Down the Rabbit Hole: Robust Proximity Search and Density Estimation in Sublinear Space

Sariel Har-Peled*
Dept. of Computer Science
University of Illinois,
Urbana, IL 61801, USA
Email: sariel@uiuc.edu

Nirman Kumar*
Dept. of Computer Science
University of Illinois,
Urbana, IL 61801, USA
Email: nkumar5@illinois.edu

***Abstract*—For a set of n points in $\mathbb{R}^d$, and parameters k and ε, we present a data structure that answers $(1+\varepsilon)$-approximate k nearest neighbor queries in logarithmic time. Surprisingly, the space used by the data-structure is $\widetilde{O}(n/k)$; that is, the space used is sublinear in the input size if k is sufficiently large. Our approach provides a novel way to summarize geometric data, such that meaningful proximity queries on the data can be carried out using this sketch. Using this we provide a sublinear space data-structure that can estimate the density of a point set under various measures, including: (i) sum of distances of k closest points to the query point, and (ii) sum of squared distances of k closest points to the query point. Our approach generalizes to other distance based estimation of densities of similar flavor.**

I. Introduction

Given a set of n points P in $\mathbb{R}^d$, the ***nearest neighbor*** problem is to construct a data structure, such that for any *query* point q it (quickly) finds the point closest to q in P. This is an important fundamental problem in computer science [1], [2], [3], [4]. Applications of nearest neighbor search include pattern recognition [5], [6], self-organizing maps [7], information retrieval [8], vector compression [9], computational statistics [10], clustering [11], data mining, learning, and many others. If one is interested in guaranteed performance and near linear space, there is no known way to solve this problem efficiently (i.e., logarithmic query time) for dimension $d > 2$.

A commonly used approach for this problem is to use Voronoi diagrams. The ***Voronoi diagram*** of P is the decomposition of $\mathbb{R}^d$ into interior disjoint closed cells, so that for each cell C there is a unique single point $\mathsf{p} \in \mathsf{P}$ such that for any point $\mathsf{q} \in \mathrm{int}(C)$ the nearest neighbor of q in P is p. Thus, one can compute the nearest neighbor to q by a point location query in the collection of Voronoi cells. In the plane, this approach leads to $O(\log n)$ query time, using $O(n)$ space, and preprocessing time $O(n \log n)$. However, in higher dimensions, this solution leads to algorithms with *exponential* dependency on the dimension. The complexity of a Voronoi diagram of n points in $\mathbb{R}^d$ is $\Theta\left(n^{\lceil d/2 \rceil}\right)$ in the worst case. By requiring slightly more space, Clarkson [12] showed a data-structure with query time $O(\log n)$, and $O\left(n^{\lceil d/2 \rceil + \delta}\right)$ space, where $\delta > 0$ is a prespecified constant (the $O(\cdot)$ notation here hides constants that are exponential in the dimension). One can tradeoff the space used and the query time [13]. Meiser [14] provided a data-structure with query time $O\left(d^5 \log n\right)$ (which has polynomial dependency on the dimension), where the space used is $O\left(n^{d+\delta}\right)$. Therefore, even for moderate dimension, the exact nearest neighbor data structure uses an exorbitant amount of storage. It is believed that there is no efficient solution for the nearest neighbor problem when the dimension is sufficiently large [15]; this difficulty has been referred to as the "curse of dimensionality".

*Approximate Nearest Neighbor (*ANN*):* In light of the above, major effort went into developing approximation algorithms for nearest neighbor search [16], [17], [18], [1], [2], [3], [4], [19]. In d dimensional Euclidean space, one can answer ANN queries, in $O(\log n + 1/\varepsilon^{d-1})$ time using linear space [16], [20]. Because of the $1/\varepsilon^{d-1}$ in the query time, this approach is only good for low dimensions. Interestingly, for this data-structure, the approximation parameter ε is not prespecified during the construction; one can provide it during the query. An alternative approach is to use Approximate Voronoi Diagrams (AVD), introduced by Har-Peled [21], which are partitions of space into regions of low total complexity, with a representative point for each region that is an ANN for any point in the region. In particular, Har-Peled showed that there is such a decomposition of size $O\left((n/\varepsilon^d)\log^2 n\right)$. This allows ANN queries to be answered in $O(\log n)$ time. Arya and Malamatos [22] showed how to build AVDs of linear complexity (i.e., $O(n/\varepsilon^d)$). Their construction uses Well Separated Pair Decompositions [23]. Further tradeoffs between query and space for AVDs were studied by Arya *et al.* [24].

k-nearest neighbor: A more general problem is the k-nearest neighbors problem where one is interested in finding the k points in P nearest to the query point q. This is widely used in pattern recognition, where the majority label is used to label the query point. Here we are interested in the more restricted problem of approximately computing the distance to the kth-nearest neighbor and finding a data point achieveing the approximation. This problem is widely used for density estimation in statistics, with $k \approx \sqrt{n}$ [25]. It is also used in meshing (with $k = d$) to compute the local feature size of a point set in $\mathbb{R}^d$ [26]. The problem also has applications in non-linear dimensionality reduction – finding low dimensional structures in data; more specifically low dimensional submanifolds embedded in Euclidean spaces. Algorithms like ISOMAP, LLE, Hessian-LLE, SDE and others, all use the k-nearest neighbor as a subroutine [27], [28], [29], [30].

Density estimation: Given distributions $\mu_1, \ldots, \mu_k$ defined over $\mathbb{R}^d$, and a query point q, we are interested in computing the *a posteriori* probabilities of q being generated by each of these distributions. This approach is used in

*Work on this paper was partially supported by NSF AF award CCF-0915984.

unsupervised learning as a way to classify a new point. Naturally, in most cases, the distributions are given implicitly; that is, one is given a large number of points sampled from each distribution. So, let μ be such a distribution, and P be a set of n samples. To estimate the density of μ at q, a standard Monte Carlo technique is to consider a ball B centered at q, and count the number of points of P inside B. Naturally, if $\mathsf{P} \cap \mathsf{B}$ is too small, this estimate is not stable. Similarly, if B is too large, then the estimate is too "smoothed" out, taking into account samples that are too far away. One possible approach to address this issue, that is used in practice [11], is to find the smallest ball centered at q that contains k points of P and use this to estimate the density of μ. Choosing the right value of k has to be done carefully – if it is too small, then the estimate is unstable, and if it is too large, it either requires the set P to be larger, or the estimate is too smoothed out to be useful (values of k that are used in practice are $\widetilde{O}(\sqrt{n})$). See Duda *et al.* [11] for more details. To do such density estimation, one needs to be able to answer, approximate or exact, k-nearest neighbor queries.

Sometimes one is interested not only in the radius of this ball centered at the query point, but also in the distribution of the points inside this ball. The average distance of a point inside the ball to its center, can be estimated by the sum of distances of the sample points inside the ball to the center. Similarly, the variance of the distance can be esimated by the sum of squared distances of the sample points inside the ball to the center of the ball. As mentioned above, density estimation is used in manifold learning and surface reconstruction. For example, Guibas *et al.* [31] recently used a similar density estimate to do manifold reconstruction.

Answering exact k-nearest neighbor queries: Given a point set $\mathsf{P} \subseteq \mathbb{R}^d$, computing the partition of space into regions where the k nearest neighbors do not change, is equivalent to computing the ***k*****th order Voronoi diagram**. Via standard lifting, this is equivalent to computing the first k levels in an arrangement of hyperplanes in $\mathbb{R}^{d+1}$ [32]. More precisely, if we are interested in the kth-nearest neighbor, we need to compute the $(k-1)$-level in this arrangement.

The complexity of the $\leq k$ levels in a hyperplane arrangement in $\mathbb{R}^{d+1}$ is $\Theta(n^{\lfloor (d+1)/2 \rfloor}(k+1)^{\lceil (d+1)/2 \rceil})$ [33]. The exact complexity of the kth-level is not well understood and achieving tight bounds on its complexity is one of the long-standing open problems in discrete geometry [34]. In particular, via an averaging argument, in the worst case the complexity of the kth-order Voronoi diagram is $\Omega\Big(n^{\lfloor (d+1)/2 \rfloor}(k+1)^{\lceil (d+1)/2 \rceil - 1}\Big)$. As such, the complexity of kth-order Voronoi diagram is $\Omega(nk)$ in two dimensions, and $\Omega(n^2 k)$ in three dimensions.

Thus, to provide a data-structure for answering k-nearest neighbor queries exactly and quickly (i.e., logarithmic query time) in $\mathbb{R}^d$, requires computing the k-level of an arrangement of hyperplanes in $\mathbb{R}^{d+1}$. The complexity of this structure is prohibitive even in two dimensions (this complexity determines the preprocessing and space needed by such a data-structure). Furthermore, naturally, the complexity of this structure increases as k increases. On the other end of the spectrum one can use partition-trees and parametric search to answer such queries using linear space and query time (roughly) $O\Big(n^{1-1/(d+1)}\Big)$ [35], [36]. One can get intermediate results using standard space/time tradeoffs [37].

Known results on approximate k-order Voronoi diagram: Similar to AVD, one can define a AVD for the k-nearest neighbor. The case $k = 1$ is the regular approximate Voronoi diagram [21], [22], [24]. The case $k = n$ is the furthest neighbor Voronoi diagram. It is not hard to see that it has a constant size approximation (see [38], although it was probably known before). Our results (see below) can be interpreted as bridging between these two extremes.

Quorum clustering: Carmi *et al.* [39] describe how to compute efficiently a partition of the given point set into clusters of k points such that the clusters are compact. Specifically, this quorum clustering repeatedly computes the smallest ball containing k points, removes this cluster and repeats, see Section II-B1 for more details. Carmi *et al.* [39] also describe a data-structure that can approximate the smallest cluster. The space of their data structure is $\widetilde{O}(n/k)$, but it can not be directly used for our purposes. Furthermore, their data-structure is for two dimensions and it can not be extended to higher dimensions, as it uses additive Voronoi diagrams (which have high complexity in higher dimensions).

