Better ε-Dependencies for Offline Approximate Nearest Neighbor Search, Euclidean Minimum Spanning Trees, and ε-Kernels

Sunil Arya*
Department of Computer Sci. & Eng.
HKUST, Hong Kong
arya@cse.ust.hk

Timothy M. Chan†
Cheriton School of Computer Science
University of Waterloo, Canada
tmchan@uwaterloo.ca

ABSTRACT

Recently, Arya, da Fonseca, and Mount [STOC 2011, SODA 2012] made notable progress in improving the ε-dependencies in the space/query-time tradeoffs for $(1+\varepsilon)$-factor approximate nearest neighbor search in fixed-dimensional Euclidean spaces. However, ε-dependencies in the preprocessing time were not considered, and so their data structures cannot be used to derive faster algorithms for offline proximity problems. Known algorithms for many such problems, including approximate bichromatic closest pair (BCP) and approximate Euclidean minimum spanning trees (EMST), typically have factors near $(1/\varepsilon)^{d/2\pm O(1)}$ in the running time when the dimension d is a constant.

We describe a technique that breaks the $(1/\varepsilon)^{d/2}$ barrier and yields new results for many well-known proximity problems, including:

- an $O((1/\varepsilon)^{d/3+O(1)}n)$-time randomized algorithm for approximate BCP,
- an $O((1/\varepsilon)^{d/3+O(1)}n\log n)$-time algorithm for approximate EMST, and
- an $O(n\log n+(1/\varepsilon)^{d/3+O(1)}n)$-time algorithm to answer n approximate nearest neighbor queries on n points.

Using additional bit-packing tricks, we can shave off the $\log n$ factor for EMST, and even move most of the ε-factors to a sublinear term.

The improvement arises from a new time bound for exact "discrete Voronoi diagrams", which were previously used in the construction of ε-kernels (or extent-based coresets), a well-known tool for another class of fundamental problems. This connection leads to more results, including:

- a streaming algorithm to maintain an approximate diameter in $O((1/\varepsilon)^{d/3+O(1)})$ time per point using $O((1/\varepsilon)^{d/2+O(1)})$ space, and
- a streaming algorithm to maintain an ε-kernel in $O((1/\varepsilon)^{d/4+O(1)})$ time per point using $O((1/\varepsilon)^{d/2+O(1)})$ space.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical problems and computations*

General Terms

Algorithms, Theory

Keywords

ε-dependencies, bichromatic closest pair, minimum spanning tree, nearest neighbor, coresets

1. INTRODUCTION

1.1 Proximity Problems

Proximity problems constitute a fundamental class of geometric problems, involving distances between points. They have been extensively studied in the literature, both because of their theoretical importance and because they have numerous applications in areas such as information retrieval, pattern recognition, and machine learning. Typical proximity problems include the following:

Bichromatic Closest Pair (BCP): Given a set of n red and blue points in d-dimensional Euclidean space, find the closest red-blue pair.

Offline Nearest Neighbor Search: Given a set of n red and blue points in d-dimensional Euclidean space, find the closest red point to each blue point.

Euclidean Minimum Spanning Tree (EMST): Given a set of n points in d-dimensional Euclidean space, construct the spanning tree that minimizes the sum of the edge lengths.

*This author's work was supported by the Research Grants Council, Hong Kong, China under project number 610012.

†This author's work was supported by NSERC, and was done while visiting the Department of Computer Science and Engineering, Hong Kong University of Science and Technology.

SoCG'14, June 8–11, 2014, Kyoto, Japan.
 ACM 978-1-4503-2594-3/14/06 ...$15.00.

Previous studies strongly suggest that many proximity problems are hard to solve exactly, except in very low dimensions. In particular, for all the above problems, the best algorithms take nearly quadratic time as the dimension increases and are thus scarcely better than brute-force search [1]. This has prompted researchers to study approximation algorithms, specifically, algorithms that can approximate the optimal solution to within a factor $1+\varepsilon$ for any given $\varepsilon > 0$.

Throughout this paper, we will focus on the setting when the dimension d is a constant, as is typical in traditional computational geometry, but the approximation parameter ε is nonconstant. (This is orthogonal to another major line of research [24, 26], which treats d as nonconstant but ε as constant; interesting recent developments have also been made along that direction [4].) Many important applications reside in spaces of relatively low dimensions (for example, ranging from 3 to 20). Previous work has already established efficient approximation algorithms that run in time linear or near-linear in n for constant d, but there are hidden factors in ε that grow rapidly as we demand greater accuracy. These factors can dominate the running time in practice. Our goal is to minimize such ε-dependencies while maintaining linear or near-linear dependencies on n.

Virtually all previous work has ε-dependencies that are of the order of $O^*((1/\varepsilon)^d)$ or $O^*((1/\varepsilon)^{d/2})$.[1] For approximate BCP, Khuller and Matias [27] in 1995 gave an $O^*((1/\varepsilon)^d n)$-time randomized algorithm, while Chan [15] at SoCG 1997 presented an $O^*((1/\varepsilon)^{d/2} n \log n)$-time algorithm. For approximate EMST, Vaidya [31] in 1988 gave an $O^*((1/\varepsilon)^d n \log n)$-time algorithm, while Callahan and Kosaraju [13] at SODA 1993 presented an $O^*(n \log n + (1/\varepsilon)^{d/2} n)$-time algorithm, which has remained unimproved for two decades. (Czumaj et al. [21, 22] have given algorithms for approximating the EMST weight but not for computing an approximate EMST.)

Data structures for *approximate nearest neighbor search* can be used to solve the proximity problems listed above. Since Arya et al. [8] presented an $O(n)$-space data structure with $O^*(\log n + (1/\varepsilon)^d)$ query time, much attention has been devoted to improving the ε-dependencies. For example, following Clarkson [20], Chan [15] described an $O^*((1/\varepsilon)^{d/2} n)$-space data structure with $O((1/\varepsilon)^{d/2} \log n)$ query time. Arya, Malamatos, and Mount [7] at STOC 2002 obtained an $O^*(n)$-space data structure with $O^*(\log n + (1/\varepsilon)^{d/2})$ query time. A tradeoff between space and time is possible; for example, to balance the ε-dependencies of the two bounds, we can get a structure with $O^*((1/\varepsilon)^{d/3} n)$ space and $O^*(\log n + (1/\varepsilon)^{d/3})$ query time. Further exciting developments have been reported recently: at STOC 2011 and SODA 2012, Arya, da Fonseca, and Mount [5, 6] obtained a better space-time tradeoff curve; for example, we can now get a structure with $O^*((1/\varepsilon)^{d/4} n)$ space and $O^*(\log n + (1/\varepsilon)^{d/4})$ query time.

Unfortunately all these newer data structures for approximate nearest neighbor search have no immediate implications to offline proximity problems, because their preprocessing time still have ε-dependencies $O^*((1/\varepsilon)^{d/2})$ or worse. The barrier for the offline problems has remained at $(1/\varepsilon)^{d/2}$.

We reduce the $O^*((1/\varepsilon)^{d/2})$ factors to $O^*((1/\varepsilon)^{d/3})$ for all three problems:

- We give a randomized algorithm for approximate BCP that runs in $O^*((1/\varepsilon)^{d/3} n)$ time, improving the previous $O^*((1/\varepsilon)^{d/2} n \log n)$ algorithm [15]. With more complicated bit-packing tricks, the time bound can even be reduced to $O^*(n + (1/\varepsilon)^{d/3} n / \log^{5/3} n)$.
- We give an algorithm for offline approximate nearest neighbor search that runs in $O^*(n \log n + (1/\varepsilon)^{d/3} n)$ time.
- We give an algorithm for approximate EMST that runs in $O^*((1/\varepsilon)^{d/3} n \log n)$ time. With more complicated bit-packing tricks, the time bound can be reduced to $O^*(n + (1/\varepsilon)^{d/3} n / \log^{2/3} n)$, strictly improving Callahan and Kosaraju's $O^*(n \log n + (1/\varepsilon)^{d/2} n)$ algorithm [13].

