Improved Approximation Algorithms for Tverberg Partitions

Sariel Har-Peled
Department of Computer Science, University of Illinois, Urbana, IL, USA

Timothy Zhou
Department of Computer Science, University of Illinois, Urbana, IL, USA

Abstract

Tverberg's theorem states that a set of n points in $\mathbb{R}^d$ can be partitioned into $\lceil n/(d+1) \rceil$ sets whose convex hulls all intersect. A point in the intersection (aka Tverberg point) is a centerpoint, or high-dimensional median, of the input point set. While randomized algorithms exist to find centerpoints with some failure probability, a partition for a Tverberg point provides a certificate of its correctness.

Unfortunately, known algorithms for computing exact Tverberg points take $n^{O(d^2)}$ time. We provide several new approximation algorithms for this problem, which improve running time or approximation quality over previous work. In particular, we provide the first strongly polynomial (in both n and d) approximation algorithm for finding a Tverberg point.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geometric spanners, vertex failures, robustness

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.51

Related Version *Full Version*: https://arxiv.org/abs/2007.08717 [17]

Funding *Sariel Har-Peled*: Work on this paper was partially supported by a NSF AF award CCF-1907400.
Timothy Zhou: Work on this paper was partially supported by a NSF AF award CCF-1907400.

Acknowledgements The authors thank Timothy Chan, Wolfgang Mulzer, David Rolnick, and Pablo Soberon-Bravo for providing useful references.

1 Introduction

Given a set P of n points in the plane and a query point q, classification problems ask whether q belongs to the same class as P. Some algorithms use the convex hull $\mathcal{CH}(P)$ as a decision boundary for classifying q. However, in realistic datasets, P may be noisy and contain outliers, and even one faraway point can dramatically enlarge the hull of P. Thus, we would like to measure how deeply q lies within P in way that is more robust against noise.

In this paper, we investigate the notion of Tverberg depth. However, there are many related measures of depth in the literature, including:

1. **Tukey depth**. The Tukey depth of q is the minimum number of points that must be removed before q becomes a vertex of the convex hull. Computing the depth is equivalent to computing the closed halfspace that contains q and the smallest number of points of P, and this takes $O(n \log n)$ time in the plane [7].
2. **Centerpoint**. In $\mathbb{R}^d$, a point with Tukey depth $n\alpha$ is an *α-centerpoint*. There is always a $1/(d+1)$-centerpoint, known simply as the centerpoint, which can be computed exactly in $O(n^{d-1})$ time [19, 7]. It can be approximated using the centerpoint of a sample [11], but getting a polynomial-time (in both n and d) approximation algorithm proved challenging. Clarkson *et al.* [11] provided an algorithm that computes a $1/4d^2$-centerpoint in roughly

$O(d^9)$ time. Miller and Sheehy [23] derandomized it to find a (roughly) $1/2d^2$-centerpoint in $n^{O(\log d)}$ time. More recently, Har-Peled and Mitchell [16] improved the running time to compute a (roughly) $1/d^2$-centerpoint in (roughly) $O(d^7)$ time.

3. **Onion depth**. Imagine peeling away the vertices of the current convex hull and removing them from P. The onion depth is the number of layers which must be removed before the point q is exposed. The convex layers of points in the plane can be computed in $O(n \log n)$ time by an algorithm of Chazelle [8]. The structure of convex layers is well-understood for random points [12] and grid points [13].
4. **Uncertainty.** Another model considers uncertainty about the locations of the points. Suppose that each point of P has a certain probability of existing, or alternatively, its location is given via a distribution. The depth of query point q is the probability that q is in the convex hull once P has been sampled. Under certain assumptions, this probability can be computed exactly in $O(n \log n)$ time [2]. Unfortunately, the computed value might be very close to zero or one, and therefore tricky to interpret.
5. **Simplicial depth.** The simplicial depth of q is the number of simplices induced by P containing it. This number can be approximated quickly after some preprocessing [1]. However, it can be quite large for a point which is intuitively shallow.

Tverberg depth

Given a set P of n points in $\mathbb{R}^d$, a **Tverberg partition** is a partition of P into k disjoint sets $P_1, \ldots, P_k$ such that $\bigcap_i \mathcal{CH}(P_i)$ is not empty. A point in this intersection is a **Tverberg point**. Tverberg's theorem states that P has a Tverberg partition into $\lceil n/(d+1) \rceil$ sets. In particular, the *Tverberg depth* (*T-depth*) of a point q is the maximum size k of a Tverberg partition such that $q \in \bigcap_{i=1}^{k} \mathcal{CH}(P_i)$.

By definition, points of T-depth $n/(d+1)$ are centerpoints for P. In the plane, Reay [25] showed that if a point has Tukey depth $k \leq |P|/3$, then the T-depth of q is k. This property is already false in three dimensions [4]. The two-dimensional case was handled by Birch [5], who proved that any set of n points in the plane can be partitioned into $n/3$ triples whose induced triangles have a common intersection point.

Computing a Tverberg point

For work on computing approximate Tverberg points, see [23, 24, 26, 9] and the references therein. Currently, no polynomial-time (in both n and d) approximation algorithm is known for computing Tverberg points. This search problem is believed to be quite hard, see [22].

Algorithms for computing an *exact* Tverberg point of T-depth $n/(d+1)$ implement the construction implied by the original proof. The runtime of such an algorithm is $d^{O(d^2)} n^{d(d+1)+1}$, see the full version [17]. As previously mentioned, the exception is in two dimensions, where the algorithm of Birch [5] runs in $O(n \log n)$ time. But even in three dimensions, we are unaware of an algorithm faster than $O(n^{13})$.

Convex combinations and Carathéodory's theorem

The challenge in finding a Tverberg point is that we have few subroutines at our disposal with runtimes polynomial in d. Consider the most basic task – given a set P of n points and a query point q, decide if q lies inside $\mathcal{CH}(P)$, and if so, compute the convex combination of q in term of the points of P. This problem can be reduced to linear programming. Currently, the fastest strongly polynomial LP algorithms run in super-polynomial time $2^{O(\sqrt{d \log d})} + O(d^2 n)$

[10, 21], where d is the number of variables and n is the number of constraints. However, any given convex combination of P representing q can be sparsified in polynomial time into a convex combination using only $d+1$ points of P. Lemma 22 describes this algorithmic version of Carathéodory's theorem.

