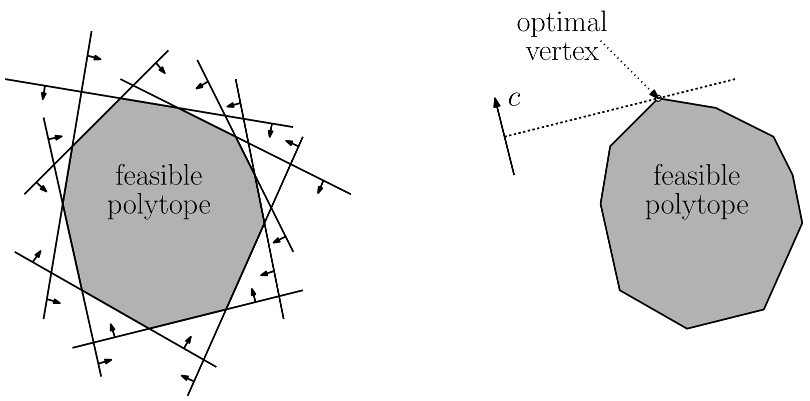

Figure 1. Left: The feasible polytope is defined by multiple halfspaces; Right: Goal is to find optimal vertex in the feasible polytope that is furthest in the direction of the objective function vector c.

Note: These lecture notes are based on the textbook "Computational Geometry" by Berg et al.*; lecture notes from [1]; and lecture notes from MIT 6.854J Advanced Algorithms class by M. Goemans*

1 The Linear Programming Problem

In linear programming (LP), we want to find a point in d dimensional space that minimizes a given linear *objective function* subject to a set of linear *constraints*. Currently, LP solvers can handle about a million dimensions, using a combination of formally analyzable algorithms, and also many ad hoc heuristics. Here, we're going to focus on an elegant algorithm that has linear expected run time when the number of dimensions is a constant.

In the LP problem, we're given a set of linear inequalities, called constraints in $\mathbb{R}^d$. Given a point $(x_1, \ldots x_d) \in \mathbb{R}^d$, we can express a constraint as $a_1x_1 + \ldots a_dx_d \leq b$, by specifying coefficients a_i, $b \in \mathbb{R}$. There is no loss in generality by assuming only these types of constraints, since we can convert other constraints to this form via multiplication by -1. Each constraint defines a halfspace in $\mathbb{R}^d$ and the intersection of halfspaces defines a (possibly empty or unbounded) polytope called the *feasible polytope*.

Next we're given a linear *objective function* to be maximized. Given a point $x \in \mathbb{R}^d$, we express the objective function as $c_1x_1 + \ldots c_dx_d$, for coefficients c_i.[1] We can think of the coefficients as a vector $c \in \mathbb{R}^d$, and then the value of the objective function for $x \in \mathbb{R}^d$ is just $x \cdot c$. Assuming general position, it's not hard to see that if a solution exists, it'll be achieved by a vertex of the feasible polytope. See Figure 1.

In general, a d-dimensional LP can be expressed as.

Minimize: $c_1x_1 + c_2x_2 + \ldots c_dx_d$.

[1]Again there is no difference between minimization or maximization since we can negate the coefficients to go from one to the other.

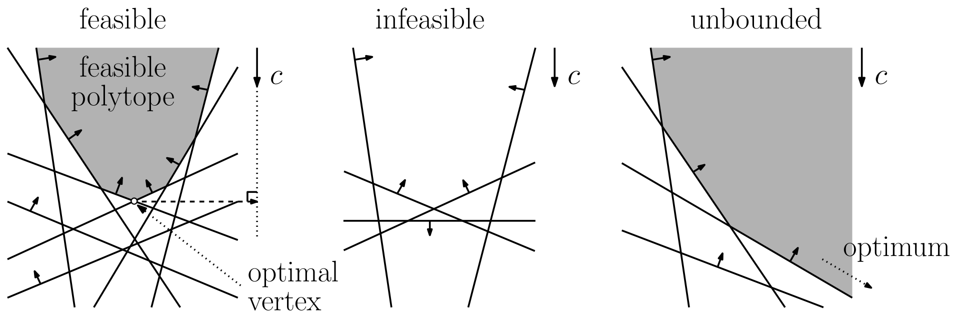

Figure 2. Possible Outcomes of a LP

Subject to:

$$\begin{aligned} a_{1,1}x_1 + \dots a_{1,d}x_d &\geq b_1 \\ a_{2,1}x_1 + \dots a_{2,d}x_d &\geq b_2 \\ &\dots \\ a_{n,1}x_1 + \dots a_{n,d}x_d &\geq b_n \end{aligned}$$

where $x_1, \dots x_d$ are the d variables,and $a_{i,j}, c_i$ and b_i are given real numbers. This can be written in matrix notation as

$$\begin{array}{ll} \text{Minimize:} & c^T x, \\ \text{Subject to:} & Ax \geq b \end{array}$$

Here c and x are d-vectors, b is an n and A is a n by d matrix, where n is the number of constraints. Note that n should be at least as large as d.

There are three possible outcomes for a given LP problem. See Figure 2

Feasible: An optimal point exists and (assuming general position) is a unique vertex of the feasible polytope.

Infeasible: The feasible polytope is empty and there is no solution

Unbounded: The feasible polytope is unbounded in the direction of c and so no finite optimal solution exists.

2 Seidel's LP Algorithm: LP in Constant Dimensions

We now discuss the incremental construction method for very efficiently solving LP in constant dimensions. There are other faster algorithms for LP when the number of dimensions is not a constant. For example, the interior point method is polynomial both the number of constraints and the dimension. The Simplex method is often used in practice, although it is only polynomial time for an input model called "smoothed analysis", where the input values all have some random perturbation. But Seidel's algorithm generally is great when the number of dimensions is (say) no more than 10 and the number of constraints is large.

Seidel's algorithm uses a technique called "Incremental Construction" which is a very useful general tool in computational geometry.

