Computational Geometry: Theory and Applications 1 (1992) 189–201
Elsevier

Farthest neighbors, maximum spanning trees and related problems in higher dimensions*

Pankaj K. Agarwal

Department of Computer Science, Duke University, Durham, NC 27706, USA

Jiří Matoušek

Department of Applied Mathematics, Charles University, Malostranske nám. 25, 11800 Praha 1, Czechoslovakia

Subhash Suri

Bellcore, Morristown, NJ 07960, USA

Communicated by Frances Yao
Submitted 15 April 1991
Accepted 18 October 1991

Abstract

Agarwal, P.K., J. Matoušek and S. Suri, Farthest neighbors, maximum spanning trees and related problems in higher dimensions, Computational Geometry: Theory and Applications 1 (1992) 189–201.

We present a randomized algorithm of expected time complexity

$$O(m^{2/3}n^{2/3}\log^{4/3} m + m\log^2 m + n\log^2 n)$$

for computing bi-chromatic farthest neighbors between n red points and m blue points in $\mathbb{E}^3$. The algorithm can also be used to compute all farthest neighbors or external farthest neighbors of n points in $\mathbb{E}^3$ in $O(n^{4/3}\log^{4/3} n)$ expected time. Using these procedures as building blocks, we can compute a Euclidean maximum spanning tree or a minimum-diameter two-partition of n points in $\mathbb{E}^3$ in $O(n^{4/3}\log^{7/3} n)$ expected time. The previous best bound for these problems was $O(n^{3/2}\log^{1/2} n)$. Our algorithms can be extended to higher dimensions.

We also propose fast and simple approximation algorithms for these problems. These approximation algorithms produce solutions that approximate the true value with a relative accuracy ε and run in time $O(n\varepsilon^{(1-k)/2}\log n)$ or $O(n\varepsilon^{(1-k)/2}\log^2 n)$ in k-dimensional space.

1. Introduction

In this paper, we consider problems in the Euclidean space $\mathbb{E}^k$, for some fixed dimension k. We present efficient algorithms for the following problems.

* A preliminary version of this paper appeared in *Proc. of 2nd Workshop on Algorithms and Data Structures, LNCS 519, Springer Verlag, August 1991.*

(1) All farthest neighbors. Given a set S of n points in $\mathbb{E}^k$, for each point $p \in S$, compute a point $q \in S$ such that $d(p, q) \geq d(p, q')$, for all $q' \in S$; q is called a *farthest neighbor* of p.

(2) Bi-chromatic farthest neighbors. Given a set R of n 'red' points and another set B of m 'blue' points in $\mathbb{E}^k$, for each point $r \in R$, compute a point $b \in B$ such that $d(r, b) \geq d(r, b')$ for all $b' \in B$; b is called a *bi-chromatic farthest neighbor* of r.

(3) External farthest neighbors. Given a set S of n points in $\mathbb{E}^k$ and its partition $S_1, S_2, \ldots, S_m$ into m subsets, for each $p \in S$, if $p \in S_i$ then compute a point $q \in S - S_i$ such that $d(p, q) \geq d(p, q')$, for all $q' \in S - S_i$; q is called an *external farthest neighbor* of p.

(4) Euclidean maximum spanning tree (EMXST). Given a set S of n points in $\mathbb{E}^k$, compute a spanning tree of S whose edges have the maximum total length among all spanning trees, where the length of an edge is the Euclidean distance between its endpoints.

(5) Minimum diameter two-partition. Given a set S of n points in $\mathbb{E}^k$, partition S into two sets such that the larger of the two diameters is a minimized.

Computing neighbors (nearest, farthest, or some intermediate ones) is a fundamental problem in Computational Geometry, with many applications. While the all nearest neighbors problem in $\mathbb{E}^k$ can be solved in (optimal) $O(n \log n)$ time [22], no algorithm of similar efficiency is known for computing the all farthest neighbors, for $k \geq 3$. Many applications, however, require a more general formulation of this problem, such as the *bi-chromatic* and the *external* neighbors problems, where for points in one set we want to find the neighbors in some other set. The bi-chromatic and external neighbors problems also have resisted optimal algorithms for dimensions $k \geq 3$. A recent result of Agarwal et al. [1] computes a 'diametral pair' in $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ time[1] for n points in k dimensions, but their algorithms does not seem to generalize to the all farthest (or, nearest) neighbors problem. The best algorithm currently known for computing all farthest neighbors (bi-chromatic or otherwise) in three dimensions runs in time $O(n^{3/2} \log^{1/2} n)$. In this paper, we present improved, albeit randomized, algorithms for the all farthest neighbors and some related problems in three and higher dimensions.