Radon partitions in polynomial time

Finding points of T-depth 2 is relatively easy. Any set of $d+2$ points in $\mathbb{R}^d$ can be partitioned into two disjoint sets whose convex hulls intersect, and a point in the intersection is a *Radon point*. Radon points can be computed in $O(d^3)$ time by solving a linear system with $d+2$ variables. Almost all the algorithms for finding Tverberg points mentioned above amplify the algorithm for finding Radon points.

Our results

The known and new results are summarized in Table 1.1. In Section 2, we review preliminary information and known results, which include the following.

1. **An exact algorithm.** The proof of Tverberg's theorem is constructive and leads to an algorithm with running time $O(n^{d(d+1)})$. It seems that the algorithm has not been described and analyzed explicitly in the literature. For the sake of completeness, we provide this analysis in the full version [17].
2. **In two dimensions.** Given a set P of n points in the plane and a query point q of Tukey depth k, Birch's theorem [5] implies that q can be covered by $\min(k, \lfloor n/3 \rfloor)$ vertex-disjoint triangles of P. One can compute k and this triangle cover in $O(n + k \log k)$ time, and use them to compute a Tverberg point of depth $\lfloor n/3 \rfloor$ in $O(n \log n)$ time.

In Section 3, we provide improved algorithms for computing Tverberg points and partitions.

1. **Projections in low dimensions.** We use projections to find improved approximation algorithms in dimensions 3 to 7, see Table 1.2. For example, in three dimensions, one can compute a point with T-depth $n/6$ in $O(n \log n)$ time.
2. **An improved quasi-polynomial algorithm.** We modify the algorithm of Miller and Sheehy to use a buffer of free points. Coupled with the algorithm of Mulzer and Werner [24], this idea yields an algorithm that computes a point of T-depth $\geq (1-\delta)n/2(d+1)^2$ in $d^{O(\log(d/\delta))}n$ time. This improves the approximation quality of the algorithm of [24] by a factor of $2(d+1)$, while keeping (essentially) the same running time.
3. **A strongly polynomial algorithm.** In Section 3.3, we present the first strongly polynomial approximation algorithm for Tverberg points, with the following caveats:
   a. the algorithm is randomized, and might fail,
   b. one version returns a Tverberg partition, but not a point that lies in its intersection,
   c. the other (inferior) version returns a Tverberg point and a partition realizing it, but not the convex combination of the Tverberg point for each set in the partition.

   Specifically, one can compute a partition of P into $n/O(d^2 \log d)$ sets, such that the intersection of their convex hulls is nonempty (with probability close to one), but without finding a point in the intersection. Alternatively, one can also compute a Tverberg point, but the number of sets in the partition decreases to $n/O(d^3 \log d)$.

Table 1.1 The known and improved results for Tverberg partition. The notation $\mathcal{O}_w$ hides terms with polylogarithmic dependency on size of the numbers, see Remark 21. The parameter δ can be freely chosen.

Depth	Running time	Ref / Comment
$n/(d+1)$	$d^{O(d^2)}n^{d(d+1)+1}$	Tverberg theorem
$d=2:\ n/3$	$O(n\log n)$	[5]: Theorem 8
$\frac{n}{2(d+1)^2}$	$n^{O(\log d)}$	Miller and Sheehy [23]
$\frac{n}{4(d+1)^3}$	$d^{O(\log d)}n$	Mulzer and Werner [24]
$\frac{n}{2d(d+1)^2}$	$\mathcal{O}_w(n^4)$	Rolnick and Soberón [26]
$\frac{(1-\delta)n}{d(d+1)}$	$(d/\delta)^{O(d)}+\mathcal{O}_w(n^4)$	Rolnick and Soberón [26]
	New results	
$\frac{(1-\delta)n}{2(d+1)^2}$	$d^{O(\log(d/\delta))}n$	Theorem 18
$\frac{n}{O(d^2\log d)}$	$O(dn)$	Lemma 19: Only partition
$\frac{n}{O(d^3\log d)}$	$O(dn+d^7\log^6 d)$	Lemma 20: Partition + point, but no convex combination
$\frac{n}{O(d^2\log d)}$	$\mathcal{O}_w(n^{5/2}+nd^3)$	Lemma 24: Weakly polynomial
$\frac{(1-\delta)n}{2(d+1)^2}$	$d^{O(\log\log(d/\delta))}\mathcal{O}_w(n^{5/2})$	Theorem 25: Weakly quasi polynomial
$\frac{(1-\delta)n}{d(d+1)}$	$O(c+c'n+d^2n\log^2 n)$ $c=d^{O(d)}/\delta^{2(d-1)}$ $c'=2^{O(\sqrt{d\log d})}$	Lemma 26: Useful for low dimensions

Table 1.2 The best approximation ratios for Tverberg depth in low dimensions, with nearly linear-time algorithms, as implied by Lemma 16. Note that the new algorithm is no longer an improvement in dimension 8. We are unaware of any better approximation algorithms (except for running the exact algorithm for Tverberg's point, which requires $n^{O(d^2)}$ time).

Dim	T. Depth	New depth	Known	Ref	Comment
3	$n/4$	$n/6$	$n/8$		
4	$n/5$	$n/9$	$n/16$	[24]	
5	$n/6$	$n/18$	$n/32$		
6	$n/7$	$n/27$	$(1-\delta)n/42$		Original paper describes a weakly
7	$n/8$	$n/54$	$(1-\delta)n/56$	[26]	polynomial algorithm. The improved
8	$n/9$	$n/81$	$(1-\delta)n/72$		algorithm is described in Lemma 26.

4. **A weakly polynomial algorithm.** Revisiting an idea of Rolnick and Soberón [26], we use algorithms for solving LPs. The resulting running time is either weakly polynomial (depending logarithmically on the relative sizes of the numbers in the input) or super-polynomial, depending on the LP solver. In particular, the randomized, strongly polynomial algorithms described above can be converted into constructive algorithms that compute the convex combination of the Tverberg point over each set in its partition. Having computed approximate Tverberg points of T-depth $\geq n/O(d^2 \log d)$, we can feed them into the buffered version of Miller and Sheehy's algorithm to compute Tverberg points of depth $\geq (1-\delta)n/2(d+1)^2$. This takes $d^{O(\log\log(d/\delta))}\mathcal{O}_w(n^{5/2})$ time, where $\mathcal{O}_w$ hides polylogarithmic terms in the size of the numbers involved, see Remark 21.
5. **Faster approximation in low dimensions.** One can compute (or approximate) a centerpoint, then repeatedly extract simplices covering it until the centerpoint is exposed. This leads to an $O_d(n^2)$ approximation algorithm [26]. Since O_d hides constants that depend badly on d, this method is most useful in low dimensions. By random sampling, we can speed up this algorithm to $O_d(n \log^2 n)$ time.

