Fast Algorithms for Geometric Consensuses

Sariel Har-Peled
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
sariel@illinois.edu

Mitchell Jones
Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
mfjones2@illinois.edu

Abstract

Let P be a set of n points in $\mathbb{R}^d$ in general position. A median hyperplane (roughly) splits the point set P in half. The *yolk* of P is the ball of smallest radius intersecting all median hyperplanes of P. The *egg* of P is the ball of smallest radius intersecting all hyperplanes which contain exactly d points of P.

We present exact algorithms for computing the yolk and the egg of a point set, both running in expected time $O(n^{d-1} \log n)$. The running time of the new algorithm is a polynomial time improvement over existing algorithms. We also present algorithms for several related problems, such as computing the Tukey and center balls of a point set, among others.

2012 ACM Subject Classification Theory of computation → Computational geometry

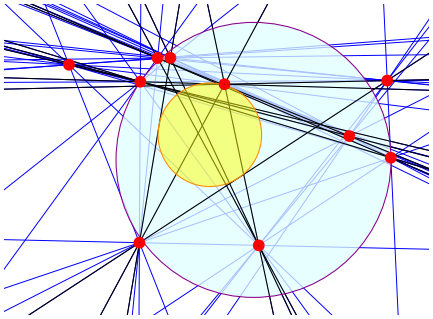
Keywords and phrases Geometric optimization, centerpoint, voting games

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.50

Related Version A full version of this paper is available at `https://arxiv.org/abs/1912.01639`.

Funding *Sariel Har-Peled*: Supported in part by NSF AF award CCF-1907400.
Mitchell Jones: Supported in part by NSF AF award CCF-1907400.

Acknowledgements The authors thank Joachim Gudmundsson for bringing the problem of computing the yolk to our attention. The second author thanks Sampson Wong for discussions on computing the yolk in higher dimensions. We also thank Timothy Chan for useful comments (in particular, the improved algorithm for the yolk in 3D, see Remark 26).



1 Introduction

Voting games and the yolk. Suppose there is a collection of n voters in $\mathbb{R}^d$, where each dimension represents a specific ideology. In a fixed dimension, each voter maintains a value along this continuum representing their stance on a given ideology. One can interpret $\mathbb{R}^d$ as a *policy space*, and each point in $\mathbb{R}^d$ represents a single policy. In the Euclidean spatial model, a voter $p \in \mathbb{R}^d$ always prefers policies which are closer to p under the Euclidean norm. For two policies $x, y \in \mathbb{R}^d$ and a set of voters $P \subset \mathbb{R}^d$, x *beats* y if more voters in P prefer policy x compared to y. A plurality point is a policy which beats all other policies in $\mathbb{R}^d$. For $d = 1$, the plurality point is the median voter (when n is odd) [3]. However for $d > 1$,

a plurality point is not always guaranteed to exist [18]. It is known that one can test if a plurality point exists (and if so, compute it) in $O(dn \log n)$ time [10]. Note that the plurality point is a point of Tukey depth $\lceil n/2 \rceil$ – in general this is the largest possible Tukey depth any point can have; while the centerpoint is a point that guarantees a "respectable" minority of size at least $n/(d+1)$.

Since plurality points may not always exist, one generalization of a plurality point is the yolk [17]. A hyperplane is a **median hyperplane** if the number of voters lying in each of the two closed halfspaces is at least $\lceil n/2 \rceil$. The **yolk** is the ball of smallest radius intersecting all such median hyperplanes. Note that when a plurality point exists, the yolk has radius zero (equivalently, all median hyperplanes intersect at a common point).

We also consider the following restricted problem. A hyperplane is **extremal** if and only if it passes through d points, under the assumption that the points are in general position. The **extremal yolk** is the ball of smallest radius intersecting all extremal median hyperplanes. Importantly, the yolk and the extremal yolk are different problems – the radius of the yolk and extremal yolk can differ [21].

The egg of a point set. A problem related to computing the yolk is the following: For a set of n points P in $\mathbb{R}^d$, compute the smallest radius ball intersecting all extremal hyperplanes of P (i.e., all hyperplanes passing through d points of P). Such a ball is the **egg** of P. See Figure 1.1 for an illustration of the yolk and egg of a point set.

Linear programs with many implicit constraints. The problem of computing the egg can be written as a linear program (LP) with $\Theta(n^d)$ constraints, defined implicitly by the point set P. One can apply Seidel's algorithm [19] (or any other linear time LP solver in constant dimension) to obtain an $O(n^d)$ expected time algorithm for computing the egg (or the yolk, with a bit more work). However, as each d-tuple of points forms a constraint, it is natural to ask if one can obtain a faster algorithm in this setting. Specifically, we are interested in the following problem: Let I be an instance of a d-dimensional LP specified via a set of n entities P, where each k-tuple of P induces a linear constraint in I, for some (constant) integer k. The problem is to efficiently solve I, assuming access to some additional subroutines.

1.1 Previous work

The yolk. Let P be a set of n points in $\mathbb{R}^d$. Both the yolk and extremal yolk have been studied in the literature. The first polynomial time exact algorithm for computing the yolk in $\mathbb{R}^d$ was by Tovey in $O\big(n^{(d+1)^2}\big)$ time – in the plane, the running time can be improved to $O(n^4)$ [22]. Following Tovey, the majority of results have focused on computing the yolk in the plane. In 2018, de Berg et al. [10] gave an $O(n^{4/3} \log^{1+\varepsilon} n)$ time algorithm (for any fixed $\varepsilon > 0$) for computing the yolk. Obtaining a faster exact algorithm remained an open problem. Gudmundsson and Wong [12, 13] presented a $(1+\varepsilon)$-approximation algorithm with $O(n \log^7 n \log^4 \varepsilon^{-1})$ running time. An unpublished result of de Berg et al. [8] achieves a randomized $(1+\varepsilon)$-approximation algorithm for the extremal yolk running in expected time $O(n\varepsilon^{-3} \log^3 n)$.

The egg. The egg of a point set in $\mathbb{R}^d$ can be computed by solving a linear program with $\Theta(n^d)$ constraints. The egg is a natural extension to computing the yolk, and thus obtaining faster exact algorithms is of interest. The authors are not aware of any previous work on this specific problem. Bhattacharya et al. [2] gave an algorithm which computes the smallest radius ball intersecting a set of m hyperplanes in $O(m)$ time, when $d = O(1)$, by formulating the problem as an LP (see also Lemma 11). However we emphasize that in our problem the set of hyperplanes are implicitly defined by the point set P, and is of size $\Theta(n^d)$ in $\mathbb{R}^d$.