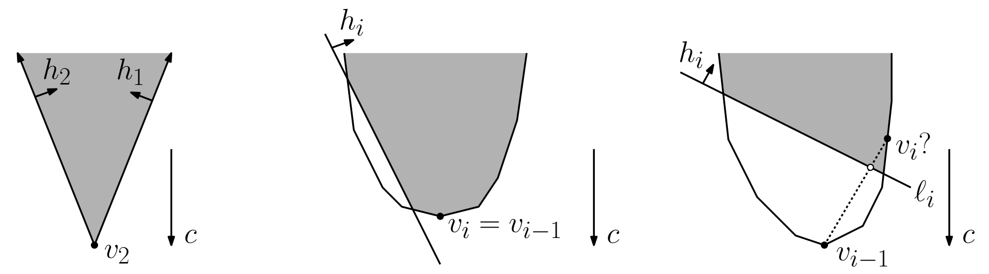

Figure 3. Left: Starting the incremental construction; Right: Proof that new optimum lies on ℓ_i

2.1 Initialization

Recall that we are given n halfspaces $\{h_1, \ldots h_n\}$ in $\mathbb{R}^d$, and an objective vector c, with indices c_i for all $i \in [1, d]$. We want to compute the vertex of the feasible polytope that is the most extreme in the direction of c.

We will initially assume that the LP is bounded and that we have d halfspaces that provide us with an initial feasible point. Our approach will be to add halfspaces one at a time and successively update this feasible point.

First, we create a set of initial d bounding halfspaces. Assume that there is some maximum value M that any variable can take on. Then, we will add d constraints of the following form for all variables:

$$\forall i \in [1, d],\ x_i \leq M \text{ if } c_i < 0 \text{ and } -x_i \leq M \text{ otherwise.}$$

Then ww perturb these constraints by small random amounts to ensure that the hyperplanes associated with them all intersect. These will be our initial d constraints. See Figure 3 left.

If one of these "max value" constraints turns out to intersect the point that is eventually output, then we know that the original LP is unbounded. In this way, we can detect unbounded LPs.

In the bounded case, we'll assume that there is a unique solution, which we call the optimal vertex. This follows via the general position assumption (or by rotating the planes slightly).

2.2 Incremental Algorithm

Recall that we have n input halfspaces. We can imagine adding these halfspaces, $h_1, h_2, \ldots h_n$, and with each addition, updating the current optimum vertex if necessary. The feasible polytope gets smaller with each halfplane addition and so the value of the objective function can only decrease. In Figure 4, the y-coordinate of the feasible vertex decreases.

For $i : 1 \leq i \leq n$, let v_i be the optimum vertex after halfspace h_i is added. There are two cases that can occur when h_i is added. In the easy case, v_{i-1} lies in the halfspace h_i, and so already satisfies the constraint. Then, $v_i = v_{i-1}$ (See Figure 3 middle).

In the hard case, v_{i-1} is not in the halfspace h_i, since it violates the i-th constraint. In this case, the following lemma shows that v_i must lie on the hyperplane that bounds h_i.

Lemma 1. *After adding halfspace h_i, if the LP is still feasible but $v_i \neq v_{i-1}$, then v_i lies on the hyperplane bounding h_i.*

Proof: Let ℓ_i denote the bounding hyperplane for h_i. Assume by contradiction that v_i does not lie on hyperplane ℓ_i (see Figure 3, right). Now consider the line segment between v_{i-1} and v_i. First note that this line segment must cross hyperplane ℓ_i since v_i is in halfspace h_i and v_{i-1} is not.

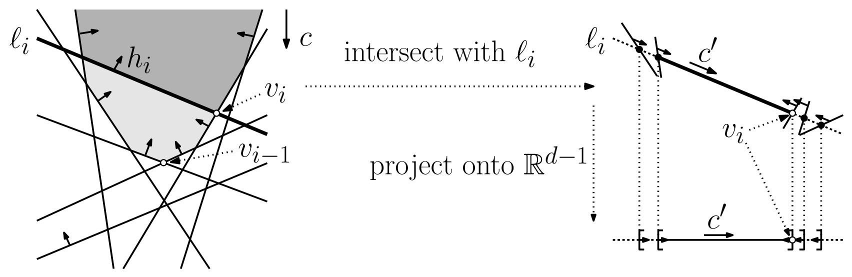

Figure 4. Projection during the incremental construction.

Further, the entire line segment is in the region bounded by the first $i-1$ halfspaces, and so by convexity, the part of the line segment that is in halfspace h_i is also in the region bounded by the first i halfspaces.

The objective function is minimized on this line segment at the point v_{i-1}. Since the objective function is linear, it must be non-increasing as we move from v_i to v_{i-1}. Thus, there is a point on hyperplane ℓ_i with objective function at most equal to that of point v_i. But this contradicts the uniqueness property, so v_i must be on hyperplane ℓ_i. □

2.3 Recursively updating v_i

Consider the case where v_{i-1} does not lie in halfspace h_i (Figure 4, left). Again, let ℓ_i denote the hyperplane bounding h_i. We basically project everything onto that hyperplane and solve a $d-1$ dimensional LP. In particular, we first project c onto ℓ_i to get the vector c' (Figure 4, right). Next intersect each of the halfspaces $h_1, \dots h_{i-1}$ with ℓ_i. Each projection is a $d-1$ dimensional halfspace that lies on hyperplane ℓ_i. Finally, since ℓ_i is a $d-1$ dimensional hyperplane, we can project ℓ_i onto $\mathbb{R}^{d-1}$ with a 1-to-1 mapping. Then, we apply this mapping to all the other vectors to get a LP in $\mathbb{R}^{d-1}$ with $i-1$ constraints.

