Approximating Minimization Diagrams and Generalized Proximity Search[*]

Sariel Har-Peled[†] Nirman Kumar[‡]

ABSTRACT

We investigate the types of minimization diagrams (i.e., lower envelopes) that can be approximated efficiently in $\mathbb{R}^d$ (d is a constant). We present a new technique for computing approximate minimization diagrams, and we use it to build data-structures for generalized proximity search. The resulting data-structures have near linear size and can answer proximity queries in logarithmic time. For example, one can answer quickly proximity queries for multiplicative and additive weighted Voronoi diagrams, and Minkowski metrics for fat (convex) bodies. Our technique is more general, allowing more general distance functions.

1. INTRODUCTION

Given a set of functions $\mathcal{F} = \left\{ f_i : \mathbb{R}^d \to \mathbb{R} \;\middle|\; i = 1, \ldots, n \right\}$ their minimization diagram is the function $f_{\min}(\mathsf{q}) = \min_{1 \le i \le n} f_i(\mathsf{q})$, for $\mathsf{q} \in \mathbb{R}^d$. 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 can compute $f_{\min}(\mathsf{q})$ quickly. This typically requires $O\left(n^{O(d)}\right)$ space if one is interested in logarithmic query time. If one is restricted to using linear space, then the query time deteriorates to $O\left(n^{1-O(1/d)}\right)$ [Mat92, Cha10]. There is substantial work on bounding the complexity of the lower envelope in various cases, how to compute it efficiently, and performing range searching on them, see the book by Sharir and Agarwal [SA95].

[†]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; `sariel@uiuc.edu`; `http://www.uiuc.edu/~sariel/`.

[‡]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; `nkumar5@illinois.edu`; `http://www.cs.uiuc.edu/~nkumar5/`.

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 $\mathcal{X}$, 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.

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 $i = 1, \ldots, n$. Now, we look at the graph of these functions as a surface of dimension d in $\mathbb{R}^{d+1}$. 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 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. Going the other way, we have a much more general problem. 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. We give a short introduction to nearest neighbor and the approximate versions as the problem provides direct intuition for the problem of approximating minimization diagrams.

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 sublinear query time. In d-dimensional Euclidean space (i.e., $\mathbb{R}^d$) this is facilitated by answering point location queries using a Voronoi diagram [dBCvKO08]. 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 [Cla88] 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 tradeoff the space used and the query time [AM93]. Meiser [Mei93] 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

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

where $\mathsf{n_q} \in \mathsf{P}$ is the nearest neighbor to q in P. Considerable amount of work was done on this problem, see [Cla06] and references therein.

In high dimensional Euclidean space, Indyk and Motwani showed that ANN can be reduced to a small number of near neighbor queries [IM98, HIM12]. Next, using locality sensitive hashing they provide a data-structure that answers ANN queries in time (roughly) $\widetilde{O}\left(n^{1/(1+\varepsilon)}\right)$ and preprocessing time and space $\widetilde{O}\left(n^{1+1/(1+\varepsilon)}\right)$; here the $\widetilde{O}(\cdot)$ hides terms polynomial in $\log n$ and $1/\varepsilon$ i.e. their exponents do not involve d. This was improved to $\widetilde{O}\left(n^{1/(1+\varepsilon)^2}\right)$ query time, and preprocessing time and space $\widetilde{O}\left(n^{1+1/(1+\varepsilon)^2}\right)$ [AI08, AI08]. These bounds are near optimal [MNP06].

In low dimensions (i.e., $\mathbb{R}^d$ for small d), one can use linear space (independent of ε) and get ANN query time $O(\log n + 1/\varepsilon^{d-1})$ [AMN+98, Har11]. The tradeoff 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 [Har01], which are partition of space into regions, desirably of low 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 $O\left((n/\varepsilon^d)\log^2 n\right)$, such that ANN queries can be answered in $O(\log n)$ time. Arya and Malamatos [AM02] showed how to build AVDs of linear complexity (i.e., $O(n/\varepsilon^d)$). Their construction uses Well-Separated Pair Decomposition [CK95]. Further tradeoffs between query and space for AVDs were studied by Arya *et al.* [AMM09].

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 AVDs extend to ℓ_p metrics [Har01, AM02] as well. However, these constructions fail miserably even for the simplest extension – of multiplicative weighted Voronoi diagrams. Here, every site 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. It is known that, even in the plane, multiplicative Voronoi diagrams can have quadratic complexity, and they usually do not comply with 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 problem can not be solved in $\Omega\left(n^{4/3}\right)$ time [Eri96]. This implies that no multiplicative approximation for proximity search for segments is possible, if one insists on near linear preprocessing, and logarithmic query time.

For what cases fast proximity search is possible? So, consider by 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 the space used by the data structure should be near linear.

The challenge. Given the above, one needs to first extract the key properties of a distance functions which permit approximation. Drawing intuition from the simple distance function defined by the Euclidean norm, we can formulate two key required properties on the functions, so that their lower envelopes can be well approximated:

(A) **From afar, they are all the same.** From "sufficiently far", the lower envelope of a set of functions should approximately look like that of a single function. That is, in standard Euclidean case, given a set of points P, from far away enough, the distance to any query point to any point in P, is (approximately) the same as to any other point of P. That is, from far away, we can pick any arbitrary point in P to represent it from afar. Maybe, more disturbingly, in the Euclidean case, the quality of this approximation improves as the query point get further away from the point-set.
Of course, this does not hold even in the multiplicative Voronoi diagram case.