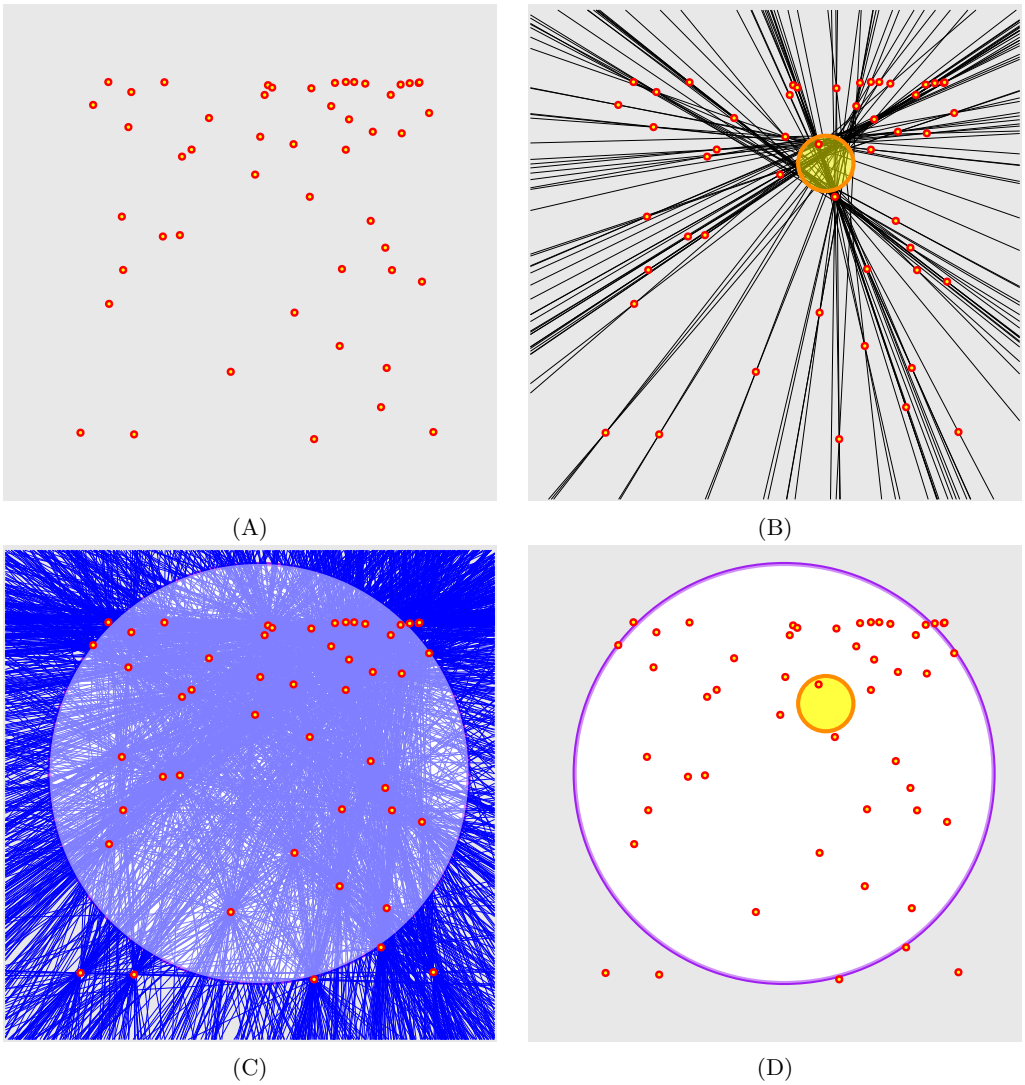


Figure 1.1 (A) Points. (B) Median lines and the extremal yolk. (C) All lines and the egg. (D) Points with the extremal yolk and the egg.

Table 1.1 Some previous work on the yolk and our results. Existing algorithms are deterministic, while the running time of our algorithms holds in expectation.

$d = 2$	$(1+\varepsilon)$-approx	Exact	Our results (Exact)
Extremal yolk	$O(n\varepsilon^{-3}\log^3 n)$ [8]	$O(n^{4/3}\log^{1+\epsilon} n)$ [10]	$O(n\log n)$ Theorem 16
Yolk	$O(n\log^7 n\log^4 \varepsilon^{-1})$ [13]	$O(n^{4/3}\log^{1+\epsilon} n)$ Variant of [10]	$O(n\log n)$ Theorem 25
$d = 3$			
Yolk	?	$O(n^3)$ Known techniques	$O(n^2)$ Remark 26
$d > 3$			
Extremal yolk	?	$O(n^d)$ Known techniques	$O(n^{d-1}\log n)$ Theorem 16
Yolk	?	$O(n^d)$ Known techniques	$O(n^{d-1}\log n)$ Theorem 25

Implicit LPs. In 2004, Chan [4] developed a framework for solving LPs with many implicit constraints (the motivation was to obtain an efficient algorithm for computing the Tukey depth of a point set). Informally, suppose that each input set P of entities maps to a set $\mathcal{H}(P)$ of implicit constraints. For n entities P and a candidate solution, suppose one can decide if the candidate solution violates any constraints of $\mathcal{H}(P)$ in $D(n)$ time. Additionally, assume that from P, one can construct $r = O(1)$ sets $P_1, \ldots, P_r$, each of size at most n/c (for some constant $c > 1$) with $\mathcal{H}(P) = \bigcup_{i=1}^{r} \mathcal{H}(P_i)$. If this partition step can be performed in $D(n)$ time, then both assumptions imply that the resulting LP can be solved in $O(D(n))$ expected time.

1.2 Our results

Note. Due to space limitations, not all results discussed below are presented in the paper. We refer the reader to the full version of the paper on arXiv [14].

In this paper we revisit Chan's algorithm for solving LPs with many implicitly defined constraints [4]. The technique leads to efficient algorithms for the following problems. Throughout, let $P \subset \mathbb{R}^d$ be a set of n points in general position:

(A) The yolk (and extremal yolk) of P can be computed *exactly* in $O(n^{d-1}\log n)$ expected time. Hence in the plane, the yolk can be computed *exactly* in $O(n\log n)$ expected time. This improves all existing algorithms (both exact and approximate) [22, 10, 13, 12, 8] for computing the yolk in the plane, and our algorithm easily generalizes to higher dimensions. See Table 1.1 for a summary of our results and previous work.

(B) By a straight-forward modification of the above algorithm, see Lemma 17, implies that the egg of P can be computed in $O(n^{d-1}\log n)$ expected time. The authors are not aware of any previous work on this specific problem.

(C) Let $H_k(P)$ be the collection of all open halfspaces which contain at least $n-k$ points of P. Consider the convex polygon $\mathcal{T}_k = \cap_{h\in H_k(P)} h$. Observe that $\mathcal{T}_0$ is the convex hull of P, with $\mathcal{T}_0 \supseteq \mathcal{T}_1 \supseteq \cdots$. The centerpoint theorem implies that $\mathcal{T}_{n/(d+1)}$ is non-empty (and contains the centerpoint). The Tukey depth of a point q in the minimal k such that $q \in \mathcal{T}_k \setminus \mathcal{T}_{k+1}$.

When $\mathcal{T}_k$ is non-empty, the **center ball** of P is the ball of largest radius contained inside $\mathcal{T}_k$. For $\mathcal{T}_k$ empty, we define the **Tukey ball** of P as the smallest radius ball intersecting all halfspaces of $H_k(P)$.
In the full version of the paper [14] we show that the Tukey ball and center ball can both be computed in $\widetilde{O}\big(k^{d-1}\big[1+(n/k)^{\lfloor d/2\rfloor}\big]\big)$ expected time. Here, $\widetilde{O}$ hides polylogarithmic factors in n. In particular when k is a (small) constant, a point of Tukey depth k can be computed in time $\widetilde{O}(n^{\lfloor d/2\rfloor})$. This improves Chan's $O(n^{d-1}\log n)$ expected time algorithm for deciding if there is a point of Tukey depth at least k [4].

