A. C. 1980. Optimal expected-time algorthms for closest point problems. *ACM Transactions on Mathematical Software 6*, 4 (Dec.), 563–580.

Callahan, P. B. 1995. *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. Ph. D. thesis, Dept. Comput. Sci., Johns Hopkins University, Baltimore, Maryland.

Callahan, P. B. and Kosaraju, S. R. 1993. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms* (1993), pp. 291–300.

Callahan, P. B. and Kosaraju, S. R. 1995. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM 42*, 67–90.

Clarkson, K. L. 1989. An algorithm for geometric minimum spanning trees requiring nearly linear expected time. *Algorithmica 4*, 461–469.

Dwyer, R. A. 1991. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete Comput. Geom. 6*, 343–367.

Eppstein, D. 1999. Spanning trees and spanners. In J.-R. Sack and J. Urrutia Eds., *Handbook of Computational Geometry*. Elsevier Science Publishers B.V. North-Holland.

Krznaric, D., Levcopoulos, C., and Nilsson, B. 1997. Minimum spanning trees in d-dimensions. In *Proc. 5th European Sympos. on Algorithms* (1997), pp. 341–349.

Mehlhorn, K. and Näher, S. 1995. LEDA: a platform for combinatorial and geometric computing. *Commun. ACM 38*, 96–102.

Pugh, W. 1990. Skip lists: a probabilistic alternative to balanced trees. *Comm. ACM 33*, 6, 668–676.

Reinelt, G. 1991. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing 3*, 4, 376–384.

Shewchuk, J. R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In M. C. Lin and D. Manocha Eds., *Applied Computational Geometry: Towards Geometric Engineering*, Volume 1148 of *Lecture Notes in Computer Science*, pp. 203–222. Springer-Verlag.