Our BCP and EMST algorithms (the versions without bit tricks) are relatively simple and do not require the recent advances by Arya, da Fonseca, and Mount [5, 6]. Instead it is based on a new result on the computation of exact *discrete Voronoi diagrams* and *discrete upper envelopes*, studied by Chan [18] in a different context (see the next subsection).

Our algorithm for offline approximate nearest neighbor search requires techniques from Arya, da Fonseca, and Mount's STOC 2011 paper [5], but not their subsequently improved SODA 2012 paper. More generally, we describe a method for (online) approximate nearest neighbor search that provides preprocessing-time/query-time rather than space/query-time tradeoffs, specifically achieving $O^*(n \log n + (1/\varepsilon)^{d/3} n)$ preprocessing time and $O^*(\log n + (1/\varepsilon)^{d/3})$ query time.

1.2 Extent Problems

Discrete Voronoi diagrams and discrete upper envelopes were actually proposed by Chan [18] to solve a different class of problems related to extent measures of a point set. Typical such problems include the following:

Diameter (or Farthest Pair): Given a set of n points in d-dimensional Euclidean space, determine the maximum distance between any pair of points.

Width: Given a set of n points in d-dimensional Euclidean space, find the smallest width of a slab that contains all the points.

For the diameter problem, there is a straightforward $O^*((1/\varepsilon)^{d/2} n)$-time approximation algorithm [3] that simply computes the pair of minimal and maximal points along each of $O((1/\varepsilon)^{(d-1)/2})$ uniformly spaced directions. Chan [17, 18] obtained alternative algorithms with $O^*(n + (1/\varepsilon)^d)$ running time, one of which was based on discrete upper envelopes.

For other extent-related problems, Agarwal, Har-Peled, and Varadarajan [2] have provided a general framework based on the notion of a kernel (or "coreset"). A subset Q of a point set P is said to be an *ε-kernel* of P if, for any slab that contains Q, a $(1+\varepsilon)$-expansion of the slab contains P. It turns out that there exist ε-kernels of size

[1] Throughout the paper, we will use the notation O^* to hide factors $1/\varepsilon^c$ (also written as E^c), where c is independent of the dimension d, typically not more than 1 or 2 (in some cases it may even be negative). This will help simplify the bounds and the presentation of the algorithms, and shift attention away from the less important issue of optimizing c.

$O((1/\varepsilon)^{(d-1)/2})$, and this bound is tight in the worst case. Intuitively, the convex hull of an ε-kernel provides a good approximation to the convex hull of the entire set of points, and this fact allows us to approximate many extent measures efficiently by solving the problems on a much smaller subset. (This includes certain problems, such as minimum-width annulus, which at first do not seem directly related to convex hulls, but can be transformed into one that does in a higher dimension, by linearization.)

Agarwal, Har-Peled, and Varadarajan's paper [2] originally gave an $O^*(n+(1/\varepsilon)^{3d/2})$-time algorithm to construct an ε-kernel of size $O((1/\varepsilon)^{(d-1)/2})$. Chan [18] improved the running time to $O^*(n+(1/\varepsilon)^d)$ by using his algorithm for discrete Voronoi diagrams.

The connection of nearest-neighbor-type problems with ε-kernels may sound unexpected at first, but the heart of Arya, da Fonseca, and Mount's recent result on approximate nearest neighbors [5, 6] lies in a problem called *approximate polytope membership queries*, and similarities between this and the ε-kernel problem are apparent. Both problems are about approximating a convex polytope, and algorithms for both problems exploit a well known construction by Dudley [23] and Bronshteyn and Ivanov [12]. Discrete Voronoi diagrams are what is required to turn this construction into efficient algorithms.

With our new result on discrete Voronoi diagrams, we can obtain an $O^*(n+(1/\varepsilon)^{d/2}\sqrt{n})$-time algorithm for the diameter problem and the ε-kernel problem. This is a modest improvement over the previous $O^*(n+(1/\varepsilon)^d)$ algorithm in the special case when n is smaller than $(1/\varepsilon)^d$. (This time bound does not apply to the width problem, however, since computing an approximate width of the ε-kernel still requires $O^*((1/\varepsilon)^d)$ time.)

We obtain a more interesting improvement in the context of *streaming* algorithms. In the (one-pass) streaming model, the points arrive one by one, we have limited amount of space at any time, and we need to maintain an approximate solution of the points seen so far. For example, consider the diameter problem. The straightforward algorithm [3] which maintains the minimal and maximal points along each of the $O((1/\varepsilon)^{(d-1)/2})$ directions can be implemented as a streaming algorithm which uses $O((1/\varepsilon)^{(d-1)/2})$ space and requires $O((1/\varepsilon)^{(d-1)/2})$ time to process a new point. While the space usage is optimal, the processing time might not be, but no better results have ever been published.

To get streaming algorithms for other extent-based problems, the main tool is again ε-kernels. Agarwal, Har-Peled, and Varadarajan [2] gave a streaming algorithm to maintain an ε-kernel of size $O((1/\varepsilon)^{(d-1)/2})$ with $O^*((1/\varepsilon)^{d/2}\log^d n)$ space and $O((1/\varepsilon)^d)$ time per point. Chan [18] gave another streaming algorithm to maintain an ε-kernel of size $O^*((1/\varepsilon)^d)$ with $O^*((1/\varepsilon)^d)$ space and $O(1)$ time per point. Zarrabi-Zadeh [32] gave yet another streaming algorithm to maintain an ε-kernel of size $O^*((1/\varepsilon)^{d/2})$ with $O^*((1/\varepsilon)^{d/2})$ space and $O^*((1/\varepsilon)^{d/2})$ time per point. While the space usage of the last algorithm is near optimal, again the processing time might not be.

Our new results are as follows:

- We give a streaming algorithm that uses $O^*((1/\varepsilon)^{d/2})$ space and maintains an approximate diameter in $O^*((1/\varepsilon)^{d/3})$ time per point.
- We give a streaming algorithm that uses $O((1/\varepsilon)^{(d-1)/2})$ space and requires $O^*((1/\varepsilon)^{d/4})$ time per point, and that maintains an ε-kernel of size $O((1/\varepsilon)^{(d-1)/2})$. For example, we can report an approximate width whenever requested in $O^*((1/\varepsilon)^d)$ additional time, by running an existing width algorithm on the ε-kernel [17].

The result for diameter requires techniques from Arya, da Fonseca, and Mount's STOC 2011 paper [5]. More generally, we describe how to support approximate farthest neighbor queries in $O^*((1/\varepsilon)^{d/3})$ time and point insertions in $O^*((1/\varepsilon)^{d/3})$ time. (See [9] for a previous work on approximate farthest neighbor queries.)

2. DISCRETE VORONOI DIAGRAMS AND DISCRETE UPPER ENVELOPES

In the *d-dimensional discrete Voronoi diagram (DVD)* problem, we are given a set P of n points from an $E\times\cdots\times E$ axis-parallel grid in $\mathbb{R}^d$. We are also given an $F\times\cdots\times F$ axis-parallel grid Ξ with $F\le E$. We want to find the exact nearest neighbor in P to each grid point in Ξ. The output thus consists of a matrix of F^d elements.

A slight generalization is the $(d+1)$-*dimensional discrete upper envelope (DUE)* problem: Here, we are also given a weight $w_p\in\mathbb{R}$ for each point $p\in P$. We want to find the point $p\in P$ that maximizes $p_1\xi_1+\cdots+p_d\xi_d+w_p$ for each $\xi\in\Xi$. (For a point p, its j-th coordinate is denoted by p_j.)

DVD is a special case of DUE by a standard lifting map, i.e., by setting $w_p=-(p_1^2+\cdots+p_d^2)/2$.

The trivial algorithm requires $O(F^d n)$ time. Chan [18, Corollary 2.2] (see also the work by Breu et al. [11]) obtained the following result by a simple algorithm that uses induction in the dimension.