(D) For a set $Q \subseteq \mathbb{R}^d$, let $\mathsf{conv}(Q)$ denote the convex hull of Q. For a given integer k let $\mathcal{C}(P,k) = \left\{\mathsf{conv}(Q) \;\middle|\; Q \in \binom{P}{k}\right\}$, where $\binom{P}{k}$ is the set of all k-tuples of points of P. We define the ***k*-ball** of P as the smallest radius ball intersecting all convex bodies in $\mathcal{C}(P,k)$.
While one may be tempted to apply the techniques discussed so far for implicit LPs, there is a faster algorithm using $(\leq k)$-sets. When k is constant, in [14] we present an algorithm for computing the k-ball in $O(n^{\lfloor d/2\rfloor}+n\log n)$ expected time. As such, the smallest ball intersecting all triangles induced by triples of a set of n points in $\mathbb{R}^3$ can be computed in $O(n\log n)$ expected time.

In [14], we present another application of Chan's technique for solving implicit LP-type problems.

(E) Given a set L of n lines in the plane, the **crossing distance** between two points $p, q \in \mathbb{R}^2$ is the number of lines of L intersecting the segment pq. Given a point $q \in \mathbb{R}^2$ not lying on any lines of L, the disk of smallest radius containing all vertices of $\mathcal{A}(L)$, within crossing distance at most k from q, can be computed, in $O(n\log n)$ expected time.

2 Preliminaries

▶ Notation. Throughout, the O hides factors which depend (usually exponentially) on the dimension d. Additionally, the $\widetilde{O}$ notation hides factors of the form $\log^c n$, where c may depend on d.

2.1 LP-type problems

An LP-type problem, introduced by Sharir and Welzl [20], is a generalization of a linear program. Let $\mathcal{H}$ be a set of constrains and f be an objective function. For any $\mathcal{B} \subseteq \mathcal{H}$, let $f(\mathcal{B})$ denote the value of the optimal solution for the constraints of $\mathcal{B}$. The goal is to compute $f(\mathcal{H})$. If the problem is infeasible, let $f(\mathcal{H}) = \infty$. Similarly, define $f(\mathcal{H}) = -\infty$ if the problem is unbounded.

▶ **Definition 1.** *Let $\mathcal{H}$ be a set of constraints, and let $f : 2^{\mathcal{H}} \to \mathbb{R} \cup \{\infty, -\infty\}$ be an objective function. The tuple $(\mathcal{H}, f)$ forms an **LP-type problem** if the following properties hold:*

(A) *Monotonicity. For any $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{H}$, we have $f(\mathcal{B}) \leq f(\mathcal{C})$.*

(B) *Locality. For any $\mathcal{B} \subseteq \mathcal{C} \subseteq \mathcal{H}$ with $f(\mathcal{C}) = f(\mathcal{B}) > -\infty$, and for all $s \in \mathcal{H}$, $f(\mathcal{C}) < f(\mathcal{C}+s) \iff f(\mathcal{B}) < f(\mathcal{B}+s)$, where $\mathcal{B}+s = \mathcal{B}\cup\{s\}$.*

A **basis** of $\mathcal{H}$ is an inclusion-wise minimal subset $\mathcal{b} \subseteq \mathcal{H}$ with $f(\mathcal{b}) = f(\mathcal{H})$. The **combinatorial dimension** δ is the maximum size of any feasible basis of any subset of $\mathcal{H}$. Throughout, we consider δ to be constant. For a basis $\mathcal{b} \subseteq \mathcal{H}$, we say that $h \in \mathcal{H}$ **violates** the current solution induced by $\mathcal{b}$ if $f(\mathcal{b}+h) > f(\mathcal{b})$. LP-type problems with n constraints can be solved in randomized time $O(n)$, hiding constants depending (exponentially) on δ [7], where the bound on the running time holds with high probability.

2.2 Implicit LPs using Chan's algorithm

Our algorithms will need the following result of Chan [4] on solving LPs with implicitly defined constraints.

▶ **Lemma 2** ([4]). *Let $(\mathcal{H}, f)$ be an LP-type problem of constant combinatorial dimension δ, and let c_δ be a constant that depends only on δ. Let $\psi, c > 1$ be fixed constants, such that $c_\delta \log^\delta \psi < c$. For an input space Π, suppose that there is a function $g : \Pi \to 2^{\mathcal{H}}$ which maps inputs to constraints. Furthermore, assume that for any input $P \in \Pi$ of size n, we have:*

- **(I)** *When $n = O(1)$, a basis for $g(P)$ can be computed in constant time.*
- **(II)** *For a basis $\mathcal{b}$, one can decide if $\mathcal{b}$ satisfies $g(P)$ in $D(n)$ time.*
- **(III)** *In $D(n)$ time, one can construct sets $P_1, \ldots, P_\psi \in \Pi$, each of size at most n/c, such that $g(P) = \bigcup_{i=1}^{\psi} g(P_i)$.*

Then a basis for $g(P)$ can be computed in $O(D(n))$ expected time, assuming that $D(n/k) = O(D(n)/k)$, for all positive integers $k \leq n$.

2.3 Duality, levels, and zones

2.3.1 Duality

▶ **Definition 3** (Duality). *The **dual hyperplane** of a point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ is the hyperplane $p^\star$ defined by the equation $x_d = -p_d + \sum_{i=1}^{d-1} x_i p_i$. The **dual point** of a hyperplane h defined by $x_d = a_d + \sum_{i=1}^{d-1} a_i x_i$ is the point $h^\star = (a_1, a_2, \ldots, a_{d-1}, -a_d)$.*

▶ Fact 4. Let p be a point and let h be a hyperplane. Then p lies above h if and only if the hyperplane $p^\star$ lies below the point $h^\star$.

Given a set of objects T (e.g., points in $\mathbb{R}^d$), we let $T^\star = \{x^\star \mid x \in T\}$ denote the dual set of objects.

2.3.2 k-Levels

▶ **Definition 5** (Levels). *For a collection of hyperplanes H in $\mathbb{R}^d$, the **level** of a point $p \in \mathbb{R}^d$ is the number of hyperplanes of H lying on or below p. The k**-level** of H is the union of points in $\mathbb{R}^d$ which have level equal to k. The $(\leq k)$**-level** of H is the union of points in $\mathbb{R}^d$ which have level at most k.*

By Fact 4, if h is a hyperplane which contains k points of P lying on or above it, then the dual point $h^\star$ is a member of the k-level of $P^\star$.

2.3.3 Zones of surfaces

For a set of hyperplanes H, we let $\mathcal{A}(H)$ denote the arrangement of H and $\mathsf{V}(\mathcal{A}(H))$ denote the vertices of the arrangement of H.

