2013 IEEE 54th Annual Symposium on Foundations of Computer Science

Approximating Minimization Diagrams and Generalized Proximity Search

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*—We investigate the classes of functions whose minimization diagrams can be approximated efficiently in $\mathbb{R}^d$. We present a general framework and a data-structure that can be used to approximate the minimization diagram of such functions. The resulting data-structure has near linear size and can answer queries in logarithmic time. Applications include approximating the Voronoi diagram of (additively or multiplicatively) weighted points. Our technique also works for more general distance functions, such as metrics induced by convex bodies, and the nearest furthest-neighbor distance to a set of point sets. Interestingly, our framework works also for distance functions that do not obey the triangle inequality. For many of these functions no near-linear size approximation was known before.**

I. Introduction

Let $\mathcal{F} = \left\{ f_i : \mathbb{R}^d \to \mathbb{R} \;\middle|\; i = 1, \ldots, n \right\}$, be a set of functions. The minimization diagram of $\mathcal{F}$ is the function $f_{\min}(\mathsf{q}) = \min_{i=1,\ldots,n} f_i(\mathsf{q})$, for any $\mathsf{q} \in \mathbb{R}^d$. By viewing the graphs of these functions as manifolds in $\mathbb{R}^{d+1}$, the graph of the minimization diagram, also known as the ***lower envelope*** of $\mathcal{F}$, is the manifold that can be viewed from an observer at $-\infty$ on the x_{d+1} axis. Given a set of functions $\mathcal{F}$ as above, many problems in Computational Geometry can be viewed as computing the minimization diagram; that is, one preprocesses $\mathcal{F}$, and given a query point q, one needs to compute $f_{\min}(\mathsf{q})$ quickly. This typically requires $n^{O(d)}$ space if one is interested in logarithmic query time. If one is restricted to using linear space, then the query time deteriorates to $O\left(n^{1-O(1/d)}\right)$ [2], [3]. There is substantial work on bounding the complexity of the lower envelope in various cases, how to compute it efficiently, and performing range search on them; see the book by Sharir and Agarwal [4].

Nearest neighbor.

One natural problem that falls into this framework is the nearest neighbor (NN) search problem. Here, given a set P of n data points in a metric space $\mathcal{X}$, we need to preprocess P, such that given a query point $\mathsf{q} \in \mathcal{X}$, one can find (quickly) the point $\mathsf{n}_\mathsf{q} \in \mathsf{P}$ closest to q. Nearest neighbor search is a fundamental task used in numerous domains including machine learning, clustering, document retrieval, databases, statistics, and many others.

*Work on this paper was partially supported by NSF AF award CCF-0915984, and NSF AF award CCF-1217462. The full version of this paper is available at [1].

To see the connection to lower envelopes, consider a set of data points $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ in $\mathbb{R}^d$. Next, consider the set of functions $\mathcal{F} = \{f_1, \ldots, f_n\}$, where $f_i(\mathsf{q}) = \|\mathsf{q} - \mathsf{p}_i\|$, for $i = 1, \ldots, n$. The graph of f_i is the set of points $\left\{(\mathsf{q}, f_i(\mathsf{q})) \mid \mathsf{q} \in \mathbb{R}^d\right\}$ (which is a cone in $\mathbb{R}^{d+1}$ with apex at $(\mathsf{p}_i, 0)$). Clearly the NN problem is to evaluate the minimization diagram of the functions at a query point q.

More generally, given a set of n functions, one can think of the minimization diagram defining a "distance function", by analogy with the above. The distance of a query point here is simply the "height" of the lower envelope at that point.

Exact nearest neighbor.

The exact nearest neighbor problem has a naive linear time algorithm without any preprocessing. However, by doing some nontrivial preprocessing, one can achieve a sub-linear query time. In $\mathbb{R}^d$, this is facilitated by answering point location queries using a Voronoi diagram [5]. However, this approach is only suitable for low dimensions, as the complexity of the Voronoi diagram is $\Theta\left(n^{\lceil d/2 \rceil}\right)$ in the worst case. Specifically, Clarkson [6] showed a data-structure with query time $O(\log n)$ time, 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 trade-off the space used and the query time [7]. Meiser [8] 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 $O\left(n^{d+\delta}\right)$. These solutions are impractical even for data-sets of moderate size if the dimension is larger than two.

Approximate nearest neighbor.

In typical applications however, it is usually sufficient to return an ***approximate nearest neighbor*** (**ANN**). Given an $\varepsilon > 0$, a $(1+\varepsilon)$-ANN, to a query point q, is a point $y \in \mathsf{P}$, such that

$$\|\mathsf{q} - y\| \leq (1+\varepsilon) \|\mathsf{q} - \mathsf{n}_\mathsf{q}\|,$$

where $\mathsf{n}_\mathsf{q} \in \mathsf{P}$ is the nearest neighbor to q in P. Considerable amount of work was done on this problem, see [9] and references therein.

Indyk and Motwani showed that ANN queries can be reduced to a small number of near neighbor queries [10], [11]. For high dimensional Euclidean spaces, they used locality sensitive hashing to solve the near neighbor problem and provided a data-structure that answers ANN queries in time (roughly) $\widetilde{O}\left(n^{1/(1+\varepsilon)}\right)$ with preprocessing time and space

0272-5428/13 $26.00 © 2013 IEEE
DOI 10.1109/FOCS.2013.82

$\widetilde{O}\left(n^{1+1/(1+\varepsilon)}\right)$; here the $\widetilde{O}(\cdot)$ hides terms polynomial in $\log n$ and $1/\varepsilon$. This was improved to $\widetilde{O}\left(n^{1/(1+\varepsilon)^2}\right)$ query time, and preprocessing time and space $\widetilde{O}\left(n^{1+1/(1+\varepsilon)^2}\right)$ [12]. These bounds are near optimal [13].

In low dimensions (i.e., $\mathbb{R}^d$ for small d), one can use linear space (independent of ε) and get ANN query time $O(\log n + 1/\varepsilon^{d-1})$ [14], [15]. The trade-off for this logarithmic query time is of course an exponential dependence on d. Interestingly, for this data-structure, the approximation parameter ε is not prespecified during the construction; one needs to provide it only during the query. An alternative approach is to use Approximate Voronoi Diagrams (AVD), introduced by Har-Peled [16], which is a partition of space into regions, of near-linear total complexity, typically 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 $O\big((n/\varepsilon^d)\log^2 n\big)$, such that ANN queries can be answered in $O(\log n)$ time. Arya and Malamatos [17] showed how to build AVD's of linear complexity (i.e., $O(n/\varepsilon^d)$). Their construction uses Well-Separated Pair Decomposition [18]. Further trade-offs between query time and space for AVD's were studied by Arya *et al.* [19].

Generalized distance functions: motivation.

The algorithms for approximate nearest neighbor, extend to various metrics in $\mathbb{R}^d$, for example the well known ℓ_p metrics. In particular, previous constructions of AVD's extend to ℓ_p metrics [16], [17] as well. However, these constructions fail even for a relatively simple and natural extension; specifically, multiplicative weighted Voronoi diagrams. Here, every site p, in the given point set P, has a weight ω_{p}, and the "distance" of a query point q to p is $f_{\mathsf{p}}(\mathsf{q}) = \omega_{\mathsf{p}} \|\mathsf{q}-\mathsf{p}\|$. The function f_{p} is the natural distance function induced by p. As with ordinary Voronoi diagrams, one can define the weighted Voronoi diagram as a partition of space into disjoint regions, one for each site p, such that in the region for p the function f_{p} is the one realizing the minimum among all the functions induced by the points of P. It is known that, even in the plane, multiplicative Voronoi diagrams can have quadratic complexity, and the minimizing distance function usually does not obey the triangle inequality. Intuitively, such multiplicative Voronoi diagrams can be used to model facilities where the price of delivery to a client depends on the facility and the distance. Of course, this is only one possible distance function, and there are many other such functions that are of interest (e.g., multiplicative, additive, etc.).

When fast proximity and small space is not possible.

Consider a set of segments in the plane, and we are interested in the nearest segment to a query point. Given n such segments and n such query points, this is an extension of Hopcroft's problem, which requires only to decide if there is any of the given points on any of the segments. There are lower bounds (in reasonable models) that show that Hopcroft's problem cannot be solved faster than $\Omega\left(n^{4/3}\right)$ time [20]. This implies that no multiplicative-error approximation for proximity search in this case is possible, if one insists on near linear preprocessing, and logarithmic query time.

When is fast ANN possible.