THEOREM 2.1. *We can solve the DVD and DUE problem in $O^*(n+E^d)$ time.*

The running time is thus $O^*(n)$ when n is large enough to exceed the second term E^d. Unfortunately we do not know how to reduce the second term (other than by $O^*(1)$ factors) while keeping the first term close to $O(n)$. The trivial lower bound is $\Omega(n+F^d)$, but in our applications, $F\ll E$ (specifically $F=\sqrt{E}$), so a large gap remains.

Nevertheless, in the following corollary, we notice that an improvement to the upper bound is possible when $F\ll E$ and $n\ll E^d$. The idea is simple almost to the point of being obvious, but it turns out to be sufficient to give new results for our applications.

COROLLARY 2.2. *We can solve the DVD and DUE problem in $O^*(F^{d-k}(n+E^k))$ time for any given integer $k\le d$.*

PROOF. We simply decompose the d-dimensional grid Ξ into F^{d-k} k-dimensional subgrids and apply Theorem 2.1 to each subgrid. Note that when solving the problem for a k-dimensional subgrid of Ξ, e.g., in which the values of $\xi_{k+1},\ldots,\xi_d$ are fixed, we can make the point set P k-dimensional by projecting each point $p\in P$ to $(p_1,\ldots,p_k)$ and setting its new weight to $p_{k+1}\xi_{k+1}+\cdots+p_d\xi_d+w_p$.

(An alternative solution is to directly modify Chan's DUE algorithm [18] using a different base case, where we switch to the trivial algorithm when the dimension is reduced to $d-k$.) □

COROLLARY 2.3. *For $F = \sqrt{E}$, we can solve the DVD and DUE problem in $O^*(n + E^{d/2}\sqrt{n})$ time.*

PROOF. If $n \geq E^d$, we pick $k = d$ and the time bound is $O^*(n)$. Otherwise, we pick k so that E^k approximates n, to within a factor of E; then the bound is $O^*(E^{(d-k)/2}(n + E^k)) = O^*((E^{d/2}/\sqrt{n})n)$. □

3. BICHROMATIC CLOSEST PAIR

As an illustration of the power of Corollary 2.3, we first consider the approximate bichromatic closest pair (BCP) problem.

We begin by solving a special case when the red and the blue point sets are well-separated. A pair of point sets are *σ-well-separated* if they can be enclosed within two hypercubes of side length r, such that the distance between the two hypercubes is at least σr. Our solution of this special case uses DVDs and is inspired by a technique of Dudley [23] and Bronshteyn and Ivanov [12], originally intended for a different problem (polytope approximation).

LEMMA 3.1. *Given a σ-well-separated pair of a red point set X and a blue point set Y for a sufficiently large constant σ, we can solve the approximate BCP problem in $O^*(\min\{n + (1/\varepsilon)^{d/2}\sqrt{n},\, n^2\}) \leq O^*((1/\varepsilon)^{d/3}n)$ time.*

PROOF. First round the points in X and Y to an axis-parallel uniform grid of side length εr; this incurs an additive error of $O(\varepsilon r)$ only, and thus a multiplicative error of $1 + O(\varepsilon)$.

Find an axis-parallel hyperplane h between the two enclosing hypercubes that is of distance $\Omega(r)$ away from both balls. Consider the portion of this hyperplane h within the convex hull of the two hypercubes. Build a uniform grid Ξ inside this portion with side length $\sqrt{\varepsilon}r$. For each grid point $\xi \in \Xi$ (which we call a *helper*), find its nearest neighbor x_ξ in X and its nearest neighbor y_ξ in Y.

We claim that the closest pair (x_ξ, y_ξ) over all $\xi \in \Xi$ is a $(1 + O(\varepsilon))$-approximate BCP. To see this, suppose (x^*, y^*) is the actual closest red-blue pair. There exists a helper $\xi^* \in \Xi$ that is of distance $O(\sqrt{\varepsilon}r)$ from the intersection of the line x^*y^* with h. Let z^* be the closest point on the line x^*y^* to ξ^*. Consider the right triangle $x^*z^*\xi^*$. Since $\|\xi^*z^*\| \leq O(\sqrt{\varepsilon}r)$ and $\|x^*\xi^*\| \geq \Omega(r)$, we have $\sin\angle x^* \leq O(\sqrt{\varepsilon})$ and $\cos\angle x^* \geq 1 - O(\varepsilon)$. Thus, $\|x^*\xi^*\| \leq \|x^*z^*\|/(1-O(\varepsilon))$. By considering the right triangle $y^*z^*\xi^*$, we similarly have $\|\xi^*y^*\| \leq \|z^*y^*\|/(1 - O(\varepsilon))$. It follows that $\|x_{\xi^*}y_{\xi^*}\| \leq \|x_{\xi^*}\xi^*\| + \|\xi^*y_{\xi^*}\| \leq \|x^*\xi^*\| + \|\xi^*y^*\| \leq (\|x^*z^*\| + \|z^*y^*\|)/(1-O(\varepsilon)) = (1+O(\varepsilon))\|x^*y^*\|$.

The computation of all the x_ξ's and y_ξ's reduces precisely to the DVD problem, with $E = O(1/\varepsilon)$ and $F = O(1/\sqrt{\varepsilon})$. The $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n})$ time bound then follows immediately from Corollary 2.3. On the other hand, the $O(n^2)$ time bound follows from the trivial algorithm that checks all pairs of points.

The first bound is $O^*((1/\varepsilon)^{d/3}n)$ when $n \geq (1/\varepsilon)^{d/3}$; the second bound is $O((1/\varepsilon)^{d/3}n)$ when $n \leq (1/\varepsilon)^{d/3}$. □

We now solve the general BCP problem.

THEOREM 3.2. *We can solve the approximate BCP problem in $O^*((1/\varepsilon)^{d/3}n)$ expected time.*

PROOF. Consider the approximate decision problem: given a value ℓ, confirm that every red-blue pair has distance greater than ℓ, or report a red-blue pair of distance at most $(1 + \varepsilon)\ell$. The original problem can be reduced to the decision problem by known randomized techniques [16]. (Alternatively, we can run a randomized linear-time algorithm of Khuller and Matias [27] to compute a constant-factor approximation, then do binary search with $O(\log(1/\varepsilon))$ iterations.)

We solve the approximate decision problem by building a uniform grid of side length ℓ/σ for a sufficiently large constant σ. First identify all nonempty grid cells by hashing in linear expected time. Examine each pair of nonempty grid cells that are of distance at most ℓ. If this pair of grid cells have distance less than $(1 - 2\sqrt{d}/\sigma)\ell$ and contain points of both colors, then we have found a pair of points of distance at most ℓ. Otherwise, the two grid cells are $O(\sigma)$-well-separated, and we can apply Lemma 3.1 to the subsets X_i and Y_i of points inside the two cells (or more precisely, the red points in X_i and the blue points in Y_i, and vice versa).

Each point participates in a constant number of pairs (X_i, Y_i). Thus, $\sum_i(|X_i| + |Y_i|) = O(n)$. The total cost of applying Lemma 3.1 is therefore bounded by $O^*((1/\varepsilon)^{d/3}n)$. □

With a more careful analysis, one can verify that the above time bound is $O((1/\varepsilon)^{d/3+c}n)$ for a *negative* constant c (specifically, $c \approx -1/2, -7/12, -2/3$ for $d \equiv 0, 1, 2 \pmod 3$ respectively). This improves over the ε-dependencies of previous algorithms for all constant dimensions $d \geq 4$.

4. EUCLIDEAN MINIMUM SPANNING TREE

In this section, we consider the approximate Euclidean minimum spanning tree (EMST) problem. Reductions of EMST to BCP (in both the exact and approximate setting) have been given before [1, 13, 28]; for example, Callahan and Kosaraju's method [13] increases the running time by a logarithmic factor, while Krznaric, Levcopoulos, and Nilsson's method [28] increases the running time by a constant factor, assuming that the bound exceeds $n \log n$. We describe a solution that is simpler than these previous reductions and works specifically in the approximate setting.