(B) **Separation.** For any two set of functions, there should be roughly some notion of separation between them. If a point lies very close to a one of those sets, that is, in the neighborhood where the lower envelope is defined by one of those sets, one can search for the nearest function in that set, without looking at the other set. Moreover, this valid neighborhood should be defined roughly by the separation between the sets of functions.
This roughly corresponds to the triangle inequality in the ℓ_p case. Indeed, given two point sets P and Q that are relatively far from each other, if the query point is close enough to P, the nearest neighbor to the query is in P, as by the triangle inequality, the query point cannot be close to both the sets.

1.1 Our results

We give a precise and abstract formulation of the above conditions and prove that under those assumptions we can indeed efficiently approximate the lower envelope. Using this framework, one cane quickly evaluate approximately the lower envelope for large class of functions that rises 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. Specific cases, which seem interesting and useful, where the new data-structure can be used include:

(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 $f_i(\mathsf{q}) = w_i \left\| \mathsf{q} - \mathsf{p}_i \right\|$. For these functions computing approximately the lower envelope function is equivalent to finding the approximate nearest neighbor for the multiplicative weighted distance functions or equivalently construction an approximate multiplicative Voronoi diagram. The approach of Arya and Malamatos [AM02] to construct AVD's using WSPD's fails for this problem, as that construction relies on the 1-Lipschitz property of the distance function. When arbitrary weights are allowed to be multiplied, this does not hold. If one follows the construction by letting the cell sizes depend on the weights w_i, the size of the AVD has a dependence on the spread of the weights w_i, which is not desirable. Our general technique does not run into this problem as we do not exploit the Lipschitz property of distance functions (indeed our distance functions are not Lipschitz).
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, v their distance as induced by C, denoted by $\left\| \mathsf{u} - \mathsf{v} \right\|_C$, 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 $f_i(\mathsf{q}) = \left\| \mathsf{p}_i - \mathsf{q} \right\|_{C_j}$, for $i = 1, \ldots, n$. With different convex bodies for each point the distance function induced by the collection no longer defines a metric and this makes the problem complicated. Existing techniques for AVD and ANN cannot be readily applied because of this reason. Since our framework is formulated very abstractly, these challenges present little difficulty to it. In fact, by concentrating on the desired conditions to be met by the functions amenable to our framework, we can extract the geometrical "reason" why these distance functions are still amenable to efficient proximity search. In particular the fatness of the associated convex bodies turns out to be sufficient for us to approximate the associated distance function, see Section 4.2. The negative example for the case of segments presented above, indicates that the condition is also necessary.

Paper organization. In Section 2 we define our basic framework and prove some basic properties. Since we are trying to make our frame as inclusive as possible, its description is somewhat abstract. In Section 3, we describe the construction of the AVD and its associated data-structure. We describe in Section 4 some specific cases where the new AVD construction can be used. We conclude in Section 5.

2. 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 and is a standard first step in many ANN algorithms.

2.1 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 $\mathsf{d}(\mathsf{q}, \mathsf{P}) = \min_{x \in \mathsf{P}} \left\| \mathsf{q} - x \right\|$. For a number $\ell > 0$, the ***grid*** of sidelength ℓ, denoted by G_ℓ, is the natural tiling of $\mathbb{R}^d$, with cubes of sidelength ℓ. 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 corresponds to a cell in a quadtree having the unit (hyper)cube as the root region.

Definition 2.1. *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 2.2. *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 2.3. *Given a function f and $\mathsf{q} \in \mathbb{R}^d$ their separation is $\mathsf{sep}(\mathsf{q}, f) = f(\mathsf{q})$. Given two functions f and g, their* **separation** *$\mathsf{sep}(f, g)$ 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* **separation** *is*

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

Example 2.4. To decipher these somewhat cryptic definitions, the reader might want to consider the standard settings of regular Voronoi diagrams. Here, we have a set P of n points. The ith point $\mathsf{p}_i \in \mathsf{P}$ induces the natural function $f_i(\mathsf{q}) = \left\| \mathsf{q} - \mathsf{p}_i \right\|$. We have:

(A) The graph of f_i in $\mathbb{R}^{d+1}$ is a cone with an apex at $(\mathsf{p}_i, 0)$.

(B) The ℓ sublevel set of f_i is a ball of radius ℓ centered at p_i.

(C) The separation of q from f_i is the Euclidean distance between q and p_i.

(D) Consider two subsets of points $X, Y \subseteq \mathsf{P}$ and let $\mathcal{F}_X$ and $\mathcal{F}_Y$ be the corresponding sets of functions. The separation $\ell = \mathsf{sep}(\mathcal{F}_X, \mathcal{F}_Y)$ is the minimum radius of balls centered at points of X and Y, such that there are two balls from the two sets that intersect; that is, ℓ is half the minimum distance between a point of X and a point of Y. In particular, if the union of balls of radius ℓ centered at X are connected i.e. $(\mathcal{F}_X)_{\preceq \ell}$ is connected, and similarly for Y, then $(\mathcal{F}_X \cup \mathcal{F}_Y)_{\preceq \ell}$ is connected. This is the critical value where two connected components of the sublevel set merge.

