6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms December 1, 2008

Lecture 22

Lecturer: Michel X. Goemans

In this lecture, we introduce Seidel's algorithm [3] to solve linear programs with n constraints in dimension d, when the dimension is small. The expected running time of Seidel's algorithm is $O(d!n)$, i.e. it is strongly polynomial for fixed dimension d (*strongly*, since it does not depend on the size of the input coefficients). Then, we use Seidel's algorithm to develop a randomized convex-hull algorithm in an arbitrary dimension d which is the best possible when $d \geq 4$.

1 Linear Programming in Fixed Dimension

In this section, we fix the dimension d. We wish to find a strongly-polynomial time algorithm to solve linear programming.

1.1 Seidel's Algorithm

Let H be a set of n inequalities. Each inequality corresponds to a half-space h determined by a hyperplane. Let $LP(H)$ be the linear program that minimizes $c^T x$ subject to the constraints:

$$x \in \bigcap_{h \in H} h, \quad x \in \mathbb{R}^d.$$

To make the description of the algorithm simpler, we make the following two assumptions:

1. Bounded: the feasible region is bounded, i.e. there exists M such that, for any feasible x, $-M \leq x_i \leq M$ for all $i = 1, 2, \ldots, d$.

   This assumption can be enforced by ficticiously imposing a large bounding box, and whenever one of the inequalities of this bounding box is tight at optimum, we know that the linear program is unbounded.

2. Non-degenerate: the intersection of any $d+1$ hyperplanes is empty. In 2-D, non-degeneracy means that there do not exist three lines meeting at the same point.

   If H does not meet this assumption, we can use some standard tricks like perturbation to make it non-degenerate. This can be handled by doing so-called lexicographic perturbation.

These two assumptions imply that for any $H' \subseteq H$, $LP(H')$ has a unique solution $x(H')$. Seidel's algorithm will actually apply to a more general class of problems than linear programming, but here we'll focus on linear programming. What is actually needed in the generalization is that the unique solution $x(H')$ is defined by a basis:

Definition 1 *A subset $B \subseteq H'$ is called a basis of the linear program $LP(H')$ if $x(B) = x(H')$ and B is minimal.*

Seidel's algorithm solves the linear program H incrementally as follows. Chooes uniformly $h \in H$. Solve the linear program with h removed, and get a solution x. If the solution x satisfies h, then return x. If the solution x does not satisfy h, we impose the condition that h is satisfied at equality, and eliminate one variable. Then solve the linear program with $d-1$ variables and $n-1$ inequalities. The correctness of this algorithm was proved in the last lecture.

In Seidel's algorithm, we can stop the recursion when we have either n constraints in $d = 1$ variable (which takes $O(n)$ time to solve), or 1 constraint in d variables (which takes $O(d)$ time to optimize over our ficticious bounding box).

1.2 Analysis of Running Time

Let $T(d, n)$ be the expected running time of Seidel's algorithm on an instance with n inequalities and d variables. To find a recursive relation for $T(d, n)$, note that we first recursively solve an LP with $n-1$ inequalities and d variables, which takes time $T(d, n-1)$. If the solution x satisfies the removed constraint h (which takes $O(d)$ time to check), we are done and simply return the d coordinates of x. If x does not satisfy h, we first reduce the LP to only $d-1$ variables in $O(dn)$ time (it takes $O(d)$ time to eliminate one variable in each constraint) using the constraint h, and then solve the LP with $n-1$ inequalities and $d-1$ variables in $T(d-1, n-1)$ time. The probability that x does not satisfy h is d/n, since the optimal solution is determined by exactly d inequalities and we have selected an inequality uniformly at random. This is the important step in the analysis and is known as *backward analysis*.

By the analysis above, we have

$$\begin{aligned} T(d, n) &= T(d, n-1) + O(d) + \frac{d}{n}\left(O(dn) + T(d-1, n-1)\right) \\ &= T(d, n-1) + \frac{d}{n} T(d-1, n-1) + O(d^2). \end{aligned}$$

The base cases are $T(1, n) = O(n)$ and $T(d, 1) = O(d)$.

Using this recursive relation, we can prove by induction on $d + n$ that

Claim 1

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

Proof: The base case is satisfied. We need to check the induction step. Suppose that

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

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

Since

$$\sum_{1 \le i \le d} \frac{i^2}{i!} \cdot d!(n-1) + \frac{d}{n} \sum_{1 \le i \le d-1} \frac{i^2}{i!} \cdot (d-1)!(n-1) + d^2 \le \left(\sum_{1 \le i \le d} \frac{i^2}{i!}\right) d! n,$$

the claim also holds for $T(d, n)$.

The second equality in the claim follows from the fact that $\sum_{i=1}^{\infty} \frac{i^2}{i!}$ is finite. □

Thus, we have shown a strongly polynomial time algorithm to solve linear programs in a fixed small dimension d.

1.3 Improvement (Matousek, Sharir, Welzl [2])

Although the expected running time of Seidel's algorithm is strongly-polynomial in n, it increases exponentially when d increases (more precisely, the dependence on d is $2^{O(d \log d)}$). In this subsection, we briefly introduce an improvement to Seidel's algorithm which gives a subexponential bound in d.

We consider the linear program as follows. The LP algorithm $LP(H, C)$ takes as input a candidate set C (that plays the role of a basis), and returns x as well as a basis B. Initially, we call $LP(H, C)$ with $C = \emptyset$.

