On the Budgeted Hausdorff Distance Problem

Sariel Har-Peled
Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA

Benjamin Raichel
Department of Computer Science; University of Texas at Dallas; 800 W. Campbell Road; Richardson, TX 75080, USA

Abstract

Given a set P of n points in the plane, and a parameter k, we present an algorithm, whose running time is $O\left(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\right)$, with high probability, that computes a subset $Q^\star \subseteq P$ of k points, that minimizes the Hausdorff distance between the convex-hulls of $Q^\star$ and P. This is the first subquadratic algorithm for this problem if k is small.

Keywords and phrases Convex hull, Hausdorff distance, Randomization

Digital Object Identifier 10.57717/cgt.v4i2.57

Funding *Sariel Har-Peled*: Work on this paper was partially supported by NSF AF award CCF-2317241.
Benjamin Raichel: Work on this paper was partially supported by NSF AF CAREER award CCF-1750780 and NSF AF award CCF-2311179.

1 Introduction

Given a set of points P in $\mathbb{R}^d$, a natural goal is to find a small subset of it that represents the point set well. This problem has attracted a lot of interest over the last two decades, and this subset of P is usually referred to as a *coreset* [3, 1]. An alternative approximation is provided by the largest enclosed ellipsoid inside $\mathcal{C}(P)$ (here $\mathcal{C}(P)$ denotes the *convex-hull* of P) or the smallest area bounding box of P (not necessarily axis-aligned). This provides a constant approximation to the projection width of P in any direction v – that is, the projection of P into the line spanned by v is contained in the projection of the ellipsoid after appropriate constant scaling. One can show that in two dimensions, there is a subset $Q \subseteq P$ (i.e., a coreset) of size $O(1/\sqrt{\varepsilon})$ such that the projection width of P and Q is the same up to scaling by $1+\varepsilon$. See Agarwal *et al.* [3, 1] for more details.

The concept of a coreset is attractive as it provides a notion of approximating that adapts to the shape of the point set. However, an older and arguably simpler approach is to require that $\mathcal{C}(Q)$ approximates $\mathcal{C}(P)$ within a certain absolute error threshold. A natural such measure is the ***Hausdorff*** distance between sets $X, Y \subseteq \mathbb{R}^2$, which is

$$\mathsf{d}_H(X,Y) = \max\Big(\mathsf{d}(X \to Y),\, \mathsf{d}(Y \to X)\Big),$$

where

$$\mathsf{d}(X \to Y) = \max_{x\in X}\min_{y\in Y}\|xy\|. \tag{1}$$

In our specific case, the two sets are $\mathcal{C}(P)$ and $\mathcal{C}(Q)$, and let $\mathsf{D}_H(Q,P) = \mathsf{d}_H\big(\mathcal{C}(Q), \mathcal{C}(P)\big)$. The natural questions are

(I) MinCardin: Compute the smallest subset $Q \subseteq P$, such that $\mathsf{D}_H(Q,P) \le \tau$, where τ is a prespecified error threshold. Formally, let

$$\mathcal{F}_{\le\tau} = \left\{ Q \subseteq P \;\middle|\; \mathsf{D}_H(Q,P) \le \tau \right\},$$

and let $k^\star = k^\star(P,\tau) = \min_{Q \in \mathcal{F}_{\leq \tau}} |Q|$ denote the minimum cardinality of such a set Q.

(II) MinDist: Compute the subset $Q \subseteq P$ of size k, such that $\mathsf{D}_H(Q,P)$ is minimized, where k is a prespecified subset size threshold. Let $\tau^\star = \tau^\star(P,k) = \min_{Q \subseteq P : |Q| = k} \mathsf{D}_H(Q,P)$ denote the optimal radius.

The two problems are "dual" to each other – solve one, and you get a way to solve the other in polynomial time via a search on the values of the other parameter. In particular, solving both problems directly (in two dimensions) can be done via dynamic programming, but even getting a subcubic running time is not immediate in this case. Indeed, the problem seems to have a surprisingly subtle and intricate structure that make this problem more challenging than it seems at first.

Klimenko and Raichel [7] provided an $O(n^{2.53})$ time algorithm for MinCardin. Very recently, Agarwal and Har-Peled [2] provided a near-linear time algorithm for MinCardin that runs in near linear time if $k^\star = k^\star(P,\tau)$ is small. Specifically, the running time of this algorithm is $O(k^\star n \log n)$.