The separation function behaves to some extent like a distance function: (i) $\mathsf{sep}(f, g)$ always exists, and (ii) (symmetry) $\mathsf{sep}(f, g) = \mathsf{sep}(g, f)$, Also, we have $f_{\preceq \mathsf{sep}(f,g)} \neq \emptyset$. We extend the above definition to sets of functions. Note that the triangle inequality does not hold for $\mathsf{sep}(\cdot,\cdot)$.

Observation 2.5. *Suppose that f and g are two functions such that $\mathsf{sep}(f,g) > 0$ and $\mathsf{q} \in \mathbb{R}^d$. Then, $\max(\mathsf{sep}(\mathsf{q}, f), \mathsf{sep}(\mathsf{q}, g)) \geq \mathsf{sep}(f,g)$.*

Definition 2.6. *Let $B_1, B_2, \ldots, B_m$ be a finite collection of subsets of $\mathbb{R}^d$. We say that they form a* **connected system of sets** *if for any $1 \leq i \leq j \leq m$ there is a sequence of distinct indices $i = i_1, i_2, \ldots, i_k = j$ with $1 \leq i_r \leq m, 1 \leq r \leq k$ such that $B_r \cap B_{r+1} \neq \emptyset$ for $1 \leq r \leq k-1$. Namely, the intersection graph of these sets is connected. Note, that if the sets B_i are each connected, for all i, then the above states that $\bigcup_i B_i$ is* **connected**.

2.1.1 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. This property is the key insight in constructing approximate Voronoi diagrams. 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 2.7. *Given a set of functions $\mathcal{G}$, if $\mathcal{G}$ contains a single function then the* **connectivity level** *$\Delta(\mathcal{G})$ is 0; otherwise, it is the minimum $\ell \geq 0$, such that $\mathcal{G}_{\preceq \ell}$ is connected.*

Definition 2.8. *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 $\delta' \geq \delta, y_0' \geq y_0$, 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.

2.2 Conditions on the functions

To be able to compute AVD for a given set of functions we require the following:

(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 some (absolute, independent of n) constant ζ, only depending on the family of functions under consideration, such that for any $y \geq 0$ and $\varepsilon > 0$, if $f_{\preceq y} \neq \emptyset$ and $\mathsf{q} \in \mathbb{R}^d$ with $\mathsf{d}(\mathsf{q}, f_{\preceq y}) \leq (\varepsilon/\zeta)\,\mathsf{diam}(f_{\preceq y})$, then $f(\mathsf{q}) \leq (1+\varepsilon)y$. This is equivalent to $f_{\preceq y} \oplus \mathsf{B}(0, (\varepsilon/\zeta)\mathsf{diam}(f_{\preceq y})) \subseteq f_{\preceq(1+\varepsilon)y}$, 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 $|\mathcal{H}| = O\left(1/\delta^{\mathsf{c}_{\mathrm{sk}}}\right)$ and $y_0 = O\big(\Delta(\mathcal{G})(|\mathcal{G}|/\delta)^{\mathsf{c}_{\mathrm{sk}}}\big)$ such that, $\mathcal{H}$ is an (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 $f_i(\mathsf{q}) = \mathsf{sep}(\mathsf{q}, f_i)$ 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 2.1), 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 separation $\mathsf{sep}(f_i, f_j)$ is computable in $O(1)$ time.

2.2.1 Properties

The following are basic properties that the functions under consideration have. For readability, we delegate most of the (easy) proofs to the appendix.

Lemma 2.9. *(Proof in the full version.) 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 $\mathsf{sep}(f,g) > 0$ for $f \neq g$. With this convention we have the following,

Observation 2.10. *If $\Delta(\mathcal{G}) = 0$ for any non-empty subset $\mathcal{G}$ then $|\mathcal{G}| = 1$.*

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

Lemma 2.11. *(Proof in the full version.) 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 2.12. *(Proof in the full version.) Let $A_1, \ldots, A_m \subseteq \mathbb{R}^d$ be compact 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 $A_i, 1 \leq i \leq k$, form a connected system of sets.*

Lemma 2.13. *(Proof in the full version.) 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, $\Delta(\mathcal{H}) \leq (1+\delta)(1+\zeta/2)\max(y, \Delta(\mathcal{G}))$.*

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

Lemma 2.14. *(Proof in the full version.) 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 $\mathsf{sep}(\mathsf{q}, \mathcal{G}) \geq y_0$. Then we have that $\mathsf{sep}(\mathsf{q}, \mathcal{H}) \leq (1+\delta)\mathsf{sep}(\mathsf{q}, \mathcal{G})$.*

2.2.2 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 $O(|\mathcal{G}|/\delta^{\mathsf{c}_{\mathrm{sk}}})$ time. To compute $\Delta(\mathcal{G})$, we can compute the individual separation of the functions and then compute a MST on the graph defined by vertices as the functions and edge lengths as their separation. Then $\Delta(\mathcal{G})$ is the longest edge of this MST.

3. CONSTRUCTING THE AVD

The input is a set $\mathcal{F}$ of n functions satisfying the conditions of Section 2.2, and a number $0 < \varepsilon \leq 1$. We preprocess $\mathcal{F}$, such that given a query point q one can compute a $f \in \mathcal{F}$, and $\mathsf{sep}(\mathsf{q}, \mathcal{F}) \leq \mathsf{sep}(\mathsf{q}, f) \leq (1+\varepsilon)\mathsf{sep}(\mathsf{q}, \mathcal{F})$.