▶ **Definition 6** (Zone of a surface). *For a collection of hyperplanes H in $\mathbb{R}^d$, the complexity of a cell ψ in the arrangement $\mathcal{A}(H)$ is the number of faces (of all dimensions) which are contained in the closure of ψ. For a $(d-1)$-dimensional surface γ, the **zone** $\mathcal{Z}(\gamma, H)$ of γ is the subset of cells of $\mathcal{A}(H)$ which intersect γ. The **complexity of a zone** is the sum of the complexities of the cells in $\mathcal{Z}(\gamma, H)$.*

The complexity of a zone of a hyperplane is known to be $\Theta(n^{d-1})$ [11]; for general algebraic surfaces it is larger by a logarithmic factor. Furthermore, the cells in the zone of a surface can be computed efficiently using lazy randomized incremental construction [9].

▶ **Lemma 7** ([1, 9]). *Let H be a set of n hyperplanes in $\mathbb{R}^d$ and let γ be a $(d-1)$-dimensional algebraic surface of degree δ. The complexity of the zone $\mathcal{Z}(\gamma, H)$ is $O(n^{d-1}\log n)$, where the hidden constants depend on d and δ. The collection of cells in $\mathcal{Z}(\gamma, H)$ can be computed in $O(n^{d-1}\log n)$ expected time.*

3 Computing the extremal yolk

3.1 Background

▶ **Definition 8.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. A median hyperplane is a hyperplane such that each of its two closed halfspaces contain at least $\lceil n/2 \rceil$ points of P. A hyperplane is extremal if it passes through d points of P. The **extremal yolk** is the ball of smallest radius interesting all extremal median hyperplanes of P.*

We give an $O(n^{d-1}\log n)$ expected time exact algorithm computing the extremal yolk. To do so, we focus on the more general problem.

▶ Problem 9. Let $E_k(P)$ be the collection of extremal hyperplanes which contain exactly k points of P on or above it. Here, k is not necessarily constant. The goal is to compute the smallest radius ball intersecting all hyperplanes of $E_k(P)$.

We observe that computing the extremal yolk can be reduced to the above problem.

▶ **Lemma 10.** *The problem of computing the extremal yolk can be reduced to Problem 9.*

Proof. Suppose that n is even, and define the set $S_{\text{even}} = \{n/2, n/2+1, \ldots, n/2+d\}$. A case analysis shows that any extremal median hyperplane h must have exactly m points of P above or on h, where $m \in S_{\text{even}}$. Thus, computing the extremal yolk reduces to computing smallest radius ball intersecting all hyperplanes in the set $\bigcup_{m \in S_{\text{even}}} E_m(P)$.

When n is odd, a similar case analysis shows that any extremal median hyperplane must have exactly m points above or on it, where $m \in S_{\text{odd}} = \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1, \ldots, \lceil n/2 \rceil + d - 1\}$. Analogously, computing the extremal yolk with n odd reduces to computing the smallest radius ball intersecting all hyperplanes in the set $\bigcup_{m \in S_{\text{odd}}} E_m(P)$. ◀

To solve Problem 9, we apply Chan's result for solving implicit LP-type problems [4], stated in Lemma 2. We first prove that Problem 9 is an LP-type problem when the constraints are explicitly given (the following Lemma was also observed by Bhattacharya et al. [2]).

▶ **Lemma 11.** *Problem 9 when the constraints (i.e., hyperplanes) are explicitly given, is an LP-type problem and has combinatorial dimension $\delta = d+1$.*

Proof. We prove something stronger, namely that the problem can be written as a linear program, implying it is an LP-type problem. Let $\mathcal{H}$ be the set of n hyperplanes. For each hyperplane $h \in \mathcal{H}$, let $\langle a_h, x \rangle + b_h = 0$ be the equation describing h, where $a_h \in \mathbb{R}^d$, $\|a_h\| = 1$, and $b_h \in \mathbb{R}$. Because of the requirement that $\|a_h\| = 1$, for a given point $p \in \mathbb{R}^d$, the distance from p to a hyperplane h is $|\langle a_h, p \rangle + b_h|$.

The linear program has $d+1$ variables and $2n$ constraints. The $d+1$ variables represent the center $p \in \mathbb{R}^d$ and radius $\nu \geq 0$ of the egg. The resulting LP is

$$\begin{aligned} \min \quad & \nu \\ \text{subject to} \quad & \nu \geq \langle a_h, p \rangle + b_h && \forall h \in \mathcal{H} \\ & \nu \geq -\big(\langle a_h, p \rangle + b_h\big) && \forall h \in \mathcal{H} \\ & p \in \mathbb{R}^d. \end{aligned}$$

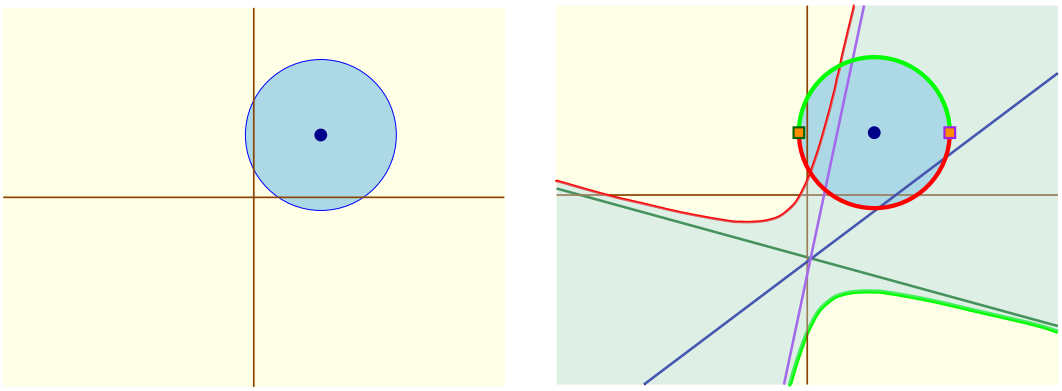


Figure 3.1 A disk and its dual.

As for the combinatorial dimension, observe that any basic feasible solution for the above linear program will be tight for at most $d+1$ of the above $2n$ constraints. Namely, these $d+1$ planes are tangent to the optimal radius ball, and as such form a basis $\mathcal{b} \subseteq \mathcal{H}$. ◀

To apply Lemma 2 we need to:

(i) design an appropriate input space,

(ii) develop a decider, and

(iii) construct a constant number of subproblems which cover the constraint space.

3.2 Building the decider

The algorithm will work in the dual space. In the dual, the interior of a ball b corresponds to a closed region $\mathsf{b}^\star$ which lies between two branches of a hyperboloid, see Figure 3.1.