Algebraically, the way we do this is: (1) set the constraint associated with hyperplane ℓ_i to equality; and (2) remove a variable and that constraint from the LP. For example, imagine there are 3 dimensions and the constraint associated with ℓ_i is $x_1 + 2x_2 - 3x_3 \leq 5$. Then we set $x_1 = 5 - 2x_2 + 3x_3$; do a substitution in the LP using this equation wherever we see the variable x_1; and then remove the variable x_1 from the LP. We can do all this in $O(d \cdot i)$ time.

2.4 Base Case

The recursion ends when we get an LP in 1-dimensional space. Then the projected objective vector just points one way or the other on the real line; and the intersection of each halfspace with ℓ_i is a ray. Computing the intersection of a collection of rays on the line can be done in linear time. For example, see the heavy solid line in Figure 4, right. The optimum point is whichever endpoint of this interval is most extreme in the direction of c'. If the interval is empty, then the feasible polytope is also empty. So when $d = 1$, we can solve the LP over i halfplanes in $O(i)$ time.

2.5 Worst-Case Analysis

Let $T(d, n)$ be the runtime of our algorithm for the LP with n constraints in d dimensional space. To analyze the runtime, let's implement the algorithm recursively.

What is $T(d, n)$? We first do $T(d, n-1)$ work to get the point x_{n-1} that is optimal for the LP with all but the last constraint. Then, in the worst case, point x_{n-1} never satisfies the n-th

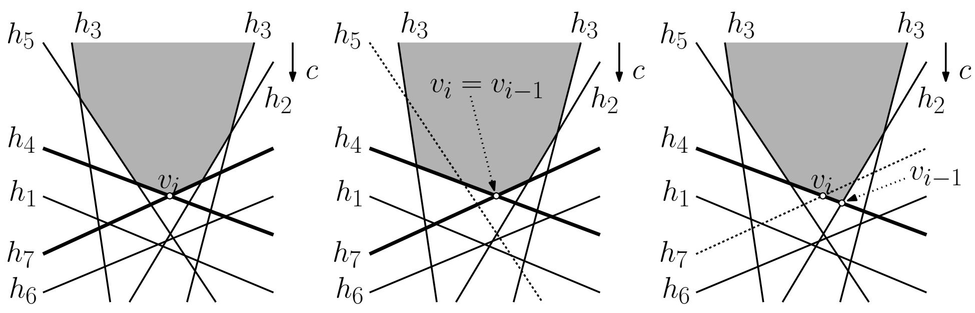

Figure 5. Backwards analysis for Randomized LP

constraint. In this case, we reduce the LP to $d-1$ dimensions, which takes $O(dn)$ time ($O(d)$ time to eliminate the variable in each constraint); and then recursively solve the new smaller dimensional LP, which takes time $T(d-1, n-1)$. So, in the worst case, we get the base case: $T(1, n) = n$; and for $d > 1$:

$$T(d, n) = T(d, n-1) + T(d-1, n-1) + O(dn)$$

Unfortunately, the solution to this recurrence is superlinear in n.

2.6 Randomization to the Rescue

The above analysis assumes we *always* require a projection, and that we *never* get the lucky case where v_{i-1} is in h_i. If we first randomly permute the hyperplanes, we can calculate the probability of the "lucky" and "unlucky" cases to get an expected runtime. Let p_i be the probability that $v_i \neq v_{i-1}$. Then the expected runtime is bounded by the following recurrence relation (multiplicative constants in the asymptotic costs set to 1 for simplicity):

$$T(d, n) = T(d, n-1) + d + p_n(dn + T(d-1, n-1))$$

So what is p_i? Assuming general position, there are exactly d halfspaces whose intersection defines the point v_i. At any step i, there have been i total halfspaces inserted, exactly d of which define the point v_i. Since the halfspaces are randomly permuted, this means that

$$p_i = \frac{d}{i}$$

For example, in Figure 5, h_7 and h_4 define the point v_i, so v_i changes iff one of these two is the last of the 7 halfspaces inserted. In this analysis, we have denoted d halfspaces as special (those that define v_i) and only then revealed the permutation order of the first i halfspaces. This technique is sometimes called *backwards analysis* or *principle of deferred decision.*

Plugging p_n back into the recurrence, we now get:

$$T(d, n) = T(d, n-1) + \frac{d}{n}T(d-1, n-1) + d^2$$

with base cases $T(1, n) = O(n)$ and $T(d, 1) = O(d)$. We can now prove the following.

Lemma 2.

$$T(d, n) = O\left(\left(\sum_{1 \leq i \leq d} \frac{i^2}{i!}\right) d!n\right) = O(d!n)$$

Proof: We show this by induction on n. For any value of d, let C_d be a constant to be solved for later. We will show that $T(d,n) \le C_d d! n$. The base case is clear.

We have:

$$\begin{aligned}
T(d,n) &= T(d,n-1) + \frac{d}{n}T(d-1,n-1) + d^2 \\
&\le C_d d!(n-1) + \frac{d}{n}C_{d-1}(d-1)!(n-1) + d^2 && \text{By IH} \\
&\le C_d d! n - C_d d! + C_{d-1} d! + d^2 \\
&\le C_d d! n
\end{aligned}$$

The last step holds if:

$$C_d d! \ge C_{d-1} d! + d^2$$

which holds if

$$C_d \ge C_{d-1} + d^2/d!$$

□

3 Higher Dimension Convex Hull Algorithms

Note: These lecture notes are based on lecture notes from MIT 6.854J Advanced Algorithms class by M. Goemans

3.1 Definitions

A *polytope* is informally a geometric object with "flat" sides. More formally, it is the convex hull of a finite number of points. Another recursive definition is:

- A 0-polytope is a point
- A 1-polytope is a line segment (edge)
- The faces (sides) of a k-polytope are (k-1)-polytopes that may have (k-2)-polytopes in common. For example a 2-polytope has sides that are line segments, which may meet at points.

A *simplex* is a k-polytope that is the convex hull of its $k+1$ vertices. Informally, it is the generalization of the idea of a triangle or tetrahedron.