A *σ-well-separated pair decomposition (or σ-WSPD)* of a point set P is a set of pairs of subsets of P, $\{(X_1, Y_1), \ldots, (X_m, Y_m)\}$, such that (i) X_i and Y_i are σ-well-separated for each i, and (ii) for any two distinct points $x, y \in P$, there exists a unique pair (X_i, Y_i) with $x \in X_i$ and $y \in Y_i$ or vice versa.

Callahan and Kosaraju [14] have shown that a σ-WSPD with $m = O(\sigma^d n)$ pairs exists and can be constructed in $O(n \log n + \sigma^d n)$ time. One description of the construction [19, 25] is based on quadtrees—each generated subset X_i or Y_i is the subset of all points of P inside some quadtree cell.

THEOREM 4.1. *We can solve the approximate EMST problem in $O^*((1/\varepsilon)^{d/3}n \log n)$ expected time.*

PROOF. Let D be a constant-factor approximation to the diameter (easily computable in linear time). Initially round points to a uniform grid with side length $\varepsilon D/n$; this incurs an additive error of $O(\varepsilon D)$ to the EMST weight, and thus a multiplicative error of $1 + O(\varepsilon)$, since the EMST weight is lower-bounded by $\Omega(D)$.

Now construct a σ-WSPD $\{(X_1, Y_1), \ldots, (X_m, Y_m)\}$ for a sufficiently large constant σ, with $m = O(n)$ and $O(n \log n)$ running time. Compute a $(1+\varepsilon)$-factor approximate BCP (x_i, y_i) between X_i and Y_i, and form a graph G with these m pairs (x_i, y_i) as the edges. Then return an MST T of G, by any textbook MST algorithm with $O(m \log n)$ running time or better.

Callahan and Kosaraju [13, Lemma 3.2] have already analyzed this approach and showed that the resulting tree T approximates the EMST by a factor of $1 + O(\varepsilon)$. However, they eventually abandoned this approach for a different one (which used a σ-WSPD with a nonconstant $\sigma = \Theta(\sqrt{1/\varepsilon})$); the reason was the lack of an efficient algorithm to solve these BCP subproblems, when their combined input size $\sum_i (|X_i| + |Y_i|)$ can be as large as quadratic in the worst case.

But notice that after the initial rounding step, the height of the quadtree is $O(\log(n/\varepsilon)) = O(\log n)$ (we may assume that $n \geq 1/\varepsilon$, for otherwise we can switch to an $O(n^2)$-time algorithm). It can then be checked that each point participates in at most $O(\log n)$ pairs (X_i, Y_i) in the WSPD construction. Thus, $\sum_i (|X_i| + |Y_i|) = O(n \log n)$. We can solve all the BCP subproblems by Lemma 3.1. The total cost is therefore bounded by $O^*((1/\varepsilon)^{d/3} n \log n)$. □

In the full version of the paper, we describe how to improve the running time of our BCP and EMST algorithms slightly to $O^*(n + (1/\varepsilon)^{d/3} n / \log^{5/3} n)$ and $O^*(n + (1/\varepsilon)^{d/3} n / \log^{2/3} n)$ respectively. The general idea is to use bit tricks to pack grid point sets and their DVDs into a smaller number of words, and thereby speed up the algorithm in Corollary 2.2 by polylog n factors.

5. OFFLINE NEAREST NEIGHBOR SEARCH

In this section, we present an algorithm for answering n offline approximate nearest neighbor queries on a set S of n points in $O^*(n \log n + (1/\varepsilon)^{d/3} n)$ time. We begin with some preliminaries. For any positive ε and query point $q \in \mathbb{R}^d$, we say that a point $p \in S$ is an *ε-nearest neighbor (ε-NN)* of q if the distance from p to q is at most $(1+\varepsilon)$ times the distance from q to its nearest neighbor in S. We assume that the set S has been scaled to lie within a ball of diameter $\varepsilon/16$ at the center of the unit hypercube $\mathbb{U} = [0,1]^d$. For queries outside $\mathbb{U}$, any point of S is an ε-NN, so it suffices to build a data structure for answering queries within $\mathbb{U}$. We will use the term *box* to refer to any axis-parallel hypercube. Define a quadtree box recursively as $\mathbb{U}$ or a box obtained by splitting any quadtree box into 2^d equal parts. For any box b and positive c, we let cb denote the box obtained by scaling b about its center by a factor of c.

Our algorithm is based on the *approximate Voronoi diagram (AVD)* construction from [7] and uses ideas similar to that in [5]. The AVD employs a quadtree-like structure called a *balanced box-decomposition (BBD) tree*, in which each node is associated with a region of space called a *cell*. The leaf cells of this tree form a subdivision of $\mathbb{U}$, and each leaf cell corresponds to the set theoretic difference of two quadtree boxes, an *outer box* and an *inner box*. Given a leaf cell Q in the tree, let b_Q denote its outer box. The following lemma summarizes the key properties of the AVD construction that are relevant to us. It follows easily from the proof of Lemmas 6.1 and 8.1 in [7].

LEMMA 5.1. *Let S be a set of n points in $\mathbb{R}^d$. It is possible to construct a BBD tree T with $O^*(n)$ nodes, where each leaf cell Q stores a* representative set *$R_Q \subset S$ satisfying the following properties:*

(i) For any point $q \in Q$, one of the points in R_Q is an ε-NN of q.

(ii) At most one point of R_Q is contained in the box $4b_Q$, and the remaining points of R_Q are contained in $cb_Q \setminus 4b_Q$, for some constant $c > 4$.

(iii) The total size of the representative sets R_Q over all the leaf cells of T is $O^*(n)$.

Moreover, it is possible to compute the tree T and the representative sets for all the leaf cells in time $O^*(n \log n)$, and the cell that contains a query point can be located in $O^*(\log n)$ time.

For any query point q, one can first locate the leaf cell that contains q and then scan the representative set R_Q to find an ε-NN of q. In order to achieve faster query times, Arya et al. [5] applied the standard lifting map and reduced the ε-NN problem for the leaf cells to approximate polytope membership queries [5]. We will use similar ideas, but in order to get fast preprocessing time, we will present a scheme that avoids the use of the lifting map. This enables us to take advantage of our new results on discrete Voronoi diagrams and enjoy significant efficiency gains.

Our main technical contribution in this section is the following lemma.

LEMMA 5.2. *Let b be a box. Let S be a set of n data points in $cb \setminus 4b$, where c is a constant. We can construct a data structure in $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n} + (1/\varepsilon)^{5d/8})$ time that allows us to answer ε-nearest neighbor queries for points in b in $O^*((1/\varepsilon)^{d/4})$ time. The space used by the data structure is $O^*((1/\varepsilon)^{d/2})$.*

PROOF. Let r denote the side length of box b. First, we round the points of S to an axis-parallel uniform grid of side length εr; this incurs an additive error of $O(\varepsilon r)$, and hence a multiplicative error of $1 + O(\varepsilon)$. In the remainder of the proof, we will use S to refer to this rounded point set.

We build a uniform grid Ξ with side length $\sqrt{\varepsilon} r$ on the boundary of the hypercube $2b$. For each grid point $\xi \in \Xi$ (which we call a *helper*), we find its nearest neighbor (denoted p_ξ) in S. This computation reduces to a DVD problem for $E = O(1/\varepsilon)$ and $F = O(1/\sqrt{\varepsilon})$ and, by Corollary 2.3, can be solved in $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n})$ time.

For each helper $\xi \in \Xi$, let C_ξ be the cone with apex at ξ, axis along vector $p_\xi \xi$, and angular radius $c'\sqrt{\varepsilon}$, where c' is a suitable constant. Consider any point $q \in b$. Let p be a nearest neighbor of q in S. We claim that there is a helper ξ^* which satisfies the following properties: (i) p_{ξ^*} is an $O(\varepsilon)$-NN of q, (ii) $\|q\xi^*\| + \|\xi^* p_{\xi^*}\| \leq (1 + O(\varepsilon))\|qp\|$, and (iii) $q \in C_{\xi^*}$.