OUR RESULTS

We first show, in Section III, that one can build a data-structure that answers k-nearest neighbor queries approximately, up to a constant factor, with query time $O(\log n)$, where the input is a set of n points in $\mathbb{R}^d$. Surprisingly, the space used by this data-structure is $O(n/k)$. This result is surprising as the complexity ***decreases*** with k. This is in sharp contrast to behavior in the exact version of the kth-order Voronoi diagram (where the complexity increases with k). Furthermore, for super-constant k the space used by this data-structure is sublinear. For example, in some applications the value of k used is $\Omega(\sqrt{n})$, and the space used in this case is a tiny fraction of the input size. This is a general reduction showing that such queries can be reduced to proximity search in an appropriate product space over n/k points computed carefully.

In Section IV, we show how to construct *approximate* k-order Voronoi diagram using space $O(\varepsilon^{-d-1} n/k)$ (here $\varepsilon > 0$ is an approximation quality parameter specified in advance). Using this data-structure one can answer $(1+\varepsilon)$-approximate k-nearest neighbor queries in $O(\log n)$ time. See Theorem IV.9 for the exact result.

General density queries: We also show, in Section V, as an application of our data-structure, how to answer more robust kinds of queries. For example, one can approximate (in roughly the same time and space as above) the sum of distances (or squared distances) from a query point to its k nearest neighbors. This is useful in approximating density measures [11]. Surprisingly, our data-structure can be used to estimate the sum of any function $f(\cdot)$ defined over the k nearest neighbor points that depends only on the distance of these points from the query point. Informally, we require that $f(\cdot)$ is monotonically increasing with distance, and it is (roughly) not super-polynomial. For example, for any constant $p > 0$, our data-structure requires sublinear space (i.e.,

$\widetilde{O}(n/k)$), and given a query point q, it can $(1+\varepsilon)$-approximate the quantity $\sum_{\mathsf{u}\in X} \|\mathsf{u}-\mathsf{q}\|^p$, where X is the set of k nearest points to q. The query time is logarithmic.

To facilitate this, in a side result, that might be of independent interest, we show how to perform point-location queries in I compressed quadtrees of total size m simultaneously in $O(\log m + I)$ time (instead of the naive $O(I \log m)$ query time) without asymptotically increasing the space needed.

If k is specified with the query: In Section VI, given a set P of n points in $\mathbb{R}^d$, we show how to build a data-structure, in $O(n \log n)$ time and using $O(n)$ space, such that given a query point and parameters k and ε, the data-structure can answer $(1+\varepsilon)$-approximate k-nearest neighbor queries in $O(\log n + 1/\varepsilon^{d-1})$ time. Unlike previous results, this is the first data-structure where *both* k and ε are specified during the query time. Previously, the data-structure of Arya *et al.* [40] required knowing ε in advance. Using standard techniques [16] to implement it, should lead to a simple and practical algorithm for this problem.

If k is not important: A relevant question is how to answer the approximate k-nearest neighbor query if one is allowed to also approximate k. Inherently, this is a completely different question that is considerably easier, and arguably less interesting. Indeed, the problem then boils down to using sampling carefully, and the problem loses much of its geometric flavor. We sketch how to solve this easier variant (this seems to be new) and discuss the difference with our main problem in Section II-A1.

Techniques used: We use quorum clustering as a starting point in our solution. In particular, we show how such clustering can be used to get a constant factor approximation to the approximate k-nearest neighbor distance using sublinear space. Next, we extend this construction and combine it with ideas used in the computation of approximate Voronoi diagrams. This results in an algorithm for computing approximate k-nearest neighbor Voronoi diagram. To extend this data-structure to answer the general density queries, as described above, requires a way to estimate the function $f(\cdot)$ for very few values (instead of k values) when answering a query. We use a coreset construction to find out which values need to be approximated. Overall, our work combines several known techniques in a non-trivial fashion, together with some new ideas, to get our new results.

Paper organization: In Section II we formally define the problem and introduce some basic tools, including quorum clustering, which is a key insight into the problem at hand. The "generic" constant factor algorithm is described in Section III. We describe the construction of the approximate k-order Voronoi diagram in Section IV. In Section V we describe how to construct a data-structure to answer density queries of various types. In Section VI we present the data-structure for answering k-nearest neighbor queries that does not require knowing k and ε in advance. We conclude in Section VII.

II. Preliminaries

A. Problem Definition

Given a set of n points P in $\mathbb{R}^d$ and a number $1 \le k \le n$, consider a point q and order the points of P by their distance from q; that is,

$$\|\mathsf{q}-\mathsf{u}_1\| \le \|\mathsf{q}-\mathsf{u}_2\| \le \cdots \le \|\mathsf{q}-\mathsf{u}_n\|,$$

where $\mathsf{P} = \{\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_n\}$. The point $\mathsf{u}_k = \mathsf{nn}_k(\mathsf{q},\mathsf{P})$ is the ***kth-nearest neighbor*** of q and $\mathsf{d}_k(\mathsf{q},\mathsf{P}) = \|\mathsf{q}-\mathsf{u}_k\|$ is the ***kth-nearest neighbor distance***. The nearest neighbor distance (i.e., $k=1$) is $\mathsf{d}(\mathsf{q},\mathsf{P}) = \min_{\mathsf{u}\in\mathsf{P}} \|\mathsf{q}-\mathsf{u}\|$. It is easy to verify that the function $\mathsf{d}_k(\mathsf{q},\mathsf{P})$ is 1-Lipschitz as stated in the following.

Observation II.1. *For any $\mathsf{p},\mathsf{u} \in \mathbb{R}^d$, k and a set $\mathsf{P} \subseteq \mathbb{R}^d$, we have that $\mathsf{d}_k(\mathsf{u},\mathsf{P}) \le \mathsf{d}_k(\mathsf{p},\mathsf{P}) + \|\mathsf{p}-\mathsf{u}\|$.*

The problem at hand is to preprocess P such that given a query point q one can compute u_k quickly. The standard ***nearest neighbor problem*** is this problem for $k=1$. In the ***$(1+\varepsilon)$-approximate kth-nearest neighbor*** problem, given q, k and $\varepsilon > 0$, one wants to find a point $\mathsf{u} \in \mathsf{P}$, such that $(1-\varepsilon)\|\mathsf{q}-\mathsf{u}_k\| \le \|\mathsf{q}-\mathsf{u}\| \le (1+\varepsilon)\|\mathsf{q}-\mathsf{u}_k\|$.

1) An easier problem – if k is not important: Consider another version of the approximate k-nearest neighbor problem where one is allowed to approximate k. That is, given a query point, one has to return the (perhaps approximate) distance to a point which is a ℓ-nearest neighbor to the query, where $k \le \ell \le (1+\varepsilon)k$. As mentioned in the introduction, this is a completely different problem from the one we consider. Here we quickly sketch a solution to this variant (which seems to be new), and discuss the difference with the more interesting problem we solve in the rest of the paper.

Indeed, one can sample the given point set, where each point is picked with probability $O(k^{-1}\varepsilon^{-2}\log n)$. It is easy to verify that the $O(\varepsilon^{-2}\log n)$ nearest neighbor to the query (in the sample) is the required approximation with high probability. Using gradations one can precompute $O(\log n)$ samples that are appropriate to any value of k. Furthermore, one can easily reduce solving this problem to answering polylogarithmic number of queries using standard approximate nearest-neighbor data-structures [41], [20]. Nevertheless, the resulting data-structure has both worse space and query time than the data-structure we present here.

In particular, since we solve here the harder variant, it is not clear why one should compromise on a weaker data-structure with worse performance. Secondly, for some of the applications, like density estimation, there might be a phase change in the distribution of distances and approximating k is not acceptable. Conversely, for such applications approximating the distances is acceptable. Finally, it is not clear how such a fuzzy data-structure can be used for the density estimation without some additional overheads, that make it inherently less applicable for such problems.

B. Basic tools

For a real positive number α and a point $\mathsf{p} = (\mathsf{p}_1, \ldots, \mathsf{p}_d) \in \mathbb{R}^d$, define $\mathsf{G}_\alpha(\mathsf{p})$ to be the grid point $(\lfloor \mathsf{p}_1/\alpha \rfloor \alpha, \ldots, \lfloor \mathsf{p}_d/\alpha \rfloor \alpha)$. We call α the ***width*** or ***sidelength*** of the ***grid*** G_α. Observe that the mapping G_α partitions $\mathbb{R}^d$ into cubic regions, which we call grid ***cells***.

Definition II.2. *A cube is a* **canonical cube** *if it is contained inside the unit cube $[0,1]^d$, it is a cell in a grid G_r, and r is a* *power of two (i.e., it might correspond to a node in a quadtree having $[0,1]^d$ as its root cell). We will refer to such a grid G_r as a* **canonical grid**. *Note, that all the cells corresponding to nodes of a compressed quadtree are canonical.*

For a ball b of radius r, and a parameter ψ, let $\boxplus(\mathsf{b},\psi)$ denote the set of all the canonical cells intersecting b, when considering the canonical grid with sidelength $2^{\lfloor \log_2 \psi \rfloor}$. Clearly, $|\boxplus(\mathsf{b},\psi)| = O\big((r/\psi)^d\big)$.