So, consider a set of geometric objects where each one of them induces a natural distance function, measuring how far a point in space is from this object. Given such a collection of functions, the nearest neighbor for a query point is simply the function that defines the lower envelope "above" the query point (i.e., the object closest to the query point under its distance function). Clearly, this approach allows a generalization of the proximity search problem. In particular, the above question becomes, for what classes of functions, can the lower envelope be approximated up to $(1+\varepsilon)$-multiplicative error, in logarithmic time? Here the preprocessing space used by the data structure should be near linear.

I-A. Our results

We characterize the conditions that are sufficient to efficiently approximate the minimization diagram of functions. Using this framework, one can quickly evaluate the lower envelope approximately for large classes of functions that arise naturally from proximity problems. Our data-structure can be constructed in near linear time, uses near linear space, and answers proximity queries in logarithmic time (for constant dimension d). Our framework is quite general and should be applicable to many distance functions, and in particular we present the following specific cases where the new data-structure can be used:

(A) **Multiplicative Voronoi diagrams.** Given a set of points P, where the ith point p_i has associated weight $w_i > 0$, for $i = 1, \ldots, n$, consider the functions $f_i(\mathsf{q}) = w_i \|\mathsf{q}-\mathsf{p}_i\|$. The minimization diagram for this set of functions, corresponds to the multiplicative weighted Voronoi diagram of the points. The approach of Arya and Malamatos [17] to construct AVD's using WSPD's fails for this problem, as that construction relies on the triangle inequality that the regular Euclidean distance possesses, which does not hold in this case.
We provide a near linear space AVD construction for this case. We are unaware of any previous results on AVD for multiplicatively weighted Voronoi diagrams.

(B) **Minkowski norms of fat convex bodies.** Given a bounded symmetric convex body C centered at the origin, it defines a natural metric; that is, for points u and v their distance, as induced by C, denoted by $\|\mathsf{u}-\mathsf{v}\|_C$, is the minimum x such that $xC + \mathsf{u}$ contains v. So, given a set of n data points $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$ and n centrally symmetric and bounded convex bodies $C_1, \ldots, C_n$, we define $f_i(\mathsf{q}) = \|\mathsf{p}_i - \mathsf{q}\|_{C_i}$, for $i = 1, \ldots, n$. Since each point induces a distance by a different convex body, this collection no longer defines a metric, and this makes the problem significantly more challenging. In particular, existing techniques for AVD and ANN cannot be readily applied. Intuitively, the fatness of the associated convex bodies turns out to be sufficient to approximate the associated distance function, see Section IV-B.

(C) **Nearest furthest-neighbor.** Consider a situation where the given input is uncertain; specifically, for the

ith point we are given a set of points $\mathsf{P}_i \subseteq \mathbb{R}^d$ where it might lie (the reader might consider the case where the ith point randomly chooses its location out of the points of P_i). There is a growing interest in how to handle such inputs, as real world measurements are fraught with uncertainty, see [21], [22], [23], [24] and references therein. In particular, in the worst case, the distance of the query point q to the ith point, is the distance from q to the furthest-neighbor of q in P_i; that is, $\mathcal{F}_i(\mathsf{q}) = \max_{\mathsf{p}\in\mathsf{P}_i} \|\mathsf{q}-\mathsf{p}\|$. Thus, in the worst case, the nearest point to the query is $\mathcal{F}(\mathsf{q}) = \min_i \mathcal{F}_i(\mathsf{q})$. Using our framework we can approximate this function efficiently, using space $\widetilde{O}(n)$, and providing logarithmic query time. Note that surprisingly, the space requirement is independent of the original input size, and only depends on the number of uncertain points.

Paper organization.

In Section II we define our framework and prove some basic properties. Since we are trying to make our framework as inclusive as possible, its description is somewhat abstract. In Section III, we describe the construction of the AVD and its associated data-structure. We describe in Section IV some specific cases where the new AVD construction can be used. We conclude in Section V.

II. Preliminaries

For the sake of simplicity of exposition, throughout the paper we assume that all the "action" takes place in the unit cube $[0,1]^d$. Among other things, this implies that all the queries are in this region. This can always be guaranteed by an appropriate scaling and translation of space. The scaling and translation, along with the conditions on functions in our framework, implies that outside the unit cube the approximation to the lower envelope can be obtained in constant time.

II-A. Notations and basic definitions

Given $\mathsf{q} \in \mathbb{R}^d$ and $\mathsf{P} \subseteq \mathbb{R}^d$ a non-empty closed set, the ***distance*** of q to P is $\mathsf{d}(\mathsf{q},\mathsf{P}) = \min_{x\in\mathsf{P}} \|\mathsf{q}-x\|$. For a number $\ell > 0$, the ***grid*** of side-length ℓ, denoted by G_ℓ, is the natural tiling of $\mathbb{R}^d$, with cubes of side-length ℓ (i.e., with a vertex at the origin). A cube $\square$ is ***canonical*** if it belongs to G_ℓ, ℓ is a power of 2, and $\square \subseteq [0,1]^d$. Informally, a canonical cube (or cell) is a region that might correspond to a cell in a quadtree having the unit cube as the root region.

Definition II.1. *To approximate a set $X \subseteq [0,1]^d$, up to distance r, consider the set $\mathsf{G}_{\approx r}(X)$ of all the canonical grid cells of G_ℓ that have a non-empty intersection with X, where $\ell = 2^{\lfloor \log_2(r/\sqrt{d}) \rfloor}$. Let $\cup\mathsf{G}_{\approx r}(X) = \bigcup_{\square\in\mathsf{G}_{\approx r}(X)} \square$, denote the union of cubes of $\mathsf{G}_{\approx r}(X)$.*

Observe that $X \subseteq \cup\mathsf{G}_{\approx r}(X) \subseteq X \oplus \mathsf{B}(0,r)$, where $\oplus$ denotes the Minkowski sum, and $\mathsf{B}(0,r)$ is the ball of radius r centered at the origin.

Definition II.2. *For $\ell \geq 0$ and a function $f : \mathbb{R}^d \to \mathbb{R}$, the ℓ **sublevel set** of f is the set $f_{\preceq\ell} = \left\{ \mathsf{p} \in \mathbb{R}^d \;\middle|\; f(\mathsf{p}) \leq \ell \right\}$. For a set of functions $\mathcal{F}$, let $\mathcal{F}_{\preceq\ell} = \bigcup_{f\in\mathcal{F}} f_{\preceq\ell}$.*

Definition II.3. *Given a function f and $\mathsf{q} \in \mathbb{R}^d$ their distance is $\mathrm{d}(\mathsf{q}, f) = f(\mathsf{q})$. Given two functions f and g, their **distance** $\mathrm{d}(f,g)$ is the minimum $l \geq 0$ such that $f_{\preceq l} \cap g_{\preceq l} \neq \emptyset$. Similarly, for two sets of function, $\mathcal{F}$ and $\mathcal{G}$, their **distance** is*

$$\mathrm{d}(\mathcal{F},\mathcal{G}) = \min_{f\in\mathcal{F}, g\in\mathcal{G}} \mathrm{d}(f,g)\,.$$

The distance function behaves to some extent like one would expect an usual metric to behave: (i) $\mathrm{d}(f,g)$ always exists, and (ii) (symmetry) $\mathrm{d}(f,g) = \mathrm{d}(g,f)$, Also, we have $f_{\preceq \mathrm{d}(f,g)} \neq \emptyset$. We extend the above definition to sets of functions. Note that the triangle inequality does not hold for $\mathrm{d}(\cdot,\cdot)$.

Observation II.4. *Suppose that f and g are two functions such that $\mathrm{d}(f,g) > 0$ and $\mathsf{q} \in \mathbb{R}^d$. Then, $\max(\mathrm{d}(\mathsf{q},f), \mathrm{d}(\mathsf{q},g)) \geq \mathrm{d}(f,g)$.*

Definition II.5. *Let $B_1, B_2, \ldots, B_m$ be n connected, nonempty sets in $\mathbb{R}^d$. This collection of sets is **connected** if $\cup_i B_i$ is connected.*

II-A1. Sketches

A key idea underlying our approach is that is that any set of functions of interest should look like a single (or a small number of functions) from "far" enough. Indeed, given a set of points $\mathsf{P} \subseteq \mathbb{R}^d$, they look like a single point (as far as distance), if the distance from $\mathcal{CH}(\mathsf{P})$ is at least $2\mathsf{diam}(\mathsf{P})/\varepsilon$.