3.1 Building blocks

3.1.1 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 separation larger or smaller than α. Formally, for a query point q a near-neighbor query answers yes if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) \leq \alpha$, and no if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) > (1+\varepsilon)\alpha$. It can return either answer if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) \in \left(\alpha, (1+\varepsilon)\alpha\right]$. If it returns yes, then it also returns a function $f \in \mathcal{G}$ such that $\mathsf{sep}(\mathsf{q}, f) \leq (1+\varepsilon)\alpha$. The query time of this data-structure is denoted by $T_{\leq}(m)$, where $m = |\mathcal{G}|$.

Lemma 3.1. *(Proof in the full version.) 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 $O\left(m/\varepsilon^d\right)$, in $O\left(m\varepsilon^{-d}\log(m/\varepsilon)\right)$ time, such that given any query point $\mathsf{q} \in \mathbb{R}^d$ one can answer an $(1+\varepsilon)$-approximate near-neighbor query for the distance α, in time $T_{\leq}(m) = O(\log(m/\varepsilon))$.*

3.1.2 Interval data structure

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

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

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

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

The time to perform an interval query is denoted by $T_r(m, \alpha, \beta)$.

Lemma 3.2. *(Proof in the full version.) 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 $O\left(m\tau^{-d-1}\log(4\beta/\alpha)\right)$, in time $O\left(m\tau^{-d-1}\log(4\beta/\alpha)\log(m/\tau)\right)$, 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 $T_r(m, \alpha, \beta, f) = O\left(\log \dfrac{m\log(4\beta/\alpha)}{\tau}\right)$.*

Lemma 3.2 readily implies that if somehow a priori we know the nearest neighbor separation lies in an interval of values of polynomial spread, then we would get the desired data-structure by just using Lemma 3.2. 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 separation function.

3.1.3 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 3.3. *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 3.4. *Given partitions Π, Ξ of a finite set X, if $\Pi \sqsubseteq \Xi$ then $|\Xi| \leq |\Pi|$.*

Definition 3.5. *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 that their union is $X'_i \in \Xi$.*

Observation 3.6. *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 $O\left(|\phi(\Pi, \Xi, i)|\right)$ time, and its size can be returned in $O(1)$ time.*

Definition 3.7. *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 3.8. *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.

Lemma 3.9. *(Proof in the full version.) Given $\mathcal{G} \subseteq \mathcal{F}$, $\ell \geq 0$, and $\varepsilon > 0$, one can compute, in $O\left(\frac{m}{\varepsilon^d}\log(m/\varepsilon)\right)$ 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 3.10. *The partition Ψ computed by Lemma 3.9 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 $\Delta(C) \leq (1+\varepsilon)\ell$.*

3.1.4 Computing a splitting distance

Definition 3.11. *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 3.12. *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 separation distance from all the functions in $\mathcal{G} \setminus C$; that is $r_C = \min_{f \in C} \min_{g \in \mathcal{G} \setminus C} \mathsf{sep}(f, g)$. Note that $r_C \geq \ell$. Now, let $r_1 \leq r_2 \leq \cdots \leq r_m$ be these separation 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 separation 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 corresponds to $r_i, \ldots, r_m$ are singletons in the partition $\Psi_1(\mathcal{G}, \ell/4)$, as the separation 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 corresponds 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

```
Search( $\mathcal{G}$, $\Upsilon$, q )
  // $\mathcal{G}$:  set of functions
  // $\Upsilon = \Psi_1(\mathcal{G}, \ell)$ for some value $\ell$
  // Invariant:  $\mathsf{sep}(\mathsf{q}, \mathcal{G}) > \ell N$
  if $|\Upsilon| = 1$ then
      return $\mathsf{sep}(\mathsf{q}, \mathcal{G}) = \min_{f \in \mathcal{G}} \mathsf{sep}(\mathsf{q}, f)$          (*)
  $x \leftarrow$ compute a splitting distance of $\Upsilon$, see Lemma 3.12

  // Perform an interval approximate nearest
  //    neighbor query on the interval $[x/8N, x8N]$
  //    for the set $\mathcal{G}$, see Lemma 3.2.
  if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) \in \left[x/8N, x8N^2\right]$ or $(1 + \frac{\varepsilon}{4})$-ANN found then
      return nearest function found by the
             $(1 + \varepsilon/4)$-approximate interval query.
  if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) < x/8N$ then
      $f \leftarrow$ 2-approximate near neighbor query on $\mathcal{G}$
             and distance $x/8$, see Lemma 3.1.
      Find cluster $C \in \Psi_1(\mathcal{G}, x/4)$, such that $f \in C$,
             see Lemma 3.9.
      return Search( $C$, $\Upsilon[C]$, q )
  if $\mathsf{sep}(\mathsf{q}, \mathcal{G}) > x8N^2$ then
      return Search( $\mathcal{G}$, $\Psi_1(\mathcal{G}, xN)$, q )          (**)
```

Figure 3.1: 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.

realizing the smallest separation 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 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 proved.

Now, compute $\Psi_1(\mathcal{G}, \ell')$ and $\Psi_1(\mathcal{G}, \ell'/4)$ using Lemma 3.9. 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. □

3.2 The search procedure

3.2.1 An initial "naive" implementation

The search procedure is presented in Figure 3.1.