For any $0 \le k < d$, a k-face of a d-polytope, P is a face of P with dimension k. A (d-1)-face is called a facet. A (d-2)-face is called a ridge. A 1-face is a edge, and a 0-face is a vertex.

A *simplicial polytope* is a polytope where every face is a simplex. Assuming general position, all polytopes are simplicial polytopes.

Every facet of a d-polytope has a *supporting hyperplane*, which is the hyperplane in dimension d that intersects the entire facet.

3.2 Number of Facets

Even outputting all facets of a polytope in high dimensions can be a challenge. In particular, the number of facets may be exponential in the dimension, as we'll show in Section 7.

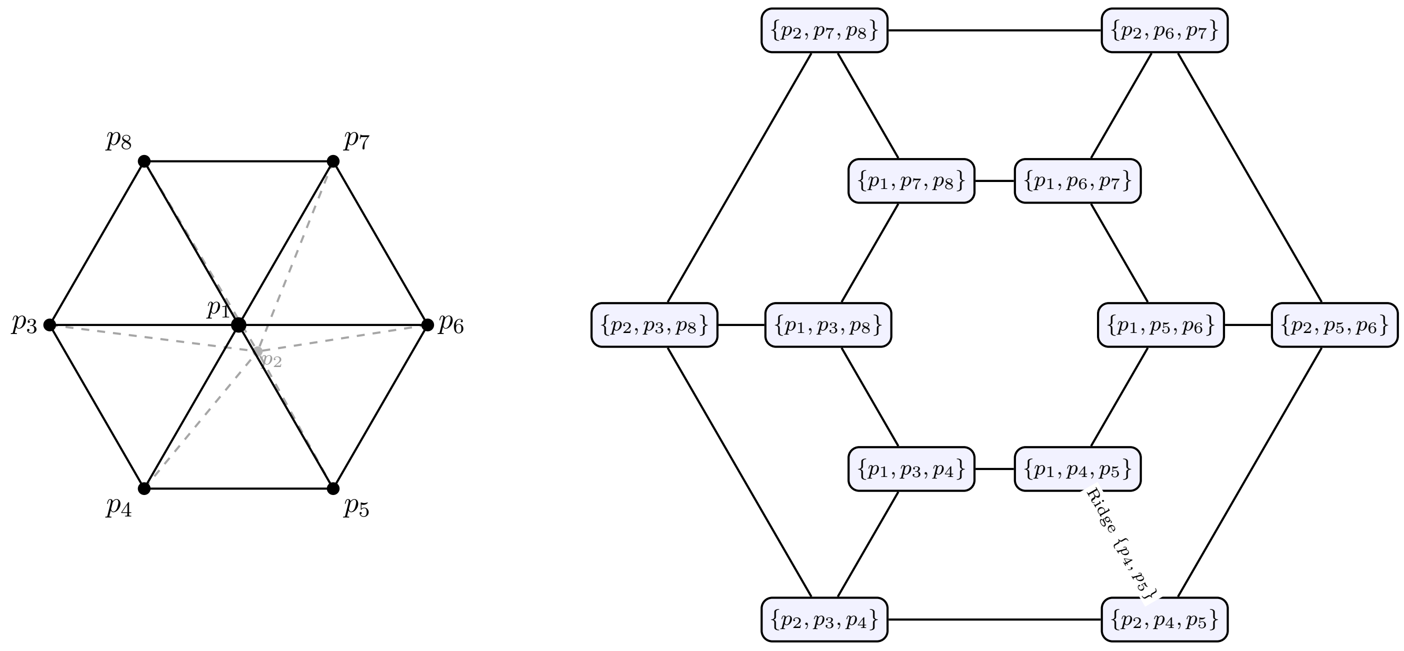

Figure 6. Left: A 3D simplicial polytope with 9 vertices and 12 facets. Right: The corresponding facet graph $\mathcal{F}(P)$ containing nodes for each of the 12 facets and edges for each of the ridges connecting a pair of facets.

3.3 Output of convex-hull algorithm

Seidel's algorithm outputs a facet graph, $\mathcal{F}(P)$:

- Vertices of $\mathcal{F}(P)$ are the facets of conv(P). Each vertex is associated with the d points that define the facet.
- Edges of $\mathcal{F}(P)$ are the ridges of conv(P). Each ridge connects two facets, whose intersection defines the ridge.

An example facet graph is given in Figure 6.

4 Seidel's Convex Hull algorithm

Seidel's algorithm has expected runtime $O(n^2 + n^{\lfloor d/2 \rfloor})$ and assumes points are in general position. For $d \geq 3$, it is optimal. Take a random permutation $p_1, p_2, \dots p_n$ of the points. Let P_i be the convex hull of $p_1, \dots p_i$. We incrementally compute $P_{d+2}, \dots, P_n$, using notions of visibility.

4.1 Preliminaries

Visibility. We make use of the following definitions about visibility.

- A facet F is *visible* from a point p, if the supporting hyperplane of F separates p from P. Otherwise F is called *obscured*.
- From the vantage of a point p, a ridge of P is called
  - *visible:* if both facets it connects are visible
  - *obscured:* if both facets are obscured
  - *a horizon ridge:* if one facet is visible and the other obscured.