Definition II.6 ($\mathsf{cl}(\mathcal{F})$). *Given a set of functions $\mathcal{G}$, if $\mathcal{G}$ contains a single function then the **connectivity level** $\mathsf{cl}(\mathcal{G})$ is 0; otherwise, it is the minimum $\ell \geq 0$, such that the collection of sets $f_{\preceq\ell}$ for $f \in \mathcal{G}$ is connected, see Definition II.5.*

Remark II.7. *It follows from Definition II.6 that at level $\ell = \mathsf{cl}(\mathcal{G})$, each of the sets $f_{\preceq\ell}$ for $f \in \mathcal{G}$ are nonempty and connected and further their union $\mathcal{G}_{\preceq\ell}$ is also connected. This can be relaxed to require that the intersection graph of the sets $f_{\preceq\ell}$ for $f \in \mathcal{G}$ is connected (this also implies they are nonempty). Notice that, if at level ℓ, the sublevel sets are connected, then the relaxed definition is equivalent to Definition II.6. However, the relaxed definition introduces more technical baggage, and for all the interesting applications we have, the sublevel sets $f_{\preceq y}$ are connected at all levels y at which they are nonempty. Therefore, in the interest of brevity, and to keep the presentation simple, we mandate that the sublevel sets be connected at ℓ. In fact, it would not harm to assume that the sublevel sets are connected whenever nonempty.*

Definition II.8. *Given a set of functions $\mathcal{G}$ and $\delta \geq 0, y_0 \geq 0$, a **$(\delta, y_0)$-sketch** for $\mathcal{G}$ is a (hopefully small) subset $\mathcal{H} \subseteq \mathcal{G}$, such that $\mathcal{G}_{\preceq y} \subseteq \mathcal{H}_{\preceq(1+\delta)y}$, for all $y \geq y_0$.*

It is easy to see that for any $\mathcal{G}, \delta \geq 0, y_0 \geq 0$, if $\mathcal{H} \subseteq \mathcal{G}$ is a (δ, y_0)-sketch, then for any $\delta' \geq \delta, y_0' \geq y_0, \mathcal{H}' \supseteq \mathcal{H}$ it is true that $\mathcal{H}'$ is a (δ', y_0')-sketch for $\mathcal{G}$. Trivially, for any $\delta \geq 0, y_0 \geq 0$, it is true that $\mathcal{H} = \mathcal{G}$ is a (δ, y_0)-sketch.

II-B. Conditions on the functions

We require that the set of functions under consideration satisfy the following conditions.

(P1) **Compactness.** For any $y \geq 0$ and $i = 1, \ldots, n$, the set $(f_i)_{\preceq y}$ is compact.

(P2) **Bounded growth.** For any $f \in \mathcal{F}$, there is a function $\lambda_f : \mathbb{R}^+ \to \mathbb{R}^+$, called the ***growth function***, such that for any $y \geq 0$ and $\varepsilon > 0$, if $f_{\preceq y} \neq \emptyset$, then $\lambda_f(y) \geq \mathsf{diam}(f_{\preceq y})/\zeta$, where ζ is an absolute constant, the ***growth constant***, depending only on the family of functions and not on n and such that if $\mathsf{q} \in \mathbb{R}^d$ with $\mathsf{d}(\mathsf{q}, f_{\preceq y}) \leq \varepsilon\lambda_f(y)$, then $f(\mathsf{q}) \leq (1+\varepsilon)y$. This is equivalent to $f_{\preceq y} \oplus \mathsf{B}(0, \varepsilon\lambda_f(y)) \subseteq f_{\preceq(1+\varepsilon)y}$, where $\mathsf{B}(\mathsf{u}, r)$ is the ball of radius r centered at u.

(P3) **Existence of a sketch.** Given $\delta > 0$ and a subset $\mathcal{G} \subseteq \mathcal{F}$, there is a $\mathcal{H} \subseteq \mathcal{G}$ with $|\mathcal{H}| = O\left(1/\delta^{\mathsf{c}_{\mathsf{sk}}}\right)$ and $y_0 = O\big(\mathsf{cl}(\mathcal{G})\big(|\mathcal{G}|/\delta\big)^{\mathsf{c}_{\mathsf{sk}}}\big)$ such that, $\mathcal{H}$ is a (δ, y_0)-sketch, where c_{sk} is some positive integer constant that depends on the given family of functions.

We also require some straightforward properties from the computation model:

(C1) $\forall \mathsf{q} \in \mathbb{R}^d$ and $1 \leq i \leq n$, the value $f_i(\mathsf{q}) = \mathrm{d}(\mathsf{q}, f_i)$ is computable in $O(1)$ time.

(C2) For any $y \geq 0, r > 0$ and i, the set of grid cells approximating the sublevel set $(f_i)_{\preceq r}$ of f_i, that is $(f_i)_{\preceq y, \approx r} = \mathsf{G}_{\approx r}\Big((f_i)_{\preceq y}\Big)$ (see Definition II.1), is computable in linear time in its size.

(C3) For any $f_i, f_j \in \mathcal{F}, 1 \leq i, j \leq n$ the distance $\mathrm{d}(f_i, f_j)$ is computable in $O(1)$ time.

We also assume that the growth function $\lambda_{(f_i)}(y)$ from Condition (P2) be in fact computable easily, i.e., in $O(1)$ time.

Remark II.9. *We will use Condition (C2) for a given y and i only for r at least $\Omega\big(\varepsilon\lambda_{(f_i)}(y)\big)$, i.e., we will use a grid on the sublevel set at a resolution typically ε times its growth function value at that level, which by Condition (C2) is also $\Omega\Big(\varepsilon\mathsf{diam}\Big((f_i)_{\preceq y}\Big)\Big)$. As such the number of grid cells in the grid used is $O(1/\varepsilon^d)$.*

Remark II.10. *Condition (C2) when used with a resolution $r = \varepsilon\lambda_{(f_i)}(y)$ requires that we precisely output the set of cells $(f_i)_{\preceq y, \approx r}$. However, it is sufficient to output a set of cells at that resolution which contains $(f_i)_{\preceq y, \approx r}$, is still contained in $f_{\preceq(1+\varepsilon)y}$ and is of size $O(1/\varepsilon^d)$. We state the stricter condition, because it is easier to digest. However it is easier and still correct to work with the relaxed condition.*

II-B1. Properties

The following are basic properties that the functions under consideration have. Their proofs are in the full version [1].

Lemma II.11. *Let $\mathcal{F}$ be a set of functions that satisfy the compactness (P1) and bounded growth (P2) conditions. Then, for any $f \in \mathcal{F}$, either $f_{\preceq 0} = \emptyset$ or $f_{\preceq 0}$ consists of a single point.*

By the above lemma, we may assume that a symbolic perturbation guarantees that $\mathrm{d}(f, g) > 0$ for $f \neq g$. With this convention we have the following,

Observation II.12. *If $\mathsf{cl}(\mathcal{G}) = 0$ for any non-empty subset $\mathcal{G}$ then $|\mathcal{G}| = 1$.*

We also assume that the quantities $\mathrm{d}(f, g)$ are distinct for all distinct pairs of functions.

Lemma II.13. *Let $f \in \mathcal{G}$ and $y \geq 0$. Suppose $\mathsf{u}, \mathsf{v} \in f_{\preceq y}$. Then, $\mathsf{uv} \subseteq \mathcal{G}_{\preceq(1+\zeta/2)y}$, where uv denotes the segment joining u to v.*

Lemma II.14. *Let $A_1, \ldots, A_m \subseteq \mathbb{R}^d$ be compact connected sets. Let uv be any segment. Suppose that $\mathsf{uv} \cap A_i \neq \emptyset$ for all $1 \leq i \leq k$ and $\mathsf{uv} \subseteq \bigcup_{i=1}^{k} A_i$. Then, the sets $A_i, 1 \leq i \leq k$, are connected.*

Lemma II.15. *Suppose we are given $\mathcal{H} \subseteq \mathcal{G} \subseteq \mathcal{F}$, $\delta \geq 0$ and $y \geq 0$, and $\mathcal{H}$ is a (δ, y)-sketch for $\mathcal{G}$. Then, $\mathsf{cl}(\mathcal{H}) \leq (1+\delta)(1+\zeta/2)\max(y, \mathsf{cl}(\mathcal{G}))$.*

The following testifies that a sketch approximates the distance to a set of functions.