To prove (i), observe there exists a helper $\xi^* \in \Xi$ at distance $O(\sqrt{\varepsilon} r)$ from the intersection of the line segment qp with the boundary of the box $2b$. Let y^* be the closest point on the line segment qp to ξ^*. Consider the right triangle $qy^*\xi^*$. Since $\|\xi^* y^*\| \leq O(\sqrt{\varepsilon} r)$ and $\|q\xi^*\| \geq \Omega(r)$, we have $\sin \angle q \leq O(\sqrt{\varepsilon})$ and $\cos \angle q \geq 1 - O(\varepsilon)$. Thus $\|q\xi^*\| \leq \|qy^*\|/(1 - O(\varepsilon))$. Similarly, by considering the

right triangle $py^*\xi^*$, we have $\|\xi^*p\| \leq \|y^*p\|/(1-O(\varepsilon))$. It follows that $\|q\xi^*\| + \|\xi^*p\| \leq (\|qy^*\| + \|y^*p\|)/(1-O(\varepsilon)) = (1+O(\varepsilon))\|qp\|$. Thus, $\|qp_{\xi^*}\| \leq \|q\xi^*\| + \|\xi^*p_{\xi^*}\| \leq \|q\xi^*\| + \|\xi^*p\| \leq (1+O(\varepsilon))\|qp\|$, which proves (i) and (ii).

To prove (iii), note that since p is a nearest neighbor of q, the ball B_q centered at q and having radius $\|qp\|$ contains no point of S in its interior. Let t denote the point where the ray emanating from q and passing through ξ^* intersects ∂B_q. Consider the cone with apex at ξ^*, axis along vector ξ^*t, and angular radius equal to $\angle p\xi^*t$. It follows from elementary geometry that p_{ξ^*} must lie within this cone (otherwise $\|\xi^*p_{\xi^*}\|$ would exceed $\|\xi^*p\|$). Since $\angle p\xi^*t$ is the exterior angle of triangle $q\xi^*p$, we have $\angle p\xi^*t = \theta + \phi = O(\sqrt{\varepsilon})$. It follows that the angle between vectors $p_{\xi^*}\xi^*$ and ξ^*q is $O(\sqrt{\varepsilon})$, from which property (iii) is immediate (for suitable constant c').

Let $R = \{p_\xi : \xi \in \Xi\}$. We have $|R| \leq |\Xi| = O(1/\varepsilon^{(d-1)/2})$. In light of (i), we can find the $O(\varepsilon)$-NN of any point $q \in b$ by scanning R and returning the closest point to q. However, this approach takes $O^*(1/\varepsilon^{d/2})$ time. In order to speed it up, we build a uniform grid Π of side length $\sqrt{\varepsilon}r$ inside b. Our strategy is to store a set of *representatives* $R_\pi \subseteq R$ with each box $\pi \in \Pi$ such that $|R_\pi| = O^*((1/\varepsilon)^{d/4})$ and, for any point q in π, one of the points of R_π is an $O(\varepsilon)$-NN. Furthermore, we will show that $\sum_\pi |R_\pi| = O^*((1/\varepsilon)^{d/2})$. This will enable us to reduce the query time to $O^*(1/\varepsilon^{d/4})$, without asymptotically increasing the space needed.

For each box $\pi \in \Pi$, we compute a helper set $H'_\pi = \{\xi : C_\xi \cap \pi \neq \emptyset\}$ and a representative set $R'_\pi = \{p_\xi : \xi \in H'_\pi\}$. Observe that for each helper ξ, the number of boxes π that intersect cone C_ξ is $O(1/\sqrt{\varepsilon})$ (since the angular radius of the cone is $O(\sqrt{\varepsilon})$ and the side length of the boxes is $\sqrt{\varepsilon}r$). Since the number of helpers is $O(1/\varepsilon^{(d-1)/2})$, it follows that $\sum_\pi |R'_\pi| \leq \sum_\pi |H'_\pi| = O((1/\varepsilon)^{d/2})$. We can also identify the boxes that intersect C_ξ in time proportional to their number. Thus, we can compute H'_π and R'_π for all π in $O^*((1/\varepsilon)^{d/2})$ time.

Let π be any box of Π and q be any point in π. Recall the helper ξ^* corresponding to point q described in properties (i)-iii). It follows from property (iii) that $\xi^* \in H'_\pi$. Note that R'_π includes p_{ξ^*} which, by property (i), is an $O(\varepsilon)$-NN of q. We say that a box $\pi \in \Pi$ is *good* if $|R'_\pi| \leq (1/\varepsilon)^{d/4}$, otherwise we say that it is *bad*. If a query point lies in a good box, then we can find its approximate nearest neighbor in $O^*((1/\varepsilon)^{d/4})$ time by examining the points of R'_π one by one and picking the closest among them. Thus, for each good box π, we let $R_\pi = R'_\pi$ be the final set of representatives. For the bad boxes, the size of the representative set R'_π is too large, but we will show that it is possible to prune its size to $O^*((1/\varepsilon)^{d/4})$.

Define the weight w_ξ of any helper ξ to be $\|\xi p_\xi\|$, that is, the distance between ξ and its nearest neighbor in S. For any point $q \in \mathbb{R}^d$ and any helper ξ, define the *weighted distance* from q to ξ to be $\|q\xi\| + w_\xi$. Consider any bad box π. We build an axis-parallel uniform grid Λ of side length $\varepsilon^{3/4}r$ on the boundary of the hypercube 2π. For each grid point $\lambda \in \Lambda$, we find its nearest neighbor (denoted h_λ) in H'_π, under the weighted distance measure. Let $H_\pi = \{h_\lambda : \lambda \in \Lambda\}$, and let $R_\pi = \{p_\xi : \xi \in H_\pi\}$. We have $|R_\pi| \leq |H_\pi| \leq |\Lambda| = O^*((1/\varepsilon)^{d/4})$. We claim that the time to compute H_π (and hence R_π) for all bad boxes $\pi \in \Pi$ is $O^*((1/\varepsilon)^{5d/8})$. Furthermore, for any point $q \in \pi$, there exists a point in R_π that is an $O(\varepsilon)$-NN of q.

To establish the bound on computation time, observe that the query points are the grid points of Λ and the data points (i.e., points of H'_π), lie on the grid Ξ. Therefore, the computation of H_π reduces to a DVD problem with additive weights, for $E = O(1/\sqrt{\varepsilon})$ and $F = O((1/\varepsilon)^{1/4})$ (it is not hard to show that all the coordinates and the weights can be rounded to be positive integers smaller than $O^*(1)$). By applying a generalization of Corollary 2.2 (see the full paper for details), this problem can be solved in $O^*(|H'_\pi| + (1/\varepsilon)^{d/4}\sqrt{|H'_\pi|}) = O^*(|H'_\pi| + (1/\varepsilon)^{d/4}|H'_\pi|/\sqrt{(1/\varepsilon)^{d/4}}) = O^*((1/\varepsilon)^{d/8}|H'_\pi|)$, where we have used the fact that $|H'_\pi| > (1/\varepsilon)^{d/4}$, since π is a bad box. Recalling that $\sum_\pi |H'_\pi| = O((1/\varepsilon)^{d/2})$, it follows that the time to compute H_π for all bad boxes π is $O^*((1/\varepsilon)^{5d/8})$, as desired.

It remains to show that for any point $q \in \pi$, there exists a point in R_π that is an $O(\varepsilon)$-NN of q. By our earlier remarks, there is a point $\xi^* \in H'_\pi$ that satisfies properties (i)-(iii). In particular, by property (ii), we have $\|q\xi^*\| + \|\xi^*p_{\xi^*}\| \leq (1+O(\varepsilon))\|qp\|$, where p is a nearest neighbor of q in S. We claim that there exists a point $\xi \in H_\pi$ that is an $O(\varepsilon)$-NN of q in H'_π, under the weighted distance measure. Assuming this claim, we can easily show that $p_\xi \in R_\pi$ would then be an $O(\varepsilon)$-NN of q in S. We have $\|qp_\xi\| \leq \|q\xi\| + \|\xi p_\xi\| = \|q\xi\| + w_\xi \leq (1+O(\varepsilon))(\|q\xi^*\| + w_{\xi^*}) = (1+O(\varepsilon))(\|q\xi^*\| + \|\xi^*p_{\xi^*}\|) \leq (1+O(\varepsilon))(1+O(\varepsilon))\|qp\| \leq (1+O(\varepsilon))\|qp\|$, as desired.