A ball b of radius r in $\mathbb{R}^d$ centered at a point p can be interpreted as a point in $\mathbb{R}^{d+1}$, denoted by $\mathsf{b}' = (\mathsf{p}, r)$. For a regular point $\mathsf{p} \in \mathbb{R}^d$, its corresponding image under this transformation is the ***mapped*** point $\mathsf{p}' = (\mathsf{p}, 0) \in \mathbb{R}^{d+1}$.

Given point $\mathsf{u} = (\mathsf{u}_1, \ldots, \mathsf{u}_d) \in \mathbb{R}^d$ we will denote its euclidean norm by $\|\mathsf{u}\|$. We will consider a point $\mathsf{u} = (\mathsf{u}_1, \mathsf{u}_2, \ldots, \mathsf{u}_{d+1}) \in \mathbb{R}^{d+1}$ to be in the product metric of $\mathbb{R}^d \times \mathbb{R}$ and endowed with the product metric norm

$$\|\mathsf{u}\|_\oplus = \sqrt{\mathsf{u}_1^2 + \cdots + \mathsf{u}_d^2} + |\mathsf{u}_{d+1}|\,.$$

It is easy to see that the above defines a norm and the following holds for it.

Lemma II.3. *For any $\mathsf{u} \in \mathbb{R}^{d+1}$ we have $\|\mathsf{u}\| \le \|\mathsf{u}\|_\oplus \le \sqrt{2}\,\|\mathsf{u}\|$.*

The distance of a point to a set under the $\|\cdot\|_\oplus$ norm is denoted by $\mathsf{d}_\oplus(\mathsf{u}, \mathsf{P})$.

Simplifying assumption: In the following, we will assume that k divides n; if not one can easily add fake points as necessary at infinity. We also assume that the point set P is contained in $[1/2, 1/2 + 1/n]^d$, where $n = |\mathsf{P}|$. This can be achieved by scaling and translation (which does not effect the distance ordering). We will assume that the queries are restricted to the unit cube $U = [0,1]^d$.

1) Quorum Clustering: Given a set of n points in $\mathbb{R}^d$ and a number $k \ge 1$, where $k|n$,we start with the smallest ball b_1 that contains k points of P. Let $\mathsf{P}_1 = \mathsf{P} \cap \mathsf{b}_1$. We continue on the set of points $\mathsf{P} \setminus \mathsf{P}_1$ by finding the smallest ball that contains k points of the remaining set of points, and so on. Let $\mathsf{b}_1, \mathsf{b}_2, \ldots, \mathsf{b}_{n/k}$ denote the set of balls found by the algorithm and let $\mathsf{P}_i = (\mathsf{P} \setminus (\mathsf{P}_1 \cup \cdots \cup \mathsf{P}_{i-1})) \cap \mathsf{b}_i$. Let c_i and r_i denote the center and radius, respectively, of b_i, $1 \le i \le n/k$. A slight symbolic perturbation can guarantee that (i) each ball b_i contains exactly k points of P, and (ii) all the centers $\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_k$ are distinct points. It is easy to see that $\mathsf{r}_1 \le \mathsf{r}_2 \le \cdots \le \mathsf{r}_{n/k} \le \mathsf{diam}(\mathsf{P})$. Such a clustering of P into n/k clusters is termed a ***quorum clustering*** and an algorithm for computing it is provided in Carmi *et al.* [39]. We assume we have a black-box procedure **QuorumCluster**(P, k) [39] that computes an ***approximate*** quorum clustering. It returns a list of balls, $(\mathsf{c}_i, \mathsf{r}_i)$, $1 \le i \le n/k$. The algorithm of Carmi *et al.* [39] provides such sequence of clusters, where each ball is a 2-approximation to the smallest ball containing k points of the remaining points. The following is an improvement over the result of Carmi *et al.* [39].

Lemma II.4. *Given a set P of n points in $\mathbb{R}^d$ and parameter k, one can compute, in $O(n \log n)$ time, a sequence of n/k balls such that:*

(A) *For every ball $(\mathsf{c}_i, \mathsf{r}_i)$ there is an associated subset P_i of k points of $\mathsf{P} \setminus (\mathsf{P}_i \cup \ldots \cup \mathsf{P}_{i-1})$ that it covers.*

(B) *The ball $(\mathsf{c}_i, \mathsf{r}_i)$ is a 2-approximation to the smallest ball covering k points in $\mathsf{P} \setminus (\mathsf{P}_1 \cup \ldots \cup \mathsf{P}_{i-1})$.*

Proof: The guarantee of Carmi *et al.* is slightly worse – their algorithm running time is $O(n \log^d n)$. They use a dynamic data-structure for answering $O(n)$ queries, that report how many points are inside a query canonical square. Since they use orthogonal range trees this requires $O(\log^d n)$ time per query. Instead, one can use dynamic quadtrees. More formally, we store the points using linear ordering [20] using any balanced data-structure. A query to decide the number of points inside a canonical node corresponds to an interval query (i.e., reporting the number of elements that are inside a query interval) and can be performed in $O(\log n)$ time. Plugging this data-structure into the algorithm of Carmi *et al.* [39] gives the desired result. ∎

III. A Constant Factor Approximation

Lemma III.1. *Let P be a set of n points in $\mathbb{R}^d$, and let $k \ge 1$ be a number such that $k|n$. Let $(\mathsf{c}_1, \mathsf{r}_1), \ldots, (\mathsf{c}_{n/k}, \mathsf{r}_{n/k})$ be the list of balls returned by* **QuorumCluster**(P, k). *Let $x = \min_{1 \le i \le n/k}(\|\mathsf{q} - \mathsf{c}_i\| + \mathsf{r}_i)$. We have that $x/5 \le \mathsf{d}_k(\mathsf{q}, \mathsf{P}) \le x$.*

Proof: For any $1 \le i \le n/k$ we have $\mathsf{b}_i = \mathsf{ball}(\mathsf{c}_i, \mathsf{r}_i) \subseteq \mathsf{ball}(\mathsf{q}, \|\mathsf{q} - \mathsf{c}_i\| + \mathsf{r}_i)$. Since $|\mathsf{b}_i \cap \mathsf{P}| \ge k$, we have $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \le \|\mathsf{q} - \mathsf{c}_i\| + \mathsf{r}_i$. As such, $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \le x = \min_{1 \le i \le n/k}(\|\mathsf{q} - \mathsf{c}_i\| + \mathsf{r}_i)$.

For the other direction, let i be the minimal integer such that $\mathsf{ball}(\mathsf{q}, \mathsf{d}_k(\mathsf{q}, \mathsf{P}))$ contains a point of P_i. Then, we have

$$\mathsf{r}_i/2 \le \mathsf{d}_k(\mathsf{q}, \mathsf{P})\,,$$

as r_i is a 2-approximation to the radius of the smallest ball that contains k points of $\mathsf{P}_i \cup \mathsf{P}_{i+1} \cup \cdots \cup \mathsf{P}_{n/k}$. We also have

$$\|\mathsf{q} - \mathsf{c}_i\| - \mathsf{r}_i \le \mathsf{d}_k(\mathsf{q}, \mathsf{P})\,,$$

as the distance from q to any $\mathsf{u} \in \mathsf{ball}(\mathsf{c}_i, \mathsf{r}_i)$ satisfies $\|\mathsf{q} - \mathsf{u}\| \ge \|\mathsf{q} - \mathsf{c}_i\| - \mathsf{r}_i$ by the triangle inequality. Putting the above together, we get

$$\begin{aligned} x &= \min_{1 \le j \le n/k}(\|\mathsf{q} - \mathsf{c}_j\| + \mathsf{r}_j) \\ &\le \|\mathsf{q} - \mathsf{c}_i\| + \mathsf{r}_i = (\|\mathsf{q} - \mathsf{c}_i\| - \mathsf{r}_i) + 2\mathsf{r}_i \\ &\le 5\mathsf{d}_k(\mathsf{q}, \mathsf{P})\,. \end{aligned}$$ ∎

Theorem III.2. *Given a set P of n points in $\mathbb{R}^d$, and a number $k \ge 1$ such that $k|n$, one can build a data-structure, in $O(n \log n)$ time that uses $O(n/k)$ space, and given any query point $\mathsf{q} \in \mathbb{R}^d$ one can find a $O(1)$-approximation to $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$ in $O(\log(n/k))$ time.*

Proof: We invoke **QuorumCluster**(P, k) to compute the clusters $(\mathsf{c}_i, \mathsf{r}_i)$, for $i = 1, \ldots, n/k$. For $i = 1, \ldots, n/k$, let $\mathsf{b}'_i = (\mathsf{c}_i, \mathsf{r}_i) \in \mathbb{R}^{d+1}$. We preprocess the set $\mathcal{B}' = \{\mathsf{b}'_1, \ldots, \mathsf{b}'_{n/k}\}$ for 2-ANN queries (in $\mathbb{R}^{d+1}$). The preprocessing time for the ANN is $O((n/k)\log(n/k))$, the space used is $O(n/k)$ and the query time is $O(\log(n/k))$ [20].

Given a query point $\mathsf{q} \in \mathbb{R}^d$ the algorithm computes a 2-ANN to $\mathsf{q}' = (\mathsf{q}, 0)$, denoted by b'_j. The algorithm returns $\|\mathsf{q}' - \mathsf{b}'_j\|_\oplus$ as the approximate distance.

