Small-dimensional Linear Programming and Convex Hulls Made Easy

Raimund Seidel

Teng Liu

April 3rd 2025

Background

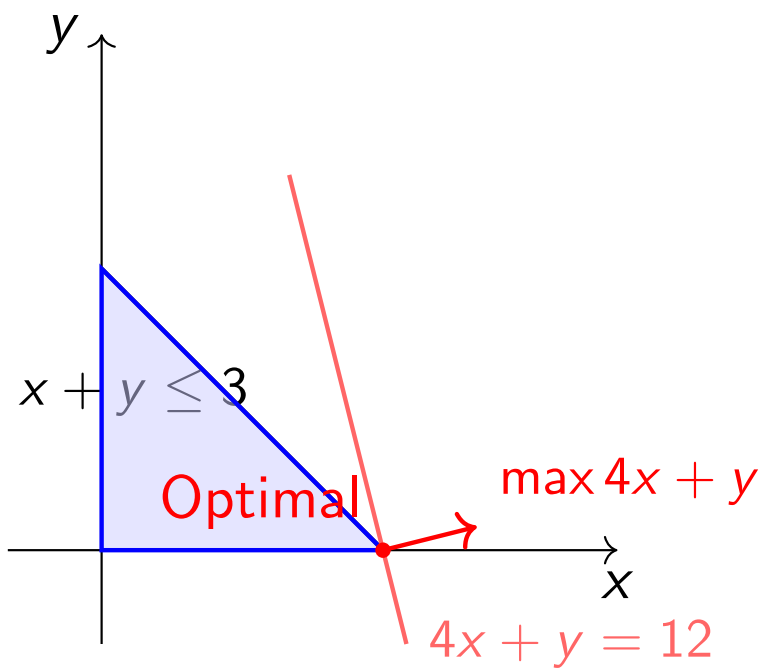
Linear programming captures one of the most canonical and influential constrained optimization problems.

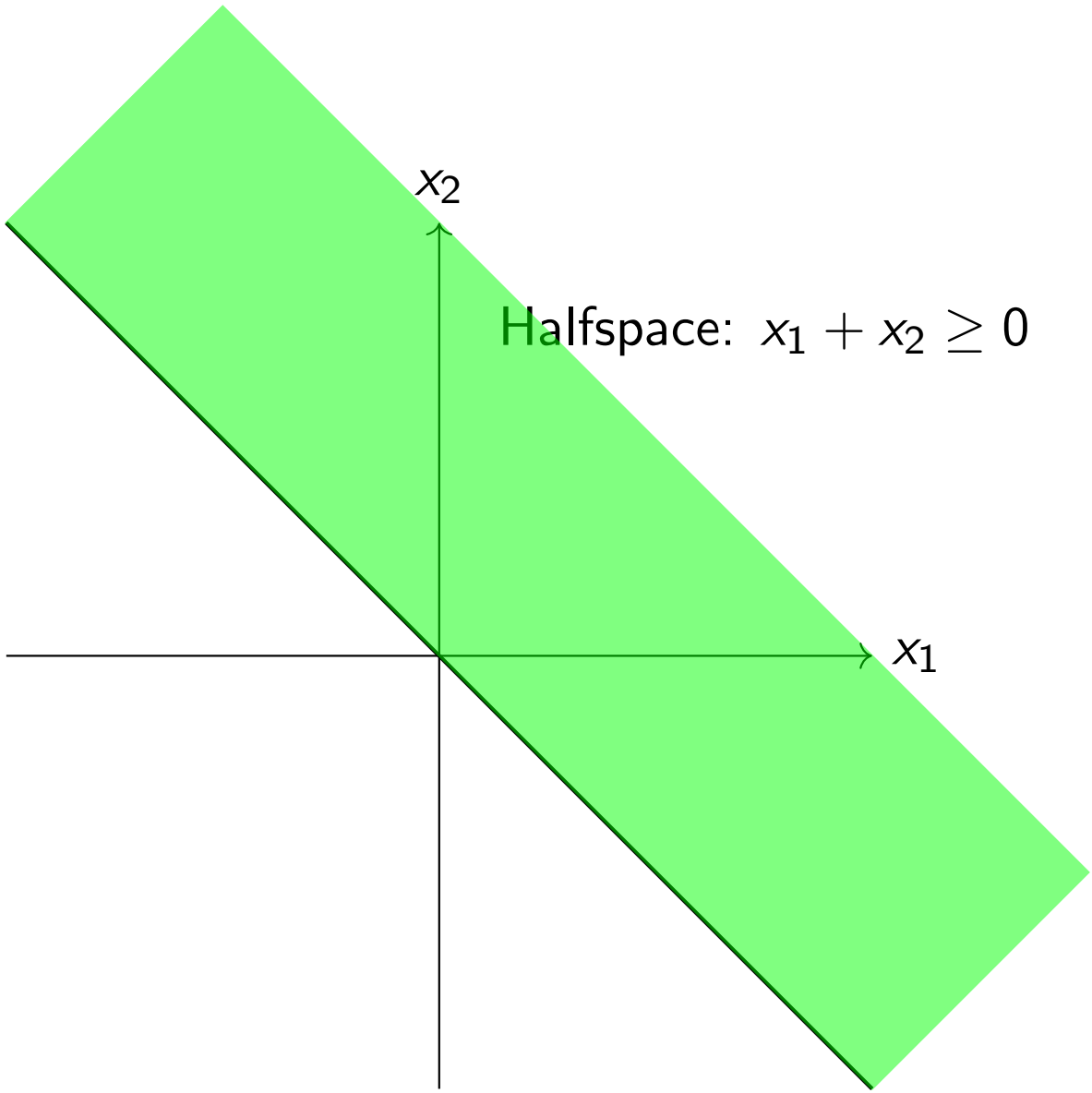
More precisely, it asks to maximize or minimize a linear objective under linear inequality and equality constraints.