Lemma 3.13. Search$(\mathcal{G}, \Upsilon, \mathsf{q})$ *returns a function* $f \in \mathcal{G}$, *such that* $\mathsf{sep}(\mathsf{q}, f) \leq (1 + \varepsilon)\mathsf{sep}(\mathsf{q}, \mathcal{G})$. *The depth of the recursion of* Search *is* $\mathsf{h} = O(\log n)$, *where* $n = |\mathcal{G}|$.

Proof. The proof is by induction on the size of Υ. If $|\Upsilon| = 1$, then the function realizing $\mathsf{sep}(\mathsf{q}, \mathcal{G})$ is returned, and the claim is true.

Let x be the computed splitting distance of Υ. Next, the procedure perform an $(1+\varepsilon/4)$-approximate interval nearest-neighbor query for q on the range $[x/8N, x8N]$. If this computed the approximate nearest neighbor then we are done.

Otherwise, it must be that either $\mathsf{sep}(\mathsf{q}, \mathcal{G}) < x/8N$ or $\mathsf{sep}(\mathsf{q}, \mathcal{G}) > 8Nx$, and significantly, we know which of the two options it is:

(A) If $\mathsf{sep}(\mathsf{q}, \mathcal{G}) < x/8N$ then doing an approximate near-neighbor query on $\mathcal{G}$ and distance $x/8$, returns a function $f \in \mathcal{G}$ such that $\mathsf{sep}(\mathsf{q}, f) \leq x/4$. Clearly, the nearest neighbor to q must be in the cluster containing f in the partition $\Psi_1(\mathcal{G}, x/4)$, and Search recurse on this cluster. Now, by induction, the returned ANN is correct.
Since x is a splitting distance of Υ, see Definition 3.11, we have $|\Upsilon|/4 \leq |\Psi_1(\mathcal{G}, x/4)|$ and $\Upsilon \sqsubseteq \Psi_1(\mathcal{G}, x/4)$. As such, since C is one of the clusters of $\Psi_1(\mathcal{G}, x/4)$, the induced partition of C by Υ (i.e., $\Upsilon[C]$), can have at most $(1 - 1/4)|\Upsilon|$ clusters.

(B) Otherwise, we have $\mathsf{sep}(\mathsf{q}, \mathcal{G}) > x \cdot 8N$. Since x is a splitting distance, we have that $|\Psi_1(\mathcal{G}, x)| \leq (7/8)|\Upsilon|$, see Definition 3.11. We recurse on $\mathcal{G}$, and a partition that has fewer clusters, and by induction, the returned answer is correct.

In each step of the recursion, the partition shrunk by at least a fraction of 7/8. As such, after a logarithmic number of recursive calls, the procedure is done. □

3.2.2 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 separation 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 $O(1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}} + \log^2 n)$ (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 $O(1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}})$.

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.

3.2.3 Sketching a sketch

To improve the efficiency of the preprocessing of 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 3.13. From this point on, when we manipulate this cluster, we do it on its sketch. To make this work set $N = n^{4\mathsf{c}_{\mathrm{sk}}}$, see $(\mathrm{P3})_{\mathrm{P}}$ and Lemma 2.13.

The only place in the algorithm where we need to compute the sketches, is in (**) in Figure 3.1. 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, in 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 distances there are far away and are thus allowable (i.e., the sketches represents the functions they correspond to well in these distances).

Importantly, whenever we compute such a new set, we do in 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 $O(1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}})$. 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 do.

3.3 The connectivity tree, and the preprocessing

Given a set of functions $\mathcal{F}$, create a 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 separation $\mathsf{sep}(X, Y)$, 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 $\mathsf{sep}(X', Y')$. 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 $\mathsf{sep}_{\approx}(X, Y)$ be the minimum x such that $\Psi_1(X \cup Y, x)$ is connected. Computing $\mathsf{sep}_{\approx}(X, Y)$ can be done by computing $\mathsf{sep}_{\approx}(f, g)$ for each pair of functions separately. This in turn, can be done by first computing $\alpha = \mathsf{sep}(f, g)$ 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 3.9.

So, in the above, we use $\mathsf{sep}_{\approx}(\cdot,\cdot)$ instead of $\mathsf{sep}(\cdot,\cdot)$, 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 3.14. *Consider a recursive call* **Search**$(\mathcal{G}, \Upsilon, \mathsf{q})$ *made during the search algorithm execution. Then* $\mathcal{G} = \mathrm{F}(u)$*, and* $\Upsilon = \left\{ \mathrm{F}(v) \;\middle|\; v \in \mathrm{L}_{\mathcal{H}}(\ell) \text{ and } v \text{ is in subtree of } u \right\}$.

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, computes 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 separation $\mathsf{sep}(\mathsf{q}, \mathrm{F}(T))$ is in the range $[x/8N, x8N^2]$. If so, we found the desired ANN.
(D) If $\mathsf{sep}(\mathsf{q}, \mathrm{F}(T)) > x8N^2$ then continue recursively on portion of T above $\mathrm{L}_T(x)$.
(E) If $\mathsf{sep}(\mathsf{q}, \mathrm{F}(T)) < x/8N$ 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 continue 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 3.1$_{\mathrm{p}}$.