Observe that, for any i, we have $\|q' - b'_i\| \le \|q' - b'_i\|_\oplus \le \sqrt{2}\,\|q' - b'_i\|$ by Lemma II.3. As such, the returned distance to b'_j is a 2-approximation to $d(q', \mathcal{B}')$; that is,

$$d_\oplus\bigl(q', \mathcal{B}'\bigr) \le \bigl\|q' - b'_j\bigr\|_\oplus \le \sqrt{2}\,\|q' - b'_j\| \le 2\sqrt{2}d\bigl(q', \mathcal{B}'\bigr) \le 2\sqrt{2}d_\oplus\bigl(q', \mathcal{B}'\bigr).$$

By Lemma III.1, $d_\oplus(q', \mathcal{B}')/5 \le d_k(P, q) \le d_\oplus(q', \mathcal{B}')$. Namely, $\bigl\|q' - b'_j\bigr\|_\oplus/(10\sqrt{2}) \le d_k(P, q) \le \bigl\|q' - b'_j\bigr\|_\oplus$, implying the claim. ■

Remark III.3. The algorithm of Theorem III.2 works for any metric space. Given a set P of n points in a metric space, one can compute n/k points in a the product space induced by adding an extra coordinate, such that approximating the distance to the kth nearest neighbor, is equivalent to answering ANN queries on the reduced point set.

IV. Approximate Voronoi Diagram for $d_k(q, P)$

Here, we are given a set P of n points in $\mathbb{R}^d$, and our purpose is to build an AVD that approximates the k-ANN distance, while using (roughly) $O(n/k)$ space.

A. Construction

1) Preprocessing:

(A) Compute a quorum clustering for P using Lemma II.4. Let the list of balls returned be $b_i = (c_i, r_i)$, for $i = 1, \ldots, n/k$.

(B) Compute an exponential grid around each quorum cluster. Specifically, let

$$\mathcal{X} = \bigcup_{i=1}^{n/k} \bigcup_{j=0}^{\lceil \log(32/\varepsilon)+1 \rceil} \boxplus\Bigl(\mathrm{ball}\Bigl(c_i, 2^j r_i\Bigr), \frac{\varepsilon}{\zeta_1 d} 2^j r_i\Bigr) \tag{1}$$

be the set of grid cells covering the quorum clusters and their immediate environ, where ζ_1 is a sufficiently large constant.

(C) Intuitively, $\mathcal{X}$ takes care of region of space immediately next to the quorum clusters. For the other regions of space, we can apply (intuitively) a construction of an approximate Voronoi diagram for the centers of the clusters (the details are somewhat more involved). To this end, lift the quorum clusters into points in $\mathbb{R}^{d+1}$, as follows

$$\mathcal{B}' = \bigl\{b'_1, \ldots, b'_{n/k}\bigr\},$$

where $b'_i = (c_i, r_i) \in \mathbb{R}^{d+1}$, for $i = 1, \ldots, n/k$. Note, that all points in $\mathcal{B}'$ belong to $U' = [0,1]^{d+1}$. We now build $(1+\varepsilon/8)$-AVD for $\mathcal{B}'$ using the algorithm of [22]. The AVD construction provides a list of canonical cubes covering $[0,1]^{d+1}$ such that locating the smallest cube containing the query point, has an associated point of $\mathcal{B}'$ that is $(1+\varepsilon/8)$-ANN to the query point. (Note, that there cubes are not necessarily disjoint. In particular, the smallest cube containing the query point q is the one determines what is the ANN of q.)

We clip this collection of cubes to the hyperplane $x_{d+1} = 0$ (i.e., we throw away cubes that do not have a face on this hyperplane). For a cube $\square$ in this collection, we denote by $\mathrm{nn}'(\square)$ the point of $\mathcal{B}'$ assigned to it. Let $\mathcal{S}$ be this resulting set of canonical cubes.

(D) Let $\mathcal{W}$ be the space decomposition resulting from overlaying the two collection of cubes $\mathcal{X}$ and $\mathcal{S}$. Formally, we compute a compressed quadtree $\mathcal{T}$ that has all the canonical cubes of $\mathcal{X}$ and $\mathcal{S}$ as nodes, and $\mathcal{W}$ is the resulting decomposition of space into cells. One can overlay two compressed quadtrees representing the two sets in linear time [42], [20]. Here a cell associated with a leaf is a canonical cube, and a cell associated with a compressed node is the set difference of two canonical cubes. Each node in this compressed quadtree contains two pointers – one to the smallest cube of $\mathcal{X}$, and one to the smallest cube of $\mathcal{S}$ that contains it. This information can be easily computed by doing a BFS on the tree.

For each cell $\square \in \mathcal{W}$ we store the following.

(I) An arbitrary representative point $\square_{\mathrm{rep}} \in \square$.

(II) The point $\mathrm{nn}'(\square) \in \mathcal{B}'$ that is associated with the smallest cell of $\mathcal{S}$ that contains this cell.

(III) A number $\beta_k(\square_{\mathrm{rep}})$ that satisfies $d_k(\square_{\mathrm{rep}}, P) \le \beta_k(\square_{\mathrm{rep}}) \le (1 + \varepsilon/4)d_k(\square_{\mathrm{rep}}, P)$.

2) Answering a query: Given a query point q, compute the leaf cell (equivalently the smallest cell) from $\mathcal{W}$ that contains q by performing a point-location query in $\mathcal{T}$. Let $\square$ be the leaf cell that contains q. Return

$$\min\Bigl(\bigl\|q' - \mathrm{nn}'(\square)\bigr\|_\oplus, \beta_k(\square_{\mathrm{rep}}) + \|q - \square_{\mathrm{rep}}\|\Bigr), \tag{2}$$

as the approximate value to $d_k(q, P)$.

One can also compute a representative point that corresponds to the returned kth-nearest neighbor distance. To this end, together with $\mathrm{nn}'(\square)$ we associate an arbitrary point p that is associated with this quorum cluster. Similarly, with $\beta_k(\square_{\mathrm{rep}})$ one stores the kth-nearest neighbor (or approximate k-nearest neighbor) to $\square_{\mathrm{rep}}$. One returns the point corresponding to the distance selected as the desired approximate kth-nearest neighbor.

B. Correctness

Lemma IV.1. *Let $\square \in \mathcal{W}$ and $q \in \square$. Then the number computed by the algorithm is an upper bound on $d_k(q, P)$.*

Proof: By Observation II.1, $d_k(q, P) \le d_k(\square_{\mathrm{rep}}, P) + \|q - \square_{\mathrm{rep}}\| \le \beta_k(\square_{\mathrm{rep}}) + \|q - \square_{\mathrm{rep}}\|$. Now, let $\mathrm{nn}'(\square) = (c, r)$. We also have, by Lemma III.1, that $d_k(q, P) \le \|q - c\| + r = \|q' - \mathrm{nn}'(\square)\|_\oplus$. As the returned value is the minimum of these two numbers, the claim holds. ■

Lemma IV.2. *Consider any query point $q \in [0,1]^d$, and let $\square$ be the smallest cell of $\mathcal{W}$ that contains the query point. Then, $d(q', \mathcal{B}') \le \|q' - \mathrm{nn}'(\square)\| \le (1+\varepsilon/8)d(q', \mathcal{B}')$.*

Proof: Observe, that space decomposition generated by $\mathcal{W}$ is a refinement of the decomposition of space (which is an AVD of $\mathcal{B}'$) generated by the Arya and Malamatos [22] construction when applied to $\mathcal{B}'$ and restricted to the d dimensional subspace we are interested in (i.e., $x_{d+1} = 0$). As such, $\mathrm{nn}'(\square)$ is exactly the point returned by the AVD for this query point before the refinement, thus implying the claim. ■

1) The query point is close to a quorum cluster of the right size:

Lemma IV.3. *Consider a query point* q*, and let* $\square \subseteq \mathbb{R}^d$ *be any subset with* $\mathsf{q} \in \square$ *such that* $\mathsf{diam}(\square) \leq \varepsilon \mathsf{d}_k(\mathsf{q}, \mathsf{P})$*. Then, for any* $\mathsf{u} \in \square$*, we have*

$$(1-\varepsilon)\mathsf{d}_k(\mathsf{q},\mathsf{P}) \leq \mathsf{d}_k(\mathsf{u},\mathsf{P}) \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q},\mathsf{P}).$$

Proof: By Observation II.1, we have

$$\begin{aligned}\mathsf{d}_k(\mathsf{q},\mathsf{P}) &\leq \mathsf{d}_k(\mathsf{u},\mathsf{P}) + \|\mathsf{u}-\mathsf{q}\| \leq \mathsf{d}_k(\mathsf{u},\mathsf{P}) + \mathsf{diam}(\square)\\ &\leq \mathsf{d}_k(\mathsf{u},\mathsf{P}) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P}).\end{aligned}$$

The other direction follows by a symmetric argument. ■

Lemma IV.4. *If the smallest* $\square \in \mathcal{W}$ *that contains* q *has diameter* $\mathsf{diam}(\square) \leq \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4$ *then the algorithm returns a distance which is between* $\mathsf{d}_k(\mathsf{q},\mathsf{P})$ *and* $(1+\varepsilon)\mathsf{d}_k(\mathsf{q},\mathsf{P})$*.*

Proof: Let $\square_{\mathsf{rep}}$ be the representative stored with the cell. Let α be the number returned by the algorithm. By Lemma IV.1 we have that $\mathsf{d}_k(\mathsf{q},\mathsf{P}) \leq \alpha$. Since the algorithm returns the minimum of two numbers one of which is $\beta_k(\square_{\mathsf{rep}}) + \|\mathsf{q} - \square_{\mathsf{rep}}\|$ we have by Lemma IV.3,

$$\begin{aligned}\alpha &\leq \beta_k(\square_{\mathsf{rep}}) + \|\mathsf{q}-\square_{\mathsf{rep}}\|\\ &\leq (1+\varepsilon/4)\mathsf{d}_k(\square_{\mathsf{rep}},\mathsf{P}) + \|\mathsf{q}-\square_{\mathsf{rep}}\|\\ &\leq (1+\varepsilon/4)(\mathsf{d}_k(\mathsf{q},\mathsf{P}) + \|\mathsf{q}-\square_{\mathsf{rep}}\|) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4\\ &\leq (1+\varepsilon/4)(\mathsf{d}_k(\mathsf{q},\mathsf{P}) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4\\ &= (1+\varepsilon/4)^2\mathsf{d}_k(\mathsf{q},\mathsf{P}) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4 \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q},\mathsf{P}),\end{aligned}$$

establishing the claim. ■

Definition IV.5. *Consider a query point* $\mathsf{q} \in \mathbb{R}^d$*. The first quorum cluster* $\mathsf{b}_i = \mathsf{ball}(\mathsf{c}_i, \mathsf{r}_i)$ *that intersects* $\mathsf{ball}(\mathsf{q}, \mathsf{d}_k(\mathsf{q},\mathsf{P}))$ *is the anchor cluster of* q*. The corresponding* **anchor point** *is* $(\mathsf{c}_i, \mathsf{r}_i) \in \mathbb{R}^{d+1}$*.*

The proof of the following lemma appears in the full version [43].