The purpose of this work is to come up with a subquadratic algorithm for the "dual" problem MinDist. An algorithm with running time $O(n^2 \log n)$ follows readily by computing all possible critical values, and performing a binary search over these values, using the procedure of [2] as a black box. The only subquadratic algorithm known previously was for the special case when P is in convex position, for which [7] gave an algorithm whose running time is $O(n \log^3 n)$ with high probability.[1]

Our main result is an algorithm that, given P and k as input, solves MinDist in $O\big(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\big)$ time, with high probability, see Theorem 8 for details. We believe the algorithm itself is technically interesting – it uses random sampling to reduce the range of interest into an interval containing $O(\sqrt{n})$ critical values. It then use the decision procedure of [2] as a way to compute the critical values in this interval, by "peeling" them one by one in decreasing order. Using random sampling for parametric search is an old idea, see [6] and references there.

2 Preliminaries

Given a point set X in $\mathbb{R}^2$, let $\mathcal{C}(X)$ denote its ***convex hull***. For two compact sets $X, Y \subset \mathbb{R}^2$, let $\mathsf{d}(X,Y) = \min_{x \in X, y \in Y} \|xy\|$ denote their distance. For a single point x let $\mathsf{d}(x,Y) = \mathsf{d}(\{x\},Y)$.

Consider two finite point sets $Q \subseteq P \subset \mathbb{R}^2$, and observe that

$$\mathsf{D}_H(Q,P) = \mathsf{d}_H\big(\mathcal{C}(Q), \mathcal{C}(P)\big) = \max_{p \in P} \mathsf{d}\big(p, \mathcal{C}(Q)\big),$$

see Eq. (1). The first equality above is by definition, and the second is since $Q \subseteq P$ and so we have that $\mathcal{C}(Q) \subseteq \mathcal{C}(P)$, and moreover the furthest point in $\mathcal{C}(P)$ from $\mathcal{C}(Q)$ is always a point in P.

In this paper we consider the following two related problems, where for simplicity, we assume that P is in general position.

[1] The general problem cannot be reduced to the convex position case. To see this, suppose P consists of 4 points, 3 forming an equilateral triangle, with the fourth at the midpoint of an edge but infinitesimally pushed towards the interior of the triangle. Then the optimal 2 point approximation is this fourth point along with the vertex opposite its edge.

▶ **Problem 1** (MinCardin). *Given a set $P \subset \mathbb{R}^2$ of n points, and a value $\tau > 0$, find the smallest cardinality subset $Q \subseteq P$ such that $\mathsf{D}_H(Q, P) \leq \tau$.*

▶ **Problem 2** (MinDist). *Given a set $P \subset \mathbb{R}^2$ of n points, and an integer k, find the subset $Q \subseteq P$ that minimizes $\mathsf{D}_H(Q, P)$ subject to the constraint that $|Q| \leq k$.*

For either problem let $Q^\star$ denote an optimal solution. For MinCardin let $k^\star = k^\star(P, \tau) = |Q^\star|$, and for MinDist let $\tau^\star = \tau^\star(P, k) = \mathsf{D}_H(Q^\star, P)$. The algorithms discussed in this paper will output the set $Q^\star$, though when it eases the exposition, we occasionally refer to $k^\star$ as the solution to MinCardin and $\tau^\star$ as the solution to MinDist.

We make use of the following result from [2]. The statement of this result in [2] only depends on $k^\star$, though it can be easily adapted to the result below which allows for querying values of k in time dependent on k and not $k^\star$, which is required for our search procedure to achieve the desired running time.

▶ **Theorem 3** ([2]). *Given as an input a point set P and parameters k and τ, let $k^\star = k^\star(P, \tau)$. There is a procedure* **`decider`**(P, τ, k)*, that in $O(nk \log n)$ time, either returns that "$k^\star > k$", or alternatively returns a set $Q^\star \subseteq P$, such that $|Q^\star| = k^\star \leq k$, and $\mathsf{D}_H(Q^\star, P) \leq \tau$.*

The above theorem readily implies that the problem MinCardin can be solved in $O(nk^\star \log n)$ time.

Given an input of size n, an algorithm runs in $O(f(n))$ time *with high probability*, if for any chosen constant $c > 0$, there is a constant α_c such that the running time exceeds $\alpha_c f(n)$ with probability $< 1/n^c$.