Our main result is a randomized algorithm for computing bi-chromatic all farthest (or nearest) neighbors in expected time $O(m^{2/3}n^{2/3} \log^{4/3} m + m \log^2 m + n \log^2 n)$ in $\mathbb{E}^3$. The algorithm can be used to compute in $O(n^{4/3} \log^{4/3} n)$ expected time the farthest neighbors and the external farthest neighbors for n points in $\mathbb{E}^3$.

[1] Throughout this paper, γ will denote an arbitrarily small but positive fixed constant.

Our algorithms can be extended to k dimensions, $k \geq 4$, for solving the bi-chromatic farthest neighbors problem in expected time $O((mn)^{1-1/(\lceil k/2 \rceil + 1 + \gamma)} + (m+n)^{1+\gamma})$, and for solving all farthest neighbors and external farthest neighbors problems in expected time $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$.

A Euclidean maximum spanning tree of n points can be computed by repeatedly invoking the (external) farthest neighbors algorithm. Thus, our result for computing the farthest neighbors leads to an improved algorithm for maximum spanning tree as well. The algorithm has expected time complexity $O(n^{4/3} \log^{7/3} n)$ in $\mathbb{E}^3$ and $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ in $\mathbb{E}^k$, for $k \geq 4$. A variant of our algorithm can also compute a Euclidean minimum spanning tree of a set of points within the same time bound. The new algorithm is somewhat simpler than the one in Agarwal et al. [1], although unlike their algorithm, we do not know how to make ours deterministic without significantly increasing the running time. Euclidean maximum spanning tree are useful for determining a minimum diameter two-partitioning of a point set, and hence we obtain a similar result for the latter problem as well.

We also propose a simple approximation scheme that, given a set S of n points and a real parameter $0 < \varepsilon < 1$, computes, for each point $p \in S$, a point $q' \in S$ that is a farthest neighbor of p with a relative error $\leq \varepsilon$. In particular, if the farthest neighbor of p is $q \in S$, then q' satisfies the following inequality

$$\frac{d(p, q')}{d(p, q)} \geq 1 - \varepsilon.$$

The algorithm runs in time $O(n\varepsilon^{(1-k)/2})$ in k-dimensional space, and does not use any data structure beyond a linked list. The method also works for computing an ε-approximation of the diameter and a maximum spanning tree.

Egecioglu and Kalantari [12] recently proposed an iterative algorithm for approximating the diameter of a k-dimensional set of points. Each iteration of their algorithm runs in time $O(n)$, and the first iteration produces a distance Δ such that the diameter is between Δ and $\sqrt{3}\Delta$. The interesting aspect of this approximation is that the bound is independent of the dimension. However, in the worst case, their algorithm cannot guarantee an approximation factor better than $\sqrt{3}$, no matter how many iterations are allowed. By contrast, our algorithm achieves ε-approximation for arbitrarily small $\varepsilon > 0$.

Our paper is organized as follows. In Section 2 we describe algorithms to compute farthest neighbors. For simplicity, we describe our algorithms in three dimensions, and sketch the details for extension to higher dimensions. In Section 3, we describe our algorithm for computing an EMXST of a set of points. Section 4 presents our approximation algorithms.

2. Computing farthest neighbors

In this section, we develop a randomized algorithm for computing bi-chromatic farthest neighbors, and then apply it to compute all farthest neighbors and

external farthest neighbors. The structure of our algorithm follows a pattern that is fairly standard for random-sampling based geometric algorithms (e.g., see [1]). First we give an efficient algorithm for the 'unbalanced' case, where the blue points greatly outnumber the red points. Then we use a randomized divide-and-conquer approach to convert a 'balanced' problem into several instances of the unbalanced problem.

2.1. *An algorithm for many blue and few red points*

Recall that we are given a set R of n red points and another set B of m blue points, and for each $r \in R$, we want to compute its farthest neighbor in B. We randomly choose a subset $B' \subset B$ of $\lfloor m/4 \rfloor$ blue points. We partition the set R of red points into subsets R_1, R_2 of sizes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, respectively, and then recursively solve the problems for R_1, B' and for R_2, B'. This gives us, for each $r_i \in R$, its farthest neighbor $b_i' \in B'$. Let S_i denote the (closed) ball of radius $d(r_i, b_i')$ centered on r_i, and let $\bar{S}_i$ denote the (open) exterior of S_i. Since b_i' is a farthest neighbor of r_i in B', $B' \subset S_i$, for all i, and the farthest neighbor of r_i in B is either b_i' or lies in $\bar{S}_i$. We are therefore interested in the points of B that fall outside the common intersection $I = \bigcap_{i=1}^{n} S_i$.

Imagine that the points of B are sorted in the order of nondecreasing their distances from a point $r_i \in R$. Then the points of $B \cap \bar{S}_i$ are those lying beyond all points of the sample B', and it is easy to see that the expected number of such points is $O(1)$, for every r_i. By the additivity of expectation, we get that the expected size of $B'' = B \setminus I$ is at most $O(n)$.