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 $U_1 = O(M/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}) = O\bigl(m(\varepsilon\delta)^{-\mathsf{c}_{\mathrm{sk}}}\bigr)$ time, see Section 2.2.2. Computing the splitting distance, using Lemma 3.12, takes $U_2 = O(M' \log M' + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}) = O\bigl(m\varepsilon^{-\mathsf{c}_{\mathrm{sk}}} \log m\bigr)$ time. Computing the interval data-structure Lemma 3.2 takes $U_3 = O\bigl(M'\varepsilon^{-d-1} \log n \log M'\bigr)$ time, and requires $S_1 = O\bigl(M'\varepsilon^{-d-1}\log n\bigr)$ 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 2.2.2.

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

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

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

As for the space, we have that by the same argumentation, that each edge requires $O(\log n \cdot (S_1/m)) = \bigl(\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\bigr)$. As such, the overall space used by the data-structure is $\bigl(n\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\bigr)$. 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 $O\left(\log^2 n + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}\right)$ time.

3.4 The result

Theorem 3.15. *Let $\mathcal{F}$ be a set of n functions in $\mathbb{R}^d$ that complies with our assumptions, see Section 2.2, 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 $O\left(n\varepsilon^{-2\mathsf{c}_{\mathrm{sk}}} \log^{2\mathsf{c}_{\mathrm{sk}}+1} n\right)$.
(C) The space used is $O\left(n\varepsilon^{-d-1-\mathsf{c}_{\mathrm{sk}}} \log^2 n\right)$.

Proof. The query time stated above is $O\left(\log^2 n + 1/\varepsilon^{\mathsf{c}_{\mathrm{sk}}}\right)$. 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 $\mathcal{T}_2$ 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. □

Corollary 3.16. *(Proof in the full version.) Let $\mathcal{F}$ be a set of n functions in $\mathbb{R}^d$ that complies with our assumptions, see Section 2.2, 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 $O\left(n/\varepsilon^{O(1)} \log^{2\mathsf{c}_{\mathrm{sk}}+1} n\right)$.
(C) The space used is $S = O\left(n\varepsilon^{O(1)} \log^2 n\right)$.

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 the $(1+\varepsilon)$-ANN among the functions of $\mathcal{F}$.

4. APPLICATIONS

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

4.1 Additive, multiplicative and weighted offset distance functions

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

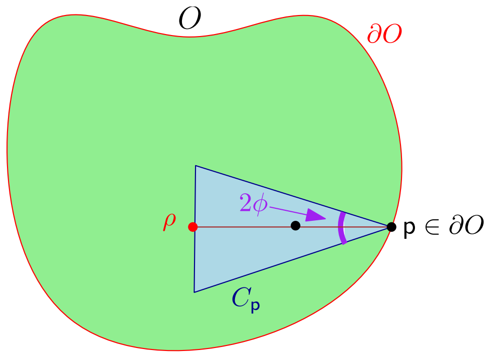

Figure 4.1: Being ϕ-cone fat.

$(f_i)_{\preceq y} = \mathsf{B}\left(\mathsf{p}_i, \frac{y-\alpha_i}{w_i}\right)$ for $y \geq \alpha_i$. Checking conditions (C1) and (C2) is trivial. As for (C3) we have the following easy lemma,

Lemma 4.1. *(Proof in the full version.) For any $1 \leq i, j \leq n$ we have*

$$\mathsf{sep}(f_i, f_j) = \max\left(\max(\alpha_i, \alpha_j), \|\mathsf{p}_i - \mathsf{p}_j\| \frac{w_i w_j}{w_i + w_j} + \frac{\alpha_i w_j + \alpha_j w_i}{w_i + w_j}\right).$$

Lemma 4.2. *(Proof in the full version.) Given $1 \leq i, j \leq n$ such that $w_i \leq w_j$. Suppose $y \geq \max(\alpha_i, \alpha_j)$. Then, $(f_j)_{\preceq y} \subseteq (f_i)_{\preceq(1+\delta)y}$ if and only if $y \geq \frac{\|\mathsf{p}_i - \mathsf{p}_j\| + (1+\delta)\alpha_i/w_i - \alpha_j/w_j}{(1+\delta)/w_i - 1/w_j}$.*

It is easy to see that compactness (P1) and bounded growth (P2) hold for the set of functions $\mathcal{F}$. The following lemma proves the sketch property (P3).

Lemma 4.3. *(Proof in the full version.) For any $\mathcal{G} \subseteq \mathcal{F}$ and $\delta > 0$ there is a (δ, y_0)-sketch $\mathcal{H} \subseteq \mathcal{G}$ with $|\mathcal{H}| = 1$ and $y_0 = 3\Delta(\mathcal{G}) |\mathcal{G}| / \delta$.*

4.1.1 Result

Theorem 4.4. *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 $w_i \geq 0$. 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 an $(1+\varepsilon)$-AVD for these sites with these additive/multiplicative distances, with near linear space complexity, and logarithmic query time. See Theorem 3.15$_p$ for the exact result.*

4.2 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 an O with a center ρ is the set $tO = \left\{ t(\mathsf{v} - \rho) + \rho \;\middle|\; \mathsf{v} \in O \right\}$.

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 $O = \mathsf{B}(\mathsf{p}, 1)$, then $F_O(\mathsf{q}) = \|\mathsf{p} - \mathsf{q}\|$. That is, this distance notion is a strict extension of the Euclidean distance.