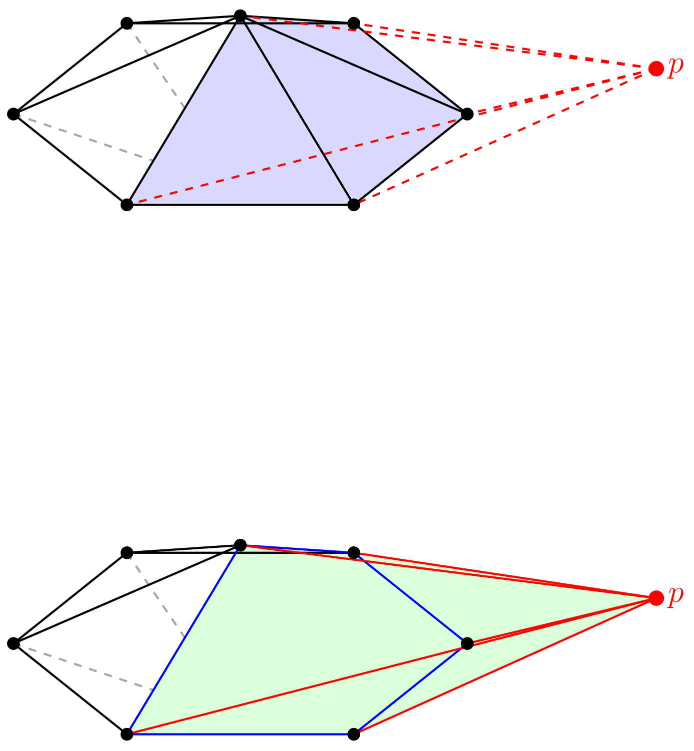

Figure 7. Top: Shaded regions are the facets visible from the point X, with dashed red line segments from X to all visible vertices. Bottom: Visible facets are removed and new facets are added. New ridges are red, horizon ridges are blue.

Ridges. There are d ridges bordering each facet. To see this, note that each facet is uniquely determined by d points. And each ridge bordering that facet is uniquely determined by $d-1$ points. This implies that each facet borders d ridges. For example, if we have the facet p_1, p_5, p_7, p_8, p_9. Then this facet borders the 5 ridges: (p_5, p_7, p_8, p_9); (p_1, p_7, p_8, p_9); (p_1, p_5, p_8, p_9); (p_1, p_5, p_7, p_9); (p_1, p_5, p_7, p_8).

4.2 The algorithm

The algorithm is incremental, keeping track of the convex hull of points $p_1, \ldots, p_{i-1}$. It adds point p_i in step i, removing all facets visible from p_i and adding in all the new facets induced by p_i. See Figure 7.

First, we randomly permute all the points in P. Then, we start out with the convex hull formed by the first d points. Then for any $i \in [d+1, n]$, let C_{i-1} be the convex hull of points $p_1, \ldots p_{i-1}$. All ridges in the current hull are maintained in a search tree, with each ridge having doubly-linked pointers to the two facets forming that ridge. The search tree for the ridges is height $O(d)$, enabling lookups and insertions in $O(d)$ time.

We will be using *normalized hyperplanes*: for length d vectors of coefficients $\vec{a}$ and variables $\vec{x}$, and real value b, the equation : $\vec{a}^T \vec{x} = b$ describes a hyperplane. But also, the same hyperplane is described by the equation $(\vec{a'})^T \vec{x} = 1$, where the vector $\vec{a'} = (1/b)\vec{a}$.

We will also use the fact that in any solution to a LP with d variables, at least d constraints will be tight. To see this recall that the LP solution is a point in d dimensional space, which is determined by the intersection of d of the hyperplane-delineated constraints (See Figure 2).

1. Find a horizon ridge of C_i. If there is no such ridge, skip all steps below. To do this, we seek a hyperplane $a^T x = 1$ such that $a^T p_i = 1$ and either (1) $a^T p_j \leq 1$ for all $j = 1, \ldots, i-1$; or (2) $a^T p_j \geq 1$ for all $j = 1, \ldots, i-1$. This hyperplane can be found by solving a linear program in d dimensions, where the variables are the possible coefficient values a_i for the hyperplane; and the i constraints are given by the inequalities above. We solve this LP using Seidel's algorithm in $O(d!i)$ expected time. In the solution to this LP, d of the constraints will be tight. So the LP solution gives a hyperplane supporting a new facet of C_i that contains p_i. Also, the $d-1$ tight constraints for points in C_{i-1} are points supporting a horizon ridge!

2. Next, use this first horizon ridge to find all visible facets and all horizon ridges, via a DFS in the facet graph. We can do this since visible facets and invisible facets are separated by horizon ridges, and so all visible facets are connected. To determine if a facet is visible during the DFS, check if p_i is on the opposite side of the halfspace supporting the facet than the points $p_1, \ldots p_{i-1}$, which are all on the same side of the halfspace. Delete all visible facets and all visible ridges.

3. Construct all new facets. Each horizon ridge corresponds to a new facet combining p_i and all the p_j points in that ridge.

4. Each new facet contains d ridges. Find each ridge in the ridge search tree, or insert it if it is a new ridge. Maintain pointers between each ridge and the two facets that neighbor that ridge.

4.3 Example for Step 1

We know that $a^T x = 1$ defines a hyperplane in $\mathbb{R}^2$. For example, consider the hyperplane $(2,1)^T(x,y) = 1$. In Step 1, we first try to find a coefficient vector a that ensures that the point p_i is on the hyperplane, and that all other points $p_1, \ldots, p_{i-1}$ are on the same side of the halfspace delineated by that hyperplane. So, if $i = 3$ and $p_1 = (1,0)$, $p_2 = (0,1)$, and $p_3 = (1,2)$, we solve the following linear program. Find variables a_1, a_2, such that:

$$\begin{aligned} (a_1, a_2)^T(1,2) &= 1 \\ (a_1, a_2)^T(1,0) &\leq 1 \\ (a_1, a_2)^T(0,1) &\leq 1 \end{aligned}$$

There is really nothing to maximize or minimize, we just want to find a feasible point, but we could just minimize an arbitrary function like $a_1 + a_2$. Finally, we create a second LP where the last two constraints have $\geq$ instead of $\leq$. Determining if there is a solution to either of these LPs determines if there is any hyperplane that supports a new facet of the convex hull of points $p_1, \ldots, p_i$.

4.4 Runtime Analysis

We assume d is a constant and that there are n points.