It remains only to establish the above claim. Let h be a nearest neighbor of q in H'_π, under the weighted distance measure. By our construction, there exists a helper $\lambda^* \in \Lambda$ at distance $O(\varepsilon^{3/4}r)$ from the intersection of the line segment qh with the boundary of the box 2π. Let z^* be the closest point on the line segment qh to λ^*. Consider the right triangle $qz^*\lambda^*$. Since $\|\lambda^*z^*| \leq O(\varepsilon^{3/4}r)$ and $\|q\lambda^*\| \geq \Omega(\sqrt{\varepsilon}r)$, we have $\sin\angle q \leq O(\varepsilon^{1/4})$ and $\cos\angle q \geq 1 - O(\sqrt{\varepsilon})$. Thus $\|q\lambda^*\| \leq \|qz^*\|/(1-O(\sqrt{\varepsilon})) = (1+O(\sqrt{\varepsilon}))\|qz^*\|$. Similarly, by considering the right triangle $hz^*\lambda^*$ and noting that $\|h\lambda^*\| \geq \Omega(r)$, we obtain $\|\lambda^*h\| \leq (1+O(\varepsilon^{3/2}))\|z^*h\|$. It follows that $\|q\lambda^*\| + \|\lambda^*h\| \leq (1+O(\sqrt{\varepsilon}))\|qz^*\| + (1+O(\varepsilon^{3/2}))\|z^*h\| = \|qh\| + O(\sqrt{\varepsilon})\|qz^*\| + O(\varepsilon^{3/2})\|z^*h\|$. Since $\|q\lambda^*\| \leq O(\sqrt{\varepsilon}r)$ and $\|\lambda^*z^*\| \leq O(\varepsilon^{3/4}r)$, we have $\|qz^*\| \leq \|q\lambda^*\| + \|\lambda^*z^*\| \leq O(\sqrt{\varepsilon}r) = O(\sqrt{\varepsilon})\|qh\|$. Thus $\|q\lambda^*\| + \|\lambda^*h\| \leq \|qh\| + O(\sqrt{\varepsilon})\cdot O(\sqrt{\varepsilon})\|qh\| + O(\varepsilon^{3/2})\|qh\| = (1+O(\varepsilon))\|qh\|$. Letting $\xi = h_{\lambda^*}$, we have $\|q\xi\| + w_\xi \leq \|q\lambda^*\| + \|\lambda^*\xi\| + w_\xi \leq \|q\lambda^*\| + \|\lambda^*h\| + w_h \leq (1+O(\varepsilon))\|qh\| + w_h \leq (1+O(\varepsilon))(\|qh\| + w_h)$. Thus $\xi \in H_\pi$ is an $O(\varepsilon)$-NN of q in H'_π, under the weighted distance measure, as desired.

Note that the preprocessing time is dominated by the $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n})$ time it takes to solve the DVD problem associated with box b, and the $O^*((1/\varepsilon)^{5d/8})$ time it takes to solve the DVD problems for all the bad boxes within b. The bound on the preprocessing time given in the statement of the lemma follows. □

We are now ready to apply the above lemma to obtain our main result on offline approximate nearest neighbor searching.

THEOREM 5.3. *Given n points in $\mathbb{R}^d$, we can build a data structure in $O^*(n\log n + (1/\varepsilon)^{d/3}n)$ time so that approximate nearest neighbor queries can be answered in $O^*(\log n + (1/\varepsilon)^{d/3})$ time.*

PROOF. First, we construct the BBD tree described in Lemma 5.1. For each leaf cell Q such that $|R_Q| > (1/\varepsilon)^{d/3}$, we build the data structure of Lemma 5.2 to answer ε-NN queries for query points in b_Q with respect to the points of R_Q (excluding the point of R_Q contained in $4b_Q$, if any). To answer queries, we first locate the leaf cell Q that contains the query point q. If $|R_Q| \leq (1/\varepsilon)^{d/3}$, we scan the set R_Q and return the closest point to q. Otherwise, we use the data structure of Lemma 5.2 to find the ε-NN of q in $O^*(1/\varepsilon)^{d/4})$ time. In either case, we can answer queries in at most $O^*(\log n + (1/\varepsilon)^{d/3})$ time.

The preprocessing time is dominated by the time it takes to construct the data structures of Lemma 5.2. For a given cell Q, the time taken is given by $O^*(|R_Q|+(1/\varepsilon)^{d/2}\sqrt{|R_Q|}+(1/\varepsilon)^{5d/8}) = O^*(|R_Q|+(1/\varepsilon)^{d/2}|R_Q|/\sqrt{(1/\varepsilon)^{d/3}}+(1/\varepsilon)^{5d/8})$ $= O^*((1/\varepsilon)^{d/3}|R_Q|)$, since we only build this data structure if $|R_Q| > (1/\varepsilon)^{d/3}$. Recalling that the total size of the representative sets R_Q over all cells Q is $O^*(n)$ and the time to construct the BBD tree is $O^*(n \log n)$, the desired bound on preprocessing time follows. □

COROLLARY 5.4. *We can answer n offline approximate nearest neighbor queries on a set of n points in $\mathbb{R}^d$ in $O^*(n \log n + (1/\varepsilon)^{d/3}n)$ time.*

6. STREAMING DIAMETER

Our streaming algorithm for diameter is based on a static data structure for answering approximate farthest neighbor queries. We present this data structure in Lemma 6.2. When we apply this in the streaming context, we will use the standard logarithmic method of Bentley and Saxe [10] to turn it into a semi-dynamic data structure (i.e., insertions are supported but not deletions).

Our data structure employs the following farthest-point analogue of Lemma 5.2. The proof of Lemma 6.1 is similar to the proof of Lemma 5.2 and has been omitted due to space limitations. For any positive ε and query point $q \in \mathbb{R}^d$, we say that a point $p \in S$ is an *ε-farthest neighbor (ε-FN)* of q if the distance from p to q is at least $(1-\varepsilon)$ times the distance from q to its farthest neighbor in S.

LEMMA 6.1. *Let b be a box. Let S be a set of n data points in $cb \setminus 4b$, where c is a constant. We can construct a data structure in $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n} + (1/\varepsilon)^{5d/8})$ time that allows us to answer ε-farthest neighbor queries for points in b in $O^*((1/\varepsilon)^{d/4})$ time. The space used by the data structure is $O^*((1/\varepsilon)^{d/2})$.*

LEMMA 6.2. *Given n points in $\mathbb{R}^d$, we can build a data structure in $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n} + (1/\varepsilon)^{5d/8})$ time so that approximate farthest neighbor queries can be answered in $O^*((1/\varepsilon)^{d/4})$ time. The space used by the data structure is $O^*((1/\varepsilon)^{d/2})$.*

PROOF. Let Δ be the diameter of the given set S of points. Let p be any point of S. In linear time, we can compute the distance Δ' of p from its farthest neighbor. It is easy to see that $\Delta' \geq \Delta/2$. Let c and c' be two sufficiently large constants. For $0 \leq i \leq \lceil \log(c/\varepsilon) \rceil$, let b_i be a ball of radius $r_i = 2^i\Delta'$ centered at p. We choose c sufficiently large such that for any query point outside the largest ball, any point of S is an ε-FN. Thus, it suffices to construct a data structure to handle the case when the query point lies inside the largest ball.