The above discussion implies that we only need to compute a farthest neighbor of each $r \in R$ in the set B''. We do this in a straight-forward manner by computing $d(r, b)$, for all $b \in B''$.

The correctness of the algorithm is obvious. As for the running time of our algorithm, I can be computed in $O(n^2)$ time by mapping the balls to half-spaces in $\mathbb{E}^4$ and computing their common intersection [9, 20]. In $O(n^2)$ time and space we can preprocess I into a data structure that supports $O(\log^2 n)$ point-in-common-intersection queries, see Aurenhammer [2]. Thus, the set B'' can be computed in total time $O(n^2 + m \log^2 n)$. Finally, the farthest neighbor of each $r \in R$ in B'' can be computed in expected $O(|R| \cdot |B''|) = O(n^2)$ time. If $T(n, m)$ denotes the maximum expected time complexity of the algorithm for the input sets of cardinalities n and m, then we have the following recurrence relation:

$$T(n, m) = O(n^2 + m \log^2 n) + 2T(\lceil n/2 \rceil, \lfloor m/4 \rfloor),$$

which solves to $T(n, m) = O(n^2 + m \log^2 n)$.

Now we make a technical modification of our algorithm, which will give the running time in a more convenient form. Namely, we partition R into $\lceil n/s \rceil$ subsets, each of size at most $s = \sqrt{m} \log m$ and, for each i, compute the farthest neighbors of R_i in B separately using the algorithm described above. The

expected running time of the whole procedure is now

$$O(\lceil n/s \rceil (s^2 + m \log^2 s)) = O(n\sqrt{m} \log m + m \log^2 m).$$

Theorem 2.1. *Let R and B be two sets of points in $\mathbb{E}^3$ with n and m points, respectively. There is an $O(n\sqrt{m} \log m + m \log^2 m)$ expected time randomized algorithm that computes, for each $r \in R$, its farthest neighbor in B.*

The same approach can be extended to higher dimensions. The only change that we need to make is to use a different point-location data structure for computing the set B''. In particular, in $\mathbb{E}^k$ we map the spheres S_i, for $1 \leq i \leq n$, to the half-spaces

$$U_i\colon\ z \leq 2\alpha_1 x_1 + \cdots + 2\alpha_k x_k + (r^2 - \alpha_1^2 - \cdots - \alpha_k^2)$$

in $\mathbb{E}^{k+1}$, where $(\alpha_1, \ldots, \alpha_k)$ and r are the center and the radius of S_i, respectively. A point $p = (p_1, \ldots, p_k)$ lies outside S_i if and only if the point $(p_1, \ldots, p_k, p_1^2 + \cdots + p_k^2)$ does not lie in U_i. Thus, $p \in I$ if and only if $p \notin \bigcap_{1 \leq i \leq n} U_i$. We preprocess the polytope formed by the intersection of half-spaces $U_i (1 \leq i \leq n)$ for point location queries using a variant of Clarkson's algorithm for point location in hyperplane arrangements [5]. This structure answers a point location query in $O(\log n)$, and requires $O(n^{\lceil k/2 \rceil + \gamma})$ time and space for preprocessing. Therefore B'' can be computed in time $O(n^{\lceil k/2 \rceil + \gamma} + m \log n)$. Analyzing in the same way as above, one can show the total running time to be $O(nm^{1 - 1/(\lceil k/2 \rceil + \gamma)} + m \log n)$.

Theorem 2.2. *Let R and B be two sets of points in $\mathbb{E}^k$ with n and m points, respectively. Then, for each point $r \in R$, its farthest neighbor in B can be computed by a randomized algorithm in expected time $O(nm^{1 - 1/(\lceil k/2 \rceil + \gamma)} + m \log n)$.*

Remark. The above approach can be used to solve *bi-chromatic all nearest neighbor* problem in the same expected time. The only change that we need to make in the above algorithm is replace $I = \bigcap_{i=1}^{n} S_i$ with $I = \bigcap_{i=1}^{n} \bar{S}_i$. We leave it to the reader to fill in the details.