Lemma 3. *Seidel's convex hull algorithm has expected runtime* $O(n^{\lfloor d/2 \rfloor} + n^2)$

Proof: The time to add point p_i is $O(i + N_i)$ where N_i is a random variable giving the number of new facets created when p_i is added. To see this, first note that step (1) takes $O(d!i)$ expected time to solve the two LPs. This is $O(i)$ time assuming d is fixed. In step (2), we delete all visible facets and ridges. Each facet takes $O(d)$ time to process, since determining if it is visible is equivalent to determining if p_i is on the opposite side of the supporting hyperplane from any other single point in $p_1, \ldots p_{i-1}$. Since this is constant time, we charge the time to delete these facets and ridges to the time they were created. In step (3), we create N_i new facets, which takes $O(N_i)$ time. In step (4), there are at most $O(dN_i)$ new ridges, each of these can be processed in the ridge tree in $O(d)$ time, so this step takes $O(d^2 N_i) = O(N_i)$ time. So the total time to process p_i is $O(i + N_i)$.

To get the expected runtime, we compute $E(N_i)$ using the principle of deferred decision. Fix one of the facets of C_i. Then, the probability that point p_i participates in this facet is d/i, given that the points are randomly permuted. Next, we note that a polytope with i vertices in d dimensions has $O(i^{\lfloor d/2 \rfloor})$ facets (See Theorem 1). So, using linearity of expectation over all $O(i^{\lfloor d/2 \rfloor})$ facets, we have $E(N_i) = O((d/i)i^{\lfloor d/2 \rfloor}) = O(i^{\lfloor d/2 \rfloor - 1})$. Thus, the expected runtime of Seidel's convex hull algorithm is, for some constants C and C':

$$\begin{aligned}\sum_{i=1}^{n} C(i + E(N_i)) &= \sum_{i=1}^{n} C'(i + i^{\lfloor d/2 \rfloor - 1}) \\ &= O(n^{\lfloor d/2 \rfloor} + n^2)\end{aligned}$$

The last step holds since for any $x \geq 0$, $\sum_{i=1}^{n} i^x \leq \sum_{i=1}^{n} n^x = n^{x+1}$. □

5 Polar Transformation

There are two key ways to create convex polytopes: (1) convex hull of a set of points; and (2) intersection of a collection of closed halfspaces. We show that these are essentially identical via a *polar transformation.* A polar transformation maps points to hyperplanes and vice versa; it's another example of duality.

Fix any point $\mathcal{O}$ in d-dimensional space. If $\mathcal{O}$ is the origin, we can view any point $p \in \mathbb{R}^d$ as a d-element vector; if $\mathcal{O}$ is not the origin then p is the vector $p - \mathcal{O}$. Given two vectors p and x, recall that $p \cdot x$ is the dot-product of p and x. Then the *polar hyperplane* of p is denoted:

$$p^* = \{x \in \mathbb{R}^d, \ p \cdot x = 1\}.$$

Clearly this is linear in the coordinates of x, and so p^* is a hyperplane in $\mathbb{R}^d$. If p is on the unit sphere centered at $\mathcal{O}$, then p^* is a hyperplane that passes through p and is orthogonal to the vector $\overrightarrow{\mathcal{O}p}$.

As p moves away from the origin along this vector, the dual hyperplane move closer to the origin, and vice versa, so that the product of their distances from the origin is always 1. See Figure 8(a).

5.1 Properties

Like with point-line duality, the polar transformation satisfies certain incidence and inclusion properties between points and hyperplanes. For example, let h be any hyperplane that does not contain $\mathcal{O}$. The *polar point* of h, denoted h^* is the point that satisfies $h^* \cdot x = 1$ for all $x \in h$.

Let p be any point in $\mathbb{R}^d$ and let h be any hyperplane in $\mathbb{R}^d$. The polar transformation satisfies the following properties. For a hyperplane h, let h^+ be the halfspace containing the origin and h^- be the other halfspace for h. See Figure 8(b).

- **Incidence Preserving:** Point p belongs to hyperplane h iff point h^* belongs to hyperplane p^*

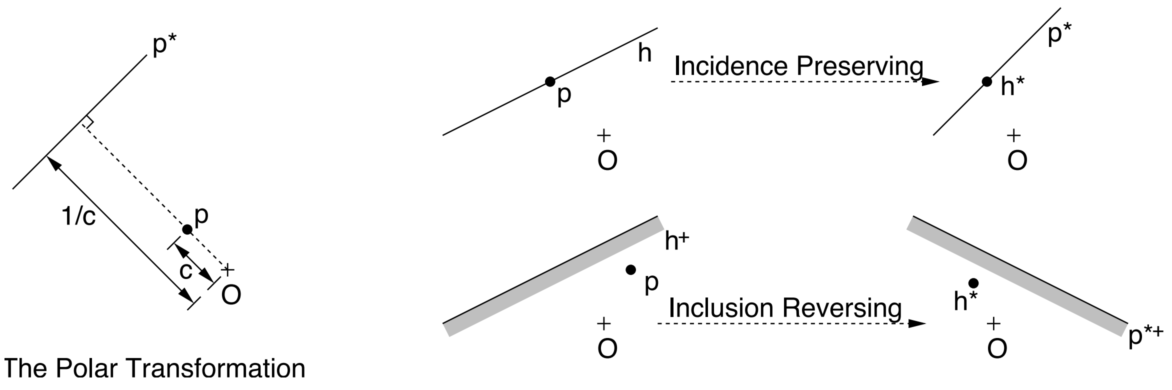

Figure 8. Polar Transform Properties

- **Inclusion Reversing:** Point p belongs to halfspace h^+ iff point h^* belongs to halfspace $(p^*)^+$. This implies that point p belongs to halfspace h^- iff point h^* belongs to halfspace $(p^*)^-$. Intuitively, the polarity transform reverses relative positions.

A bijective transformation that preserves incidence relations is called a *duality.* So the above claim shows that the polarity transform is another dualtiy.

6 Convex Hulls and Halfspace Intersection

Many of the proofs in this section are based on a writeup by Dave Broaddus. We can transform a polytope defined as the convex hull of a finite set of points to a polytope defined as the intersection of a finite set of closed halfspaces.