For $1 \leq i \leq \lceil \log(c/\varepsilon) \rceil$, generate all the boxes in a uniform grid of side length r_i/c', that intersect the annulus $b_i \setminus b_{i-1}$. Additionally, generate all boxes in a uniform grid of side length r_0/c' that intersect the ball b_0. The number of boxes generated for ball b_0 and for each annulus is $O(1)$ and so the total number of boxes is $O(\log(1/\varepsilon)) = O^*(1)$. It is easy to see that for sufficiently large constant c', each grid box b satisfies the property that its farthest neighbor is contained in the region $c''b \setminus 4b$, for a suitable constant c''. (We omit the straightforward details.) It follows that by building the data structure of Lemma 6.1 for each grid box, we can answer ε-FN queries. Given a query point q, we first identify a grid box that contains it and then use the corresponding data structure to find its ε-FN. The bound on query time, space, and preprocessing time follow from Lemma 6.1. □

Chan [17] presented an algorithm for approximating the diameter Δ of a static point set S that will be useful to us. The algorithm is based on constructing a set V of $O((1/\varepsilon)^{(d-1)/2})$ unit vectors in $\mathbb{R}^d$ such that the angle between any vector in $\mathbb{R}^d$ and one of these vectors is $O(\sqrt{\varepsilon})$. We say that a pair of points $p_v, p'_v \in S$ is ε-extreme for v if $p_v \cdot v \geq \max_{p \in S} p \cdot v - \varepsilon\Delta$ and $p'_v \cdot v \leq \min_{p \in S} p \cdot v + \varepsilon\Delta$. Chan showed that the problem of computing ε-extreme pairs for all vectors $v \in V$ can be reduced to a constant number of DUE problems for $E = O(1/\varepsilon)$ and $F = O(1/\sqrt{\varepsilon})$. Applying Corollary 2.3 implies part (a) of the following lemma. Part (b) is proved in [17] and the proof of part (c) is similar.

LEMMA 6.3. *Let S be a set of n points and Δ be the diameter of S. Let V be as described above.*

(a) We can find an ε-extreme pair of points for each vector in V in total time $O^*(n + (1/\varepsilon)^{d/2}\sqrt{n})$.

(b) Let p_x, p'_x be a farthest pair of ε-extreme points, that is, $\|p_x - p'_x\| = \max_{v \in V} \|p_v - p'_v\|$. Then $\|p_x - p'_x\|$ approximates the diameter Δ to within a factor of $1 + O(\varepsilon)$.

(c) Let $S' \subseteq S$ denote the set of all the ε-extreme points found in part (a). For any point $q \in \mathbb{R}^d$, the farthest neighbor of q in S' is an $O(\varepsilon)$-farthest neighbor of q in S.

We are now ready to present our main result on streaming diameter.

THEOREM 6.4. *There is a streaming algorithm that can maintain an approximate diameter in $O^*((1/\varepsilon)^{d/3})$ time per point using $O^*((1/\varepsilon)^{d/2})$ space.*

PROOF. (Sketch) The algorithm runs in phases, where each phase processes $(1/\varepsilon)^{d/2}$ points. Without loss of generality, we assume that $N = (1/\varepsilon)^{d/2}$ is a power of two. At the beginning of each phase, we ensure that the algorithm has the following information. Let S_c be the set of points seen until now (i.e., cumulatively, in all previous phases). For each vector in V, the algorithm maintains an ε-extreme pair of points with respect to S_c. Let $S'_c \subseteq S_c$ denote the set of all these ε-extreme points. Let q be any point in $\mathbb{R}^d$. By Lemma 6.3(c), there exists a point of S'_c that is an $O(\varepsilon)$-FN of q with respect to S_c. The algorithm also constructs the data structure of Lemma 6.2 to answer $O(\varepsilon)$-FN queries with respect to S'_c in $O^*((1/\varepsilon)^{d/4})$ time. Putting together

the two sources of error, it follows that the point returned is an $O(\varepsilon)$-FN of q with respect to S_c. By adjusting the constants, we can ensure that the point returned is an ε-FN of q. Since $|S'_c| = O((1/\varepsilon)^{(d-1)/2})$, the time to construct this data structure is $O^*((1/\varepsilon)^{3d/4})$.

Recall that we process a total of $N = (1/\varepsilon)^{d/2}$ points during a phase. We handle these points using the logarithmic rebuilding method of Bentley and Saxe [10]. Let $h = \log N = O^*(1)$. Let S be the set of points seen so far by the algorithm in the current phase. Let the binary representation of $|S|$ be $b_h b_{h-1} \ldots b_0$, where each $b_i \in \{0, 1\}$. We say that a level i is *active* if $b_i = 1$, otherwise it is said to be *inactive*. We maintain a partition of S into a collection of sets S_i corresponding to the active levels i, where $|S_i| = 2^i$. We maintain a data structure for each of the sets S_i as follows. We say that a level i is *high* if $2^i > (1/\varepsilon)^{d/3}$, otherwise we say that it is *low*. For the high active levels, we construct the data structure of Lemma 6.2 for S_i and, for the low active levels, we simply store S_i as a list of points. (Alternatively, we can compress all the low levels into one, but for the sake of convenience we present this scheme which parallels the standard use of the logarithmic method.)

We now describe the processing of a point p. Let S denote the set of points seen in the current phase, before p. Suppose that the approximate diameter being maintained becomes invalid on inserting p. It is clear then that the diameter of $S_c \cup S \cup \{p\}$ is realized by p and its farthest neighbor in $S_c \cup S$. Thus, to maintain the approximate diameter, it suffices to compute the distance of p from its ε-farthest neighbor in $S_c \cup S$, and then update the approximate diameter if necessary. We can find the ε-FN of p in S_c by searching the data structure for S'_c as described above. To find the ε-FN of p in S, we search the $O^*(1)$ data structures which together hold all these points. For data structures corresponding to high active levels, the time for finding the ε-FN in each such structure is $O^*((1/\varepsilon)^{d/4})$. Recall that the data structures at the low active levels simply store the points in lists. We can examine the points on all these lists in time $O^*((1/\varepsilon)^{d/3})$. Note that the time for finding the ε-FN is dominated by the time to search the lists at the low active levels and is given by $O^*((1/\varepsilon)^{d/3})$.

Next, we need to update the active levels and the associated data structures. We handle this in the standard way. We find the lowest level i that is inactive. We merge p and the points in the sets $S_j, 0 \le j \le i-1$, to obtain a set S_i of size 2^i. The level i now becomes active, while all levels $0 \le j \le i-1$ become inactive. We build a data structure for the set S_i at level i as follows. If level i is low, then the data structure for S_i is obtained by merging the i lists at the lower levels. The time for this operation is proportional to the number of lists merged. Otherwise, if level i is high, then we build the data structure of Lemma 6.2 for S_i and destroy the data structures at the lower levels which have now become inactive.

The total processing time for a phase (not including the time for the tasks done at the end of a phase) can be computed by summing the time for building the data structures at each level. First, we consider the time for all the low levels together. Since the data structures at the low levels are obtained by merging lists, and merging any two lists takes constant time, the total time is proportional to the number of points processed, namely, $O((1/\varepsilon)^{d/2})$. Next we bound the time for the high levels. For a given level i, note that over the course of a phase we build at most $N/2^i = (1/\varepsilon)^{d/2}/2^i$ data structures for sets of size 2^i. By Lemma 6.2, the time to construct one such data structure is $O^*(2^i + (1/\varepsilon)^{d/2}2^{i/2} + (1/\varepsilon)^{5d/8})$. Since $(1/\varepsilon)^{d/3} \le 2^i \le (1/\varepsilon)^{d/2}$, the second term dominates the first and the third terms. Thus, the total time for constructing all the data structures at level i is $O^*(((1/\varepsilon)^{d/2}/2^i) \cdot ((1/\varepsilon)^{d/2}2^{i/2})) = O^*((1/\varepsilon)^d/2^{i/2})$. Note that the construction times for the high levels form a geometric series dominated by the lowest level i such that $2^i > (1/\varepsilon)^{d/3}$. Thus the total time for constructing all the data structures at the high levels is $O^*((1/\varepsilon)^d/(1/\varepsilon)^{d/6}) = O^*((1/\varepsilon)^{5d/6})$.