2.2. An algorithm for the balanced case

We now describe our final algorithm, which is significantly faster than the previous algorithm when n and m are of the same order. This algorithm is very similar to that of Agarwal et al. [1], therefore we describe only the main idea.

$$\mathcal{D}(p)\colon\ x_4 = 2b_1 x_1 + 2b_2 x_2 + 2b_3 x_3 - (b_1^2 + b_2^2 + b_3^2).$$

We identify $\mathbb{E}^3$ with the plane $x_4 = -\infty$, so the red points lie on this plane. A crucial property of the above transform is that if we treat the hyperplane $\mathcal{D}(b)$ as a 3-variate linear function

$$h_b(x_1, x_2, x_3) = 2b_1 x_1 + 2b_2 x_2 + 2p_3 x_3 - (b_1^2 + b_2^2 + b_3^2),$$

then b is a farthest neighbor of a red point $r = (r_1, r_2, r_3)$ if and only if

$$h_b(r_1, r_2, r_3) = \min_{b' \in B} h_{b'}(r_1, r_2, r_3).$$

The problem of computing the farthest neighbor of r thus reduces to finding the blue hyperplane that lies immediately above r (in x_4 direction). This property is well-known and the reader is referred to [9–10] for details. Our second algorithm for the bi-chromatic all-farthest neighbors problem can now be outlined as follows.

1. Transform B to the set of hyperplanes $\mathcal{D}(B)$ in $\mathbb{E}^4$ and identify $\mathbb{E}^3$ with the hyperplane $x_4 = -\infty$ in $\mathbb{E}^4$.
2. Randomly choose a subset $H \subset \mathcal{D}(B)$ of size t, for some parameter t; all subsets of H of size t are chosen with equal probability.
3. Let h^- be the half space lying below the hyperplane h. Compute the common intersection $P = \bigcap_{h \in H} h^-$; P is a convex polyhedron unbounded from below.
4. Triangulate P as follows. A ridge (2-face) of P is triangulated by connecting its highest vertex (in x_4 direction) to all other vertices. Each triangle is thus bounded by two edges incident to the highest vertex of the 2-face and a third edge which is an original edge of the face. Similarly, a facet of P is triangulated by connecting its highest vertex to all vertices, edges and triangles in the boundary of the triangulated facet. Each tetrahedron is now incident to its highest vertex and is bounded by a triangle not incident to it. Finally the interior of P is decomposed by extending each of its 2-faces τ vertically downwards, that is, erecting a vertical prism whose top face is τ and which is unbounded from below.[2] Let $\{\Delta_1, \ldots, \Delta_s\}$ be the set of cells in P. Each cell Δ_i is bounded by vertical facets and by a top facet, which is a portion of a hyperplane of H.
5. Given a cell Δ_i, let $R_i \subseteq R$ denote the set of red points contained in Δ_i.
6. Let $B_i \subseteq B$ denote the set of points b such that $\mathcal{D}(B)$ either contains the top facet of Δ_i or intersects the interior of Δ_i.
7. Solve each subproblem (R_i, B_i) separately using the previous algorithm. That is, for points in R_i, find their farthest neighbors in B_i.

Lemma 2.3. *For a suitable choice of the parameter t, the expected running time of the above algorithm is* $O(m^{2/3}n^{2/3}\log^{4/3} m + m\log^2 m + n\log^2 n)$.

Proof. If $T(n, m)$ denotes the maximum running time of the algorithm for the input sets R and B of cardinalities n and m, respectively, then we have the following recurrence:

$$T(n, m) = \sum_{i=1}^{s} T(n_i, m_i) + M(n, m),$$

[2] The unbounded faces of P can be triangulated by using homogenous coordinates and adding points at infinity (called *ideal points*). See [4, 5] for details.

where $M(n, m)$ is the time needed to break up the problem of size n and m into s subproblems of sizes n_i and m_i.

We solve each subproblem directly using Theorem 2.1. Since P can be triangulated in $O(t^2)$ time into $O(t^2)$ cells, the sets R_i can be computed in $O(t^2 \log t + n \log^2 t)$ using the point location algorithm of Preparata and Tamassia [19], and the sets B_i can be computed in $O(\sum_{i=1}^{O(t^2)} m_i)$ time by tracing the planes of $B - H$ through the triangulated polytope, the expected running time of the algorithm is

$$E[T(n, m)] = O\left(E\left[\sum_{i=1}^{O(t^2)} (n_i\sqrt{m_i} \log m + m_i \log^2 m)\right]\right) + O(t^2 \log t + n \log^2 t).$$

The theory of random sampling [1, 7, 11] implies that

$$E\left[\sum_{i=1}^{O(t^2)} m_i\right] = O(mt), \qquad E\left[\sum_{i=1}^{O(t^2)} n_i\sqrt{m_i}\right] = O\left(n\sqrt{\frac{m}{t}}\right).$$

Therefore, by setting $t = \lceil n^{2/3}/(m^{1/3} \log^{2/3} m)\rceil$, we obtain

$$E[T(n, m)] = O(m^{2/3}n^{2/3} \log^{4/3} m + m \log^2 m + n \log^2 n). \qquad \square$$

We can state the main result of this section as follows.