$$\begin{aligned} \max / \min \ & c^T x \\ \text{subject to} \quad & Ax \leq e \\ & Bx \geq f \\ & Cx = g \end{aligned}$$

Background - Geometrical Interpretation

$$\begin{aligned} \max \ & 4x + y \\ \text{subject to} \quad & x + y \leq 3 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$





Hyperplane: $x_1 + x_2 = 0$

Background - Facts

Fact

an LP can be bounded, unbounded and infeasible.

Fact

Optimal solution, if exists, can always be achieved by some vertex.

Fact

If there exists any vertex with better objective value than current vertex, we always have a neighbour vertex with better value.

Background - Facts

Fact

an LP can be bounded, unbounded and infeasible.

Fact

Optimal solution, if exists, can always be achieved by some vertex.

Fact

If there exists any vertex with better objective value than current vertex, we always have a neighbour vertex with better value.

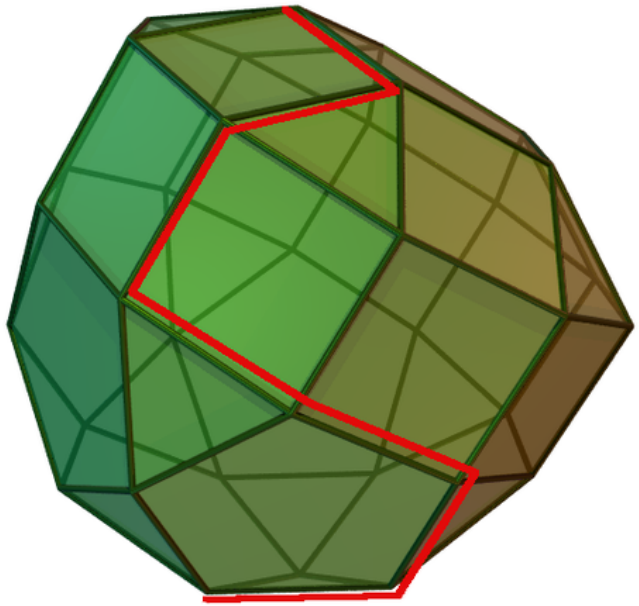
Firstly select any starting vertex.