▶ **Lemma 12.** *The dual of the set of points in a ball is the set of hyperplanes whose union forms the region enclosed between two branches of a hyperboloid.*

Proof. In $\mathbb{R}^d$ the hyperplane h defined by $x_d = \beta + \sum_{i=1}^{d-1} \alpha_i x_i$, or more compactly $\langle x, (-\alpha, 1)\rangle = \beta$, intersects a disk b centered at $p = (p_1, \ldots, p_d)$ with radius $r \iff$ the distance of h from p is at most r. That is, h intersects b if

$$\begin{aligned}
\frac{|\langle p, (-\alpha, 1)\rangle - \beta|}{\|(-\alpha, 1)\|} \leq r &\iff (\langle p, (-\alpha, 1)\rangle - \beta)^2 \leq r^2 \|(-\alpha, 1)\|^2 \\
&\iff \left(p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i\right)^2 \leq r^2(\|\alpha\|^2 + 1). \\
&\iff \frac{\left(p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i\right)^2}{r^2} - \|\alpha\|^2 \leq 1.
\end{aligned}$$

The boundary of the above inequality is a hyperboloid in the variables $p_d - \beta - \sum_{i=1}^{d-1} \alpha_i p_i$ and $\alpha_1, \ldots, \alpha_{d-1}$. This corresponds to an affine image of a hyperboloid in the dual space $\alpha \times -\beta$. ◀

Throughout, we let $\mathsf{b}^\star$ denote the region between the two branches of the hyperboloid dual to a ball b.

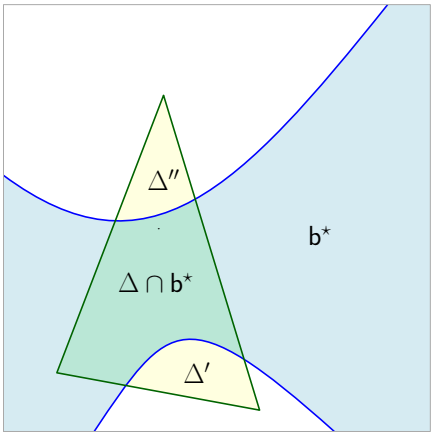


Figure 3.2 The region $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$ consists of (at most) two disjoint convex regions, Δ' and Δ''.

3.2.1 Algorithm

Given a candidate solution (i.e., a ball b in the primal) and a collection of points $Q \subseteq P$. Our goal is to construct a decider which detects if there is a hyperplane of $E_k(P)$, passing through d points of Q, which avoids the interior of the ball b. In the dual setting, the problem is to decide if there is a vertex of $\mathcal{A}(Q^\star)$ which is a member of the k-level, and is inside the region $\mathbb{R}^d \setminus \mathsf{b}^\star$.

The input. The input to the algorithm is a simplex Δ, the set of hyperplanes

$$H = P^\star \cap \Delta = \{h \in P^\star \mid h \cap \Delta \neq \emptyset\}$$

(i.e., all hyperplanes of $P^\star$ that intersect Δ), a candidate solution $\mathsf{b}^\star$, and a parameter u which is the number of hyperplanes of $P^\star$ lying completely below Δ.

The task. Decide if there is a vertex of $\mathcal{A}(P^\star)$ of the k-level in $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$. That is, there is a vertex of level k that is outside $\mathsf{b}^\star$ but inside Δ.

The decision procedure. Consider the set $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$, where Δ is a simplex, and notice that the set is the union of at most two convex regions. Indeed, the set $\mathbb{R}^d \setminus \mathsf{b}^\star$ consists of two disjoint connected components, where each component is a convex body. Intersecting a simplex Δ with each component of $\mathbb{R}^d \setminus \mathsf{b}^\star$ produces two (disjoint) convex bodies Δ' and Δ'' (it is possible that Δ' or Δ'' are empty). See Figure 3.2. Let Δ' be one of these two regions of interest. The algorithm will process Δ'' in exactly the same way.

If Δ' is empty, then no constraints are violated. Otherwise, we need to check for any violated constraints inside Δ'. Let $\partial\Delta'$ denote the boundary of Δ'. Define $H' \subseteq H$ to be the subset of hyperplanes intersecting Δ'. Observe that it suffices to check if there is a vertex v in the arrangement $\mathcal{A}(H')$ such that:

(i) v has level k in $P^\star$,
(ii) v is a member of some cell in the zone $\mathcal{Z}(\partial\Delta', H')$, and
(iii) v is contained in Δ'.

The algorithm computes $\mathcal{Z}(\partial\Delta', H')$. Next, it chooses a vertex v of the arrangement $\mathcal{A}(H')$ which lies inside Δ' and computes its level in H' (adding u to the count). The algorithm then walks around the vertices of the zone *inside* Δ', computing the level of

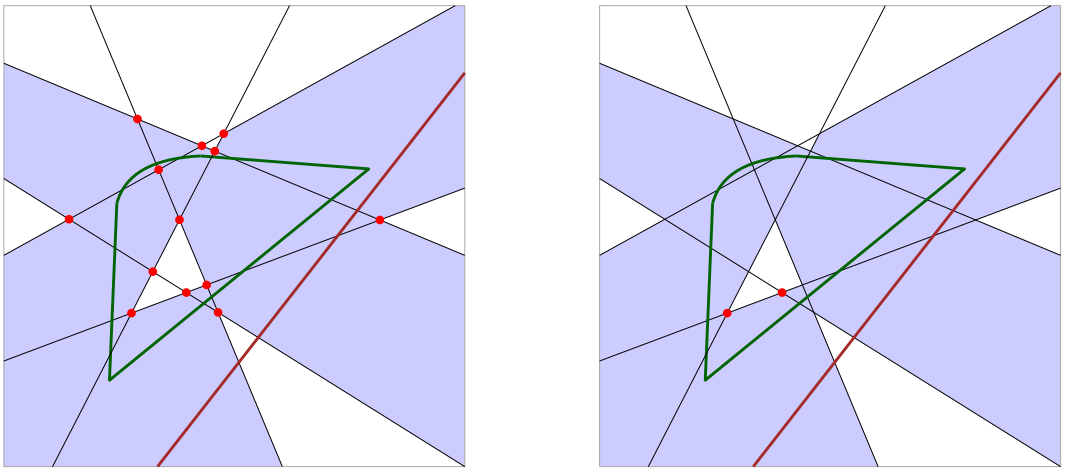


Figure 3.3 Left: A convex region Δ'. Let H' be the set of lines intersecting Δ', with one line lying completely below Δ' ($\mathsf{u} = 1$). The shaded regions are the cells of $\mathcal{A}(H')$ intersecting $\partial\Delta'$. The vertices of the cells in the zone $\mathcal{Z}(\partial\Delta', H')$ are highlighted. Right: The vertices of $\mathcal{Z}(\partial\Delta', H')$ which are part of the 3-level and contained inside Δ'.

each vertex along the walk. Note that the level between any two adjacent vertices in the arrangement differ by at most a constant (depending on d). If at any point we find a vertex of the desired level (such a vertex also lies inside Δ'), we report the corresponding median hyperplane which violates the given ball b. See Figure 3.3 for an illustration.

3.2.2 Analysis

The running time of the algorithm is proportional to the complexity of the zone $\mathcal{Z}(\partial\Delta', H')$. Because the boundary of Δ' is constructed from $d+1$ hyperplanes and the boundary of the hyperboloid, Lemma 7 implies that the zone complexity is no more than $O(|H|^{d-1}\log|H|)$. As such, our decision procedure runs in time $D(n) = O(n^{d-1}\log n)$.

3.3 Constructing subproblems

To decompose a given input into smaller subproblems, we need the notion of cuttings.