It remains to consider the time for the tasks that need to be performed at the end of a phase. Recall that at the end of a phase, for all $v \in V$, we need to update the ε-extreme pairs of points so they are valid with respect to $S_c \cup S$. We already know the ε-extreme pairs with respect to S_c. At the end of a phase, we compute the ε-extreme pairs with respect to S. By Lemma 6.3(a), this can be done in $O^*((1/\varepsilon)^{3d/4})$ time. Note that the additive projection error involved in choosing ε-extreme points is at most ε times the diameter of the point set. Also, the diameter of a point set is obviously greater than or equal to the diameter of any subset. Using these facts, it is easy to see that for any vector $v \in V$, in $O(1)$ time, we can combine the ε-extreme pairs with respect to S_c and S to obtain the ε-extreme pair with respect to $S_c \cup S$.

Putting it all together, the processing time for a phase is dominated by the time to construct the data structures associated with the high levels and the time to find the ε-FN for each inserted point. Recall that the total time to construct the data structures associated with the high levels is $O^*((1/\varepsilon)^{5d/6})$. This yields an amortized time of $O^*((1/\varepsilon)^{d/3})$ per point. Recall also that the worst-case time to find the ε-FN for each inserted point is $O^*((1/\varepsilon)^{d/3})$. Using standard techniques, we can modify the logarithmic rebuilding method to achieve a worst-case time bound that matches the amortized time bound [29, 30].

Finally, to bound the space, note that there is one data structure at each active level i. For the low levels, the total space is proportional to the sum of the sizes of the lists, which is $O^*((1/\varepsilon)^{d/3})$. For each of the $O^*(1)$ high levels, by Lemma 6.2, the data structure needs $O^*((1/\varepsilon)^{d/2})$ space. The data structure for S'_c also uses the same space. Thus, the total space used is $O^*((1/\varepsilon)^{d/2})$. □

We remark that our streaming algorithm for diameter can be trivially adapted to yield a data structure that supports ε-farthest neighbor queries and point insertions, both in time $O^*((1/\varepsilon)^{d/3})$.

7. STREAMING ε-KERNELS

THEOREM 7.1. *There is a streaming algorithm that uses $O((1/\varepsilon)^{(d-1)/2})$ space and requires $O^*((1/\varepsilon)^{d/4})$ time per point, and that maintains an ε-kernel of size $O((1/\varepsilon)^{(d-1)/2})$.*

PROOF. (Sketch) Zarrabi-Zadeh [32] has given a streaming algorithm for ε-kernels that has all the above properties, except the update time per point is $O^*((1/\varepsilon)^{d/2})$. We will show how we can use our techniques to reduce the update time to roughly a square root of this quantity. We essentially borrow the ideas of [32] and combine them with our

stronger results on discrete Voronoi diagrams. For the sake of completeness, we briefly describe the approach.

Zarrabi-Zadeh's streaming algorithm for an arbitrary stream uses a subroutine for handling *fat streams* of points. A set of points P is said to be *fat* if it is contained inside the hypercube $B = [-1, 1]^d$ and if its convex hull contains $cB+v$, for some positive constant $c \leq 1$ and some point $v \in \mathbb{R}^d$. Assume that an initial stream P is fat. Suppose that we insert more points into this stream while maintaining its fatness. Zarrabi-Zadeh's analysis shows that the update time for inserting points that can be achieved in this special situation would then carry over to an arbitrary stream.

The standard approach for computing the ε-kernel of a fat stream S is based on Dudley's method for approximating convex polytopes [23]. First, we round the points of S to an axis-parallel grid of side length $c_1\varepsilon$. Then we build a uniform grid Ξ with side length $c_2\sqrt{\varepsilon}$ on the boundary of the box $[-2, 2]^d$. For each grid point $\xi \in \Xi$, we find its nearest neighbor in the set obtained by rounding S. Chan [18] has shown that the result is an ε-kernel of S (for suitable constants c_1 and c_2).

Our algorithm for a fat stream works in phases and is similar to that in [32]. At the beginning of a phase, the algorithm knows the nearest neighbor of each point in Ξ with respect to the set of all the rounded points in the stream seen until then. Let S_c denote the set of stream points seen cumulatively in all previous phases, and let S'_c denote the set of nearest neighbors identified. By a simple packing argument, $|S'_c| = O((1/\varepsilon)^{(d-1)/2})$. In each phase, we process $(1/\varepsilon)^{(d-1)/2}$ points. These points are maintained in a simple list until the end of the phase. Clearly, this list of points together with S'_c yields an ε-kernel at any time. Let S denote the set of points seen during the current phase. At the end of a phase, we round the points of S to the nearest grid point (we use the same grid for all the phases), and find the nearest neighbors of each grid point $\xi \in \Xi$ among the rounded points of $S'_c \cup S$. This reduces to a DVD problem for $E = O(1/\varepsilon)$ and $F = O(1/\sqrt{\varepsilon})$. By Corollary 2.3, the time to solve this DVD problem is $O^*((1/\varepsilon)^{3d/4})$. Thus, the amortized time for inserting each point is $O^*((1/\varepsilon)^{d/4})$, which can be converted to a worst-case time bound by standard techniques [29, 30]. □

8. ACKNOWLEDGEMENTS

The first author would like to thank David Mount for helpful discussions.

9. REFERENCES

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. In *Proc. Sixth Annu. Sympos. Comput. Geom.*, pages 203–210, 1990.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[3] P. K. Agarwal, J. Matousek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1:189–201, 1991.

[4] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn. Beyond locality sensitive hashing. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 1018–1028, 2014.

[5] S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate polytope membership queries. In *Proc. 43rd Annu. ACM Sympos. Theory Comput.*, pages 579–586, 2011.

[6] S. Arya, G. D. da Fonseca, and D. M. Mount. Polytope approximation and the Mahler volume. In *Proc. 23rd Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 29–42, 2012.

[7] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. Assoc. Comput. Mach.*, 57:1–54, 2009.

[8] S. Arya, D. M. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45:891–923, 1998.

[9] J. Augustine, D. Eppstein, and K. A. Wortman. Approximate weighted farthest neighbors and minimum dilation stars. *Discrete Math., Algorithms and Applications*, 2(4):553–566, 2010.

[10] J. L. Bentley and J. B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[11] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17:529–533, 1995.

[12] E. M. Bronshteyn and L. D. Ivanov. The approximation of convex sets by polyhedra. *Siberian Math. J.*, 16:852–853, 1976.

[13] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graphs problems in higher dimensions. In *Proc. Fourth Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 291–300, 1993.

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

[15] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20:359–373, 1998.

[16] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.

[17] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Internat. J. Comput. Geom. Appl.*, 12(1-2):67–85, 2002.

[18] T. M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35(1-2):20–35, 2006.

[19] T. M. Chan. Well-separated pair decomposition in linear time? *Inf. Process. Lett.*, 107(5):138–141, 2008.

[20] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. Tenth Annu. Sympos. Comput. Geom.*, pages 160–164, 1994.

[21] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Approximating the weight of the Euclidean minimum spanning tree in sublinear time. *SIAM J. Comput.*, 35(1):91–109, 2005.

[22] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear time. *SIAM J. Comput.*, 39(3):904–922, 2009.

[23] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10:227–236, 1974.

[24] A. Goel, P. Indyk, and K. R. Varadarajan. Reductions among high dimensional proximity problems. In *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 769–778, 2001.

[25] S. Har-Peled. *Geometric Approximation Algorithms*. AMS Press, 2011.

[26] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[27] S. Khuller and Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. *Inf. Comput.*, 118(1):34–37, 1995.

[28] D. Krznaric, C. Levcopoulos, and B. J. Nilsson. Minimum spanning trees in d dimensions. *Nord. J. Comput.*, 6(4):446–461, 1999.

[29] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983.

[30] M. H. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Inf. Process. Lett.*, 12(4):168–173, 1981.

[31] P. M. Vaidya. Minimum spanning trees in k-dimensional space. *SIAM J. Comput.*, 17(3):572–582, 1988.

[32] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011.