Each step in Simplex Method move from one vertex to its neighbour with larger objective.

Background - Simplex Method

Firstly select any starting vertex.

Each step in Simplex Method move from one vertex to its neighbour with larger objective.



Fault-tolerant quantum algorithms - Scientific Figure on ResearchGate.

Background - Overview

Algorithm	**Complexity**	**Speed**	**Approach**
Simplex	Exponential	Fast	Edges of polyhedron
Ellipsoid	Polynomial	Slow	Ellipsoid shrinking
Interior Point	Polynomial	Fast	Through the interior

Table: Comparison of Famous LP Algorithms

Previous Work

When d, the dimension of LP (i.e. the number of variables), is considered to be constant, Megiddo showed that LP can be solved in time linear in m, the number of constraint.

Megiddo's algorithm[2] are complicated and the running time is doubly exponential, while Clarkon and Dyer improved to superexponential to d: e.g. 3^{d^2}.

Clarkon also proposed an algorithm[1] with complexity $O(d^2 m) + (\log m) O(d)^{d/2+O(1)} + O(d^4 \sqrt{m} \log m)$ but the analysis is involved.

The presented algorithm gives a simple procedure and expected complexity $O(d!m)$.

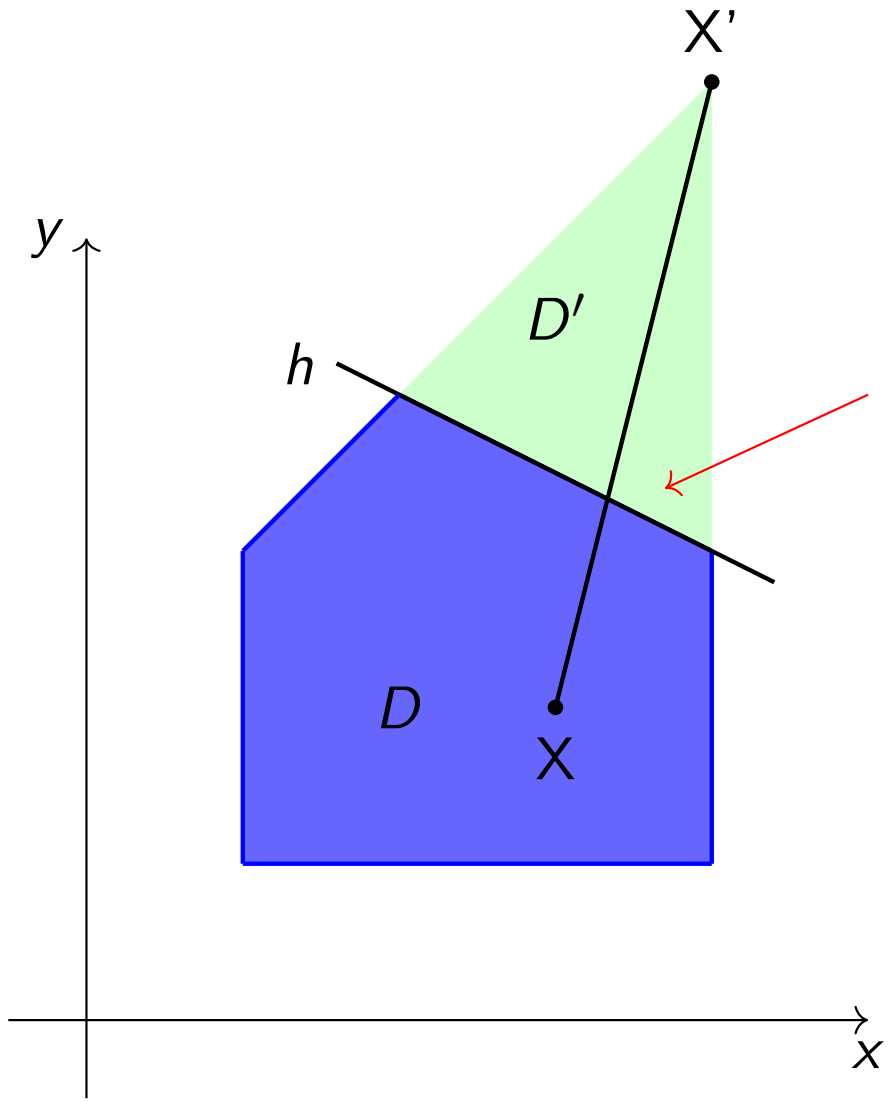
Algorithm - Sketch

This algorithm makes use of a very simple idea:

Suppose we delete one constraint $a^T x \geq b$, and somehow managed to solve the new LP. Then we check if the optimum x^* conforms with the deleted constraint.

- If it doesn't violate the constraint, i.e. $a^T x^* \geq b$, then x^* is also the optimum for original LP, because deleting one constraint gives us a larger feasible area.