▶ **Definition 13** (Cuttings). *Given n hyperplanes in $\mathbb{R}^d$, a $(1/c)$-**cutting** is a collection of interior disjoint simplices covering $\mathbb{R}^d$, such that each simplex intersects at most n/c hyperplanes. A $(1/c)$-cutting of size $O(c^d)$ can be constructed in $O(nc^{d-1})$ time [6].*

Given a simplex Δ and the set of hyperplanes $H = P^\star \cap \Delta$, we compute a $(1/c)$-cutting of H into $O(c^d)$ simplices, and clip this cutting inside Δ. For each cell in this new cutting, we compute the set of hyperplanes which intersect it, and the number of hyperplanes lying completely below the cell naively in $O(|H|)$ time. Repeating this process for the $O(c^d)$ cells implies that this decomposition procedure can be completed in $O(|H|)$ time (ignoring dependencies on d), as $(1/c)$-cuttings can be constructed in deterministic linear time for constant c [6].

The above shows that we can decompose a given input of size n into $\psi = O(c^d)$ subproblems, each of size at most n/c. Furthermore, this decomposition preserves all implicit constraints of interest (vertices of $\mathcal{A}(H)$). Choosing c to be a sufficiently large constant (possibly depending on d), to meet the requirements of Lemma 2, finishes the construction.

3.4 Putting it all together

The above discussions together with Lemma 2 and $D(n) = O(n^{d-1} \log n)$ implies the following.

▶ **Lemma 14.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a given integer k, one can compute in $O(n^{d-1} \log n)$ expected time the smallest radius ball intersecting all of the hyperplanes of $E_k(P)$*

▶ Notation. For an integer $n > 0$, let $[\![n]\!] = \{1, \ldots, n\}$.

▶ **Corollary 15.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position, and let $S \subseteq [\![n]\!]$. One can compute in $O(n^{d-1} \log n)$ expected time the smallest radius ball intersecting all of the hyperplanes of $\bigcup_{k \in S} E_k(P)$*

Proof. The algorithm is a slight modification of Lemma 14. During the decision procedure, for each vertex in the zone, we check if it is a member of the k-level for some $k \in S$. If S is of non-constant size, membership in S can be checked in constant time using hashing. ◀

3.5 Computing the extremal yolk and the egg

▶ **Theorem 16.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the extremal yolk of P in $O(n^{d-1} \log n)$ expected time.*

Proof. The result follows by applying Corollary 15 with the appropriate choice of S. When n is even, Lemma 10 tells us to choose $S = \{n/2, n/2 + 1, \ldots, n/2 + d\}$. When n is odd, we set $S = \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1, \ldots, \lceil n/2 \rceil + d - 1\}$. ◀

▶ **Lemma 17.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the egg of P in $O(n^{d-1} \log n)$ expected time.*

Proof. Follows by Corollary 15 with $S = [\![n]\!]$. (Alternatively, by directly modifying the decision procedure to check if any vertex of the zone $\mathcal{Z}(\Delta', H')$ lies inside Δ'.) ◀

3.6 An algorithm sensitive to k

Recall that to compute the extremal yolk, we reduced the problem to computing the smallest ball intersecting all hyperplanes which contain a fixed number of points of P above or on them (see Lemma 10). In particular, we developed an algorithm for Problem 9 and applied it when k is proportional to n. It is natural to ask for an algorithm for Problem 9 which is faster when k is small. The algorithm will work for all values of k. However when k is large, the running time deteriorates to the running time of the algorithm of Lemma 14.

To develop an algorithm sensitive to k, we use the result of Lemma 14 as a black-box and introduce the notion of shallow cuttings.

▶ **Definition 18** (Shallow cuttings). *Let H be a set of n hyperplanes in $\mathbb{R}^d$. A **k-shallow cutting** is a collection of simplices such that:*

- **(i)** *the union of the simplices covers the $(\leq k)$-level of H (see Definition 5), and*
- **(ii)** *each simplex intersects at most k hyperplanes of H.*

Matoušek was the first to prove existence of k-shallow cuttings of size $O((n/k)^{\lfloor d/2 \rfloor})$ [15]. When $d = 2, 3$, a k-shallow cutting of size $O(n/k)$ can be constructed in $O(n \log n)$ time [5]. For $d \geq 4$, we sketch a randomized algorithm which computes a k-shallow cutting, based on Matoušek's original proof of existence [15].

▶ **Lemma 19** (Proof sketch in [14]). *Let H be a set of n hyperplanes in $\mathbb{R}^d$. A k-shallow cutting of size $O((n/k)^{\lfloor d/2 \rfloor})$ can be constructed in $O(k(n/k)^{\lfloor d/2 \rfloor} + n \log n)$ expected time. For each simplex Δ in the cutting, the algorithm returns the set of hyperplanes intersecting Δ and the number of hyperplanes lying below Δ.*

Let $P \subset \mathbb{R}^d$ be a set of n points and let $H = P^\star$ be the set of dual hyperplanes. The algorithm itself is a randomized incremental algorithm, mimicking Seidel's algorithm for solving LPs [19]. First, compute a k-shallow cutting for the set of hyperplanes H using Lemma 19. Let $\Delta_1, \ldots, \Delta_\ell$, where $\ell = O((n/k)^{\lfloor d/2 \rfloor})$, be the collection of simplices in the cutting. For each simplex Δ_i, we have the subset $H \cap \Delta_i$ and the number of hyperplanes lying completely below H (which is at most k). For each cell Δ_i, let $g(\Delta_i)$ be the set of vertices of $\mathcal{A}(H)$ which have level k and are contained in Δ_i.

The algorithm. The input to the algorithm is a set of simplices and an initial ball b_0. Such a ball is uniquely defined by a subset of $d+1$ constraints, and this is a basis for the LP-type problem.

Begin by randomly permuting the simplices $\Delta_1, \ldots, \Delta_\ell$. At all times, the algorithm maintains a ball b_i of smallest radius which meets all the constraints defined by $\cup_{j=1}^{i} g(\Delta_i)$. In the ith iteration, the algorithm performs a *violation test*: it decides if any constraint of $g(\Delta_i)$ is violated by b_{i-1}. If so, the algorithm executes a *basis computation*, in which it computes the ball b'_i of smallest radius which obeys the constraints of $g(\Delta_i)$ and the $d+1$ constraints defining b_{i-1}. The algorithm then computes a ball b_i by invoking itself recursively on the subset of cells $\Delta_1, \ldots, \Delta_i$ with b'_i as the initial basis.

▶ **Lemma 20.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. For a given integer k, one can compute in $\widetilde{O}\big(k^{d-1}\big(1 + (n/k)^{\lfloor d/2 \rfloor}\big)\big)$ expected time the smallest radius ball intersecting all of the hyperplanes of $E_k(P)$.*

Proof. The algorithm is described above. As for the analysis, it is similar to any randomized incremental algorithm for LP-type problems. The key difference is that we are not adding a single constraint incrementally, but rather a collection of constraints in each iteration. Fortunately, this does not change the analysis of the algorithm (for further details, see the proof of Lemma 2 in [4] or the full version of our paper [14]).