Lemma IV.6. *For any query point* q*, we have that*
(i) the anchor point (c,r) *is well defined,*
(ii) $\mathsf{r} \leq 2\mathsf{d}_k(\mathsf{q},\mathsf{P})$*,*
(iii) for $\mathsf{b} = \mathsf{ball}(\mathsf{c},\mathsf{r})$ *we have* $\mathsf{b} \cap \mathsf{ball}(\mathsf{q},\mathsf{d}_k(\mathsf{q},\mathsf{P})) \neq \emptyset$*, and*
(iv) $\|\mathsf{q}-\mathsf{c}\| \leq 3\mathsf{d}_k(\mathsf{q},\mathsf{P})$*.*

Lemma IV.7. *Consider a query point* q*. If there is a cluster* $\mathsf{ball}(\mathsf{c},\mathsf{r})$ *in the quorum clustering computed, such that* $\|\mathsf{q}-\mathsf{c}\| \leq 6\mathsf{d}_k(\mathsf{q},\mathsf{P})$ *and* $\varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4 \leq \mathsf{r} \leq 6\mathsf{d}_k(\mathsf{q},\mathsf{P})$ *then the output of the algorithm is correct.*

Proof: We have

$$\frac{32\mathsf{r}}{\varepsilon} \geq \frac{32(\varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4)}{\varepsilon} \geq 8\mathsf{d}_k(\mathsf{q},\mathsf{P}) \geq \|\mathsf{q}-\mathsf{c}\|.$$

Thus, by construction, the expanded environ of the quorum cluster $\mathsf{ball}(\mathsf{c},\mathsf{r})$ contains the query point, see Eq. (1). As such the smallest quadtree cell $\square$ that contains q has sidelength at most

$$\begin{aligned}\frac{\varepsilon}{\zeta_1 d}\cdot\max(\mathsf{r}, 2\|\mathsf{q}-\mathsf{c}\|) &\leq \frac{\varepsilon}{\zeta_1 d}\cdot\max\Big(6\mathsf{d}_k(\mathsf{q},\mathsf{P}), 12\mathsf{d}_k(\mathsf{q},\mathsf{P})\Big)\\ &\leq \frac{\varepsilon}{4d}\mathsf{d}_k(\mathsf{q},\mathsf{P}),\end{aligned}$$

by Eq. (1) and if $\zeta_1 \geq 48$. As such, $\mathsf{diam}(\square) \leq \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4$, and the claim follows by Lemma IV.3. ■

2) The general case:

Lemma IV.8. *The data-structure constructed above returns* $(1+\varepsilon)$*-approximation to* $\mathsf{d}_k(\mathsf{q},\mathsf{P})$*, for any query point* q*.*

Proof: Consider the query point q and its anchor point (c,r). By Lemma IV.6, we have $\mathsf{r} \leq 2\mathsf{d}_k(\mathsf{q},\mathsf{P})$ and $\|\mathsf{q}-\mathsf{c}\| \leq 3\mathsf{d}_k(\mathsf{q},\mathsf{P})$. This implies that

$$\mathsf{d}(\mathsf{q}',\mathcal{B}') \leq \|\mathsf{q}' - (\mathsf{c},\mathsf{r})\| \leq \|\mathsf{q}-\mathsf{c}\| + \mathsf{r} \leq 5\mathsf{d}_k(\mathsf{q},\mathsf{P}). \quad (3)$$

Let the returned point, which is a $(1+\varepsilon/8)$-ANN for q' in $\mathcal{B}'$, be $(\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q}) = \mathrm{nn}'(\square)$, where $\mathsf{q}' = (\mathsf{q},0)$. We have that $\|\mathsf{q}' - (\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})\| \leq (1+\varepsilon/8)\mathsf{d}(\mathsf{q}',\mathcal{B}') \leq 6\mathsf{d}_k(\mathsf{q},\mathsf{P})$. In particular, $\|\mathsf{q}-\mathsf{c}_\mathsf{q}\| \leq 6\mathsf{d}_k(\mathsf{q},\mathsf{P})$ and $\mathsf{r}_\mathsf{q} \leq 6\mathsf{d}_k(\mathsf{q},\mathsf{P})$.

Thus, if $\mathsf{r}_\mathsf{q} \geq \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4$ or $\mathsf{r} \geq \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4$ then we are done, by Lemma IV.7. Otherwise, We have

$$\|\mathsf{q}' - (\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})\| \leq (1+\varepsilon/8)\,\|\mathsf{q}' - (\mathsf{c},\mathsf{r})\|,$$

as $(\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})$ is a $(1+\varepsilon/8)$ approximation to $\mathsf{d}(\mathsf{q}',\mathcal{B}')$. As such,

$$\frac{\|\mathsf{q}' - (\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})\|}{1+\varepsilon/8} \leq \|\mathsf{q}' - (\mathsf{c},\mathsf{r})\| \leq \|\mathsf{q}-\mathsf{c}\| + \mathsf{r}. \quad (4)$$

As $\mathsf{ball}(\mathsf{c},\mathsf{r}) \cap \mathsf{ball}(\mathsf{q},\mathsf{d}_k(\mathsf{q},\mathsf{P})) \neq \emptyset$ we have, by the triangle inequality, that

$$\|\mathsf{q}-\mathsf{c}\| - \mathsf{r} \leq \mathsf{d}_k(\mathsf{q},\mathsf{P}). \quad (5)$$

By Eq. (4) and Eq. (5) we have

$$\frac{\|\mathsf{q}' - (\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})\|}{1+\varepsilon/8} - 2\mathsf{r} \leq \|\mathsf{q}-\mathsf{c}\| - \mathsf{r} \leq \mathsf{d}_k(\mathsf{q},\mathsf{P}).$$

By the above and as $\max(\mathsf{r},\mathsf{r}_\mathsf{q}) < \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4$, we have

$$\begin{aligned}&\|\mathsf{q}-\mathsf{c}_\mathsf{q}\| + \mathsf{r}_\mathsf{q}\\ &\leq \|\mathsf{q}' - (\mathsf{c}_\mathsf{q},\mathsf{r}_\mathsf{q})\| + \mathsf{r}_\mathsf{q}\\ &\leq (1+\varepsilon/8)(\mathsf{d}_k(\mathsf{q},\mathsf{P}) + 2\mathsf{r}) + \mathsf{r}_\mathsf{q}\\ &\leq (1+\varepsilon/8)(\mathsf{d}_k(\mathsf{q},\mathsf{P}) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/2) + \varepsilon\mathsf{d}_k(\mathsf{q},\mathsf{P})/4\\ &\leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q},\mathsf{P}).\end{aligned}$$

Since the algorithm returns for q a value that is at most $\|\mathsf{q}-\mathsf{c}_\mathsf{q}\| + \mathsf{r}_\mathsf{q}$ the result is correct. ■

C. The result

Theorem IV.9. *Given a set* P *of* n *points in* $\mathbb{R}^d$*, a number* $k \geq 1$ *such that* $k|n$*, and* $0 < \varepsilon$ *sufficiently small, one can preprocess* P*, in* $O\Big(n\log n + \frac{n}{k}C_\varepsilon \log n + \frac{n}{k}C'_\varepsilon\Big)$ *time, where* $C_\varepsilon = O\big(\varepsilon^{-d}\log\varepsilon^{-1}\big)$ *and* $C'_\varepsilon = O\big(\varepsilon^{-2d+1}\log\varepsilon^{-1}\big)$*. The space used by the data-structure is* $O(C_\varepsilon n/k)$*. This data structure answers* $(1+\varepsilon)$*-approximate* k*-nearest neighbor query in* $O\Big(\log\frac{n}{k\varepsilon}\Big)$ *time. The data-structure also returns a point of* P *that is approximately the desired* k *approximate nearest neighbor.*

Proof: Computing the quorum clustering takes time $O(n\log n)$ by Lemma II.4. It is easy to see that $|\mathcal{X}| = O\big(\frac{n}{k\varepsilon^d}\log\frac{1}{\varepsilon}\big)$. From the construction in [22] we have $|\mathcal{S}| = O\big(\frac{n}{k\varepsilon^d}\log\frac{1}{\varepsilon}\big)$ (note, that since we clip the construction to a hyperplane, we get $1/\varepsilon^d$ in the bound and not $1/\varepsilon^{d+1}$). A careful implementation of this stage takes time $O\left(n \log n + |\mathcal{W}|\left(\log n + \frac{1}{\varepsilon^{d-1}}\right)\right)$. Overlaying the two compressed quadtrees representing them takes linear time in their size, that is $O(|\mathcal{X}| + |\mathcal{S}|)$.