- If it violates the constraint, then the opt value lies on the hyperplane corresponding to this constraint. We can project the feasible area onto the corresponding hyperplane, thus reducing dimension by 1. Then solve the new LP problem with lower dimension.

Algorithm - Illustration of Second Case



A better point lies on h

Algorithm - Further Problems

Some problems left to deal with:

- What if after deletion of bounding hyperplane the new problem become unbounded?
- When to stop the recurssion?
- How can we do *projection* efficiency?
- How to deal with infeasibility?

Algorithm - Unboundedness and Recurssion

The first two problems can be solved by adding a bounding box, i.e. explicit lower and upper bounds for variables $-\alpha \leq x_i \leq \alpha$ for all d variables x_i.

Recurssion stops when all original hyperplanes are deleted and it's impossible to have unbounded case.

Algorithm - Projection and Infeasibility

Projection can be done by Gaussian elimination. Since we know a constraint has to be tight, we can get an equation and then we substitute it back into constraints, which gives us an inequality system with $d - 1$ variables.

Infeasibility can then be detected in dimension 1.

Algorithm - Time Complexity

When $d = 1$, problem can be solved in $O(m)$.

When $m = 0$, problem can be solved in $O(d)$.

The interesting part is when we delete a constraint and the optimum point changes. Since original optimum point x has exactly d tight constraints (i.e. x is on exactly d hyperplanes) then deleting a constraint uniformly randomly ensures the probability of x' differs from x no more than d/m.