To do this, we need a mapping from a point to a *halfspace.* For any point $p \in \mathbb{R}^d$, define

$$p^\# = \overline{(p^*)^-} = \{x \in \mathbb{R}^d \mid x \cdot p \leq 1\}$$

This just first finds the polar hyperplane of p, and then takes the *closed* halfspace containing the origin.

Now for any set of points $P \subseteq \mathbb{R}^d$, define its *polar image* to be the intersection of these halfspaces.

$$P^\# = \{x \in \mathbb{R}^d \mid x \cdot p \leq 1, \forall p \in P\}$$

Thus, $P^\#$ is the intersection of a finite set of closed halfspaces, one for each $p \in P$. Is $P^\#$ convex? Yes, since each halfspace is convex, and the intersection of any set of convex spaces is convex. The following lemma shows that P and $P^\#$ are essentially equivalent via polarity.

6.1 Some observations

Lemma 4. *Let S be any set of points and h be any hyperplane, both S and h not containing $\mathcal{O}$. Then $S \subseteq \overline{h^+}$ iff $h^* \in S^\#$.*

Proof: Consider any point $p \in S$. By incidence-preserving and inclusion-reversing, $p \in \overline{h^+}$ iff $h^* \in p^\#$. $\square$

Define supp($S^\#$) to be the set of *bounding* halfspaces among all the halfspaces in $S^\#$.

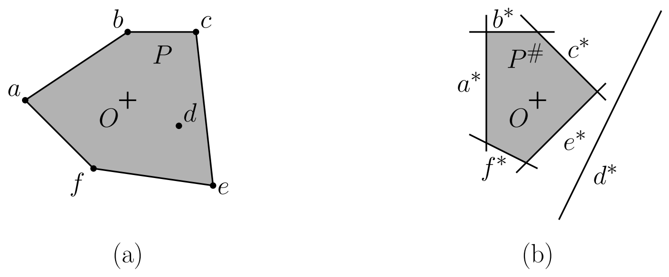

Figure 9.

Lemma 5. *Let S be any set of points not containing $\mathcal{O}$ and v be any point in S. Then $v^{\#} \in \text{supp}(S^{\#})$ iff there exists some point p satisfying*

1. $p \in v^*$
2. $p \in S^{\#}$

In the following, for a set of points S, let $\text{CH}(S)$ be the vertices of $\text{conv}\, S$.

Lemma 6. *Let S be any set of points not containing $\mathcal{O}$, but where $\text{conv}(S)$ contains $\mathcal{O}$. Then $v \in CH(S)$ iff there exists some hyperplane h such that*

1. $v \in h$
2. $S \subseteq \overline{h^+}$

Since $\mathcal{O} \in \text{conv}(S)$, $\overline{h^+}$ contains $\mathcal{O}$.

6.2 Main Proof

Lemma 7. *Let S be a set of points not containing $\mathcal{O}$, such that $P :- \text{conv}(S)$ contains $\mathcal{O}$. Then:*

1. $v \in CH(S)$ *iff* $v^{\#} \in \text{supp}(S^{\#})$
2. *For all $k \in [1, d-1]$, each k-face of P corresponds to a $(d-1-k)$-face of $P^{\#}$*

Proof: We first show that $v \in \text{CH}(S)$ implies that $v^{\#} \in \text{supp}(S^{\#})$. By Lemma 6, $v \in \text{CH}(S)$ iff there exists a hyperplane h such that $v \in h$ and $S \subseteq \overline{h^+}$. In the dual space, this requirement corresponds to a point h^* in $v*$ (by incidence preserving) that satisfies $h^* \in S^{\#}$ (by Lemma 4). Then, by Lemma 5, $v^{\#} \in \text{supp}(S^{\#})$. Showing that $v^{\#} \in \text{supp}(S^{\#})$ implies $v \in \text{CH}(S)$ just requires following the iff arrows in the reverse direction. To show part 2 of the lemma, suppose $k \in [1, d-1]]$, and consider some k-face of P that is supported by $k+1$ vertices $v_1, \ldots, v_{k+1}$. Then by, part 1 of the lemma, hyperplanes $v_1^*, \ldots, v_{k+1}^*$ are all supported by facets of $P^{\#}$. Hence, the intersection of these $k+1$ hyperplanes forma a $d-(k+1)$-face of the the polytope $P^{\#}$. $\square$