2 Background, preliminaries and known results

▶ **Definition 1.** *A **Tverberg partition** (or a **log**) of a set $P \subseteq \mathbb{R}^d$, for a point q, is a set $\ell = \{P_1, \ldots, P_k\}$ of vertex-disjoint subsets of P, each containing at most $d+1$ points, such that $q \in \mathcal{CH}(P_i)$, for all i. The **rank** of ℓ is $k = |\ell|$. The maximum rank of any log of q is its **Tverberg depth** (**T-depth**).*

*A set P_j in a log is a **batch**. For every batch $P_j = \{p_1, \ldots, p_{d+1}\}$ in the log, we store the convex coefficients $\alpha_1, \ldots, \alpha_{d+1} \geq 0$ such that $\sum_i \alpha_i p_i = q$ and $\sum_i \alpha_i = 1$. A pair (q, ℓ) of a point and its log is a **site**.*

Tverberg's theorem states that, for any set of n points in $\mathbb{R}^d$, there is a point in $\mathbb{R}^d$ with Tverberg depth $\lceil n/(d+1) \rceil$. For simplicity, we assume the input is in general position.

2.1 An exact algorithm

The constructive proof of Tverberg and Vrećica [28] implies an algorithm for computing exact Tverberg points – see [17] for details.

▶ **Lemma 2.** *Let P be a set of n points in $\mathbb{R}^d$. In $d^{O(d^2)} n^{d(d+1)+1}$ time, one can compute a point q and a partition of P into disjoint sets $P_1, \ldots, P_r$ such that $q \in \bigcap_i \mathcal{CH}(P_i)$, where $r = \lceil n/(d+1) \rceil$.*

2.2 In two dimensions

We first review Tukey depth, which is closely related to Tverberg depth in two dimensions.

▶ **Definition 3.** *The **Tukey depth** of a point q in a set $P \subseteq \mathbb{R}^d$, denoted by $\mathsf{d}_{\mathsf{TK}}(q)$, is the minimum number of points contained in any closed halfspace containing q.*

The following result is implicit in the proof of Theorem 5.2 in [6]. Chan solves the decision version of the Tukey depth problem, while we need to compute it explicitly, resulting in a more involved algorithm. For the sake of completeness, we provide the details in Appendix A.

▶ **Lemma 4** (Proof in Appendix A)**.** *Given a set P of n points in the plane and a query point q, such that $P \cup \{q\}$ is in general position, one can compute, in $O(n + k \log k)$ time, the Tukey depth k of q in P. The algorithm also computes the halfplane realizing this depth.*

For shallow points in the plane, the Tukey and Tverberg depths are equivalent, and we can compute the associated Tverberg partition.

▶ **Lemma 5** (Proof in Appendix B)**.** *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = \mathsf{d}_{\mathsf{TK}}(q) \leq n/3$. Then one can compute a log for q of rank k in $O(n + k \log k)$ time.*

The Tukey depth of a point can be as large as $\lfloor n/2 \rfloor$. Indeed, consider the vertices of a regular n-gon (for odd n), the polygon center has depth $\lfloor n/2 \rfloor$.

▶ **Lemma 6.** *Let P be a set of n points in the plane, and let q be a query point such that $P \cup \{q\}$ is in general position, and q has Tukey depth larger than $n/3$. Then, one can compute a log for q of this rank in $O(n \log n)$ time.*

Proof. Let $N = \lfloor n/3 \rfloor$. Assume q that is the origin, and sort the points of P in counterclockwise order, where p_i is the ith point in this order, for $i = 1, \ldots, n$. For $i = 1, \ldots, N$, let $\triangle_i = \triangle p_i p_{N+i} p_{2N+i}$. We claim that $\angle p_i q p_{N+i} \leq \pi$. Otherwise, there is a halfspace containing fewer than N points induced by the line passing through p_i and q. Similarly, $\angle p_{N+i} q p_{2N+i} \leq \pi$ and $\angle p_{2N+i} q p_i \leq \pi$, so that q lies inside $\triangle_i$, as desired. ◀

▶ **Theorem 7.** *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = \mathsf{d}_{\mathsf{TK}}(q)$ is the Tukey depth of q in P. Then one can compute the Tukey depth k of q, along with a log of rank $\tau = \min(\lfloor n/3 \rfloor, k)$, in $O(n + k \log k)$ time.*

Proof. Compute the Tukey depth of q in $O(n + k \log k)$ time, using the algorithm of Lemma 4. If $k \leq n/3$, then compute the log using the algorithm of Lemma 5, and otherwise using the algorithm of Lemma 6. ◀

The above implies the following theorem of Birch, which predates Tverberg's theorem.

▶ **Theorem 8** ([5])**.** *Let P be a set of $n = 3k$ points in the plane. Then there exists a partition of P into k vertex-disjoint triangles, such that their intersection is not empty. The partition can be computed in $O(n \log n)$ time.*

Proof. A centerpoint of P can be computed in $O(n \log n)$ time [7]. Such a centerpoint has Tukey depth at least k, so the algorithm of Lemma 6 partitions P into k triangles. ◀

▶ Remark 9. (A) Note that Tverberg's theorem in the plane is slightly stronger – it states that any point set with $3k - 2$ points has Tverberg depth k. In such a decomposition, some of the sets may be pairs of points or singletons.

(B) In the colored version of Tverberg's theorem in the plane, one is given $3n$ points partitioned into three classes of equal size. Agarwal *et al.* [3] showed how to compute a decomposition into n triangles covering a query point, where every triangle contains a vertex of each color (if such a decomposition exists). This problem is significantly more difficult, and their running time is a prohibitive $O(n^{11})$.

2.3 Miller and Sheehy's algorithm

Here, we review Miller and Sheehy's [23] approximation algorithm for computing a Tverberg point before describing our improvement.

Radon partitions

Radon's theorem states that a set P of $d+2$ points in $\mathbb{R}^d$ can be partitioned into two disjoint subsets P_1, P_2 such that $\mathcal{CH}(P_1) \cap \mathcal{CH}(P_2) \neq \emptyset$. This partition can be computed via solving a linear system in $d+2$ variables in $O(d^3)$ time. A point in this intersection (which is an immediate byproduct of computing the partition) is a **Radon point**.