The most expensive remaining step is to perform the k approximate nearest neighbor query for each cell in the resulting decomposition of $\mathcal{W}$, see Eq. (2) (i.e., computing $\beta_k(\square_{\mathsf{rep}})$ for each cell $\square \in \mathcal{W}$). Using the data-structure of Section VI (see Theorem VI.3) each query takes $O\left(\log n + 1/\varepsilon^{d-1}\right)$ time (alternatively, we could use the data-structure of Arya *et al.* [40]), As such, this takes

$$O\left(n \log n + |\mathcal{W}|\left(\log n + \frac{1}{\varepsilon^{d-1}}\right)\right) =$$
$$O\left(n \log n + \frac{n}{k\varepsilon^{d}} \log \frac{1}{\varepsilon} \log n + \frac{n}{k\varepsilon^{2d-1}} \log \frac{1}{\varepsilon}\right)$$

time, and this bounds the overall construction time.

The query time is a point location query and is easily seen to take time $O\left(\log\left(\frac{n}{k\varepsilon}\right)\right)$.

Finally, one needs to argue that the returned point of P is indeed the desired approximate k-nearest neighbor. It follows by going through our correctness proof and applying it to the returned point that the distance to it is indeed a $1 + O(\varepsilon)$ approximation to the k-nearest neighbor distance. We omit the tedious but straightforward details of doing so. ■

1) Using a single point for each AVD *cell:* The AVD generated can be viewed as storing two points in each cell $\square$ of the AVD. These two points are in $\mathbb{R}^{d+1}$, and for a cell $\square$, they are

(i) the point $\mathrm{nn}'(\square) \in \mathcal{B}'$, and

(ii) the point $(\square_{\mathsf{rep}}, \beta_k(\square_{\mathsf{rep}}))$.

The algorithm for $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$ can be viewed as computing the nearest neighbor of $(\mathsf{q}, 0)$ to one of the above two points using the $\|\cdot\|_{\oplus}$ norm to define the distance. Furthermore, we can use the regular $\|\cdot\|$ to resolve which one of the points to use. Using standard AVD algorithms we can subdivide each such cell $\square$ into $O\left(1/\varepsilon^{d+1} \log \varepsilon^{-1}\right)$ cells to answer this query approximately. By using this finer subdivision we can have a single point inside each cell for which the closest distance is the approximation to $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$. This incurs an increase by a factor of $O\left(1/\varepsilon^{d+1} \log \varepsilon^{-1}\right)$ in the number of cells.

V. Density estimation

Given a point set $\mathsf{P} \subseteq \mathbb{R}^d$, and a query point $\mathsf{q} \in \mathbb{R}^d$ consider the point $\mathsf{v}(\mathsf{q}) = (\mathsf{d}_1(\mathsf{q}, \mathsf{P}), \ldots, \mathsf{d}_n(\mathsf{q}, \mathsf{P}))$. This is a point in $\mathbb{R}^n$ and several problems in computational geometry can be viewed as computing some interesting function of $\mathsf{v}(\mathsf{q})$. For example one could view the nearest neighbor distance as the function that projects along the first dimension. Another motivating example is a geometric version of discrete density measures from [31]. In their problem one is interested in computing $g_k(\mathsf{q}) = \sum_{i=1}^{k} \mathsf{d}_i(\mathsf{q}, \mathsf{P})$. In this section, we show that a broad class of functions (that include g_k), can be approximated to within $(1 \pm \varepsilon)$ by a data structure requiring space $\widetilde{O}(n/k)$.

A. Basic tools

Definition V.1. *A monotonic increasing function $f : \mathbb{R}^+ \to \mathbb{R}$ is said to be* **slowly growing** *if there is a constant $c > 0$ such that for ε sufficiently small we have $(1-\varepsilon)f(x) \le f((1-\varepsilon/c)x) \le f((1+\varepsilon/c)x) \le (1+\varepsilon)f(x)$, for all $x \in \mathbb{R}^+$. The constant c is the* **growth constant** *of f. The family of slowly growing functions is denoted by $\mathcal{F}_{\mathsf{sg}}$.*

It is easy to see that the class $\mathcal{F}_{\mathsf{sg}}$ includes polynomial functions, but it does not include, for example, the function e^x. We assume that given a x we can evaluate the function $f(x)$ in constant time. In this section, we show how we can use the AVD construction to approximate any function $\mathcal{F}_{k,f}(\cdot)$ that can be expressed as

$$\mathcal{F}_{k,f}(\mathsf{q}) = \sum_{i=1}^{k} f\left(\mathsf{d}_i(\mathsf{q}, \mathsf{P})\right),$$

where $f \in \mathcal{F}_{\mathsf{sg}}$. As $f(x) = x^2$ is slowly growing we have the function above $g_k = \mathcal{F}_{k,f}$. The proof of the following lemma appears in the full version [43].

Lemma V.2. *Let $f^{\approx}_{k,f}(\mathsf{q}) = \sum_{i=\lceil k\varepsilon/8 \rceil}^{k} f(\mathsf{d}_i(\mathsf{q}, \mathsf{P}))$. Then, for any query point q, we have that $f^{\approx}_{k,f}(\mathsf{q}) \le \mathcal{F}_{k,f}(\mathsf{q}) \le (1 + \varepsilon/4) f^{\approx}_{k,f}(\mathsf{q})$.*

The next lemma exploits a coreset idea, so that we have to evaluate only few terms of the summation.

Lemma V.3. *There is a set of indices $\mathcal{I} \subseteq \left\{\lceil k\varepsilon/8 \rceil, \ldots, k\right\}$, and integer weights $w_i \ge 0$, for $i \in \mathcal{I}$, such that:*

(A) $|\mathcal{I}| = O\left(\frac{\log k}{\varepsilon}\right)$.

(B) For any query point q, we have that $\mathcal{F}^{\approx}_{k,f}(\mathsf{q}) = \sum_{i \in \mathcal{I}} w_i f(\mathsf{d}_i(\mathsf{q}, \mathsf{P}))$ is a good estimate for $f^{\approx}_{k,f}(\mathsf{q})$; that is, $(1 - \varepsilon/4)\mathcal{F}^{\approx}_{k,f}(\mathsf{q}) \le f^{\approx}_{k,f}(\mathsf{q}) \le (1 + \varepsilon/4)\mathcal{F}^{\approx}_{k,f}(\mathsf{q})$.

Furthermore, the set $\mathcal{I}$ can be computed in $O(|\mathcal{I}|)$ time.

Proof: Given a query point q consider the function $g_{\mathsf{q}} : \{1, 2, \ldots, n\} \to \mathbb{R}^+$ defined as $g_{\mathsf{q}}(i) = f\left(\mathsf{d}_i(\mathsf{q}, \mathsf{P})\right)$. Clearly, since $f \in \mathcal{F}_{\mathsf{sg}}$, it follows that g_{q} is a monotonic increasing function. The existence of $\mathcal{I}$ follows from Lemma 3.2 of [44], when $(1 \pm \varepsilon/4)$-approximating the function $f^{\approx}_{k,f}(\mathsf{q}) = \sum_{i=\lceil k\varepsilon/8 \rceil}^{k} f(\mathsf{d}_i(\mathsf{q}, \mathsf{P}))$; that is, $(1 - \varepsilon/4)\mathcal{F}^{\approx}_{k,f}(\mathsf{q}) \le f^{\approx}_{k,f}(\mathsf{q}) \le (1 + \varepsilon/4)\mathcal{F}^{\approx}_{k,f}(\mathsf{q})$. ■

1) Performing point-location in several quadtrees simultaneously:

Lemma V.4. *Consider a rooted tree T with m nodes, where the nodes are colored by I colors (i.e., a node might have several colors). Overall, assume there are $O(m)$ pairs of such $(node, color)$ associations. One can preprocess the tree in $O(m)$ time and space, such that given a query leaf v of T, one can report the nodes $v_1, \ldots, v_I$ in $O(I)$ time. Here, v_i is the lowest node in the tree along the path from the root to v that is colored with color i.*

Proof: Let us start with the naive solution – perform a DFS on T, and keep an array $\mathcal{X}$ of I entries storing the latest node of each color encountered so far along the path from the root to the current node. Storing a snapshot of this array $\mathcal{X}$ at each node would require $O(mI)$ space. But then one can answer a query in $O(I)$ time. As such, the challenge is to reduce the required space.

To this end, interpret the DFS to be a Eulerian traversal of the tree. The traversal has length $2m - 2$, and every edge traveled contains updates to the array $\mathcal{X}$. Indeed, if the **DFS** traverses down from a node u to a child node w, the updates would be updating all the colors that are stored in w, to indicate that w is the lowest node for these colors. Similarly, if the **DFS** goes up from w to u, we restore all the colors stored in w to their value just before the **DFS** visited w. Now, the **DFS** traversal of T becomes a list of $O(m)$ updates. For each leaf we know its location in this list of updates, and we are interested in the last update before it in the list for each one of the I colors.

So, let L be this list of updates. At each kth update, for $k = tI$ for some integer t, store a snapshot of the array of colors as updated if we scan the list from the beginning till this point. Clearly, all these snapshots can be computed in $O(m)$ time, and require $O((m/I)I) = O(m)$ space.

Now, given a query leaf v, we go to its location in the list L, and jump back to the last snapshot stored. We copy this snapshot, and then we scan the list from the snapshot till v. This would require updating the array of colors at most $O(I)$ times, and can be done in $O(I)$ time overall. ■

Lemma V.5. *Given I compressed quadtrees $\mathcal{D}_1, \ldots, \mathcal{D}_I$ of total size m in $\mathbb{R}^d$, one can preprocess them in $O(m \log I)$ time, using $O(m)$ space, such that given a query point q, one can perform a point-location queries in all I quadtrees, simultaneously for q, in $O(\log m + I)$ time.*

Proof: Overlay all these compressed quadtrees together. Since we are overlaying together I quadtrees, and this is equivalent to merging I sorted lists [20], this takes $O(m \log I)$ time. Let $\mathcal{D}$ denote the resulting compressed quadtree. Note, that any node of $\mathcal{D}_i$, for $i = 1, \ldots, I$, must be a node in $\mathcal{D}$.