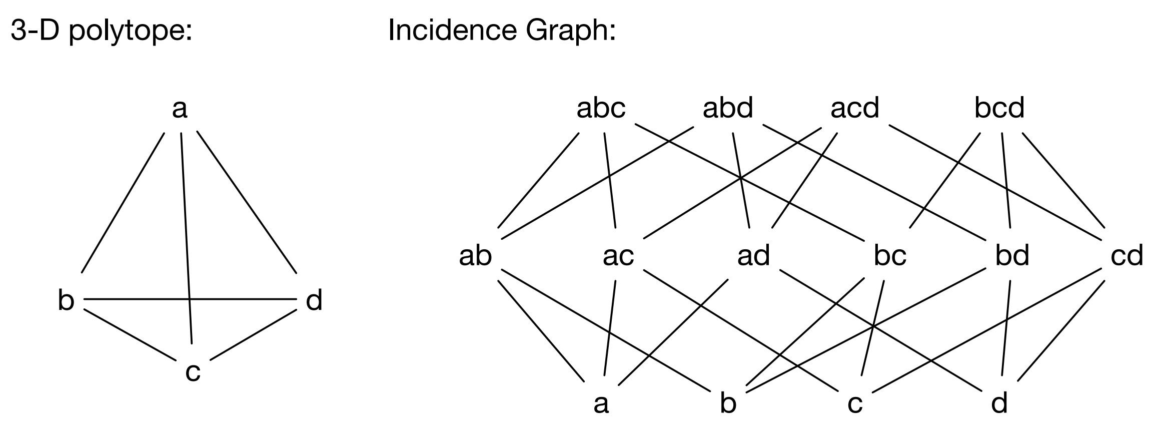

Figure 10. Left: Polytope; Right: Incidence graph for all faces

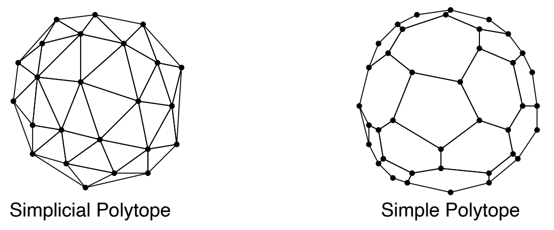

Figure 11. Simplicial and Simple Polytopes

6.3 Some Observations

Incidence Graphs. Figure 10 illustrates an incidence graph for a simplex over 4 vertices in 3 dimensions. Each vertex in the top row is a 3-face (facet), defined by 3 of the 4 vertices. Each vertex in the next row is a 2-face, defined by 2 of the 4 vertices. Each vertex in the bottom row is a 1-face (i.e. point), defined by one vertex. An edge in the incidence graph connects two faces if one of the faces is included in the other.

Two observations. First, the incidence graph of the simplex in the polar plane, can be read bottom up by just taking the polar halfplane v^* for each vertex v in the incidence graph, and thinking of each face as the intersection of a collection of these halfplanes. Second, for a simplex, there are exactly $d+1$ facets. But for an arbitrary polytope that is the convex hull of n points, there may be many more facets.

Simple and Simplicial Polytopes. If a polytope is the convex hull of a set of points in $\mathbb{R}^d$ in general position, then for all $0 \leq j \leq d-1$, each j-face is a j-simplex. Such a polytope is called *simplicial* (see Figure 11.)

In the dual view, consider a polytope that is the intersection of n halfspaces in general position. Each j-face for $0 \leq j \leq d-1$ is the intersection of exactly $d-j$ hyperplanes. Such a polytope is said to be *simple*. In simple polytopes, each vertex is incident to exactly d facets. Thus, the local region around any vertex is equivalent to a simplex.

Among all polytopes with a fixed number of vertices, simplicial polytopes maximize the number of facets. To see this, note that if there is a degeneracy (i.e. d+1 points on one facet), perturbing some point on this facet will break it into multiple facets. Dually, among all polytopes with a fixed number of facets, simple polytopes maximize the number of vertices.

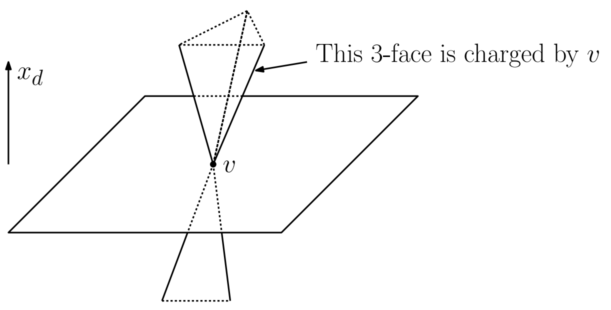

Figure 12.

7 Bounding Facets and Vertices

So, how many facets are in a convex hull defined by n points in d dimensions? Based on the polar duality result of Lemma 7, this question is equivalent to the question: How many vertices are in the convex polytope that is the intersection of n halfspaces in d dimensions.

The following beautiful theorem (also due to Seidel) resolves both questions.

Theorem 1. *A polytope in $\mathbb{R}^d$ that is the convex hull of n points has $O(n^{\lfloor d/2 \rfloor})$ facets. A polytope in $\mathbb{R}^d$ that is the intersection of n halfspaces has $O(n^{\lfloor d/2 \rfloor})$ vertices.*

Proof: We will prove the polar form of the theorem. Consider a polytope defined by intersection of n halfspaces in general position. By the discussion in the last section, this gives rise to a simple polytope. Suppose by convention that x_d is the vertical axis. Then given a face, its *highest* and *lowest* vertices are defined as those having the maximum and minimum x_d coordinates, respectively. Assuming symbolic perturbation, there will be no ties. Our proof is based on a charging argument. We start with a charge at each vertex.

Consider some vertex v. Note that there are d edges (1-faces) that are incident to v (See Figure 12 for example in $\mathbb{R}^5$). Consider a horizontal, i.e. orthogonal to x_d, hyperplane that passes through v. Since no two points have the same x_d coordinate, at least $\lceil d/2 \rceil$ of the edges must lie on the same side of this hyperplane.

Hence, there is a face of dimension at least $\lceil d/2 \rceil$ that spans these edges and is incident to v (e.g. the 3-face above v in Figure 12). So v is either the highest or lowest vertex on this face. We assign v's charge to this face. Thus, we charge every vertex to a face of dimension at least $\lceil d/2 \rceil$, and every such face will be charged at most twice.

So how many charges are there in total? The number of j faces is at most $\binom{n}{d-j}$, since each j face is the intersection of $d-j$ halfspaces. Thus, the total number of charges is at most:

$$\begin{aligned} 2\sum_{j=\lceil d/2 \rceil}^{d-1} \binom{n}{d-j} &= 2\sum_{i=1}^{\lfloor d/2 \rfloor} \binom{n}{i} \\ &\leq 2\sum_{i=1}^{\lfloor d/2 \rfloor} n^i \\ &= O(n^{\lfloor d/2 \rfloor}) \end{aligned}$$

The second step holds since $\binom{n}{x} \leq n^x$. The last step holds, since for $n \geq 2$, the sum is geometric and so equals a constant (namely $n/(n-1)$) times its largest summand. □

Is this bound tight? Yes. There is a family of polytopes called cyclic polytopes which match this asymptotic bound.

References

[1] David Mount. Computational Geometry. http://www.cs.umd.edu/class/fall2016/cmsc754/Lects/cmsc754-fall16-lects.pdf, 2016.