Theorem 2.4. *Let R and B be two sets of points in $\mathbb{E}^3$ with n and m points, respectively. There is an $O(m^{2/3}n^{2/3} \log^{4/3} m + m \log^2 m + n \log^2 n)$ expected time randomized algorithm that computes, for each $r \in R$, its farthest neighbor in B.*

As in the previous section, the same approach works also in higher dimensions. The only difference is that we choose t to be some suitable constant, and in Step 7 solve the subproblem using Theorem 2.1 only if $n_i < m_i^{1/(\lceil k/2\rceil + \gamma)} \log m_i$. Otherwise, we solve the subproblem recursively; see [1] for details. The expected running time of the algorithm now becomes

$$O((mn)^{1-1/(\lceil k/2\rceil + 1 + \gamma)} + (m + n)\log m).$$

We thus have the following.

Theorem 2.5. *Let R and B be two sets of points in $\mathbb{E}^k$ with n and m points, respectively. Then there is an $[O((mn)^{1-1/(\lceil k/2\rceil + 1 + \gamma)} + (m + n)\log m)]$ expected time randomized algorithm that computes, for each $r \in R$, its farthest neighbor in B.*

Remark. Our second algorithm can also be modified for computing bi-chromatic nearest neighbors problem within the same time bound. In this case, we map $\mathbb{E}^3$ to the hyperplane $x_4 = +\infty$ and, for every red point r, we determine the blue hyperplane lying immediately below r.

By setting $B = R$, we can compute, for each point $r \in R$, its farthest neighbor within R using Theorems 2.4 and 2.5. We have the following result.

Corollary 2.6. *Given a set S of n points in $\mathbb{E}^k$, one can compute, for each point $p \in S$, its farthest neighbor in S in randomized expected time $O(n^{4/3} \log^{4/3} n)$ for $k = 3$, and in time $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ for $k > 3$.*

2.3. Computing external farthest neighbors

We are given a set S of n points and its partition $\Pi = \{S_1, S_2, \ldots, S_m\}$ into m subsets. For each $p \in S$, we want to compute its external farthest neighbor.

We will use the standard divide-and-conquer technique. If $m = 1$, there is nothing to do and we stop, otherwise we divide Π into two subsets Π', Π'' as follows. Let $S' = \bigcup_{S_i \in \Pi'} S_i$ and $S'' = \bigcup_{S_i \in \Pi''} S_i$. If there is an S_i such that $|S_i| \geq 2n/3$, we set $\Pi' = \{S_i\}$ and $\Pi'' = \Pi - \Pi'$. Otherwise, we divide Π into two subsets Π', Π'', such that $|S'|$, $|S| \leq 2n/3$. Such a partition can be computed in $O(n)$ time. We recursively solve the subproblems (S', Π'), (S'', Π), that is, for each $p \in S'$ (resp. $p \in S''$) compute its external farthest neighbor q' with respect to Π' (resp. Π'').

Finally, for each $p \in S'$ (resp. $p \in S''$), we compute its bi-chromatic farthest neighbor q in S'' (resp. S') using Theorem 2.4. We return q, if $d(p, q) \geq d(p, q')$, and q' otherwise.

The correctness of the algorithm is evident, and the running time $T(M, n)$ obeys the following recurrence, assuming that $M(n)$ denotes the time complexity of the bi-chromatic farthest neighbors problem:

$$T(m, n) \leq T(m_1, n_1) + T(m - m_1, n - n_1) + M(n),$$

where $T(1, n) = O(1)$, $|S'| = n_1$, $|\Pi'| = m_1$, and either $m_1 = 1$ or n_1, $n - n_1 \leq 2n/3$. This recurrence solves to $M(n)$, because $M(n) \geq n^{1+\delta}$, for some fixed $\delta > 0$. We thus obtain the following result.

Theorem 2.7. *Given a set S of n points in $\mathbb{E}^k$ and its partition $S_1, \ldots, S_m$ into m subsets, we can compute, for each $p \in S$, its external farthest neighbor in randomized expected time $O(n^{4/3} \log^{4/3} n)$ for $k = 3$, and in time $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ for $k > 3$.*

3. Euclidean maximum spanning tree and clustering

In this section, we give a randomized algorithm for computing a Euclidean maximum spanning tree (EMXST) of n points in $\mathbb{E}^k$. EMXSTs find applications in clustering. For instance, given an EMXST of n points, we can find a *minimum diameter* 2-clustering of the points in $O(n)$ additional time, see Monma and Suri [17]. In particular, they show that any two coloring of an EMXST of S gives a minimum diameter 2-clustering of S.

A maximum spanning tree can be computed by repeatedly invoking the external farthest neighbors algorithm. The algorithm is based on the following well-known property.

Let $\{S_1, S_2\}$ be an arbitrary partition of the point set S, and suppose that the pair p_1^*, p_2^* maximizes the distance over all pairs of points $p_1 \in S_1$ and $p_2 \in S_2$. Then there exists a maximum spanning tree of S containing the edge (p_1^*, p_2^*). It is used in the following well-known algorithm for computing a maximum spanning tree, see [14].