Given a query point q, we need to extract the I nodes in the original quadtrees $\mathcal{D}_i$, for $i = 1, \ldots, I$, that contain the query point (these nodes can be compressed nodes). So, let $\Box$ be the leaf node of $\mathcal{D}$ containing the query point q. Consider the path π from the root to the node of $\Box$. We are interested in the lowest node of π that belongs to $\mathcal{D}_i$, for $i = 1, \ldots, I$. To this end, color all the nodes of $\mathcal{D}_i$ that appear in $\mathcal{D}$ by color i, for $i = 1, \ldots, I$. Now, we build the data-structure of Lemma V.4 for $\mathcal{D}$. We can use this data-structure to answer the desired query in $O(I)$ time. ■

B. The data-structure

We are given $\mathsf{P} \subseteq \mathbb{R}^d$ set of n points, a function $f \in \mathcal{F}_{\mathsf{sg}}$, an integer $1 \le k \le n$, and $\varepsilon > 0$ sufficiently small. Here we describe how to build a data-structure to approximate $\mathcal{F}_{k,f}(\cdot)$.

1) Construction: In the following, let $\alpha = O(c)$ be a sufficiently large constant, where c is the growth constant of f (see Defnition V.1). Consider the coreset $\mathcal{I}$ from Lemma V.3. For each $i \in \mathcal{I}$ we compute, using Theorem IV.9, a data structure (i.e., a compressed quadtree) $\mathcal{D}_i$ for answering $(1 + \varepsilon/\alpha)$-approximate ith nearest neighbor distance queries for P. We then overlay all these quadtrees into a single quadtree, using Lemma V.5.

Answering a Query.: Given a query point q, we perform simultaneous point-location query in $\mathcal{D}_1, \ldots, \mathcal{D}_I$, by using $\mathcal{D}$, as described in Lemma V.5. This results in a $(1 + \varepsilon/4c)$ approximation z_i to $\mathsf{d}_i(\mathsf{q}, \mathsf{P})$, for $i \in \mathcal{I}$, and takes $O(\log m + I)$ time, where m is the size of $\mathcal{D}$, and $I = |\mathcal{I}|$. We return $z = \sum_{i \in \mathcal{I}} w_i f(z_i)$ where w_i are the weights associated with the members of the coreset from Lemma V.3.

Bounding the quality of approximation.: We only prove the upper bound on z. The proof for the lower bound is similar. As the z_i are $(1 \pm \varepsilon/4c)$ approximations to $\mathsf{d}_i(\mathsf{q}, \mathsf{P})$ we have, $(1 - \varepsilon/4c) z_i \le \mathsf{d}_i(\mathsf{q}, \mathsf{P})$, for $i \in \mathcal{I}$, and it follows from definitions that,

$$(1 - \varepsilon/4) w_i f\Bigl(z_i\Bigr) \le w_i f\Bigl((1 - \varepsilon/4c) z_i\Bigr) \le w_i f\Bigl(\mathsf{d}_i(\mathsf{q}, \mathsf{P})\Bigr),$$

for $i \in \mathcal{I}$. Therefore,

$$(1 - \varepsilon/4) z = (1 - \varepsilon/4) \sum_{i \in \mathcal{I}} w_i f(z_i) \le \sum_{i \in \mathcal{I}} w_i f\Bigl(\mathsf{d}_i(\mathsf{q}, \mathsf{P})\Bigr) = \mathcal{F}^{\approx}_{k,f}(\mathsf{q}). \tag{6}$$

Using Eq. (6) and Lemma V.3 it follows that,

$$(1 - \varepsilon/4)^2 z \le (1 - \varepsilon/4) \mathcal{F}^{\approx}_{k,f}(\mathsf{q}) \le f^{\approx}_{k,f}(\mathsf{q}). \tag{7}$$

Finally, by Eq. (7) and Lemma V.2 we have,

$$(1 - \varepsilon/4)^2 z \le f^{\approx}_{k,f}(\mathsf{q}) \le \mathcal{F}_{k,f}(\mathsf{q}).$$

Therefore we have, $(1 - \varepsilon) z \le (1 - \varepsilon/4)^2 z \le \mathcal{F}_{k,f}(\mathsf{q})$, as desired (i.e., this is equivalent to $z \le \mathcal{F}_{k,f}(\mathsf{q})/(1 - \varepsilon)$).

Preprocessing space and time analysis.: We have that $I = |\mathcal{I}| = O\bigl(\varepsilon^{-1} \log k\bigr)$. Let $C_x = O\bigl(x^{-d} \log x^{-1}\bigr)$. By Theorem IV.9 the total size of all the $\mathcal{D}_i$s (and thus the size of the resulting data-structure) is

$$S = \sum_{i \in \mathcal{I}} O\Bigl(C_{\varepsilon/\alpha} \frac{n}{i}\Bigr) = O\Bigl(C_{\varepsilon/\alpha} \frac{n \log k}{k \varepsilon^2}\Bigr). \tag{8}$$

Indeed, the maximum of the terms involving n/i is $O(n/k\varepsilon)$ and $I = O\bigl(\varepsilon^{-1} \log k\bigr)$. By Theorem IV.9 the total time taken to construct all the $\mathcal{D}_i$ is

$$\sum_{i \in \mathcal{I}} O\Bigl(n \log n + \frac{n}{i} C_{\varepsilon/\alpha} \log n + \frac{n}{i} C'_{\varepsilon/\alpha}\Bigr) =$$
$$O\Bigl(\frac{n \log n \log k}{\varepsilon} + \frac{n \log n \log k}{k \varepsilon^2} C_{\varepsilon/\alpha} + \frac{n \log k}{k \varepsilon^2} C'_{\varepsilon/\alpha}\Bigr),$$

where $C'_x = O\bigl(x^{-2d+1} \log x^{-1}\bigr)$. The time to construct the final quadtree is $O(S \log I)$, but this is subsumed by the construction time above.

2) The result: Summarizing the above, we get the following result.

Theorem V.6. *For P be a set of n points in $\mathbb{R}^d$. Given any $f \in \mathcal{F}_{\mathsf{sg}}$, an integer $1 \le k \le n$ and $\varepsilon > 0$, one can build a data-structure to approximate $\mathcal{F}_{k,f}(\cdot)$. Specifically, we have:*

(A) *The construction time is $O(C_1 n \log n \log k)$, where $C_1 = O\bigl(\varepsilon^{-2d-1} \log \varepsilon^{-1}\bigr)$.*

(B) *The space used is $O\Bigl(C_2 \dfrac{n}{k} \log k\Bigr)$, where $C_2 = O\bigl(\varepsilon^{-d-2} \log \varepsilon^{-1}\bigr)$.*

(C) *For any query point q, the data-structure computes a number z, such that $(1 - \varepsilon) z \le \mathcal{F}_{k,f}(\mathsf{q}) \le (1 + \varepsilon) z$.*

(D) *The query time is $O\Bigl(\log n + \dfrac{\log k}{\varepsilon}\Bigr)$.*

(The O notation here hides constants that depends on f.)

VI. ANN QUERIES WHERE k AND ε ARE PART OF THE QUERY

We present here a data-structure with query time $O\big(\log n + 1/\varepsilon^{d-1}\big)$ that does not require knowing either k or ε during the preprocessing stage, and both are specified during query time (together with the query point). Unlike our main result, this data-structure requires linear space. Previous data-structures required knowing ε in advance [40].

A. Rough approximation

Observe that a fast constant approximation to $\mathsf{d}_k(\mathsf{q}, \mathsf{P})$ is implied by Theorem III.2 if k is known in advance. We next describe a polynomial approximation when k is not available during preprocessing. The proof of the following lemma appears in the full version [43].

Lemma VI.1. *Given a set* P *of* n *points in* $\mathbb{R}^d$*, one can preprocess it, in* $O(n \log n)$ *time, such that given any query point* q *and* $1 \leq k \leq n$*, one can find a number* R *satisfying,* $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq R \leq n^c \mathsf{d}_k(\mathsf{q}, \mathsf{P})$ *in* $O(\log n)$ *time. The result is correct with high probability i.e. at least* $1 - 1/n^{c-2}$ *for any constant* $c \geq 4$*.*

The following lemma, whose proof appears in the full version [43], refines this approximation.

Lemma VI.2. *Given a set* P *of* n *points in* $\mathbb{R}^d$*, one can preprocess it, in* $O(n \log n)$ *time, so that given a query point* q *and an estimate* R *satisfying* $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq R \leq n^{O(1)} \mathsf{d}_k(\mathsf{q}, \mathsf{P})$*, then one can output a number* β *satisfying,* $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq \beta \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P})$*, in* $O\big(\log n + 1/\varepsilon^{d-1}\big)$ *time. Furthermore, one can return a point* $\mathsf{p} \in \mathsf{P}$ *such that* $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq \|\mathsf{q} - \mathsf{p}\| \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P})$*.*

B. The result

Theorem VI.3. *Given a set of* n *points* P *in* $\mathbb{R}^d$*, one can preprocess them, in* $O(n \log n)$ *time, into a data structure of size* $O(n)$*, such that given a query point* q*, an integer* $1 \leq k \leq n$ *and a* $\varepsilon > 0$ *one can compute, in* $O\big(\log n + 1/\varepsilon^{d-1}\big)$ *time, a number* β *such that* $\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq \beta \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P})$*. The data-structure also returns a point* $\mathsf{p} \in \mathsf{P}$ *such that* $(1-\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P}) \leq \|\mathsf{q} - \mathsf{p}\| \leq (1+\varepsilon)\mathsf{d}_k(\mathsf{q}, \mathsf{P})$*.*

C. A generalization – Weighted version of k ANN

We consider a generalization of the k ANN problem where as usual we are given a set of points $\mathsf{P} \subseteq \mathbb{R}^d$, weights $w_\mathsf{p} \geq 0$ for each $\mathsf{p} \in \mathsf{P}$ and a number $\varepsilon > 0$. A query consists of a point q and a number $x \geq 0$ and we are required to output a distance d, where $d \leq (1+\varepsilon)D$. Where D is the smallest distance such that such that

$$\sum_{\|\mathsf{q}-\mathsf{p}\| \leq D} w_\mathsf{p} \geq x.$$

The usual k ANN problem is just the specialization with $w_\mathsf{p} = 1$ for all p and $x = k$. We remark that the algorithm for k ANN presented in this section, works with minor changes for this generalization as well, even when ε is supplied with the query point. Furthermore, the run time and space complexity remain unchanged and do not depend on the weights w_p.