Let p be a point in $\mathbb{R}^d$. It is a **convex combination** of points $\{p_1, \ldots, p_m\}$ if there are $\alpha_1, \ldots, \alpha_m \in [0,1]$ such that $p = \sum_{i=1}^m \alpha_i p_i$ and $\sum_i \alpha_i = 1$.

▶ **Lemma 10** ([23, 17]: Sparsifying conv. combination)**.** *Let p be a point in $\mathbb{R}^d$, and let $P = \{p_1, \ldots, p_m\} \subseteq \mathbb{R}^d$ be a point set. Furthermore, assume that we are given p as a convex combination of points of P. Then, one can compute a convex combination representation of p that uses at most $d+1$ points of P. This takes $O(md^3)$ time.*

▶ **Lemma 11** ([23, 17])**.** *Given $d+2$ sites $(p_1, \ell_1), \ldots, (p_{d+2}, \ell_{d+2})$ of rank r in $\mathbb{R}^d$, where the logs ℓ_i are disjoint, one can compute a site (p, ℓ) of rank $2r$ in $O(rd^5)$ time.*

A recycling algorithm for computing a Tverberg point

The algorithm of Miller and Sheehy maintains a collection of sites. Initially, it converts each input point $p \in P$ into a site $(p, \{p\})$ of rank one. The algorithm then merges $d+2$ sites of rank r into a site of rank $2r$, using Lemma 11. Before the merge, the input logs use $r(d+2)(d+1)$ points in total. After the merge, the new log uses only $2(d+1)r$ points. The algorithm recycles the remaining $(d+2)(d+1)r - 2(d+1)r = d(d+1)r$ points by reinserting them into the collection as singleton sites of rank one. When no merges are available, the algorithm outputs the maximum-rank site as the approximate Tverberg point.

Analysis

For our purposes, we need a slightly different way of analyzing the above algorithm of Miller and Sheehy [23], so we provide the analysis in detail.

Let 2^h be the rank of the output site. The number of points in all the logs is n. Since a site of rank 2^i has at most $(d+1)2^i$ points of P in its log, and there are at most $d+1$ sites of each rank, n is at most $\sum_{i=0}^h (d+1)^2 2^i$. As such,

$$n \leq \sum_{i=0}^{h} (d+1)^2 2^i = (d+1)^2 (2^{h+1} - 1) \iff \frac{n}{(d+1)^2} + 1 \leq 2^{h+1} \implies 2^h \geq \left\lceil \frac{n}{2(d+1)^2} \right\rceil. \tag{2.1}$$

Every site in the collection is associated with a history tree that describes how it was computed. Thus, the algorithm execution generates (conceptually) a forest of such history trees, which is the **history** of the computation.

▶ **Lemma 12.** *Let 2^h be the maximum rank of a site computed by the algorithm. The total number of sites in the history that are of rank (exactly) 2^{h-i} is at most $(d+2)^{i+1} - 1$.*

Proof. There are at most $T(0) = d+1$ nodes of rank 2^{h-0} in the history, and each has $d+2$ children of rank 2^{h-1}. In addition, there might be $d+1$ trees in the history whose roots have rank 2^{h-1}. Thus, $T(1) = d+1+(d+2)T(0)$. By induction, there are at most

$$T(i) = d+1+(d+2)T(i-1) = (d+2)^{i+1} - 1$$

nodes in the history of rank 2^{h-i}. ◀

▶ **Lemma 13.** *Let 2^h be the maximum rank of a site computed by the algorithm. The total amount of work spent (directly) by the algorithm (throughout its execution) at nodes of rank $\geq 2^{h-i}$ is $O(d^{4+i}n)$. In particular, the overall running time of the algorithm is $O(d^4 n^{1+\log_2 d})$.*

Proof. There are at most $(d+2)^{i+1}$ nodes of rank 2^{h-i} in the history. The rank of such a node is $r_i = O(\frac{n}{2^{i+1}(d+1)^2})$, and the amount of work spent in computing the node is $O(r_i d^6) = O\big(d^4 n/2^i\big)$. As such, the total work in the top i ranks of the history is proportional to $L_i = \sum_{j=0}^{i} (d+2)^j \cdot d^4 n/2^j = O(d^{4+i}n)$.

Since $h \leq \log_2 n$, the overall running time is $O(L_h) = O(d^{4+\log_2 n} n) = O(d^4 n^{1+\log_2 d})$. ◀

3 Improved Tverberg approximation algorithms

3.1 Projections in low dimensions

▶ **Lemma 14.** *Let P be a set of n points in three dimensions. One can compute a Tverberg point of P of depth $n/6$ (and the log realizing it) in $O(n \log n)$ time.*

Proof. Project the points of P to two dimensions, and compute a Tverberg point p and a partition for it of size $n/3$. Lifting from the plane back to the original space, the point p lifts to a vertical line ℓ, and every triangle lifts to a triangle which intersects ℓ. Pick a point q on this line which is the median of the intersections. Now, pair every triangle intersecting ℓ above q with a triangle intersecting ℓ below it. This partitions P into $n/6$ sets, each of size 6, such that the convex hull of each sets contains q. Within each set, compute at most 4 points whose convex hull contains q. These points yield the desired log for q of rank $n/6$. ◀

▶ Remark 15. Lemma 14 seems innocent enough, but to the best of our knowledge, it is the best one can do in near-linear time in 3D. The only better approximation algorithm we are aware of is the one suggested by Tverberg's theorem. It yields a point of Tverberg depth $n/4$, but its running time is probably at least $O(n^{13})$ (see Lemma 2).

As observed by Mulzer and Werner [24], one can repeat this projection mechanism. Since Mulzer and Werner [24] bottom their recursion at dimension 1, their algorithm computes a point of Tverberg depth $n/2^d$ (in three dimensions, depth $n/8$). Applying this projection idea but bottoming at two dimensions, as above, yields a point of Tverberg depth $n/(3 \cdot 2^{d-2})$.

▶ **Lemma 16.** *Let P be a set of n points in four dimensions. One can compute a Tverberg point of P of depth $n/9$ (and the log realizing it) in $O(n \log n)$ time.*

More generally, for d even, and a set P of n points in $\mathbb{R}^d$, one can compute a point of depth $n/3^{d/2}$ in $d^{O(1)} n \log n$ time. For d odd, we get a point of depth $n/(2 \cdot 3^{(d-1)/2})$.