Algorithm EMXST

Initialization: Make each point $p \in S$ a component consisting of a singleton vertex p.

1. If the number of components in the spanning forest is more than one, then execute Steps (2) through (5). Otherwise, output the component and stop.
2. Let $V_1, V_2, \ldots, V_m$ be the components of the spanning forest. Solve the external farthest neighbors problem for S with respect to $V_1, \ldots, V_m$.
3. For each component V_i, let p_i^*, q_i^* be a pair of points that maximizes the distance over all pairs $p_i \in V_i$ and $q_i \in S - V_i$.
4. Pick the pairs (p_i^*, q_i^*) one by one in the non-increasing order of their distances, and add the edge $p_i^* q_i^*$ to the spanning forest if it does not create a cycle.
5. Go to Step (1).

end Algorithm EMXST.

Since each time Steps 1–4 are executed, the number of components reduces by at least half, and since Steps 1–4 require $O(n^{4/3} \log^{4/3} n)$ time in $\mathbb{E}^3$ and $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ in $\mathbb{E}^k$. we obtain the following result.

Theorem 3.1. *Given a set S of n points in $\mathbb{E}^k$, its Euclidean maximum spanning tree and a minimum-diameter two-partition can be computed by a randomized algorithm in expected time $O(n^{4/3} \log^{7/3} n)$ for $k = 3$, and in time $O(n^{2-2/(\lceil k/2 \rceil + 1 + \gamma)})$ for $k > 3$.*

4. Approximation algorithms

We give a simple approximation scheme for computing bi-chromatic all-farthest neighbors. By the results of the previous sections, this leads to similar approximation algorithms also for the all-farthest neighbors, the external farthest neighbors, the maximum spanning tree, and the minimum-diameter two-partitioning.

Let R and B be two sets of points in $\mathbb{E}^k$, where $|R| = n$ and $|B| = m$, and let ε be a real parameter. We want to approximate the farthest neighbors of R in B. Specifically, let $b \in B$ be a farthest neighbor of $r \in R$, and let $b' \in B$ be another point. We say that b' is an *ε-approximate farthest neighbor* of r if

$$\frac{d(r, b')}{d(r, b)} \geq 1 - \varepsilon.$$

To illustrate the method, we first consider the problem in two dimensions.

We form a net Φ of uniformly spaced s directions by partitioning the *unit sphere* (circle in two dimensions) into s equal parts. In particular, the ith member of Φ, say, ϕ_i is directed from the origin to $(\cos 2\pi i/s, \sin 2\pi i/s)$.

For direction ϕ_i, let $l(\phi_i)$ denote the line whose normal vector is ϕ_i, and let $B(\phi_i)$ denote a point of B that maximizes the linear function

$$l(\phi_i)\colon x \cos\frac{2\pi i}{s} + y \sin\frac{2\pi i}{s} = 1.$$

(That is, $B(\phi_i)$ is an extreme point of B in direction ϕ.) At a high level, our approximation algorithm consists of the following steps.

Algorithm APPROXIMATION
1. Compute $B(\Phi) = \{B(\phi) \mid \phi \in \Phi\}$.
2. For each $r \in R$, find its farthest neighbor in $B(\Phi)$.
end Algorithm

The approximation potential of the above algorithm is established by the following lemma.

Lemma 4.1. *For $s \geq \pi/\sqrt{2\varepsilon}$, the algorithm* APPROXIMATION *computes ε-approximate farthest neighbors of R in B.*

Proof. Consider a point $r \in R$, and assume that $b \in B$ is a (true) farthest neighbor of r. Since $b \in B$ is a farthest neighbor of r, all points of B lie in the disk of radius $d(r, b)$ and center r. There is a direction $\phi \in \Phi$ that makes an angle of no more than π/s with the direction determined by the pair (r, b). Consider the line $l(\phi)$ and the support point $B(\phi)$. We claim that $B(\phi)$ is an ε-approximate farthest neighbor of r. Indeed, let δ be the distance from r to the line $l(\phi)$. Clearly, $d(r, b) \geq d(r, B(\phi)) \geq \delta$. Hence,

$$\frac{d(r, B(\phi))}{d(r, b)} \geq \frac{\delta}{d(r, b)} \geq \cos\frac{\pi}{s} = 1 - 2\sin^2\frac{\pi}{2s} \geq 1 - \frac{\pi^2}{2s^2} \geq 1 - \varepsilon.$$

Since Step 2 of algorithm APPROXIMATION returns a farthest neighbor among $B(\Phi)$, $d(r, B(\phi))$ is a lower bound on our approximation, which proves the lemma. □