VII. CONCLUSIONS

In this paper, we presented a data-structure for answering approximate k nearest neighbor queries in $\mathbb{R}^d$ (here d is a constant). Our data-structure has the surprising property that the space required is $\widetilde{O}(n/k)$. It is easy to verify that up to noise this is the best one can do for this problem. This data-structure also suggests a natural way of compressing geometric data, such that the resulting sketch can be used to answer meaningful proximity queries on the original data. We then used this data-structure to answer various proximity queries using roughly the same space and query time.

We also presented a data-structure for answering such queries where both k and ε are specified during query time. This data-structure is simple and practical.

There are many interesting questions for further research.

(A) In the vein of the authors recent work [45], it is to easy to verify that our results extends in a natural way to metrics of low doubling dimensions ([45] describes what an approximate Voronoi diagram is for doubling metric). It also seems believable that the result would extend to the problem where the data is high dimensional but the queries arrive from a low dimensional manifold.

(B) It is natural to ask what one can do for this problem in high dimensional Euclidean space. In particular, can one get query time close to the one required for approximate nearest neighbor [17]. Of particular interest is getting a query time that is sublinear in k and n while having subquadratic space and preprocessing time.

(C) The dependency on ε in our data-structures is probably not optimal. One can probably get space/time tradeoffs, as done by Arya *et al.* [24].

ACKNOWLEDGMENT

The authors thank Pankaj Agarwal and Kasturi Varadarajan for useful discussions on the problems studied in this paper.

REFERENCES

[1] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.

[2] B. Chazelle, "Technical perspective: finding a good neighbor, near and fast," *Commun. ACM*, vol. 51, no. 1, p. 115, 2008.

[3] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, pp. 117–122, 2008.

[4] K. L. Clarkson, "Nearest-neighbor searching and metric space dimensions," in *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, G. Shakhnarovich, T. Darrell, and P. Indyk, Eds. The MIT Press, 2006, pp. 15–59.

[5] E. Fix and J. Hodges, "Discriminatory analysis. nonparametric discrimination: Consistency properties," *Technical Report 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, TX*, 1949.

[6] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[7] T. Kohonen, *Self-Organizing Maps*, ser. Springer Series in Information Sciences. Springer, Berlin, 2001, vol. 30.

[8] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” *Commun. ACM*, vol. 18, pp. 613–620, 1975.

[9] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1991.

[10] L. Devroye and T. Wagner, “Handbook of statistics,” in *Nearest neighbor methods in discrimination*, P. R. Krishnaiah and L. N. Kanal, Eds. North-Holland, 1982, vol. 2.

[11] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: Wiley-Interscience, 2001.

[12] K. L. Clarkson, “A randomized algorithm for closest-point queries,” *SIAM J. Comput.*, vol. 17, pp. 830–847, 1988.

[13] P. K. Agarwal and J. Matoušek, “Ray shooting and parametric search,” *SIAM J. Comput.*, vol. 22, pp. 540–570, 1993.

[14] S. Meiser, “Point location in arrangements of hyperplanes,” *Inform. Comput.*, vol. 106, pp. 286–303, 1993.

[15] M. Minsky and S. Papert, *Perceptrons*. The MIT Press, 1969.

[16] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions,” *J. Assoc. Comput. Mach.*, vol. 45, no. 6, pp. 891–923, 1998. [Online]. Available: http://www.cs.umd.edu/ mount/Papers/dist.pdf

[17] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” in *Proc. 30th Annu. ACM Sympos. Theory Comput.*, 1998, pp. 604–613. [Online]. Available: http://theory.lcs.mit.edu/ indyk/nndraft.ps

[18] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, “Efficient search for approximate nearest neighbor in high dimensional spaces,” *SIAM J. Comput.*, vol. 2, no. 30, pp. 457–474, 2000. [Online]. Available: http://epubs.siam.org/sam-bin/dbq/article/34717

[19] S. Har-Peled, P. Indyk, and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” *Theory Comput.*, p. to appear, 2012.

[20] S. Har-Peled, *Geometric Approximation Algorithms*. Amer. Math. Soc., 2011.

[21] ——, “A replacement for Voronoi diagrams of near linear size,” in *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, 2001, pp. 94–103. [Online]. Available: http://cs.uiuc.edu/ sariel/papers/01/avoronoi/

[22] S. Arya and T. Malamatos, “Linear-size approximate Voronoi diagrams,” in *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, 2002, pp. 147–155.

[23] 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.*, vol. 42, pp. 67–90, 1995.

[24] S. Arya, T. Malamatos, and D. M. Mount, “Space-time tradeoffs for approximate nearest neighbor searching,” *J. Assoc. Comput. Mach.*, vol. 57, no. 1, pp. 1–54, 2009.

[25] B. Silverman, *Density estimation for statistics and data analysis*, ser. Monographs on statistics and applied probability. Chapman and Hall, 1986.

[26] J. Ruppert, “A Delaunay refinement algorithm for quality 2-dimensional mesh generation,” *J. Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.

[27] J. B. Tenenbaum, “Mapping a manifold of perceptual observations,” *Adv. Neur. Inf. Proc. Sys. 10*, pp. 682–688, 1998.

[28] M. Bernstein, V. D. Silva, J. C. Langford, and J. B. Tenenbaum, “Graph approximations to geodesics on embedded manifolds,” 2000.

[29] T. Martinetz and K. Schulten, “Topology representing networks,” *Neural Netw.*, vol. 7, no. 3, pp. 507–522, Mar. 1994.

[30] K. Q. Weinberger and L. K. Saul, “Unsupervised learning of image manifolds by semidefinite programming,” *Int. J. Comput. Vision*, vol. 70, no. 1, pp. 77–90, Oct. 2006.

[31] L. J. Guibas, Q. Mérigot, and D. Morozov, “Witnessed k-distance,” in *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, 2011, pp. 57–64.

[32] F. Aurenhammer, “Voronoi diagrams: A survey of a fundamental geometric data structure,” *ACM Comput. Surv.*, vol. 23, pp. 345–405, 1991.

[33] K. L. Clarkson and P. W. Shor, “Applications of random sampling in computational geometry, II,” *Discrete Comput. Geom.*, vol. 4, pp. 387–421, 1989. [Online]. Available: http://cm.bell-labs.com/who/clarkson/rs2m.html

[34] J. Matoušek, *Lectures on Discrete Geometry*. Springer, 2002. [Online]. Available: http://kam.mff.cuni.cz/ matousek/dg.html

[35] ——, “Efficient partition trees,” *Discrete Comput. Geom.*, vol. 8, pp. 315–334, 1992.

[36] T. M. Chan, “Optimal partition trees,” in *Proc. 26th Annu. ACM Sympos. Comput. Geom.*, 2010, pp. 1–10.

[37] P. K. Agarwal and J. Erickson, “Geometric range searching and its relatives,” in *Advances in Discrete and Computational Geometry*, B. Chazelle, J. E. Goodman, and R. Pollack, Eds. Amer. Math. Soc., 1998.

[38] S. Har-Peled, “Constructing approximate shortest path maps in three dimensions,” *SIAM J. Comput.*, vol. 28, no. 4, pp. 1182–1197, 1999.

[39] P. Carmi, S. Dolev, S. Har-Peled, M. J. Katz, and M. Segal, “Geographic quorum systems approximations,” *Algorithmica*, vol. 41, no. 4, pp. 233–244, 2005.

[40] S. Arya, T. Malamatos, and D. M. Mount, “Space-time tradeoffs for approximate spherical range counting,” in *Proc. 16th ACM-SIAM Sympos. Discrete Algorithms*, 2005, pp. 535–544.

[41] B. Aronov and S. Har-Peled, “On approximating the depth and related problems,” *SIAM J. Comput.*, vol. 38, no. 3, pp. 899–921, 2008.

[42] M. de Berg, H. Haverkort, S. Thite, and L. Toma, “Star-quadtrees and guard-quadtrees: I/o-efficient indexes for fat triangulations and low-density planar subdivisions,” *Comput. Geom. Theory Appl.*, vol. 43, pp. 493–513, jul 2010.

[43] S. Har-Peled and N. Kumar, “Down the rabbit hole: Robust proximity search and density estimation in sublinear space,” *CoRR*, vol. abs/1111.2942, 2011.

[44] S. Har-Peled, “Coresets for discrete integration and clustering,” in *Proc. 26th Conf. Found. Soft. Tech. Theoret. Comput. Sci.*, 2006, pp. 33–44.

[45] S. Har-Peled and N. Kumar, “Approximate nearest neighbor search for low dimensional queries,” in *Proc. 22nd ACM-SIAM Sympos. Discrete Algorithms*, 2011, pp. 854–867.