Proof. As mentioned above, the basic idea is due to Mulzer and Werner. Project the four-dimensional point set onto the plane spanned by the first two coordinates (i.e. "eliminate" the last two coordinates), and compute a $n/3$ centerpoint using Theorem 8. Translate the space so that this centerpoint lies at the origin. Now, consider each triangle in the original four-dimensional space. Each triangle intersects the two-dimensional subspace formed by the first two coordinates. Pick an intersection point from each lifted triangle. On this set of $n/3$ points, living in this two dimensional subspace, apply again the algorithm of Theorem 8 to compute a Tverberg point of depth $(n/3)/3 = n/9$. The resulting centerpoint p is now contained in $n/9$ triangles, where every vertex is contained in an original triangle of points. That is, p has depth $n/9$, where each group consists of 9 points in four dimensions. Now, sparsify each group into 5 points whose convex hull contains p.

The second part of the claim follows from applying the above argument repeatedly. ◀

3.2 An improved quasi-polynomial algorithm

The algorithm of Miller and Sheehy is expensive at the bottom of the recursion tree, so we replace the bottom with a faster algorithm. We use a result of Mulzer and Werner [24]:

▶ **Theorem 17** ([24]). *Given a set P of n points in $\mathbb{R}^d$, one can compute a site of rank $\geq n/4(d+1)^3$ (together with its log) in $d^{O(\log d)}n$ time.*

Let $\delta \in (0,1)$ be a small constant. We modify the algorithm of Miller and Sheehy by keeping all the singleton sites of rank one in a buffer of *free* points. Initially, all points are in the buffer. Whenever the buffer contains at least δn points, we use the algorithm of Theorem 17 to compute a site of rank $\rho \geq \delta n/8(d+1)^3$, where ρ is a power of two. (If the computed rank is too large, we throw away entries from the log until the rank reaches a power of two.) We insert this site into the collection of sites maintained by the algorithm. As in Miller and Sheehy's algorithm, we repeatedly merge $d+2$ sites of the same rank to get a site of double the rank. In the process, the points thrown out from the log are recycled into the buffer. Whenever the buffer size exceeds δn, we compute a new site of rank ρ. This process stops when no sites can be merged and the number of free points is less than δn.

▶ **Theorem 18.** *Given a set P of n points in $\mathbb{R}^d$, and a parameter $\delta \in (0,1)$, one can compute a site of rank at least $\frac{(1-\delta)n}{2(d+1)^2}$ (together with its log) in $d^{O(\log(d/\delta))}n$ time.*

Proof. When the algorithm above stops, there are at least $(1-\delta)n$ points in its logs. Arguing as in Eq. (2.1), the output site has rank

$$2^h \geq \frac{(1-\delta)n}{2(d+1)^2}.$$

We now consider runtime. The algorithm maintains only nodes of rank $\rho \geq 2^{h-H}$, where

$$H = \left\lceil \log_2 \frac{2^h}{\rho} \right\rceil \leq \left\lceil \log_2 \frac{n/2(d+1)^2}{\delta n/8(d+1)^3} \right\rceil = 1 + \log_2 \frac{4(d+1)}{\delta} = O(\log(d/\delta)).$$

The total work spent on merging nodes with these ranks is equivalent to the work in Miller and Sheehy's algorithm for such nodes. By Lemma 13, the total work performed is $O(d^{4+H}n)$.

As for the work associated with the buffer, observe that Theorem 17 is invoked $(d+2)^{H+1}$ times (this is the number of nodes in the history of rank 2^{h-H}). Each invocation takes $d^{O(\log d)}n$ time, so the total running time of the algorithm is

$$d^{O(H)}n + (d+1)^{H+1}d^{O(\log d)}n = d^{O(\log(d/\delta))}n. \qquad \blacktriangleleft$$

3.3 A strongly polynomial algorithm

▶ **Lemma 19.** *Let P be a set of n points in $\mathbb{R}^d$. For $N = \Theta(d^2 \log d)$, consider a random coloring of P by $k = \lfloor n/N \rfloor$ colors, and let $\{P_1, \ldots, P_k\}$ be the resulting partition of P. With probability $\geq 1 - k/d^{O(d)}$, this partition is a Tverberg partition of P.*

Proof. Let $\varepsilon = 1/(d+1)$. The VC dimension of halfspaces in $\mathbb{R}^d$ is $d+1$ by Radon's theorem. By the ε-net theorem [18, 15], a sample from P of size

$$N = \left\lceil \frac{8(d+1)}{\varepsilon} \log(16(d+1)) \right\rceil = \Theta(d^2 \log d)$$

is an ε-net with probability $\geq 1 - 4\big(16(d+1)\big)^{-2(d+1)}$. Then $P_1, \ldots, P_k$ are all ε-nets with the probability stated in the lemma.

Now, consider the centerpoint q of P. We claim that $q \in \mathcal{CH}(P_i)$, for all i. Indeed, assume otherwise, so that there is a separating hyperplane between q and some P_i. Then the halfspace induced by this hyperplane that contains q also contains at least εn points of P, because q is a centerpoint of P. But this contradicts that P_i is an ε-net for halfspaces. ◀

▶ **Lemma 20.** *Let P be a set of n points in $\mathbb{R}^d$. For $N = O(d^3 \log d)$, consider a random coloring of P by $k = \lfloor n/N \rfloor = \Omega(\frac{n}{d^3 \log d})$ colors, and let $\{P_1, \ldots, P_k\}$ be the resulting partition of P. One can compute, in $O(d^7 \log^6 d)$ time, a Tverberg point that lies in $\bigcap_i \mathcal{CH}(P_i)$. This algorithm succeeds with probability $\geq 1 - k/d^{O(d)}$.*

Proof. Compute a $1/(2d^2)$-centerpoint q for P using the algorithm of Har-Peled and Jones [16], and repeat the argument of Lemma 19 with $\varepsilon = 1/(2d^2)$. With the desired probability, $q \in \mathcal{CH}(P_i)$ for all i. ◀

3.4 A weakly polynomial algorithm

Let $T_{\mathsf{LP}}(n, m)$ be the time to solve an LP with n variables and m constraints. Using interior point methods, one can solve such LPs in weakly polynomial time. The fastest known method runs in $\mathcal{O}_w(mn + n^3)$ [29] or $\mathcal{O}_w(mn^{3/2})$ [20], where $\mathcal{O}_w$ hides polylogarithmic terms that depends on n, m, and the width of the input numbers.