Lemma II.16. *Let $\mathcal{H} \subseteq \mathcal{G}$ be sets of functions, where $\mathcal{H}$ is a (δ, y_0)-sketch for $\mathcal{G}$ for some $\delta \geq 0$ and $y_0 \geq 0$. Let q be a point such that $\mathrm{d}(\mathsf{q}, \mathcal{G}) \geq y_0$. Then we have that $\mathrm{d}(\mathsf{q}, \mathcal{H}) \leq (1+\delta)\mathrm{d}(\mathsf{q}, \mathcal{G})$.*

II-B2. Computing the connectivity level

We implicitly assume that the above relevant quantities can be computed efficiently. For example given some $\delta > 0$, and y_0 as per the bound in condition (P3), a (δ, y_0)-sketch can be computed in time $O(|\mathcal{G}|/\delta^{\mathsf{c}_{\mathsf{sk}}})$ time. We also assume that $\mathsf{cl}(\mathcal{G})$ can be computed efficiently without resorting to the "brute force" method. The brute force method computes the individual distance of the functions and then computes a MST on the graph defined by vertices as the functions and edge lengths as their distance. Then $\mathsf{cl}(\mathcal{G})$ is the longest edge of this MST.

III. Constructing the AVD

The input is a set $\mathcal{F}$ of n functions satisfying the conditions of Section II-B, and a number $0 < \varepsilon \leq 1$. We preprocess $\mathcal{F}$, such that given a query point q one can compute a $f \in \mathcal{F}$, where $\mathrm{d}(\mathsf{q}, \mathcal{F}) \leq \mathrm{d}(\mathsf{q}, f) \leq (1+\varepsilon)\mathrm{d}(\mathsf{q}, \mathcal{F})$.

III-A. Building blocks

III-A1. Near neighbor

Given a set of functions $\mathcal{G}$, a real number $\alpha \geq 0$, and a parameter $\varepsilon > 0$, a ***near-neighbor*** data-structure $\mathcal{D}_{near} = \mathcal{D}_{\mathrm{nr}}(\mathcal{G}, \varepsilon, \alpha)$ can decide (approximately) if a point has distance larger or smaller than α. Formally, for a query point q a near-neighbor query answers yes if $\mathrm{d}(\mathsf{q}, \mathcal{G}) \leq \alpha$, and no if $\mathrm{d}(\mathsf{q}, \mathcal{G}) > (1+\varepsilon)\alpha$. It can return either answer if $\mathrm{d}(\mathsf{q}, \mathcal{G}) \in \Big(\alpha, (1+\varepsilon)\alpha\Big]$. If it returns yes, then it also returns a function $f \in \mathcal{G}$ such that $\mathrm{d}(\mathsf{q}, f) \leq (1+\varepsilon)\alpha$. The query time of this data-structure is denoted by $T_{\preceq}(m)$, where $m = |\mathcal{G}|$. The proof of the following appears in the full version [1].

Lemma III.1. *Given a set of m functions $\mathcal{G} \subseteq \mathcal{F}$, $\alpha > 0$ and $\varepsilon > 0$. One can construct a data structure (which is a compressed quadtree), of size $O\big(m/\varepsilon^d\big)$, in $O\big(m\varepsilon^{-d}\log(m/\varepsilon)\big)$ time, such that given any query point $\mathsf{q} \in \mathbb{R}^d$ one can answer a $(1+\varepsilon)$-approximate near-neighbor query for the distance α, in time $T_{\preceq}(m) = O(\log(m/\varepsilon))$.*

III-A2. Interval data structure

Given a set of functions $\mathcal{G}$, real numbers $0 < \alpha \leq \beta$, and $\varepsilon > 0$, the ***interval data structure*** returns for a query point q, one of the following:

(A) If $\mathrm{d}(\mathsf{q}, \mathcal{G}) \in \Big[\alpha, \beta\Big]$, then it returns a function $g \in \mathcal{G}$ such that $\mathrm{d}(\mathsf{q}, g) \leq (1+\varepsilon)\mathrm{d}(\mathsf{q}, \mathcal{G})$. It might also return such a function for values outside this interval.

(B) "$\mathrm{d}(\mathsf{q}, \mathcal{G}) < \alpha$". In this case it returns a function $g \in \mathcal{G}$ such that $\mathrm{d}(\mathsf{q}, g) < \alpha$.

(C) "$\mathrm{d}(\mathsf{q}, \mathcal{G}) > \beta$".

The time to perform an interval query is denoted by $T_r(m, \alpha, \beta)$. The proof of the following appears in the full version [1].

Lemma III.2. *Given a set of m functions $\mathcal{G}$, an interval $[\alpha, \beta]$ and an approximation parameter $\tau > 0$, one can construct an interval data structure of size $O\big(m\tau^{-d-1}\log(4\beta/\alpha)\big)$, in time $O\big(m\tau^{-d-1}\log(4\beta/\alpha)\log(m/\tau)\big)$, such that given a query point q one can answer $(1+\tau)$-approximate nearest neighbor query for the distances in the interval $[\alpha, \beta]$, in time $T_r(m, \alpha, \beta, f) = O\left(\log \frac{m\log(4\beta/\alpha)}{\tau}\right)$.*

Lemma III.2 readily implies that if somehow a priori we know that the nearest neighbor distance lies in an interval of values of polynomial spread, then we would get the desired data-structure by just using Lemma III.2. To overcome this unbounded spread problem, we would first argue that, under our assumptions, there are only linear number of intervals where interesting things happen to the distance function.

III-A3. Connected components of the sublevel sets

Given a finite set X and a partition of it into disjoint sets $X = X_1 \cup \cdots \cup X_k$, let this partition be denoted by $\langle X_1, \ldots, X_k\rangle_X$. For $1 \leq i \leq k$, each X_i is a ***part*** of the partition.

Definition III.3. *For two partitions $P_A = \langle A_1, \ldots, A_k\rangle_X$ and $P_B = \langle B_1, \ldots, B_l\rangle_X$ of the same set X, P_B is a* **refinement** *of P_A, denoted by $P_B \sqsubseteq P_A$, if for any B_i there exists a set A_{j_i}, such that $B_i \subseteq A_{j_i}$. In the other direction, P_A is a* **coarsening** *of P_B.*

Observation III.4. *Given partitions Π, Ξ of a finite set X, if $\Pi \sqsubseteq \Xi$ then $|\Xi| \leq |\Pi|$.*

Definition III.5. *Given partitions $\Pi = \langle X_1, \ldots, X_k\rangle_X \sqsubseteq \Xi = \langle X'_1, \ldots, X'_{k'}\rangle_X$, let $\phi(\Pi, \Xi, i)$ be the function that return the set of indices of sets in Π whose union is $X'_i \in \Xi$.*

Observation III.6. *Given partitions $\Pi \sqsubseteq \Xi$ of a set X with n elements. The partition function $\phi(\Pi, \Xi, \cdot)$ can be computed in $O(n)$ time. For any $1 \leq i \leq |\Xi|$, the set $\phi(\Pi, \Xi, i)$ can be returned in $O\big(|\phi(\Pi, \Xi, i)|\big)$ time, and its size can be returned in $O(1)$ time.*

Definition III.7. *For $\mathcal{G} \subseteq \mathcal{F}$ and $\ell > 0$, consider the intersection graph of the sets $f_{\preceq \ell}$, for all $f \in \mathcal{G}$. Each connected component is a* **cluster** *of $\mathcal{G}$ at level ℓ. And the partition of $\mathcal{G}$ by these clusters, denoted by $\mathsf{C}(\mathcal{G}, \ell)$, is the* **$\ell$-clustering** *of $\mathcal{G}$.*

The values ℓ at which the ℓ-clustering of $\mathcal{F}$ changes are intuitively the critical values when the sublevel set of $\mathcal{F}$ changes and which influence the AVD. These values are critical in trying to decompose the nearest neighbor search on $\mathcal{F}$ into a search on smaller sets.

Observation III.8. *If $0 \leq a \leq b$ then $\mathsf{C}(\mathcal{G}, a) \sqsubseteq \mathsf{C}(\mathcal{G}, b)$.*

The following lemma testifies that we can approximate the ℓ-clustering quickly, for any number ℓ. The proof is in the full version [1].