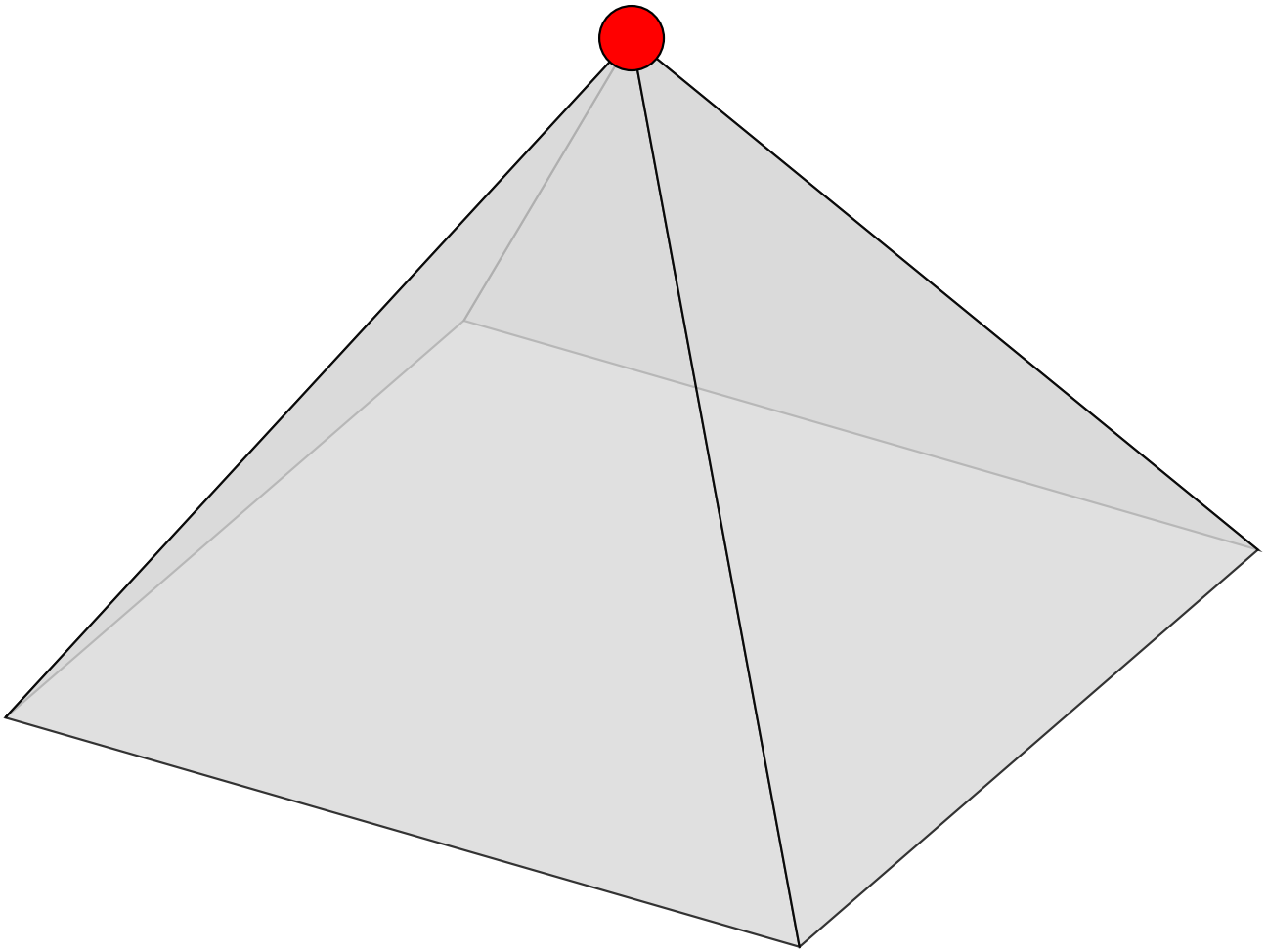
Algorithm - Time Complexity

When $d = 1$, problem can be solved in $O(m)$.

When $m = 0$, problem can be solved in $O(d)$.

The interesting part is when we delete a constraint and the optimum point changes. Since original optimum point x has exactly d tight constraints (i.e. x is on exactly d hyperplanes) then deleting a constraint uniformly randomly ensures the probability of x' differs from x no more than d/m.

Algorithm - Degenerate Vertex



Algorithm - Time Complexity

Even if there are $k > d$ constraints are tight at x, the optimum point for original LP, at most d of them has the property that deleting it leads a better optimum.

Algorithm - Time Complexity

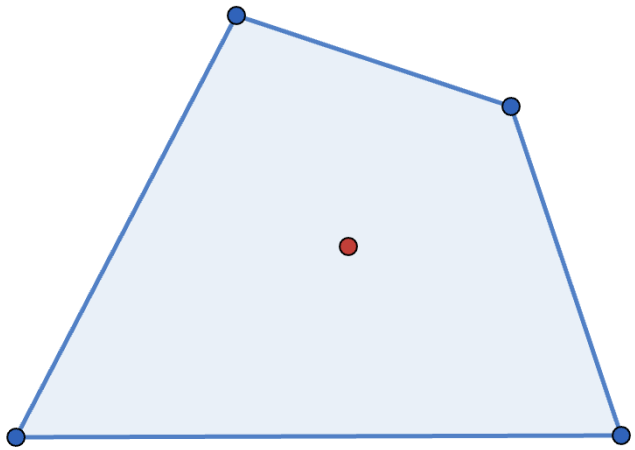
$$T(d,m) \leq \begin{cases} O(m) & \text{if } d = 1, \\ O(d) & \text{if } m = 1, \\ T(d,m-1) + O(d) + \frac{d}{m} O(dm) + & \\ \frac{d}{m} T(d-1,m-1) & \text{otherwise.} \end{cases}$$

Expand everything and we get:

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

Convex Hull

For a point set S on $\mathbb{R}^d$, assuming general position (no $d+1$ points lie in a common hyperplane), we consider construction of convex hull P of point set S.