Theorem 4.2. *There is an $O((n+m)\varepsilon^{-1/2})$ time algorithm for computing ε-approximation of bi-chromatic farthest neighbors in $\mathbb{E}^2$.*

Proof. A straightforward implementation of the algorithm APPROXIMATION leads to $O((n+m)\varepsilon^{-1/2})$ bound: the set $B(\Phi)$ can be computed at the cost of $O(m)$ per direction, and a farthest neighbor of $r \in R$ can be found by checking each of the $O(\varepsilon^{-1/2})$ candidates in the set $B(\Phi)$. □

Remark. Let us remark that at a slight increase in the conceptual complexity, we can improve the time complexity of the algorithm to $O((n+m)\log 1/\varepsilon)$, as follows. Instead of spending linear time per direction, we do a divide-and-conquer on the set of directions. This approach computes the set $B(\Phi)$ in $O(m \log 1/\varepsilon)$ time. Then, to find farthest neighbors, we compute the farthest-point Voronoi diagram of $B(\Phi)$ and locate points of R in it by point-location techniques. The latter step takes

$$O\left(n \log \frac{1}{\varepsilon} + \frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon}\right) \text{ time.}$$

The algorithm for higher dimensions is essentially the same as in two dimensions: we form a dense net Φ of direction vectors, by dividing the direction sphere into a grid, compute the support point $B(\phi)$ for each direction $\phi \in \Phi$, and report, for each point $r \in R$, its farthest neighbor in $B(\Phi)$. (The support point $B(\phi)$ is the one that maximizes the linear function $x \cdot \phi = 1$). The following lemma is straightforward.

Lemma 4.3. *There exists a set Φ of unit vectors in $\mathbb{E}^k$, where $|\Phi| = O(\varepsilon^{-(k-1)/2})$ such that for an arbitrary unit vector α, we can find a vector $\phi \in \Phi$ satisfying* $\arccos(\alpha, \phi) \leqslant \varepsilon^{1/2}$. *The set Φ can be found in $O(|\Phi|)$ time for any fixed dimension k.*

To show that the farthest neighbor reported by the algorithm APPROXIMATION, indeed, is an ε-approximation even for higher dimensions, we proceed as follows.

Consider a point $r \in R$ and its (true) farthest neighbor $b \in B$. Choose a direction $\phi \in \Phi$ such that the angle between ϕ and the vector $\vec{rb}$ is at most $\sqrt{\varepsilon}$, as guaranteed by Lemma 4.3. Let H be the two-dimensional linear space spanned by the vectors ϕ and $\vec{rb}$. Observe now that we are in the two-dimensional case, and the analysis of Lemma 4.1 can be applied. We thus have the following theorem.

Theorem 4.4. *Given two sets of points R and B in $\mathbb{E}^k$, with $|R| = n$ and $|B| = m$, and an $\varepsilon > 0$, there is an $O((n+m)\varepsilon^{(1-k)/2})$ time algorithm for computing an ε-approximation of bi-chromatic all-farthest neighbors between R and B.*

Of course, the diameter and all-farthest neighbors of a set of n points can also be found within the same bounds. We can also compute an ε-approximation of external farthest neighbors by repeatedly finding the ε-approximate bi-chromatic farthest neighbors, instead of the true neighbors, in the algorithm of Section 2.3.

Finally, by using ε-approximate external farthest neighbors in Step 2 of the algorithm EMXST, we can also find an ε-approximate maximum spanning tree. The justification for the last claim lies in the following observation. Let $\{S_1, S_2\}$ be a partition of S, let $d(p_1^*, p_2^*) = \max\{d(p_1, p_2) \mid p_1 \in S_1, p_2 \in S_2\}$, and let $p' \in S_1, p'' \in S_2$ be such that $d(p', p'')/d(p_1^*, p_2^*) \geq 1 - \varepsilon$. Then, there exists an ε-approximate maximum spanning tree of S containing the edge (p', p''). Since our algorithm always adds the ε-approximation of an external farthest edge between two components of a partition, the preceding observation implies that the resulting spanning tree is an ε-approximation.

Theorem 4.5. *Given a set S of n points in $\mathbb{E}^k$ and an $\varepsilon > 0$, we can compute ε-approximate external farthest neighbors of S in time $O(n\varepsilon^{(1-k)/2} \log n)$. An ε-approximation of a maximum spanning tree or a minimum diameter two-partitioning of S can be found in time $O(n\varepsilon^{(1-k)/2} \log^2 n)$.*

5. Conclusion

We have considered the problem of computing bi-chromatic farthest or nearest neighbors in k-dimensional space. These problems arise quite naturally in computational geometry applications. Despite their fundamental nature, optimal algorithms are still not known for solving them in dimensions greater than two. A traditional method for solving these problems is the so-called *locus* approach: build a Voronoi diagram for the set of points B, and then find a neighbor for each point $r \in R$ through point-location. When combined with a batching technique, this scheme yields a (slightly) subquadratic algorithm in any fixed dimension k and for any reasonable metric. Our contribution in this paper has been to improve the exponent significantly by incorporating random-sampling, which allows us to break the problem into several small subproblems. A key difference between the batching-based locus approach and the random-sampling approach is that while, in the former method, only one of the sets is partitioned among the subproblems, in the latter, both sets are simultaneously subdivided.