Lemma III.9. *Given $\mathcal{G} \subseteq \mathcal{F}$, $\ell \geq 0$, and $\varepsilon > 0$, one can compute, in $O\big(\frac{m}{\varepsilon^d}\log(m/\varepsilon)\big)$ time, a partition $\Psi = \Psi_\varepsilon(\mathcal{G}, \ell)$, such that $\mathsf{C}(\mathcal{G}, \ell) \sqsubseteq \Psi \sqsubseteq \mathsf{C}(\mathcal{G}, (1+\varepsilon)\ell)$, where $m = |\mathcal{G}|$.*

Remark III.10. *The partition Ψ computed by Lemma III.9 is monotone, that is, for $\ell \leq \ell'$ and $\varepsilon \leq \varepsilon'$, we have $\Psi_\varepsilon(\mathcal{G}, \ell) \sqsubseteq \Psi_{\varepsilon'}(\mathcal{G}, \ell')$. Moreover, for each cluster $C \in \Psi_\varepsilon(\mathcal{G}, \ell)$, we have that $\mathsf{cl}(C) \leq (1+\varepsilon)\ell$.*

III-A4. Computing a splitting distance

Definition III.11. *Given a partition $\Psi = \Psi_\varepsilon(\mathcal{G}, \ell)$ of $\mathcal{G}$, with $m = |\Psi|$ clusters, a distance x is a* **splitting distance** *if $m/4 \leq |\Psi_1(\mathcal{G}, x/4)|$ and $|\Psi_1(\mathcal{G}, x)| \leq (7/8)m$.*

Lemma III.12. *Given a partition $\Psi = \Psi_\varepsilon(\mathcal{G}, \ell)$ of $\mathcal{G}$, one can compute a splitting distance for it, in expected $O(n(\log n + t))$ time, where $n = |\mathcal{G}|$ and t is the maximum cluster size in Ψ.*

Proof: For each cluster $C \in \Psi$, let r_C be its distance from all the functions in $\mathcal{G} \setminus C$; that is $r_C = \min_{f \in C} \min_{g \in \mathcal{G}\setminus C} \mathrm{d}(f, g)$. Note that $r_C \geq \ell$. Now, let $r_1 \leq r_2 \leq \cdots \leq r_m$ be these distances for the m clusters of Ψ. We randomly pick a cluster $C \in \Psi$ and compute $\ell' = r_C$ for it by brute force – computing the distance of each function of C with the functions of $\mathcal{G} \setminus C$.

Let i be the rank of $\ell' = r_C$ among $r_1, \ldots, r_m$. With probability $1/2$, we have that $m/4 \leq i \leq (3/4)m$. If so we have that:

(A) All the clusters that correspond to $r_i, \ldots, r_m$ are singletons in the partition $\Psi_1(\mathcal{G}, \ell/4)$, as the distance of each one of these clusters is larger than ℓ'. We conclude that $|\Psi_1(\mathcal{G}, \ell'/4)| \geq m/4$.

(B) All the clusters of Ψ that correspond to $r_1, \ldots, r_i$ are contained inside a larger cluster of $\Psi_1(\mathcal{G}, \ell')$ (i.e., they were merged with some other cluster). But then, the number of clusters in $\Psi_1(\mathcal{G}, \ell')$ is at most $(7/8)m$. Indeed, put an edge between such a cluster, to the cluster realizing the smallest distance with it. This graph has at least $e \geq m/4$ edges, and it is easy to see that each component of size at least 2 in the underlying undirected graph has the same number of edges as vertices. As such the number of singleton components is at most $m - e$ while the number of components of size at least 2 is at most $e/2$. It follows that the total number of components is at most $m - e/2 \leq 7m/8$. Since each such component corresponds to a cluster in $\Psi_1(\mathcal{G}, \ell')$ the claim is

```
Search( 𝒢, ϒ, q )
  // 𝒢: set of functions
  // ϒ = Ψ_1(𝒢, ℓ) for some value ℓ
  // Invariant: d(q, 𝒢) > ℓN
  if |ϒ| = 1 then
      return d(q, 𝒢) = min_{f∈𝒢} d(q, f)            (*)
  x ← compute a splitting distance of ϒ, see Lemma III.12

  // Perform an interval approximate nearest
  //     neighbor query on the interval [x/8N, x8N]
  //     for the set 𝒢, see Lemma III.2.
  if d(q, 𝒢) ∈ [x/8N, x8N^2] or (1 + ε/4)-ANN found then
      return nearest function found by the
                (1 + ε/4)-approximate interval query.
  if d(q, 𝒢) < x/8N then
      f ← 2-approximate near neighbor query on 𝒢
                and distance x/8, see Lemma III.1.
      Find cluster C ∈ Ψ_1(𝒢, x/4), such that f ∈ C,
                see Lemma III.9.
      return Search( C, ϒ[C], q )
  if d(q, 𝒢) > x8N^2 then
      return Search( 𝒢, Ψ_1(𝒢, xN), q )            (**)
```

Figure III.1: Search algorithm: We are given a query point q, and an approximation parameter $\varepsilon > 0$. The quantity N is a parameter to be specified shortly. Initially, we call this procedure on the set of functions $\mathcal{F}$ with Υ being the partition of $\mathcal{F}$ into singletons (i.e., $\ell = 0$). Here, $\Upsilon[C]$ denotes the partition of C induced by the partition Υ.

proved.

Now, compute $\Psi_1(\mathcal{G}, \ell')$ and $\Psi_1(\mathcal{G}, \ell'/4)$ using Lemma III.9. With probability at least half they have the desired sizes, and we are done. Otherwise, we repeat the process. In each iteration we spend $O(n(\log n + t))$ time, and the probability for success is half. As such, in expectation the number of rounds needed is constant. ■

III-B. The search procedure

III-B1. An initial "naive" implementation

The search procedure is presented in Figure III.1. The following lemma, whose proof appears in the full version [1], proves its correctness.

Lemma III.13. **Search**$(\mathcal{G}, \Upsilon, \mathsf{q})$ *returns a function $f \in \mathcal{G}$, such that $\mathrm{d}(\mathsf{q}, f) \leq (1+\varepsilon)\mathrm{d}(\mathsf{q}, \mathcal{G})$. The depth of the recursion of* **Search** *is $\mathsf{h} = O(\log n)$, where $n = |\mathcal{G}|$.*

III-B2. But where is the beef? Modifying **Search** *to provide fast query time*

The reader might wonder how we are going to get an efficient search algorithm out of **Search**, as the case that Υ is a single cluster, still requires us to perform a scan on all the functions in this cluster and compute their distance from the query point q. Note however, we have the invariant that the distance of interest is polynomially larger than the connectivity level of each of the clusters of Υ. In particular, precomputing for all the sets of functions such that (*) might be called on, their $\varepsilon/8$-sketches, and answering the query by computing the distance on the sketches, reduces the query time to $O(1/\varepsilon^{\mathsf{c_{sk}}} + \log^2 n)$ (assuming that we precomputed all the data-structures used by the query process). Indeed, an interval query takes $O(\log n)$ time, and there $O(\log n)$ such queries. The final query on the sketch takes time proportional to the sketch size which is $O(1/\varepsilon^{\mathsf{c_{sk}}})$.

As such, the major challenge is not making the query process fast, but rather building the search structure quickly, and arguing that it requires little space.

III-B3. Sketching a sketch

To improve the efficiency of the preprocessing for **Search**, we are going to use sketches more aggressively. Specifically, for each of the clusters of Υ, we can compute their δ-sketches, for $\delta = \varepsilon/(8\mathsf{h}) = O(\varepsilon/\log n)$, see Lemma III.13. From this point on, when we manipulate this cluster, we do it on its sketch. To make this work set $N = n^{4\mathsf{c_{sk}}}$, see $(\mathrm{P3})_{\mathrm{p4}}$ and Lemma II.15.

The only place in the algorithm where we need to compute the sketches, is in (**) in Figure III.1. Specifically, we compute $\Psi_1(\mathcal{G}, xN)$, and for each new cluster $C \in \Psi_1(\mathcal{G}, xN)$, we combine all the sketches of the clusters $D \in \Upsilon$ such that $D \subseteq C$ into a single set of functions. We then compute a δ-sketch for this set, and this sketch is this cluster from this point on. In particular, the recursive calls to **Search** would send the sketches of the clusters, and not the clusters themselves. Conceptually, the recursive call would also pass the minimum distance where the sketches are active – it is easy to verify that we use these sketches only at distances that are far away and are thus allowable (i.e., the sketches represent the functions they correspond to, well in these distances).

Importantly, whenever we compute such a new set, we do so for a distance that is bigger by a polynomial factor (i.e., N) than the values used to create the sketches of the clusters being merged. Indeed, observe that $x > \ell$ and as such xN is N times bigger than ℓ (an upper bound on the value used to compute the input sketches).

As such, all these sketches are valid, and can be used at this distance (or any larger distance). Of course, the quality of the sketch deteriorates. In particular, since the depth of recursion is h, the worst quality of any of the sketches created in this process is at most $(1+\delta)^{\mathsf{h}} \leq 1 + \varepsilon/4$.