Definition 4.5. *Let $O \subseteq \mathbb{R}^d$ be a star shaped object centered at ρ. Let r (resp. R) be the radius of the largest (resp. smallest) ball centered at ρ that is contained in (resp. contains) O. We say that O is **α-fat** if $R \le \alpha r$.*

Definition 4.6. *Let O be a star shaped object centered at ρ and $0 < \phi < \pi$. For a boundary point $\mathsf{p} \in \partial O$ consider the cone C_{p} with apex p, with its axis being the segment joining p to ρ, and its base being a $(d-1)$-dimensional ball with center and lying in the subspace orthogonal to the axis, with an apex angle 2ϕ. We say that O is **ϕ-cone fat** if for every boundary point $\mathsf{p} \in \partial O$, we have that C_{p} is entirely contained in O, see Figure 4.1.*

It is not true that any α-fat object O is also ϕ-cone fat for some ϕ, even possibly where ϕ depends on α.

Lemma 4.7. *(Proof in the full version.) Let O be a α-fat object. If O is convex then O is also ϕ-cone fat where $\phi = \tan^{-1}(1/\alpha)$.*

If a star shaped object O is α-fat and ϕ-cone fat for some constants $\alpha > 0$ and $0 < \phi < \pi$ we say it is ***(α, ϕ)-fat***. Given a set $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ of n star shaped objects, where O_i is centered at p_i, for $i = 1, \ldots n$. 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.

We next argue that $\mathcal{F}$ complies with the framework of Section 2.2. Using standard techniques, we can compute the quantities required in conditions (C1)–(C3)$_{\mathrm{p}}$. Also, trivially we have that condition (P1)$_{\mathrm{p}}$ is satisfied as the sublevel sets are dilutions of the O_i and are thus compact by definition. The next few lemmas establish that both bounded growth (P2) and the sketch property (P3) are also true, if the objects are also fat.

Lemma 4.8. *(Proof in the full version.) Given $\alpha > 0$ and $0 < \phi < \pi$, suppose O is a star shaped object that is (α, ϕ)-fat. Then for any $c \ge 2\alpha/\sin\phi$ and any $y \ge 0, \varepsilon > 0$ we have that $yO \oplus \mathsf{B}(0, (\varepsilon/c)\mathsf{diam}(yO)) \subseteq (1+\varepsilon)yO$; that is, $(F_O)_{\preceq y} \oplus \mathsf{B}\Big(0, (\varepsilon/c)\mathsf{diam}\Big((F_O)_{\preceq y}\Big)\Big) \subseteq (F_O)_{\preceq (1+\varepsilon)y}$.*

By the above lemma we can take the growth constant (P2)$_{\mathrm{p}}$ for the set of functions F_{O_i,p_i} to be $\zeta = c = 2\alpha/\sin\phi$. If the object O is α-fat but not ϕ-cone fat for any constant $\phi > 0$ then for it may be that it scaling distance function grows arbitrarily quickly and thus fail to comply with our framework, see Figure 4.2. It is not hard to see that Lemma 4.8 implies that bounded growth (P2) is satisfied for all the functions $f_1, \ldots, f_n$ when the objects under consideration $O_1, \ldots, O_n$ are (α, ϕ)-fat. To show that condition (P3) is satisfied, is slightly harder.

Lemma 4.9. *(Proof in the full version.) Let $\mathcal{O}$ be a set of n star shaped objects $O_1, O_2, \ldots, O_n$. Let $\alpha \ge 1$ and $0 < \phi < \pi$ be any constants. Suppose that $O_1, \ldots, O_n$ are (α, ϕ)-fat. Then, for any $\delta > 0$, there is a subset $\mathcal{I} \subseteq \{1, 2, \ldots, n\}$ with $|\mathcal{I}| = O(\delta^{-d})$, such that for all $y \ge 0$, we have*

$$\bigcup_{i \in [n]} yO_i \subseteq \bigcup_{i \in \mathcal{I}} (1+\delta) yO_i.$$

Moreover, for every $i \in \mathcal{I}$ we have that $\mathsf{diam}(O_i) = \Omega(\max_i \mathsf{diam}(O_i))$.

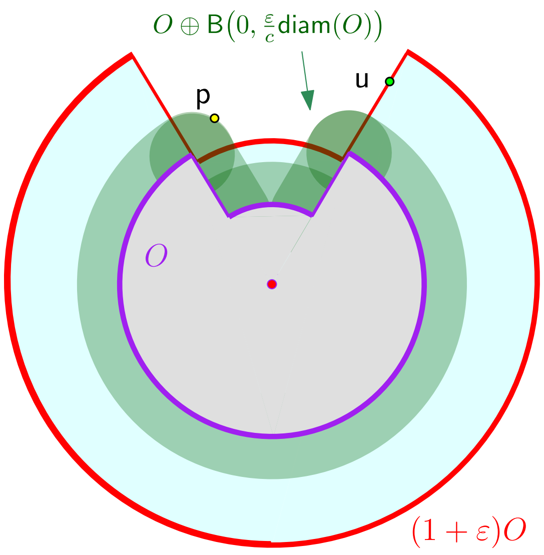
$O \oplus \mathsf{B}(0, \frac{\varepsilon}{c}\mathsf{diam}(O))$
p
u
O
$(1+\varepsilon)O$

Figure 4.2: The object O is fat but not cone fat. In particular, the point p is in $O \oplus \mathsf{B}(0, (\varepsilon/c)\mathsf{diam}(O))$ but not in $(1+\varepsilon)O$. In particular, the scaling distance function is discontinuous at u.