3 Algorithm

3.1 The canonical set

Given an instance P, k of MinDist, let $Q^\star$ denote an optimal solution. Recall that

$$\tau^\star = \mathsf{D}_H(Q^\star, P) = \max_{p \in P} \mathsf{d}(p, \mathcal{C}(Q^\star)).$$

Assume that $\tau^\star > 0$, which can easily be determined by checking if $|\mathsf{V}(\mathcal{C}(P))| > k$, where $\mathsf{V}(\mathcal{C}(P))$ denotes the set of vertices of $\mathcal{C}(P)$. Let

$$p = \arg\max_{p' \in P} \mathsf{d}(p', \mathcal{C}(Q^\star)),$$

and let q be its projection onto $\mathcal{C}(Q^\star)$, i.e. $\tau^\star = \|pq\|$. Observe that q either lies on a vertex of $\mathcal{C}(Q^\star)$ or in the interior of a bounding edge. Since $Q^\star \subseteq P$, we can conclude that $\tau^\star$ is either (i) the distance between two points in P, or (ii) the distance from a point in P to the line passing through two other points from P. Note that, in case (ii), q must be the orthogonal projection of p on to the line ℓ supporting the edge, and that p must be the furthest point from ℓ out of the points that lie in one of its two defining halfplanes. In particular, for an ordered pair $a, b \in P$ define $\ell_{a,b}$ as the line through a and b, directed from a to b, and let $P_{a,b}$ be the subset of P lying in the halfspace bounded by and to the left of $\ell_{a,b}$. We thus define the following two sets.

$$\mathcal{V} = \left\{ \|xy\| \;\middle|\; x, y \in P \right\} \qquad \text{and} \qquad \mathcal{L} = \left\{ \max_{p \in P_{a,b}} \mathsf{d}(p, \ell_{a,b}) \;\middle|\; a, b \in P \right\}. \tag{2}$$

The set $\Xi = \mathcal{V} \cup \mathcal{L}$ is the ***canonical set*** of distance values (i.e., the set of all *critical* values). By the above discussion, we have $\tau^\star \in \Xi$.

Observe that $\mathcal{V}$ and $\mathcal{L}$ (and hence Ξ) have quadratic size. Thus we will not explicitly compute these sets. Instead we will search over $\mathcal{V}$ using the following "median" selection procedure.

▶ **Theorem 4** ([4]). *Given a set $P \subset \mathbb{R}^2$ of n points, and an integer $k > 0$, with high probability, in $O(n^{4/3})$ time, one can compute the value of rank k in $\mathcal{V}$.*

For values in $\mathcal{L}$, the algorithm samples values and searches over them, using a procedure loosely inspired by [6]. For that we have the following standard lemma, whose proof we include for completeness.

▶ **Lemma 5** ([5]). *Let $P \subset \mathbb{R}^2$ be a set of n points. Then in $O(n \log n)$ time one can build a data structure such that for any query vector $\overrightarrow{u}$, in $O(\log n)$ time, it returns the point of P extremal in the direction $\overrightarrow{u}$, i.e. the point maximizing the dot product with $\overrightarrow{u}$. Let $\texttt{extremal}(\overrightarrow{u})$ denote this query procedure.*

Proof. Let $\mathsf{V}(\mathcal{C}(P)) = \{q_1, \ldots, q_k\}$ be labelled in clockwise order. Let $U(q_i)$ be the set of unit vectors $\overrightarrow{u}$ such that when we translate P so that q_i lies at the origin, then $\overrightarrow{u}$ lies in the exterior angle between the normals of $q_{i-1}q_i$ and q_iq_{i+1}. Observe that $\texttt{extremal}(\overrightarrow{u}) = q_i$ precisely when $u \in U(q_i)$. Moreover, the $U(q_i)$ define a partition of the set of all unit vectors into k sets. Thus if we maintain these intervals in an array, sorted in clockwise order, then in $O(\log k) = O(\log n)$ time we can binary search to find which interval $\overrightarrow{u}$ falls in. It takes $O(n \log n)$ time to compute $\mathcal{C}(P)$ and thus the data structure. ◀

In the next section, given a directed line ℓ, we use the above lemma to make extremal queries for the normal of ℓ lying in its left defining halfplane. This lets us evaluate extreme points for lines supporting edges of the current hull, as well as allows us to sample values from $\mathcal{L}$, for which we have the following.