Significantly, before using such a sketch, we would shrink it by computing a $\varepsilon/8$-sketch of it. This would reduce the sketch size to $O(1/\varepsilon^{\mathsf{c_{sk}}})$. Note, however, that this still does not help us as far as recursion - we must pass the larger δ-sketches in the recursive call of (**).

This completes the description of the search procedure. It is still unclear how to precompute all the data-structures required during the search. To do that, we need to better understand what the search process does.

III-C. The connectivity tree, and the preprocessing

Given a set of functions $\mathcal{F}$, consider the tree tracking the connected components of the MST of the functions. Formally, initially we start with n singletons (which are the leafs of the tree) that are labeled with the value zero, and we store them

in a set $\mathcal{F}$ of active nodes. Now, we compute for each pair of sets of functions $X, Y \in \mathcal{F}$ the distance $\mathbb{d}(X,Y)$, and let X', Y' be the pair realizing the minimum of this quantity. Merge the two sets into a new set $Z = X' \cup Y'$, create a new node for this set having the node for X' and Y' as children, and set its label to be $\mathbb{d}(X', Y')$. Finally, remove X' and Y' from $\mathcal{F}$ and insert Z into it. Repeat till there is a single element in $\mathcal{F}$. Clearly, the result is a tree that tracks the connected components of the MST.

To make the presentation consistent, let $\mathbb{d}_{\approx}(X,Y)$ be the minimum x such that $\Psi_1(X \cup Y, x)$ is connected. Computing $\mathbb{d}_{\approx}(X,Y)$ can be done by computing $\mathbb{d}_{\approx}(f,g)$ for each pair of functions separately. This in turn, can be done by first computing $\alpha = \mathbb{d}(f,g)$ and observing that r is between $\alpha/4$ and α. In particular, r must be a power of two, so there are only 3 candidate values to consider, and which is the right one can be decided using Lemma III.9.

So, in the above, we use $\mathbb{d}_{\approx}(\cdot,\cdot)$ instead of $\mathbb{d}(\cdot,\cdot)$, and let $\mathcal{H}$ be the resulting tree. For a value ℓ, let $\mathrm{L}_{\mathcal{H}}(\ell)$ be the set of nodes such that their label is smaller than ℓ, but their parent label is larger than ℓ. It is easy to verify that $\mathrm{L}_{\mathcal{H}}(\ell)$ corresponds to $\Psi = \Psi_1(\mathcal{F}, \ell)$; indeed, every cluster $C \in \Psi$ corresponds to a node $u \in \mathrm{L}_{\mathcal{H}}(\ell)$, such that the set of functions stored in the leaves of the subtree of u, denoted by $\mathrm{F}(u)$ is C. The following can be easily proved by induction.

Lemma III.14. *Consider a recursive call* **Search**$(\mathcal{G}, \Upsilon, \mathsf{q})$ *made during the search algorithm execution. Then* $\mathcal{G} = \mathrm{F}(u)$, *and* $\Upsilon = \left\{ \mathrm{F}(v) \;\middle|\; v \in \mathrm{L}_{\mathcal{H}}(\ell) \text{ and } v \text{ is in the subtree of } u \right\}$.

That is, a recursive call of **Search** *corresponds to a subtree of* $\mathcal{H}$.

Of course, not all possible subtrees are candidates to be such a recursive call. In particular, **Search** can now be interpreted as working on a subtree T of $\mathcal{H}$, as follows:

(A) If T is a single node u, then find the closet function to $\mathrm{F}(u)$. Using the sketch this can be done quickly.
(B) Otherwise, compute a distance x, such that the number of nodes in the level $\mathrm{L}_T(x)$ is roughly half the number of leaves of T.
(C) Using interval data-structure determine if the distance $\mathbb{d}(\mathsf{q}, \mathrm{F}(T))$ is in the range $[x/8N, x8N^2]$. If so, we found the desired ANN.
(D) If $\mathbb{d}(\mathsf{q}, \mathrm{F}(T)) > x8N^2$ then continue recursively on portion of T above $\mathrm{L}_T(x)$.
(E) If $\mathbb{d}(\mathsf{q}, \mathrm{F}(T)) < x/8N$ then we know the node $u \in \mathrm{L}_T(x)$ such that the ANN belongs to $\mathrm{F}(u)$. Continue the search recursively on the subtree of T rooted at u.

That is, **Search** breaks T into subtrees, and continues the search recursively on one of the subtrees. Significantly, every such subtree has constant fraction of the size of T, and every edge of T belongs to a single such subtree.

The preprocessing now works by precomputing all the data-structures required by **Search**. Of course, the most natural approach would be to precompute $\mathcal{H}$, and build the search tree by simulating the above recursion on $\mathcal{H}$. Fortunately, this is not necessary, we simulate running **Search**, and investigate all the different recursive calls. We thus only use the above $\mathcal{H}$ in analyzing the preprocessing running time. See Figure III.1$_{\mathrm{p6}}$.

In particular, given a subtree T with m edges, the corresponding partition Υ would have at most m sets. Each such set would have a δ-sketch, and we compute a $\varepsilon/8$-sketch for each one of these sketches. Namely, the input size here is $M = O(m/\delta^{\mathsf{c}_{\mathrm{sk}}})$. Computing the $\varepsilon/8$-sketches for each one of these sketches reduces the total number of functions to $M' = O(m/\varepsilon^{\mathsf{c}_{\mathrm{sk}}})$, and takes $U_1 = O(M/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}) = O\big(m(\varepsilon\delta)^{-\mathsf{c}_{\mathrm{sk}}}\big)$ time, see Section II-B2. Computing the splitting distance, using Lemma III.12, takes $U_2 = O(M' \log M' + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}) = O\big(m\varepsilon^{-\mathsf{c}_{\mathrm{sk}}} \log m\big)$ time. Computing the interval data-structure Lemma III.2 takes $U_3 = O\big(M'\varepsilon^{-d-1} \log n \log M'\big)$ time, and requires $S_1 = O\big(M'\varepsilon^{-d-1} \log n\big)$ space. This breaks T into edge disjoint subtrees $T_1, \ldots, T_t$, and we compute the search data-structure for each one of them separately (each one of these subtrees is smaller by a constant fraction of the original tree). Finally, we need to compute the δ-sketches for the clusters sent to the appropriate recursive calls, and this takes $U_4 = O(M/\delta^{\mathsf{c}_{\mathrm{sk}}})$, by Section II-B2.

Every edge of the tree T gets charged for the amount of work spent in building the top level data-structure. That is, the top level amortized work each edge of T has to pay is

$$\begin{aligned} &O\Big(\left(U_1 + U_2 + U_3 + U_4\right)/m\Big) \\ &= O\Big(\left(\varepsilon\delta\right)^{-\mathsf{c}_{\mathrm{sk}}} + \varepsilon^{-\mathsf{c}_{\mathrm{sk}}} \log m + \varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n + \delta^{-2\mathsf{c}_{\mathrm{sk}}}\Big) \\ &= O\big(\varepsilon^{-2\mathsf{c}_{\mathrm{sk}}} \log^{2\mathsf{c}_{\mathrm{sk}}} n\big)\,, \end{aligned}$$

assuming $\mathsf{c}_{\mathrm{sk}} \geq 2$. Since an edge of T gets charged at most $O(\log n)$ times by this recursive construction, we conclude that the total preprocessing time is $O\big(n\varepsilon^{-2\mathsf{c}_{\mathrm{sk}}} \log^{2\mathsf{c}_{\mathrm{sk}}+1} n\big)$.

As for the space, we have by the same argumentation, that each edge requires $O(\log n \cdot (S_1/m)) = \big(\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\big)$. As such, the overall space used by the data-structure is $\big(n\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\big)$. As for the query time, it boils down to $O(\log n)$ interval queries, and then scanning one $O(\varepsilon)$-sketch. As such, this takes $O\big(\log^2 n + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}\big)$ time.

III-D. The result

Our main result is the following,

Theorem III.15. *Let $\mathcal{F}$ be a set of n functions in $\mathbb{R}^d$ that complies with our assumptions, see Section II-B, and has sketch constant $\mathsf{c}_{\mathrm{sk}} \geq d$. Then, one can build a data-structure to answer ANN for this set of functions, with the following properties:*

(A) The query time is $O(\log n + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}})$.
(B) The preprocessing time is $O\big(n\varepsilon^{-2\mathsf{c}_{\mathrm{sk}}} \log^{2\mathsf{c}_{\mathrm{sk}}+1} n\big)$.
(C) The space used is $O\big(n\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\big)$.