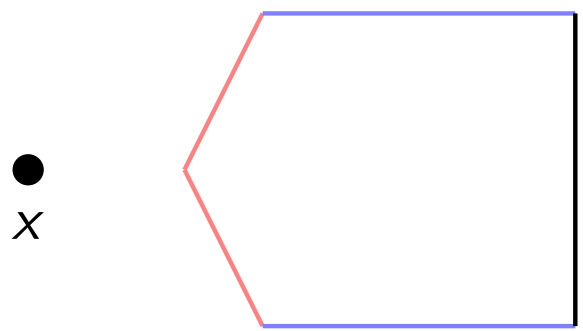


Terminology

Let p be some point in $\mathbb{R}^d$ in general position with respect to S.

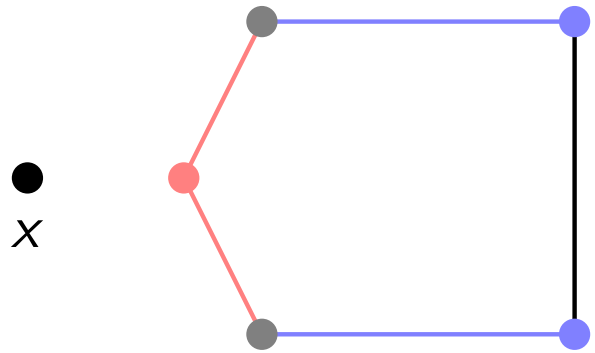
We call a facet F of P *visible* from p iff the hyperplane spanned by F seperates P and p; a facet *obscured* otherwise.

We call a face G *visible* from p iff it's only containted in facets of P that are visible from p. *Obscured* faces are defined analogously. We call G a face *horizon* iff it's contained in some visible and some obscured facet.



Visible

Obscured



Visible

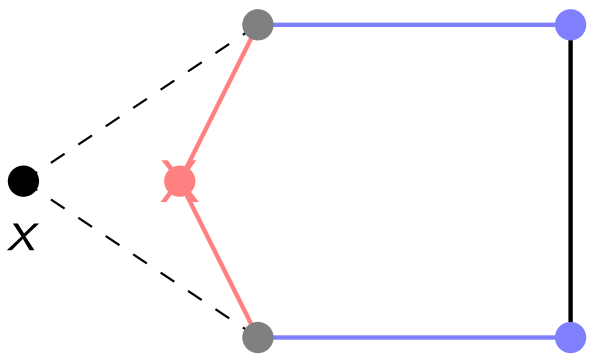
Obscured

Horizon

Convex Hull - Facts

Consider an incremental algorithm, from original convex hull P to construct $P' = conv(P \bigcup x)$ where x is the new point, and we have the following observations:

- no visible face of P is a face of P';
- all obsecured and horizon faces of P are faces of P';
- for each horizon face G of P, the pyramid $conv(G \bigcup x)$ is a face of P';
- this yields all faces of P'.



Visible

Obscured

Horizon

New Facet

Convex Hull Sketch

1. Locate a facet F of P visible from x. If no such F exists then $P' = P$.
2. Determine set of facets and ridges of P visible from x and horizon ridges. Delete all visible facets and ridges.
3. For each horizon ridge G, generate new facet $conv(G \bigcup \{x\})$ (a new node for facet graph).
4. Generate new edges (i.e. ridges of P') for new node in facet graph.