We suspect that the correct time complexity for these problems is close to linear. Achieving that goal, however, seems quite difficult at this point, at least for high dimensions. At the same time, no lower bound better than $\Omega(n \log n)$ is known. Recently Agarwal and Matoušek have developed a deterministic algorithm for computing bi-chromatic farthest neighbors of n red points and m blue points in $\mathbb{E}^k$ in time

$$O((mn)^{1-\frac{1}{\lfloor k/2 \rfloor + 1 + \gamma}} + (m+n)^{1+\gamma})$$

[2], which yields deterministic algorithm for all the problems considered here with the same asymptotic time complexity.

The topic of approximation algorithms is still mostly unexplored. Our result shows that linear or near-linear algorithms are indeed possible if one is willing to settle for an ε-approximation. The 'constant factor' of our algorithm grows exponentially with dimension in $1/\varepsilon$. It may be possible to reduce this constant to something like $(\log 1/\varepsilon)^k$, which would be interesting.

References

[1] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf and E. Welzl, Euclidean minimum spanning trees and bichromatic closest pairs, Discrete Comput. Geom. 6 (1991) 407–422.

[2] P.K. Agarwal and J. Matoušek, Ray shooting queries and parametric search, Tech. Rept. CS-1991-21, Dept. Computer Science, Duke University, 1991.

[3] F. Aurenhammer, Improved algorithms for discs and balls using power diagrams, J. Algorithms 9 (1988) 151–161.

[4] B. Chazelle, How to search in history, Inform. and Control 64 (1985) 77–99.

[5] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, Combinatorica 10 (1990) 229–249.

[6] K. Clarkson, Fast expected-time and approximate algorithms for geometric minimum spanning tree, Proc. 16th Annual Symposium on Theory of Computing (1984) 342–348.

[7] K. Clarkson, A randomized algorithm for closest-point queries, SIAM J. Comput. 17 (1988) 830–847.

[8] K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II, Discrete Comput. Geom. 4 (1989) 387–421.

[9] H. Edelsbrunner, An acyclicity theorem for cell complexes in d dimensions, Combinatorica 10 (1990) 251–260.

[10] H. Edelsbrunner, Algorithms in Combinatorial Geometry (Springer, Berlin, 1987).

[11] H. Edelsbrunner and R. Seidel, Voronoi diagrams and arrangements, Discrete Comput. Geom. 1 (1985) 25–44.

[12] H. Edelsbrunner and M. Sharir, A hyperplane incidence problem with applications to counting distances. Proc. of International Symposium on Algorithms, LNCS 450 (Springer, Berlin, 1990).

[13] O. Egecioglu and B. Kalantari, Approximating the diameter of a set of points in the Euclidean space, Technical report, Rutgers University, 1989.

[14] H.N. Gabow and R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. System Sci. 30 (1985) 209–221.

[15] R.L. Graham and P. Hell, On the history of minimum spanning tree problem, Ann. Hist. Comput. 7 (1985) 43–57.

[16] D. Haussler and E. Welzl, ϵ-nets and simplex range queries, Discrete Comput. Geom. 2 (1987) 127–151.

[17] C. Monma, M. Paterson, S. Suri and F.F. Yao, Computing Euclidean maximum spanning trees, Algorithmica 5 (1990) 407–419.

[18] C. Monma and S. Suri, Partitioning points and graphs to minimize the maximum or the sum of diameters, Proceedings of 6th International Conference on the Theory and Applications of Graphs (Wiley, New York, 1989).

[19] F.P. Preparata and M.I. Shamos, Computational Geometry (Springer, Berlin, 1985).

[20] F. Preparata and R. Tamassia, Efficient spatial point location, Workshop on Algorithms and Data Structures, LNCS 382 (Springer, Berlin, 1989) 3–11.

[21] R. Seidel, A convex hull algorithm optimal for point sets in even dimensions, University of British Columbia, Vancouver, 1981.

[22] P. Vaidya, Minimum spanning trees in k-dimensional space, SIAM J. Comput. 17 (1988) 572–582.

[23] P. Vaidya, An $O(n \log n)$ algorithm for the all-nearest-neighbor problem, Discrete Comput. Geom. 4 (1989) 101–115.

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