▶ Remark 21. The *error* of the LP solver, see [29], for a prescribed parameter ε, is the distance of the computed solution from an optimal one. Specifically, let R be the maximum absolute value of any number in the given instance. In $O((nm + n^3) \log^{O(1)} n \log(n/\varepsilon))$ time, the LP solver can find an assignment to the variables which is $\mathcal{E}$ close to complying with the LP constraints, where $\mathcal{E} \leq \varepsilon nmR$ [29]. That is, the LP solver can get arbitrarily close to a true solution. This is sufficient to compute an exact solution in polynomial time if the input is made out of integer numbers with polynomially bounded values, as the running time then depends on the number of bits used to encode the input. We use $\mathcal{O}_w(\cdot)$ to denote such **weakly polynomial** running time.

▶ **Lemma 22** (Carathéodory via LP)**.** *Given a set P of n points in $\mathbb{R}^d$ and query point q, one can decide if $q \in \mathcal{CH}(P)$, and if so, output a convex combination of $d+1$ points of P that is equal to q. The running time of this algorithm is $O(T_{\mathsf{LP}}(n, n+d) + nd^3) = \mathcal{O}_w(nd + nd^3)$.*

Alternatively, one can compute such a point in $2^{O(\sqrt{d \log d})} + O(d^2 n)$ time.

Proof. We write the natural LP for representing q as a point in the interior of $\mathcal{CH}(P)$. This LP has n variables (one for each point), and $n + 2d + 2$ constraints (all variables are positive, the points sum to q, and the coefficients sum to 1), where an equality constraint counts as two constraints. If the LP computes a solution, then we can sparsify it using Lemma 10.

The alternative algorithm writes an LP to find a hyperplane separating q from P. First, it tries to find a hyperplane which is vertically below P and above q. This LP has d variables, and it can be solved in $O(2^{\sqrt{d \log d}} + d^2 n)$ time [27, 10, 21]. If there is no such separating hyperplane, then the algorithm provides d points whose convex hull lies below q and tries again. This time, it computes a separating hyperplane below q and above P. Again, if no such separating hyperplane exists, then the algorithm returns d points whose convex hull lies above q. The union of the computed point sets above and below q contains at most $2d$ points. Now, write the natural LP with $2d$ variables as above, and solve it using linear-time LP solvers in low dimensions. This gives the desired representation. ◀

▶ Remark 23. Rolnick and Soberón [26] achieved a result similar to that of Lemma 22, but they used binary search to find the $d+1$ coefficients in the minimal representation. Thus their running time is $O(T_{\mathsf{LP}}(n, n+d)d \log n)$.

Using Lemma 19, we get the following.

▶ **Lemma 24.** *Let P be a set of n points in $\mathbb{R}^d$. For $N = O(d^2 \log d)$, one can compute a site q of rank $k = n/O(d^2 \log d)$ in $\mathcal{O}_w(n^{5/2} + nd^3)$ time. The algorithm succeeds with probability $\geq 1 - k/d^{O(d)}$.*

Proof. We compute a Tverberg partition using Lemma 19. Next we write an LP for computing an intersection point q that lies in the interior of the $k = n/O(d^2 \log d)$ sets. This LP has $O(n+d)$ constraints and n variables, and it can be solved in $\mathcal{O}_w(n^{5/2})$ time [20]. We then sparsify the representations over each of the k sets. This requires $O(d^2 \log d \cdot d^3)$ time per set and hence $O(nd^3)$ time overall. The result is a site q with a log of rank k, as desired. ◀

▶ **Theorem 25.** *Given a set P of n points in $\mathbb{R}^d$, and a parameter $\delta \in (0,1)$, one can compute a site of rank at least $\frac{(1-\delta)n}{2(d+1)^2}$ (together with its log) in $d^{O(\log\log(d/\delta))}\mathcal{O}_w(n^{5/2})$ time.*

Proof. We modify the algorithm of Theorem 18 to use Lemma 24 to compute a Tverberg point on the points in the buffer. Since the gap between the top rank of the recursion tree and rank computed by Lemma 24 is $O(\log(d/\delta))$, it follows that the algorithm uses only the top $O(\log\log(d/\delta))$ levels of the recursion tree, and the result follows. ◀

3.5 Faster approximation in low dimensions

▶ **Lemma 26.** *Let P be a set of n points in $\mathbb{R}^d$, and let $\delta \in (0,1)$ be some parameter. One can compute a Tverberg point of depth $\geq (1-\delta)n/d(d+1)$, together with its log, in $O(d^{O(d)}/\delta^{2(d-1)} + 2^{O(\sqrt{d \log d})}n + d^2 n \log^2 n)$ time. The algorithm succeeds with probability close to one.*

Proof. Let $\varepsilon = 1/2(d+1)$, and let R be a random sample from P of size

$$m = O(d\varepsilon^{-1}\delta^{-2} \log \varepsilon^{-1}) = O(d^2\delta^{-2} \log d).$$

This sample is a (ε, δ)-relative approximation for P with probability close to one [14]. Compute a centerpoint c for the sample using "brute force" in $O(m^{d-1})$ time [7]. Now, repeatedly use the algorithmic version of Carathéodory's theorem (Lemma 22) to extract a simplex that contains c. One can repeat this process $\lfloor n/d(d+1) \rfloor$ times before getting stuck, since every simplex contains at most d points in any halfspace passing through c. Naively, the running time of this algorithm is $2^{O(\sqrt{d \log d})}n^2$.

However, one can do better. The number of simplices extracted by the above algorithm is $L = \lfloor n/d(d+1) \rfloor$. For $i = 1, \ldots L$, let $b_i = \lfloor n/(d+1) \rfloor - d(i-1)$ be a lower bound on the Tukey depth of c at the beginning of the ith iteration of the above extraction algorithm. At this moment, the point set has $n_i = n - (d+1)(i-1) \geq n/2$ points. Hence the relative Tukey depth of c is $\varepsilon_{L-i} = b_{L-i}/n_{L-i}$. In particular, an ε_i-net R_i for halfspaces has size $r_i = O(\frac{d}{\varepsilon_i} \log \frac{1}{\varepsilon_i})$. If r_i is larger than the number of remaining points, then the sample consists of the remaining points of P. The convex hull of such a sample contains c with probability close to one, so one can apply Lemma 22 to R_i to get a simplex that contains P (if the sample fails, then the algorithm resamples). The algorithm adds the simplex to the output log, removes its vertices from P, and repeats.