It is well-known that in expectation, the algorithm performs $O((n/k)^{\lfloor d/2 \rfloor})$ violation tests and $O(\log^{d+1}(n/k))$ basis computations [20]. Since each simplex Δ_i intersects $O(k)$ hyperplanes of H, each of these subroutines can be implemented in $O(k^{d-1} \log k)$ time using Lemma 14. Finally, we account for the time needed to construct the shallow cutting – by Lemma 19 this can be done in $O(k(n/k)^{\lfloor d/2 \rfloor} + n \log n)$ expected time. ◀

4 Computing the (continuous) yolk

▶ **Definition 21.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. The **continuous yolk** of P is the ball of smallest radius intersecting all median hyperplanes of P.*

In contrast to Definition 8, we emphasize that the (continuous) yolk must intersect all median hyperplanes defined by P (not just extremal median hyperplanes).

As before, the algorithm works in the dual space. For an integer k, let $H_k(P)$ be the collection of halfspaces containing exactly k points of P on or above it. Equivalently, $P^\star$ is the collection of hyperplanes defined by P in the dual space, and $\big(H_k(P)\big)^\star$ is the k-level of $P^\star$. Our problem can be restated in the dual space as follows.

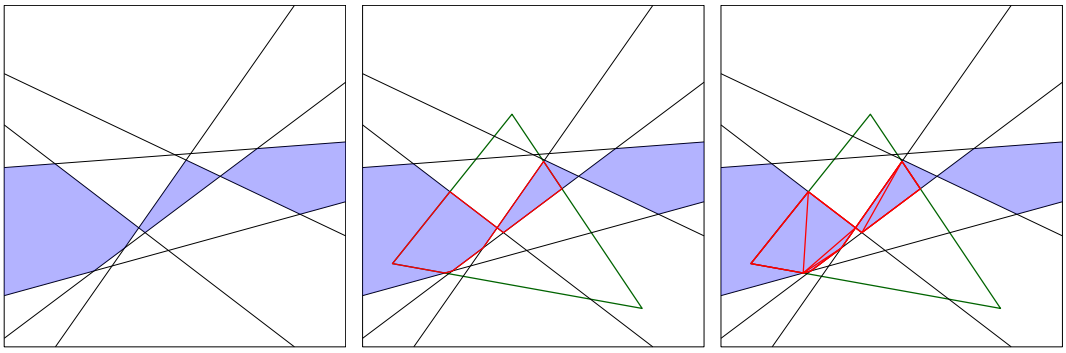


Figure 4.1 Left: A set of lines and the cells of the 3-level. Middle: A simplex Δ, with the portion of the 3-level inside Δ. Right: Triangulating the portion of the 3-level contained inside Δ. All red triangles together with the lower dimensional faces of the 3-level form the set of constraints $g(\Delta)$.

▶ Problem 22. Let P be a set of points in $\mathbb{R}^d$ in general position and let k be a given integer. Compute the ball b of smallest radius so that all points in the k-level of $P^\star$ are contained inside the region $\mathsf{b}^\star$.

Let $L_k(P) = \big(H_k(P)\big)^\star$ denote the set of all points in the k-level of $P^\star$. Note that $L_k(P)$ consists of points which are either contained in the interior of some ℓ-dimensional flat, where $0 \leq \ell \leq d-1$, or in the interior of some d-dimensional cell of $\mathcal{A}(P^\star)$.

We take the same approach as the algorithm of Theorem 16 – building a decider subroutine, and showing that the input space can be decomposed into subproblems efficiently. However the problem is more subtle, as the collection of constraints (i.e., median hyperplanes) is no longer a finite set.

The input space. The input consists of a simplex Δ. The algorithm, in addition to Δ, maintains the set of hyperplanes

$$H = P^\star \cap \Delta = \{h \in P^\star \mid h \cap \Delta \neq \emptyset\},$$

and a parameter u which is equal to the number of hyperplanes of $P^\star$ lying completely below Δ.

The implicit constraint space. Each input Δ maps to a region R which is the portion of the k-level $L_k(P)$ contained inside Δ. For each d-dimensional cell in R, we compute its bottom-vertex triangulation (see, e.g., [16, Section 6.5]), and collect all of these simplices, and all lower dimensional faces of R, into a set $g(\Delta)$, see Figure 4.1.

Let Ξ be the collection of all simplices formed from $d+1$ vertices of the arrangement $\mathcal{A}(P^\star)$. We let $\mathcal{H}$ be the union of the sets $g(\Delta)$ over all simplices $\Delta \in \Xi$. To see why this suffices, each simplex in the input space is a simplex generated by a cutting algorithm. One property of cutting algorithms [6] is that the simplices returned are induced by hyperplanes of $P^\star$. Indeed, each simplex has (at most) $d+1$ vertices, and upon inspection of the cutting algorithm, each vertex is defined by d hyperplanes of $P^\star$. There are a finite number of simplices Δ to consider, and each Δ induces a fixed subset of constraints $g(\Delta) \subseteq \mathcal{H}$.

As such, $\mathcal{H}$ forms our constraint set, where each constraint is of constant size (depending on d). Clearly, a solution satisfies all constraints of $\mathcal{H}$ if and only if the solution intersects all hyperplanes in the set $H_k(P)$. For a given subset $\mathcal{C} \subseteq \mathcal{H}$, the objective function is the minimum radius ball b such that all regions of $\mathcal{C}$ are contained inside the region $\mathsf{b}^\star$. In particular, the problem of computing the minimum radius ball b such that $\mathsf{b}^\star$ contains all points of $L_k(P)$ in its interior is an LP-type problem of constant combinatorial dimension.

Constructing subproblems. For a given input simplex Δ (along with the set $H = P^\star \cap \Delta$ and the number u) a collection of subproblems $\Delta_1, \ldots, \Delta_\psi$ (with the corresponding sets H_i and numbers u_i for $i = 1, \ldots, \psi$) can be constructed as described in Section 3.3, by computing a cutting of the planes H and clipping this cutting inside Δ. In particular, we have that $\bigcup_i g(\Delta_i) = g(\Delta)$. Strictly speaking, we have not decomposed the constraints of $g(\Delta)$ (as required by Lemma 2), but rather have decomposed the region which is the union of the constraints of $g(\Delta)$. This step is valid, as a solution satisfies the constraints of $\bigcup_i g(\Delta_i)$ if and only if it satisfies the constraints of $g(\Delta)$.

The decision procedure. Given a candidate solution $\mathsf{b}^\star$, the problem is to decide if $\mathsf{b}^\star$ contains $g(\Delta)$ in its interior. The decision algorithm itself is similar as in the proof of Theorem 16. Consider the set $\Delta \cap (\mathbb{R}^d \setminus \mathsf{b}^\star)$, where Δ is a simplex, and notice that it is the union of the most two convex regions. Let Δ' be one of these two regions of interest. Observe that it suffices to check if there is a point on the boundary of Δ' which is part of the k-level. Let $H' \subseteq H$ be the subset of hyperplanes intersecting Δ'.