▶ **Corollary 6.** *Given a set $P \subset \mathbb{R}^2$ of n points, after $O(n \log n)$ preprocessing time, one can return, in $O(\log n)$ time, a value sampled uniformly at random from $\mathcal{L}$.*

Proof. Sample uniformly at random a pair of points from P, and then use Lemma 5 for the normal to the line passing through this pair of points. ◀

3.2 The algorithm in stages

The input is a set P of n points, and a parameter k. The task at hand is to compute the minimum distance $\tau^\star$, such that there is a subset $Q \subseteq P$ of size k, such that $\mathsf{D}_H(Q, P) \le \tau^\star$.

Searching and testing for the optimal value.

The algorithm maintains an interval (r, R), such that the following invariants are maintained:
(I) $k^\star(P, r) > k$,
(II) $k^\star(P, R) \le k$, and
(III) $\tau^\star(P, k) \in (r, R)$.

(The first two conditions are actually implied by the last condition, though for clarity we list all three.) In the following, let $\delta > 0$ denote an infinitesimal, see Remark 7 below. Given a value $\tau \in (r, R)$, one can decide if $\tau = \tau^\star(P, k)$, by running $\texttt{decider}(P, \tau, k)$ and $\texttt{decider}(P, \tau - \delta, k)$, see Theorem 3. If $\texttt{decider}(P, \tau - \delta, k)$ returns that $k^\star(P, \tau - \delta) > k$ and $k^\star(P, \tau) = k$ then clearly τ is the desired optimal value. In this case, the algorithm returns this value and stops.

▶ Remark 7 (Running $\texttt{decider}(P, \tau - \delta, k)$). The above algorithm can be described without using infinitesimals, but this is somewhat cleaner. Over-simplifying, the algorithm of **decider** boils down to deciding if a circular interval graph has a cover of size k. Originally, the intervals are closed, but instead we can treat them as open when solving for r (i.e., this corresponds to the $r - \delta$ case). This essentially requires implementing two versions of decider for the closed/open cases, respectively. Both versions have the same asymptotic running time.

Updating the current interval.

After testing if $\tau = \tau^\star(P, k)$ for a value $\tau \in (r, R)$ as described above, if $\tau \neq \tau^\star(P, k)$ then the algorithm can update the current interval. Indeed, if $\texttt{decider}(P, \tau, k)$ returns that $k^\star(P, \tau) > k$, then the algorithm sets the current interval to (τ, R). Otherwise, $\texttt{decider}(P, \tau - \delta, k)$ returned that $k^\star(P, \tau - \delta) \leq k$ and so the algorithm sets the current interval to (r, τ).

Stage I: Handling pairwise distances.

The algorithm sets the initial interval to $(0, \infty)$. (Recall as discussed above that we can assume $\tau^\star > 0$.) The algorithm then binary searches over all pairwise distance from $\mathcal{V} = \binom{P}{2}$ by using the distance selection procedure of Theorem 4, in the process repeatedly updating the current interval as described above. If $\tau^\star \in \mathcal{V}$, then the algorithm will terminate when the search considers this value. Otherwise, this search reduces the current interval to two consecutive pairwise distances from $\mathcal{V}$, $r < R$, such that $\tau^\star \in (r, R)$ and the current interval (r, R) contains no pairwise distance of P in its interior.

Stage II: Sampling edge-vertex distances.

The algorithm samples a set Π of $O(n^{3/2} \log n)$ values from $\mathcal{L}$, see Eq. (2), using Corollary 6. Let U be the subset of values of Π that lie inside the current interval. The algorithm binary searches over U, repeatedly updating the current interval as described above (by doing median selection so that U's cardinality halves at each iteration). If $\tau^\star \in U$ then the algorithm will terminate when the search considers this value. Otherwise, the search further reduces to the interval to $I' = (r', R')$. (Which as discussed below, with high probability, contains $O(\sqrt{n})$ values from $\mathcal{L}$.)

Stage III: Peeling the critical edge-vertex distances.