Convex Hull - Step 1

Step 1: Locate a facet F of P visible from x. If no such F exists then $P' = P$.

This can be done by linear programming with $d + 1$ variables indicating parameters of hyperplane and each vertex gives a seperating constraint (the target hyperplane should seperate all vertices and x). The objective function is the max distance of vertices and target hyperplane.

If such hyperplane exists, it must touch some visible vertex v. Then we iterate all facets containing v, among which we can find a seperating facet.

According to the LP algorithm, **this can be done in $O(i)$ time in i-th iteration**.

Convex Hull - Step 2

Step 2: Find all visible facets and ridges, then delete them.

This can be done by Depth-First-Search from the facet determined in step 1. **The time cost is proportional to number of visible facets**, which can be accounted into the creation step of these facets, amortizedly speaking.

Convex Hull - Time Complexity

Step 3: Generate new facet for each horizon ridge.

All horizon ridges can found in step 2 and adding new nodes in facet graph **cost clearly time proportional to number of new facets** N_i.

Convex Hull - Time Complexity

Step 4: Generate new edges in facet graph.

Here we need to pair up facets (i.e. nodes in facet graph) to add new edges. For each new facet, we can iterate all ridges (represented as $d-1$-tuple of vertices index). Then by Radix Sort or Trie Tree we can simply identify the corresponding old facet of each ridge.

This can be done in $O(i+N_i)$ time.

Convex Hull - Time Complexity

Put points of S in random order $p_1, \cdots, p_n$.

For i-th iteration, we compute P_i out of P_{i-1} and the running time is $O(i + N_i)$, where N_i is the number of of facets constaining p_i in P_i.

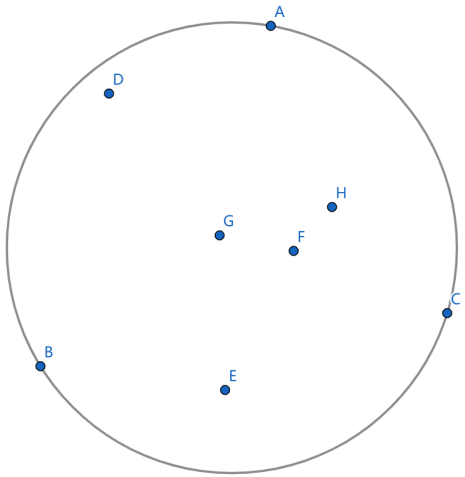
There are at most $i^{\lfloor \frac{d}{2} \rfloor}$ facets and the probability that a vertex is contained in a facet is d/i. So the expected running time for this iteration is $i^{\lfloor \frac{d}{2} \rfloor - 1}$, leading to total complexity of $n^{\lfloor \frac{d}{2} \rfloor}$.
Notice this procedure doesn't work when $d \leq 3$.

Smallest Enclosing Circle

An algorithm inspired by this paper is proposed by Emo Welzl[3].

Smallest Enclosing Circle

Given a set of points P in Euclidean plane, compute the smallest circle that contains them all.



Smallest Enclosing Circle

An efficient random algorithm with surprisingly simple procedure:

```
def Smallest_Enclosing_Circle(P, R):
    ```input : P and R are finite sets of points in the plane, |R| <= 3
    ```output: Smallest circle enclosing P with R on the boundary

    if P is empty or |R| == 3 then:
        return trivial(R)

    p = random_select(P)
    D = Smallest_Enclosing_Circle(P - {p}, R)
    if p in D then:
        return D
    else:
        return Smallest_Enclosing_Circle(P - {p}, R + {p})
```

References I

Kenneth L. Clarkson.
Las vegas algorithms for linear and integer programming when the dimension is small.
J. ACM, 42(2):488–499, March 1995.

Nimrod Megiddo.
Linear programming in linear time when the dimension is fixed.

J. ACM, 31(1):114–127, January 1984.

Emo Welzl.
Smallest enclosing disks (balls and ellipsoids).
In Hermann Maurer, editor, *New Results and New Trends in Computer Science*, pages 359–370, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.