Proof: The query time for the algorithm described above is $O\big(\log^2 n + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}\big)$. To get the improved query time, we observe that **Search** performs a sequence of point-location queries in a sequence of interval near neighbor data-structures (i.e., compressed quadtrees), and then it scans a set of functions of size $O(1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}})$ to find the ANN. We take all these quadtrees spread through our data-structure, and assign

them priority, where a quadtree $\mathcal{Q}_1$ has higher priority than a compressed quadtree $\mathcal{Q}_2$ if $\mathcal{Q}_1$ is queried after $\mathcal{Q}_2$ for any search query. This defines an acyclic ordering on these compressed quadtrees. Overlaying all these compressed quadtrees together, one needs to return for the query point, the leaf of the highest priority quadtree that contains the query point. This can be easily done by scanning the compressed quadtree, and for every leaf computing the highest priority leaf that contains it (observe, that here we are overlaying only the nodes in the compressed quadtrees that are marked by some sublevel set – nodes that are empty are ignored).

A tedious but straightforward induction implies that doing a point-location query in the resulting quadtree is equivalent to running the search procedure as described above. Once we found the leaf that contains the query point, we scan the sketch associated with this cell, and return the computed nearest-neighbor. ∎

An important corollary is the following,

Corollary III.16. *Let $\mathcal{F}$ be a set of n functions in $\mathbb{R}^d$ that complies with our assumptions, see Section II-B, and has sketch constant $\mathsf{c}_{\mathrm{sk}} \geq d$. Then, one can build a data-structure to answer ANN for this set of functions, with the following properties:*

(A) The improved query time is $O(\log n)$.

(B) The preprocessing time is $O\Big(n/\varepsilon^{O(1)} \log^{2\mathsf{c}_{\mathrm{sk}}+1} n\Big)$.

(C) The space used is $S = O\Big(n/\varepsilon^{O(1)} \log^2 n\Big)$.

In particular, we can compute an AVD of complexity $O(S)$ for the given functions. That is, one can compute a space decomposition, such that every region has a single function associated with it, and for any point in this region, this function is a $(1+\varepsilon)$-ANN among the functions of $\mathcal{F}$. Here, a region is either a cube, or the set difference of two cubes.

Proof: We build the data-structure of Theorem III.15, except that instead of linearly scanning the sketch during the query time, we preprocess each such sketch for an exact point-location query; that is, we compute the lower envelope of the sketch and preprocess it for vertical ray shooting [25]. This would require $O\Big(1/\varepsilon^{O(1)}\Big)$ space for each such sketch, and the linear scanning that takes $O(1/\varepsilon^{O(1)})$ time, now is replaced by a point-location query that takes $O\Big(\log 1/\varepsilon^{O(1)}\Big) = O(\log 1/\varepsilon) = O(\log n)$, as desired.

As for the second part, observe that every leaf of the compressed quadtree is the set difference of two canonical grid cells. The lower envelope of the functions associated with such a leaf, induce a partition of this leaf into regions with total complexity $O\Big(1/\varepsilon^{O(1)}\Big)$. ∎

IV. Applications

We present some concrete classes of functions that satisfy our framework, and for which we construct AVD's efficiently.

IV-A. Multiplicative distance functions with additive offsets

As a warm-up we present the simpler case of additively offset multiplicative distance functions. The results of this section are almost subsumed by more general results in Section IV-B. Here the sublevel sets look like expanding balls but there is a time lag before the balls even come into existence i.e. sublevel sets are empty up-to a certain level, this corresponds to the additive offsets. In Section IV-B the sublevel sets are more general fat bodies but there is no additive offset. The results in the present section essentially give an AVD construction of approximate weighted Voronoi diagrams. More formally, we are given a set of points $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$. For $i = 1, \ldots, n$, the point p_i has weight $w_i > 0$, and a constant $\alpha_i \geq 0$ associated with it. We define $f_i(\mathsf{q}) = w_i \|\mathsf{q} - \mathsf{p}_i\| + \alpha_i$. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$. We have, $(f_i)_{\preceq y} = \emptyset$ for $y < \alpha_i$ and $(f_i)_{\preceq y} = \mathsf{B}\Big(\mathsf{p}_i, \frac{y-\alpha_i}{w_i}\Big)$ for $y \geq \alpha_i$.

The proof that $\mathcal{F}$ complies with the framework of Section II-B is present in the full version [1].

We thus get the following result.

Theorem IV.1. *Consider a set P of n points in $\mathbb{R}^d$, where the ith point p_i has additive weight $\alpha_i \geq 0$ and multiplicative weight $w_i > 0$. The ith point induces the additive/multiplicative distance function $f_i(\mathsf{q}) = w_i \|\mathsf{q} - \mathsf{p}_i\| + \alpha_i$. Then one can compute a $(1+\varepsilon)$-AVD for these distance functions, with near linear space complexity, and logarithmic query time. See Theorem III.15 for the exact bounds.*

IV-B. Scaling distance – generalized polytope distances

Let $O \subseteq \mathbb{R}^d$ be a compact set homeomorphic to $\mathsf{B}(0,1)$ and containing a "center" point ρ in its interior. Then O is ***star shaped*** if for any point $\mathsf{v} \in O$ the entire segment $\rho\mathsf{v}$ is also in O. Naturally, any convex body O with any center $\rho \in O$ is star shaped. The ***t-scaling*** of O with a center ρ is the set $tO = \Big\{ t(\mathsf{v} - \rho) + \rho \;\Big|\; \mathsf{v} \in O \Big\}$.

Given a star shaped object O with a center ρ, the ***scaling distance*** of a point q from O is the minimum t, such that $\mathsf{p} \in tO$, and let $F_O(\mathsf{q})$ denote this distance function. Note that, for any $y \geq 0$, the sublevel set $(F_O)_{\preceq y}$ is the y-scaling of O, that is $(F_O)_{\preceq y} = yO$.

Note that for a point $\mathsf{p} \in \mathbb{R}^d$, if we take $O = \mathsf{B}(\mathsf{p}, 1)$ with center p, then $F_O(\mathsf{q}) = \|\mathsf{p} - \mathsf{q}\|$. That is, this distance notion is a strict extension of the Euclidean distance.

Henceforward, for this section, we assume that an object O contains the origin in its interior and the origin is the designated center, unless otherwise stated.

Definition IV.2. *Let $O \subseteq \mathbb{R}^d$ be a star shaped object centered at ρ. We say that O is α-fat if there is a number r such that, $\mathsf{ball}(\rho, r) \subseteq O \subseteq \mathsf{ball}(\rho, \alpha r)$.*

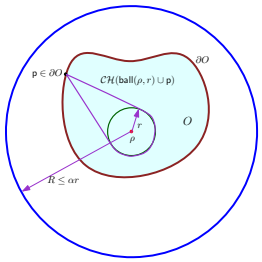

Figure IV.1: Being α-rounded fat.

Definition IV.3. *Let O be a star shaped object centered at ρ. We say that O is α-rounded fat if there is a radius r such that, (i) $\mathsf{ball}(\rho, r) \subseteq O \subseteq \mathsf{ball}(\rho, \alpha r)$ and, (ii) For every point p in the boundary of O, the cone $\mathcal{CH}(\mathsf{ball}(\rho, r) \cup \mathsf{p})$, lies within O, see Figure IV.1.*

By definition any α-rounded fat object is also α-fat. However, it is not true that a α-fat object is necessarily α'-rounded fat for any α', that is even allowed to depend on α. The following useful result is easy to see.

Lemma IV.4. *Let O be a α-fat object. If O is convex then O is also α-rounded fat.*

Given a set $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ of n star shaped objects, consider the set $\mathcal{F}$ of n scaling distance functions, where the ith function, for $i = 1, \ldots, n$ is $f_i = F_{O_i}$. We assume that the boundary of each object O_i has constant complexity.

The proof that $\mathcal{F}$ complies with the framework of Section II-B is present in the full version [1].

We conclude that for α-rounded fat objects, the scaling distance functions they define, falls under our framework. We thus get the following result.