The algorithm now continues the search on the interval $I' = (r', R')$ and critical values in it, $I' \cap \Xi = I' \cap \mathcal{L}$. In particular, the solution computed by $\texttt{decider}(P, R', k)$ is a set $Q \subseteq P$ of size $\leq k$ such that $\mathsf{D}_H(Q, P) \leq R'$. For every edge on the boundary of $\mathcal{C}(Q)$ the algorithm now computes the point from P furthest away from the line supporting the edge (among the points in the halfplane not containing $\mathcal{C}(Q)$), using extremal queries from Lemma 5. Let α be the largest such computed value over all the edges, and observe that $\alpha = \mathsf{D}_H(Q, P)$.[2] If $\alpha < R'$, then $\alpha \geq \tau^\star(P, k)$. The algorithm tests if $\alpha = \tau^\star(P, k)$, and if so it terminates. Otherwise, it must be that the optimal value lies in the interval (r', α). As $\alpha \in (r', R')$

[2] $\mathsf{D}_H(Q, P)$ must be realized at a value from $\mathcal{L}$ as Stage I eliminated $\mathcal{V}$ values, and thus it sufficed to consider furthest distances to the lines supporting edges rather than the edges themselves, since at the maximum such value they must align.

and $\alpha \in \mathcal{L}$, our new interval (r', α) has at least one less value from $\mathcal{L}$. The algorithm now continues to the next iteration of Stage III.

The case when $\alpha = R'$ (i.e., the higher end of the active interval) is somewhat more subtle. The algorithm calls $\texttt{decider}(P, k, \alpha - \delta)$ to compute a set Q' that realizes $k^\star(P, \alpha - \delta)$, where δ is an infinitesimal. Observe that $k^\star(P, \alpha - \delta) \le k$, as otherwise $\alpha = R'$ was the desired optimal value. Let $\beta = \mathsf{D}_H(Q', P)$, which can be computed in a similar fashion using Q' as α was computed using Q. The algorithm tests if $\beta = \tau^\star(P, k)$, and if so it terminates. Otherwise, by the same reasoning used above for α, we can conclude our new interval (r', β) has at least one fewer value from $\mathcal{L}$, and thus the algorithm continues to the next iteration of Stage III on the interval (r', β).

3.3 Analysis

Correctness.

The correctness of the algorithm is fairly immediate given the discussion above. Namely, the algorithm maintains an interval (r, R) with the invariant that $\tau^\star(P, k) \in (r, R)$ (where initially this interval is $(0, \infty)$). In each step of each stage a value $\tau \in (r, R)$ that is either from $\mathcal{V}$ (in Stage I) or from $\mathcal{L}$ (in Stages II and III) is determined. For this value τ we then update the current interval as described above. Namely, we query $\texttt{decider}(P, \tau, k)$ and $\texttt{decider}(P, \tau - \delta, k)$. If these calls return that $k^\star(P, \tau) \le k$ and $k^\star(P, \tau - \delta) > k$ then $\tau = \tau^\star(P, k)$ and the algorithm terminates. Otherwise, if $k^\star(P, \tau) > k$ the algorithm proceeds on (τ, R), and if $k^\star(P, \tau - \delta) \le k$ then it proceeds on (r, τ). In either case the interval contains at least one fewer value from Ξ, and thus eventually the algorithm must terminate with the value $\tau^\star(P, k)$.

Running time analysis.

In Stage I the algorithm performs a binary search over $\mathcal{V} = \binom{P}{2}$. This is done using the distance selection procedure of Theorem 4, which with high probability takes $O(n^{4/3})$ time to determine each next query value. Each query is answered using the $O(nk \log n)$ time $\texttt{decider}(P, \cdot, k)$ from Theorem 3. Thus in total Stage I takes $O\big((n^{4/3} + nk \log n) \log n\big)$ time with high probability. Here, by the union bound, a polynomial number of high probability events (i.e. the events that each call to selection occurs in $O(n^{4/3})$ time), all occur simultaneously with high probability.

In Stage II the algorithm samples $O(n^{3/2} \log n)$ values from $\mathcal{L}$ using the $O(\log n)$ time sampling procedure of Corollary 6. Next, the algorithm binary searches over these values (this time directly), again using $\texttt{decider}(P, \cdot, k)$. Thus in total Stage II takes $O(n^{3/2} \log^2 n + nk \log^2 n)$ time.

Stage III begins with some interval (r', R'). Let $X = |\mathcal{L} \cap (r', R')|$. In each iteration of Stage III, for some subset $Q \subseteq P$ of size at most k, the algorithm computes $\alpha = \mathsf{D}_H(Q, P)$. This is done using at most k calls to the $O(\log n)$ query time Lemma 5. (This same step is potentially done a second time for $\beta = \mathsf{D}_H(Q', P)$). Each iteration of Stage III also performs a constant number of calls to $\texttt{decider}(P, \cdot, k)$, thus is total one iteration takes $O(k \log n + nk \log n) = O(nk \log n)$ time. As argued above each iteration of Stage III reduces the number of values from $\mathcal{L}$ in the active interval by at least 1, and thus runs for at most X iterations. Thus the total time of Stage III is $O(Xnk \log n)$.