Since the algorithm invokes Lemma 22 $O(n/d^2)$ times, the running time is bounded by $2^{O(\sqrt{d \log d})} n/d^2 + O(d^2 \sum_i r_i)$. Since $n_{L-i} \geq n/2$ and $b_{L-i} \geq i \cdot d$, we have that $\varepsilon_{L-i} \geq (id/(n/2) = 2id/n$. Therefore,

$$\sum_i r_i = O(n) + \sum_{i=1}^{L-1} r_{L-i} = O(n) + \sum_{i=1}^{L-1} O\left(\frac{dn}{2id} \log \frac{n}{id}\right) = O(n \log^2 n). \quad \blacktriangleleft$$

References

1 Peyman Afshani, Donald R. Sheehy, and Yannik Stein. Approximating the simplicial depth. *CoRR*, abs/1512.04856, 2015. arXiv:1512.04856.

2 Pankaj K. Agarwal, Sariel Har-Peled, Subhash Suri, Hakan Yildiz, and Wuzhou Zhang. Convex hulls under uncertainty. *Algorithmica*, 79(2):340–367, 2017. doi:10.1007/s00453-016-0195-y.

3 Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. Algorithms for center and Tverberg points. *ACM Trans. Algo.*, 5(1):5:1–5:20, 2008. doi:10.1145/1435375.1435380.

4 David Avis. The m-core properly contains the m-divisible points in space. *Pattern Recognit. Lett.*, 14(9):703–705, 1993. doi:10.1016/0167-8655(93)90138-4.

5 B. J. Birch. On 3N points in a plane. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):289–293, 1959. doi:10.1017/S0305004100034071.

6 T. M. Chan. Geometric applications of a randomized optimization technique. *Disc. Comput. Geom.*, 22(4):547–567, 1999. doi:10.1007/PL00009478.

7 T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In J. Ian Munro, editor, *Proc. 15th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 430–436. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982853.

8 B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31(4):509–517, 1985. doi:10.1109/TIT.1985.1057060.

9 Aruni Choudhary and Wolfgang Mulzer. No-dimensional Tverberg theorems and algorithms. In Sergio Cabello and Danny Z. Chen, editors, *Proc. 36th Int. Annu. Sympos. Comput. Geom.* (SoCG), volume 164 of *LIPIcs*, pages 31:1–31:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.31.

10 K. L. Clarkson. Las Vegas algorithms for linear and integer programming. *J. Assoc. Comput. Mach.*, 42:488–499, 1995. doi:10.1145/201019.201036.

11 Kenneth L. Clarkson, David Eppstein, Gary L. Miller, Carl Sturtivant, and Shang-Hua Teng. Approximating center points with iterative Radon points. *Int. J. Comput. Geom. Appl.*, 6:357–377, 1996. doi:10.1142/S021819599600023X.

12 Ketan Dalal. Counting the onion. *Random Struct. Alg.*, 24(2):155–165, 2004. doi:10.1002/rsa.10114.

13 S. Har-Peled and B. Lidicky. Peeling the grid. *SIAM J. Discrete Math.*, 27(2):650–655, 2013. doi:10.1137/120892660.

14 S. Har-Peled and M. Sharir. Relative (p, ε)-approximations in geometry. *Disc. Comput. Geom.*, 45(3):462–496, 2011. doi:10.1007/s00454-010-9248-1.

15 Sariel Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, 2011.

16 Sariel Har-Peled and Mitchell Jones. Journey to the center of the point set. In Gill Barequet and Yusu Wang, editors, *Proc. 35th Int. Annu. Sympos. Comput. Geom.* (SoCG), volume 129 of *LIPIcs*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.41.

17 Sariel Har-Peled and Timothy Zhou. Improved approximation algorithms for Tverberg partitions. *CoRR*, abs/2007.08717, 2020. arXiv:2007.08717.

18 D. Haussler and E. Welzl. ε-nets and simplex range queries. *Disc. Comput. Geom.*, 2:127–151, 1987. doi:10.1007/BF02187876.

19 Shreesh Jadhav and Asish Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Disc. Comput. Geom.*, 12:291–312, 1994. doi:10.1007/BF02574382.

20 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In Venkatesan Guruswami, editor, *Proc. 56th Annu. IEEE Sympos. Found. Comput. Sci.* (FOCS), pages 230–249. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.23.

21 J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996. doi:10.1007/BF01940877.

22 Frédéric Meunier, Wolfgang Mulzer, Pauline Sarrabezolles, and Yannik Stein. The rainbow at the end of the line - A PPAD formulation of the colorful carathéodory theorem with applications. In Philip N. Klein, editor, *Proc. 28th ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1342–1351. SIAM, 2017. doi:10.1137/1.9781611974782.87.

23 Gary L. Miller and Donald R. Sheehy. Approximate centerpoints with proofs. *Comput. Geom.*, 43(8):647–654, 2010. doi:10.1016/j.comgeo.2010.04.006.

24 Wolfgang Mulzer and Daniel Werner. Approximating Tverberg points in linear time for any fixed dimension. *Disc. Comput. Geom.*, 50(2):520–535, 2013. doi:10.1007/s00454-013-9528-7.

25 John R. Reay. Several generalizations of Tverberg's theorem. *Israel Journal of Mathematics*, 34(3):238–244, 1979. doi:10.1007/BF02760885.

26 David Rolnick and Pablo Soberón. Algorithms for Tverberg's theorem via centerpoint theorems. *CoRR*, abs/1601.03083, 2016. arXiv:1601.03083.

27 R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Disc. Comput. Geom.*, 6:423–434, 1991. doi:10.1007/BF02574699.

28 Helge Tverberg and Siniša Vrećica. On generalizations of Radon's theorem and the ham sandwich theorem. *Euro. J. Combin.*, 14(3):259–264, 1993. doi:10.1006/eujc.1993.1029.

29 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proc. 52nd Annu. ACM Sympos. Theory Comput.* (STOC), pages 775–788. ACM, 2020. doi:10.1145/3357713.3384309.

A Tukey depth in two dimensions

Restatement of Lemma 4. *Given a set P of n points in the plane and a query point q, such that $P \cup \{q\}$ is in general position, one can compute, in $O(n + k \log k)$ time, the Tukey depth k of q in P. The algorithm also computes the halfplane realizing this depth.*

Proof. In the dual, $q^\star$ is a line, and the task is to find a point on this line which minimizes the number of lines of $P^\star$ (i.e., set of lines dual to the points of P) strictly below it. More precisely, one has to also solve the upward version, and return the minimum of the two solutions. Handling the downward version first, every point $p \in P$ has a dual line $p^\star$. The portion of $q^\star$ that lies above $p^\star$ is a closed ray on $q^\star$. As such, we have a set of rays on the line (which can be interpreted as the x-axis), and the task is to find a point on the line contained in the minimum number of rays. (This is known as linear programming with violations in one dimension.)