Lemma 4.10. *(Proof in the full version.) Let $\alpha \ge 1$ and $0 < \phi < \pi$ be constants. Let O be a star shaped object that is (α, ϕ)-fat, and let $\delta > 0$. Let $\mathsf{u} \in \mathbb{R}^d$ with $\|\mathsf{u}\| \le \delta \mathsf{diam}(O)/c$ where $c = 2\alpha/\sin\phi$. Then we have that $\mathsf{u} + O \subseteq (1+\delta)O$.*

Lemma 4.11. *(Proof in the full version.) For $i = 1, \ldots, n$, let O_i be a star shaped object in $\mathbb{R}^d$ centered at a point p_i. Let $\mathcal{O} = \{O_1, \ldots, O_n\}$, $\mathsf{P} = \{\mathsf{p}_1, \ldots, \mathsf{p}_n\}$, and $\mathcal{F} = \left\{f_i \,\middle|\, 1 \le i \le n\right\}$, where $f_i = F_{O_i,\mathsf{p}_i}$, for $i = 1, \ldots, n$. For $i = 1, \ldots, n$, let R_i denote the radius of the smallest ball centered at p_i that contains O_i, and let $R = \max_i R_i$. Then, $\Delta(\mathcal{F}) \ge \mathsf{diam}(\mathsf{P})/(2nR)$.*

Lemma 4.12. *(Proof in the full version.) Consider the setting of Lemma 4.11. Given $\delta > 0$, there is a index set $\mathcal{I} \subseteq \{1, \ldots, n\}$ with $|\mathcal{I}| = O(\delta^{-d})$ and $y_0 = O(l \cdot n/\delta)$ such that the functions $\left\{f_j \,\middle|\, j \in \mathcal{I}\right\}$ form a (δ, y_0)-sketch, where $l = \Delta(\mathcal{F})$.*

We conclude that for cone fat objects, the scaling distance function they define falls under out framework.

4.2.1 The result

Theorem 4.13. *Consider a set $\mathcal{O}$ of (α, ϕ)-fat objects in $\mathbb{R}^d$, where α and ϕ are constants. Then one can compute an $(1+\varepsilon)$-AVD for the scaling distance functions induced by $\mathcal{O}$, with near linear space complexity, and logarithmic query time. See Theorem 3.15$_{\mathrm{p}}$ for the exact result.*

Note, that the result in Theorem 4.13 covers any symmetric convex metric. Indeed, given a convex symmetric shape C centered in 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). In this case, of course, the triangle inequality holds, and by scaling space we can make C fat (this does not effect the scaling distances), and now Theorem 4.13 applies. Of course, Theorem 4.13 is considerably more general, allowing each of the points to induce a different scaling distance function, and it does not have to comply with the triangle inequality.

5. CONCLUSIONS

In this paper, we investigated classes of functions whose minimization diagrams can be approximated efficiently. We defined abstract properties that if they are satisfied by the functions, then they are amenable to our framework. We then presented a preprocessing algorithm, taking near linear space and preprocessing time, and a query algorithm taking logarithmic time for approximating the minimization diagram. Maybe more 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 multiplicative 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 (which the AVD generated has near linear complexity for any constant dimension). The frame also works for Minkowski metrics of fat convex bodies. However, our main result applies to even more general distance functions.

Several questions remain open for further research:

(A) The foremost is whether the additional polylog factors on the space necessary? In particular, it seems unlikely that using WSPDs directly, as done by Arya and Malamatos, should work in the most general settings, so reducing the logarithmic dependency seems quite interesting.

(B) Specifically, can the Arya and Malamatos construction [AM02] be somehow adapted to this framework, possibly with some additional constraints on the functions, to get a linear space construction?

(C) 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.)

6. REFERENCES

[AI08] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[AM93] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:540–570, 1993.

[AM02] S. Arya and T. Malamatos. Linear-size approximate Voronoi diagrams. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 147–155, 2002.

[AMM09] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. Assoc. Comput. Mach.*, 57(1):1–54, 2009.

[AMN$^+$98] 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.*, 45(6):891–923, 1998.

[Cha10] T. M. Chan. Optimal partition trees. In *Proc. 26th Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 2010.

[CK95] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.

[Cla88] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[Cla06] K. L. Clarkson. Nearest-neighbor searching and metric space dimensions. In G. Shakhnarovich, T. Darrell, and P. Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[dBCvKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. H. Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 3rd edition, 2008.

[Eri96] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.*, 16:389–418, 1996.

[Har01] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

[Har11] S. Har-Peled. *Geometric Approximation Algorithms.* Amer. Math. Soc., 2011.

[HIM12] S. Har-Peled, P. Indyk, and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *Theory Comput.*, 8:321–350, 2012. Special issue in honor of Rajeev Motwani.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[Mat92] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[Mei93] S. Meiser. Point location in arrangements of hyperplanes. *Inform. Comput.*, 106:286–303, 1993.

[MNP06] R. Motwani, A. Naor, and R. Panigrahi. Lower bounds on locality sensitive hashing. In *Proc. 22nd Annu. ACM Sympos. Comput. Geom.*, pages 154–157, 2006.

[SA95] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications.* Cambridge University Press, New York, 1995.