Theorem IV.5. *Consider a set $\mathcal{O}$ of α-rounded fat objects in $\mathbb{R}^d$, for some constant α. Then one can compute a $(1+\varepsilon)$-AVD for the scaling distance functions induced by $\mathcal{O}$, with near linear space complexity, and logarithmic query time. See Theorem III.15 and Corollary III.16 for the exact bounds.*

Note, that the result in Theorem IV.5 covers any symmetric convex metric. Indeed, given a convex symmetric shape C centered at the origin, the distance it induces for any pair of points $\mathsf{p}, \mathsf{u} \in \mathbb{R}^d$, is the scaling distance of C centered p to u (or, by symmetry, the scaling distance of p from C centered at u). Under this distance $\mathbb{R}^d$ is a metric space, and of course, the triangle inequality holds. By an appropriate scaling of space, which does not affect the norm (except for scaling it) we can make C fat, and now Theorem IV.5 applies. Of course, Theorem IV.5 is considerably more general, allowing each of the points to induce a different scaling distance function, and the distance induced does not have to obey the triangle inequality.

IV-C. Nearest furthest-neighbor

For a set of points $\mathsf{S} \subseteq \mathbb{R}^d$ and a point q, the ***furthest-neighbor distance*** of q from S, is $\mathcal{F}_{\mathsf{S}}(\mathsf{q}) = \max_{\mathsf{s} \in \mathsf{S}} \|\mathsf{q} - \mathsf{s}\|$; that is, it is the furthest one might have to travel from q to arrive to a point of S. For example, S might be the set of locations of facilities, where it is known that one of them is always open, and one is interested in the worst case distance a client has to travel to reach an open facility. The function $\mathcal{F}_{\mathsf{S}}(\cdot)$ is known as the ***furthest-neighbor Voronoi*** diagram, and while its worst case combinatorial complexity is similar to the regular Voronoi diagram, it can be approximated using a constant size representation (in low dimensions), see [26].

Given n sets of points $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in $\mathbb{R}^d$, we are interested in the distance function $\mathcal{F}(\mathsf{q}) = \min_i \mathcal{F}_i(\mathsf{q})$, where $\mathcal{F}_i(\mathsf{q}) = \mathcal{F}_{\mathsf{P}_i}(\mathsf{q})$. This quantity arises naturally when one tries to model uncertainty; indeed, let P_i be the set of possible locations of the ith point (i.e., the location of the ith point is chosen randomly, somehow, from the set P_i). Thus, $\mathcal{F}_i(\mathsf{q})$ is the worst case distance to the ith point, and $\mathcal{F}(\mathsf{q})$ is the worst-case nearest neighbor distance to the random point-set generated by picking the ith point from P_i, for $i = 1, \ldots, n$. We refer to $\mathcal{F}(\cdot)$ as the ***nearest furthest-neighbor*** distance, and we are interested in its approximation.

A naive solution to this problem would maintain a data structure for computing the furthest neighbor approximately for each of the P_i and then just compute the minimum of those distances. A data-structure to compute a $1-\varepsilon$ approximation to the furthest neighbor takes $O(1/\varepsilon^d)$ space for $O(1/\varepsilon^d)$ query time, see [26] although this was probably known before. Thus the entire data structure would take up total space of $O(n/\varepsilon^d)$ with a query time of $O(n/\varepsilon^d)$. By using our general framework we can speed up the computation. We can show that $\mathcal{F}_i$, for $i = 1, \ldots, n$ satisfy the conditions (P1) – (P3) and (C1)–(C3). The details are present in the full version [1].

We thus get the following result.

Theorem IV.6. *Given n point sets $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in $\mathbb{R}^d$ with a total of m points, and a parameter $\varepsilon > 0$, one can preprocess the points into an AVD, of size $\widetilde{O}(n)$, for the nearest furthest-neighbor distance defined by these point sets. One can now answer $(1+\varepsilon)$-approximate NN queries for this distance in $O(\log n)$ time. (Note, that the space and query time used, depend only on n, and not on the input size.)*

Proof: We only need to show how to get the improved space and query time. Observe that every one of the sets P_i can be replaced by a subset $\mathsf{S}_i \subseteq \mathsf{P}_i$, of size $O(1/\varepsilon^d \log(1/\varepsilon))$, such that for any point $\mathsf{q} \in \mathbb{R}^d$, we have that $\mathcal{F}_{\mathsf{S}_i}(\mathsf{q}) \leq \mathcal{F}_{\mathsf{P}_i}(\mathsf{q}) \leq (1+\varepsilon/4)\mathcal{F}_{\mathsf{S}_i}(\mathsf{q})$. Such a subset can be computed in $O(|\mathsf{P}_i|)$ time, see [26][1]. We thus perform this transformation for each one of the uncertain point sets $\mathsf{P}_1, \ldots, \mathsf{P}_n$, which reduces the input size to $O(n/\varepsilon^d \log(1/\varepsilon))$. We now apply our main result to the distance functions induced by the reduced sets $\mathsf{S}_1, \ldots, \mathsf{S}_n$. ∎

V. Conclusions

In this paper, we investigated what classes of functions have minimization diagrams that can be approximated efficiently – where our emphasis was on distance functions. We defined a general framework and the requirements on the distance functions to fall under it. For this framework, we presented a new data-structure, with near linear space and preprocessing time. This data-structure can evaluate (approximately) the minimization diagram of a query point in logarithmic time. Surprisingly, one gets an AVD (approximate Voronoi diagram) of this complexity; that is, a decomposition of space with near linear complexity, such that for every region of this decomposition a single function serves as an ANN for all points in this region.

We also showed some interesting classes of functions for which we get this AVD. For example, additive and multi-

[1]One computes an appropriate exponential grid, of size $O(1/\varepsilon^d \log(1/\varepsilon))$, and picks from each grid cell one representative point from the points stored inside this cell.

plicative weighted distance functions. No previous results of this kind were known, and even in the plane, multiplicative Voronoi diagrams have quadratic complexity in the worst case (for which the AVD generated has near linear complexity for any constant dimension). The framework also works for Minkowski metrics of fat convex bodies, and nearest furthest-neighbor. However, our main result applies to even more general distance functions.

Several questions remain open for further research:

(A) Are the additional polylog factors in the space necessary? In particular, it seems unlikely that using WSPD's directly, as done by Arya and Malamatos [17], should work in the most general settings, so reducing the logarithmic dependency seems quite interesting. Specifically, can the Arya and Malamatos construction [17] be somehow adapted to this framework, possibly with some additional constraints on the functions, to get a linear space construction?

(B) On the applications side, are constant degree polynomials a good family amenable to our framework? Specifically, consider a polynomial $\tau(x)$ that is positive for all $x \geq 0$. Given a point u, we associate the distance function $f(\mathsf{q}) = \tau(\|\mathsf{q} - \mathsf{u}\|)$ with u. Given a set of such distance functions, under which conditions, can one build an AVD for these functions efficiently? (It is not hard to see that in the general case this is not possible, at least under our framework.)

References

[1] S. Har-Peled and N. Kumar, "Approximating minimization diagrams and generalized proximity search," *CoRR*, 2013, http://arxiv.org/abs/1304.0393.

[2] J. Matoušek, "Efficient partition trees," *Discrete Comput. Geom.*, vol. 8, pp. 315–334, 1992.

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

[4] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*. New York: Cambridge University Press, 1995. [Online]. Available: http://us.cambridge.org/titles/catalogue.asp?isbn=0521470250

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008. [Online]. Available: http://www.cs.uu.nl/geobook/

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

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

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

[9] 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. MIT Press, 2006, pp. 15–59.

[10] 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

[11] S. Har-Peled, P. Indyk, and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," *Theory Comput.*, vol. 8, pp. 321–350, 2012, special issue in honor of Rajeev Motwani.

[12] 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.

[13] R. Motwani, A. Naor, and R. Panigrahi, "Lower bounds on locality sensitive hashing," in *Proc. 22nd Annu. ACM Sympos. Comput. Geom.*, 2006, pp. 154–157.

[14] 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

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

[16] ——, "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/

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

[18] 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.

[19] 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.

[20] J. Erickson, "New lower bounds for Hopcroft's problem," *Discrete Comput. Geom.*, vol. 16, pp. 389–418, 1996.

[21] N. N. Dalvi, C. Ré, and D. Suciu, "Probabilistic databases: Diamonds in the dirt," *Commun. ACM*, vol. 52, no. 7, pp. 86–94, 2009.

[22] C. C. Aggarwal, *Managing and Mining Uncertain Data*. Springer, 2009.

[23] P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang, "Nearest-neighbor searching under uncertainty," in *Proc. 31st ACM Sympos. Principles Database Syst.*, 2012, pp. 225–236.

[24] P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang, "Nearest neighbor searching under uncertainty ii," in *Proc. 32nd ACM Sympos. Principles Database Syst.*, 2013, p. to appear.

[25] 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.

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