To this end, compute $\mathcal{Z}(\partial\Delta', H')$. For each $(d-1)$-dimensional face f of Δ', the collection of regions $\Xi = \{f \cap s \mid s \in \mathcal{Z}(\partial\Delta', H')\}$ forms a $(d-1)$-dimensional arrangement restricted to f. Furthermore, the complexity of this arrangement lying on f is at most $O(n^{d-1} \log n)$. Notice that the level of all points in the interior of a face of Ξ is constant, and two adjacent faces (sharing a boundary) have their level differ by at most a constant. The algorithm picks a face in Ξ, computes the level of an arbitrary point inside it (adding u to the count). Then, the algorithm walks around the arrangement, exploring all faces, using the level of neighboring faces to compute the level of the current face. If at any step a face has level k, we report that the input (Δ, H, u) violates the candidate solution $\mathsf{b}^\star$.

Analysis of the decision procedure. We claim the running time of the algorithm is proportional to the complexity of the zone $\mathcal{Z}(\partial\Delta', H')$. Indeed, for each $(d-1)$-dimensional face f of Δ' (where f may either be part of a hyperplane or part of the boundary of $\mathsf{b}^\star$), we can compute the set $\{f \cap s \mid s \in \mathcal{Z}(\partial\Delta', H')\}$ in time proportional to the total complexity of $\mathcal{Z}(\partial\Delta', H')$ (assuming we can intersect a hyperplane with a portion of a constant degree surface efficiently). The algorithm then computes the level of an initial face naively in $O(|H'|)$ time, and computing the level of all other faces can be done in $O(|\mathcal{Z}(\partial\Delta', H')|)$ time by performing a graph search on the arrangement.

Because the boundary of Δ' is constructed from $d+1$ hyperplanes and the boundary of the hyperboloid, Lemma 7 implies that the zone complexity is $O(|H|^{d-1} \log |H|)$. As such, our decision procedure runs in time $D(n) = O(n^{d-1} \log n)$.

▶ **Lemma 23.** *Problem 22 can be solved in $O(n^{d-1} \log n)$ expected time, where $n = |P|$.*

Proof. Follows by plugging the above discussion into Lemma 2. ◀

By modifying the decision procedure appropriately, we also obtain a similar result to Corollary 15.

▶ **Corollary 24.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position, and let $S \subset [\![n]\!]$. The smallest ball intersecting all hyperplanes in $\bigcup_{k \in S} H_k(P)$ can be computed in $O(n^{d-1} \log n)$ expected time.*

▶ **Theorem 25.** *Let $P \subset \mathbb{R}^d$ be a set of n points in general position. One can compute the yolk of P in $O(n^{d-1} \log n)$ expected time.*

Proof. The result follows by applying Corollary 24 with the appropriate choice of S. When n is even, Lemma 10 tells us to choose $S = \{n/2, n/2+1, \ldots, n/2+d\}$. When n is odd, we set $S = \{\lceil n/2 \rceil, \lceil n/2 \rceil + 1, \ldots, \lceil n/2 \rceil + d - 1\}$. ◀

▶ Remark 26. In $\mathbb{R}^3$, one can shave the $O(\log n)$ factor to obtain an $O(n^2)$ expected time algorithm for the yolk. We modify the decision procedure as follows, which avoids computing the zone $\mathcal{Z}(\partial\Delta', H')$. For each 2D face f of Δ', simply compute the arrangement of the set of lines $\{f \cap h \mid h \in H\}$ on f in $O(n^2)$ time. As before, we perform a graph search on this arrangement, computing the level of each face. If any time we discover a point on the boundary of Δ' of the desired level, we report that the given input violates the given candidate solution.

5 Conclusion

The natural open problem is to improve the running times for computing the yolk (and extremal yolk) even further. It seems believable, that for $d > 3$, the log factors in Theorem 16 and Theorem 25 might not be necessary. We leave this as an open problem for further research.

References

1. Boris Aronov, Marco Pellegrini, and Micha Sharir. On the zone of a surface in a hyperplane arrangement. *Discrete Comp. Geom.*, 9:177–186, 1993. doi:10.1007/BF02189317.
2. Binay K. Bhattacharya, Shreesh Jadhav, Asish Mukhopadhyay, and Jean-Marc Robert. Optimal algorithms for some intersection radius problems. *Computing*, 52(3):269–279, 1994. doi:10.1007/BF02246508.
3. Duncan Black. On the rationale of group decision-making. *Journal of Political Economy*, 56(1):23–34, 1948. doi:10.1086/256633.
4. Timothy 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.
5. Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comp. Geom.*, 56(4):866–881, 2016. doi:10.1007/s00454-016-9784-4.
6. Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comp. Geom.*, 9:145–158, 1993. doi:10.1007/BF02189314.
7. Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995. doi:10.1145/201019.201036.
8. Mark de Berg, Jonathan Chung, and Joachim Gudmundsson. Computing the yolk in spatial voting games, 2019.
9. Mark de Berg, Katrin Dobrindt, and Otfried Schwarzkopf. On lazy randomized incremental construction. *Discrete Comp. Geom.*, 14(3):261–286, 1995. doi:10.1007/BF02570705.
10. Mark de Berg, Joachim Gudmundsson, and Mehran Mehr. Faster algorithms for computing plurality points. *ACM Trans. Algorithms*, 14(3):36:1–36:23, 2018. doi:10.1145/3186990.
11. Herbert Edelsbrunner, Raimund Seidel, and Micha Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. Comput.*, 22(2):418–429, 1993. doi:10.1137/0222031.
12. Joachim Gudmundsson and Sampson Wong. Computing the yolk in spatial voting games without computing median lines. In *33th Conf. Artificial Intell.* (AAAI), 2019.
13. Joachim Gudmundsson and Sampson Wong. Computing the yolk in spatial voting games without computing median lines. *CoRR*, abs/1902.04735, 2019. arXiv:1902.04735.
14. Sariel Har-Peled and Mitchell Jones. Fast algorithms for geometric consensuses. *CoRR*, abs/1912.01639, 2019. arXiv:1912.01639.

15 Jiří Matoušek. Reporting points in halfspaces. *Comput. Geom.*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

16 Jiří Matoušek. *Lectures on Discrete Geometry*, volume 212 of *Grad. Text in Math.* Springer, 2002. doi:10.1007/978-1-4613-0039-7/.

17 Richard D. McKelvey. Covering, dominance, and institution-free properties of social choice. *American Journal of Political Science*, 30(2):283–314, 1986. doi:10.2307/2111098.

18 Ariel Rubinstein. A note about the "nowhere denseness" of societies having an equilibrium under majority rule. *Econometrica*, 47(2):511–514, 1979. doi:10.2307/1914198.

19 Raimund Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comp. Geom.*, 6:423–434, 1991. doi:10.1007/BF02574699.

20 Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems. In *9th Symp. on Theoretical Aspects of Comput. Sci.* (STACS), pages 569–579, 1992. doi:10.1007/3-540-55210-3_213.

21 Richard E. Stone and Craig A. Tovey. Limiting median lines do not suffice to determine the yolk. *Social Choice and Welfare*, 9(1):33–35, 1992. doi:10.1007/BF00177668.

22 Craig A. Tovey. A polynomial-time algorithm for computing the yolk in fixed dimension. *Math. Program.*, 57:259–277, 1992. doi:10.1007/BF01581084.