Observe that since Stage II sampled a set Π of $O(n^{3/2} \log n)$ values from the $O(n^2)$ sized set $\mathcal{L}$, the interval between any two consecutive values of Π with high probability has $O(\sqrt{n})$ values from $\mathcal{L}$. As the interval $I' = (r', R')$ returned by Stage II is such an interval, with high probability $X = O(\sqrt{n})$. As the running time of Stage II dominates the running time of Stage I (with high probability), we thus have that with high probability the total time of all stages is

$$O(n^{3/2}\log^2 n + (\log n + X)nk\log n) = O(n^{3/2}\log^2 n + n^{3/2}k\log n + nk\log^2 n)$$
$$= O(n^{3/2}(k + \log n)\log n).$$

Slightly improving the running time.

Observe that if the algorithm samples $O(nt\log n)$ values in stage II, then with high probability the last two stages take

$$O\left(nt\log^2 n + \left(\frac{n^2}{nt} + \log n\right)kn\log n\right)$$

time. Solving for t, we have

$$nt\log^2 n = (n^2/t)k\log n \implies t^2 = nk/\log n.$$

Thus, setting $t = \sqrt{nk/\log n}$, and including the running time of stage I, we get the improved high probability running time bound

$$O\left(n^{4/3}\log n + nt\log^2 n + \left(\frac{n^2}{nt} + \log n\right)kn\log n\right)$$
$$= O\big(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\big).$$

In summary, we get the following result.

▶ **Theorem 8.** *Given an instance of MinDist, consisting of a set $P \subset \mathbb{R}^2$ of n points and an integer k, the above algorithm computes a set $Q^\star \subseteq P$, of size k, that realizes the minimum Hausdorff distance between the convex-hulls of P and $Q^\star$ among all such subsets – that is, $\tau^\star(P,k) = \mathsf{D}_H(P,Q^\star)$. The running time of the algorithm is $O\big(n^{3/2}\sqrt{k}\log^{3/2} n + kn\log^2 n\big)$ with high probability.*

We remark that under the reasonable assumption that $k = O(n/\log n)$ the running time can be stated more simply as $O(n^{3/2}\sqrt{k}\log^{3/2} n)$.

4 Conclusions

The most interesting open problem left by our work is whether one can get a near-linear running time if k is small. Even beating $O(n^{4/3})$ seems challenging. On the other hand, if one is willing to use $2k$ points then a near linear running time is achievable [7]. However, using less than $2k$ points without increasing the Hausdorff distance in near linear time seems challenging.

References

1 P. K. Agarwal, S. Har-Peled, and K. Varadarajan. Geometric approximation via coresets. In J. E. Goodman, J. Pach, and E. Welzl, editors, *Combinatorial and Computational Geometry*, Math. Sci. Research Inst. Pub. Cambridge, New York, NY, USA, 2005. URL: `https://sarielhp.org/p/04/survey/`.

2 Pankaj K. Agarwal and Sariel Har-Peled. Computing optimal kernels in two dimensions. In *Proc. 39th Int. Annu. Sympos. Comput. Geom.* (SoCG), LIPIcs, page to appear. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://arxiv.org/abs/2207.07211.

3 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004. doi:10.1145/1008731.1008736.

4 Timothy M. Chan and Da Wei Zheng. Hopcroft's problem, log-star shaving, 2d fractional cascading, and decision trees. *CoRR*, abs/2111.03744, 2021. URL: https://arxiv.org/abs/2111.03744, arXiv:2111.03744.

5 D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theor. Comput. Sci.*, 27:241–253, 1983.

6 Sariel Har-Peled and Benjamin Raichel. The Fréchet distance revisited and extended. *ACM Trans. Algorithms*, 10(1):3:1–3:22, 2014. doi:10.1145/2532646.

7 Georgiy Klimenko and Benjamin Raichel. Fast and exact convex hull simplification. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *Proc. 41th Conf. Found. Soft. Tech. Theoret. Comput. Sci.* (FSTTCS), volume 213 of *LIPIcs*, pages 26:1–26:17, Wadern, Germany, 2021. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2021.26.