Let $R_{\mapsto}$ (resp. $L_{\mapsfrom}$) be the set of points that corresponds to heads of rays pointing to the right (resp. to the left) by $P^\star$ on $q^\star$. Let R_i be the set of $\lfloor n/2^i \rfloor$ rightmost points of $R_{\mapsto}$, for $i = 0, \ldots, h = \lfloor \log_2 n \rfloor$. Using median selection, each set R_i can be computed from R_{i-1} in $|R_{i-1}|$ time. As such, all these sets can be computed in $\sum_i O(n/2^i) = O(n)$ time. The sets $L_0, L_1, \ldots, L_h$ are computed in a similar fashion. For all i, we also compute the rightmost point $\boldsymbol{r}_{i-1}$ of $R_{i-1} \setminus R_i$ (which is the rightmost point in $R \setminus R_i$). Similarly, $\boldsymbol{\ell}_i$ is the leftmost point of $L_{i-1} \setminus L_i$, for $i = 1, \ldots, h$. Let $\boldsymbol{r}_0$ (resp. $\boldsymbol{\ell}_0$) be the rightmost (resp. leftmost) point of R_0 (resp. L_0).

Now, compute the maximum j such that $\curlyvee_j$ is to the left of ℓ_j. Observe that $(R \setminus R_j) \cup (L \setminus L_j)$ form a set of rays that their intersection is non-empty (i.e., feasible). Similarly, the set of rays $(R \setminus R_{j+1}) \cup (L \setminus L_{j+1})$ is not feasible, and any set of feasible rays must be created by removing at least $|R_{j+1}| = |L_{j+1}| = \lfloor n/2^{j+1} \rfloor$ rays from $R_{\mapsto} \cup L_{\mapsfrom}$. Hence, in linear time, we have computed a 4-approximation to the minimum number of rays that must be removed for feasibility. In particular, if S is a set of rays such that $(R_{\mapsto} \cup L_{\mapsfrom}) \setminus S$ is feasible, then $(R_{j-1} \cup L_{j-1}) \setminus S$ is also feasible. Namely, if the minimum size of such a removeable set S is k, then we have computed a set of $O(k)$ rays, such that it suffices to solve the problem on this smaller set.

In the second stage, we solve the problem on $R_{j-1} \cup L_{j-1}$. We first sort the points, and then for each point in this set, we compute how many rays must be removed before it lies in the intersection of the remaining rays. Given a location p on the line, we need to remove all the rays of R_{j-1} (resp. L_{j-1}) whose heads lie to the right (resp. left) of p. This can be done in $O(k \log k)$ time by sweeping from left to right and keeping track of the rays that need to be removed.

As such, we can solve the LP with violations on the line in $O(n + k \log k)$ time, where k is the minimum number of violated constraints. A 4-approximation to k can be computed in $O(n)$ time.

Now we return to the Tukey depth problem. First we compute a 4-approximation, denoted by $\widetilde{k_\downarrow}$, for the minimum number of lines crossed by a vertical ray shot down from a point on $q^\star$. Similarly, we compute $\widetilde{k_\uparrow}$. If $4\widetilde{k_\downarrow} < \widetilde{k_\uparrow}$, then we compute $k_\downarrow$ exactly and return the point on $q^\star$ that realizes it. (In the primal, this corresponds to a closed halfspace containing q along with exactly $k_\downarrow$ points of P.) Similarly, if $\widetilde{k_\downarrow} > 4\widetilde{k_\uparrow}$, then we compute $k_\uparrow$ exactly, and return it as the desired solution. In the remaining case, we compute both quantities and return the minimum of the two. ◀

B Tverberg partitions in two dimensions

Restatement of Lemma 5. *Let P be a set of n points in the plane, let q be a query point such that $P \cup \{q\}$ is in general position, and suppose that $k = \mathsf{d_{TK}}(q) \leq n/3$. Then one can compute a log for q of rank k in $O(n + k \log k)$ time.*

Proof. Using the algorithm of Lemma 4, compute the Tukey depth of q and the closed halfplane $\hbar^+$ realizing it, where q lies on the line $\hbar$ bounding $\hbar^+$. This takes $O(n + k \log k)$ time. By translation and rotation, we can assume that q is the origin and $\hbar$ is the x-axis. Let $P^+ = P \cap \hbar^+$ be the k points realizing the Tukey depth of q, and let $P^- = P \setminus P^+$ be the set of points below the x-axis, so that $|P^-| = n - k \geq 2k$.

Consider the counterclockwise order of the points of P^- starting from the negative side of the x-axis. Let T be the set containing the first and last k points in this order, computed in $O(n)$ time by performing median selection twice. Let $T = \{t_1, \ldots, t_{2k}\}$ be the points of T sorted in counterclockwise order. Similarly, let $P^+ = \{p_1, \ldots, p_k\}$ be the points of P^+ sorted in counterclockwise order starting from the positive side of the x-axis, see Figure B.1.

Let $\triangle_i = \triangle p_i t_i t_{k+i}$, for $i = 1, \ldots, k$. We claim that $q \in \triangle_i$ for all i. To this end, let ℓ_i be the line passing through the origin and p_i, and let ℓ_i^+ denote the halfspace it induces to the left of the vector $\overrightarrow{qp_i}$. Then ℓ_i^+ must contain t_i, as otherwise, $\left|P \cap \ell_i^+\right| < k$, contradicting the Tukey depth of q. Namely, the segment $p_i t_i$ intersects the negative side of the x-axis. A symmetric argument, applied to the complement halfplane, implies that the segment $t_{k+i} p_i$ intersects the positive side of the x-axis. Then the origin q is contained in $\triangle_i$, as claimed.

Computing these triangles, we have found a log for q of rank k in $O(n + k \log k)$ time. ◀

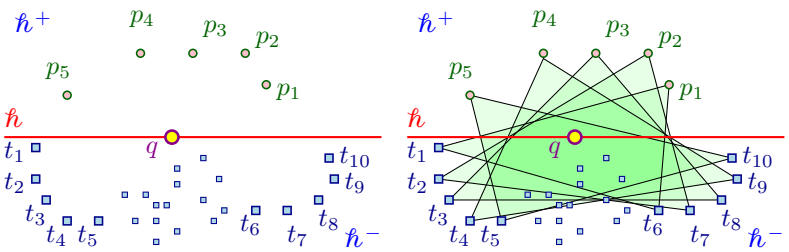

Figure B.1 Illustration of the proof of Lemma 5.