The algorithm proceeds as follows. If $H = C$, then return C. If $H \neq C$, choose h randomly among $H - C$. We recursively call $LP(H - \{h\}, C)$ and get a basis B. If h is satisfied by the solution defined by B, then return B. Otherwise, we call $LP(H, \text{basis}(B, h))$, where $\text{basis}(B, h)$ denotes an optimal basis for $LP(B \bigcup \{h\})$.

Claim 2 *The expected running time is*

$$O\left(e^{2\sqrt{d \log(n/\sqrt{d})} + O(\sqrt{d}) + O(\log n)}\right).$$

When d is fixed, the running time is a polynomial of n. When n is fixed, the running time is $O(e^{\sqrt{d}})$, subexponential in d.

Use a trick due to Clarkson (through random sampling), one can show that linear programs with n inequalities in d dimensions can be solved in

$$O(d^2 n + e^{\sqrt{d \log d}})$$

time. This is the best bound currently known that is independent on the size of entries. See Goldwasser [1] for a discussion.

2 Convex Hull

Given n points $x_1, \ldots, x_n \in \mathbb{R}^d$. Let P be the convex hull of $x_1, \ldots, x_n$. For $d = 2$ and $d = 3$, P can be found in $O(n \log n)$ time. In the previous lecture, we showed several algorithms that solve 2-dimensional convex hull in $O(n \log n)$ time.

Throughout this section, we assume that the points $x_1, \ldots, x_n$ are in general position, meaning that any $d+1$ points do not lie on the same hyperplane. If that's not the case, a standard perturbation argument can be used.

2.1 Outputs of Convex Hull Algorithms

In dimension 2, it is sufficient to output the vertices of the convex hull in counterclockwise order. In this subsection, we introduce what the output is for a general d.

Definition 2 *For any $0 \leq k < d$, a k-face of a d-dimensional 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 called an edge. A 0-face is called a vertex.*

Definition 3 *A simplicial polytope is a polytope where every face is a simplex.*

Since the points $x_1, \ldots, x_n$ are in general position, the convex hull P is a simplicial polytope.

The convex hull algorithm outputs a facet graph $\mathcal{F}(P)$. The vertices of $\mathcal{F}(P)$ are all facets of P. The edges of $\mathcal{F}(P)$ correspond to the ridges of P, connecting two facets shared by the ridge (Figure 1).

For general d, one can show that the number of facets of P is $O(n^{\lfloor d/2 \rfloor})$. Since the convex hull algorithm needs to output all the facets of P, the running time of any such algorithm is at least $\Omega(n^{\lfloor d/2 \rfloor})$.

2.2 Convex Hull Algorithms

Clarkson and Shor '89 developed a randomized algorithm to compute convex hull in $O(n \log n + n^{\lfloor d/2 \rfloor})$ expected time. Chazelle '93 developed a deterministic algorithm in $O(n \log n + n^{\lfloor d/2 \rfloor})$ time. These algorithms are optimal by the analysis in the previous subsection.

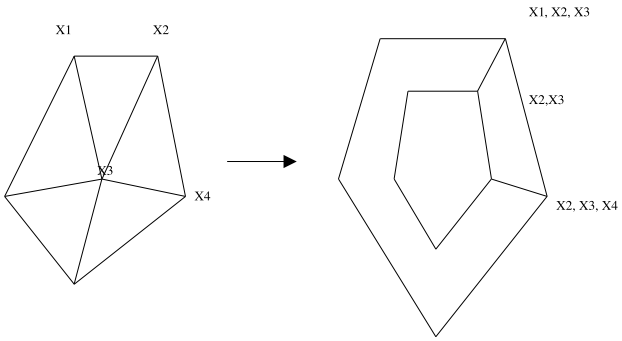

Figure 1: The figure on the left is part of a 3-dimensional simplicial polytope with four vertices labeled x_1, x_2, x_3, x_4. On the right is the corresponding facet graph, where the faces $x_1x_2x_3$, $x_2x_3x_4$, and the edge x_2x_3 are labeled.

We will illustrate Seidel's algorithm [3], which has running time $O(n^2 + n^{\lfloor d/2 \rfloor})$. For $d = 2$ and $d = 3$, Seidel's algorithm takes time $O(n^2)$, which is not optimal. But for larger d, Seidel's algorithm is optimal, and is considerably simpler.

We take a random permutation $x_1, x_2, \ldots, x_n$ of the points. Let P_i be the convex hull of the points $x_1, \ldots, x_i$.

Initially $P_{d+1} = \text{conv}(x_1, \ldots, x_{d+1})$ is a d-dimensional simplex. $\mathcal{F}(P_{d+1})$ is the complete graph on $d+1$ points. We incrementally compute $P_{d+2}, \ldots, P_n$. To do this, we need the following definitions.

Definition 4 *A facet F of a polytope P is visible from a point x_i if the supporting hyperplane of F seperates x_i from P. Otherwise, F is called obscured.*

Definition 5 *A ridge of a polytope P is called visible from a point x_i if both facets it connects are visible, and obscured if both facets are obscured. A ridge is called a horizon ridge if one of the facets it connects is visible and the other is obscured.*

To compute the convex hull P_i when adding a new point x_i, Seidel's algorithm performs the following four steps.

Step 1 Find one visible facet F if one exists. If there is no visible facet, we are done. This step can be done using linear programming in $O(d!i)$ time. Indeed we would like to find a hyperplane $a^T x \leq b$ (where the unknowns are $a \in \mathbb{R}^d$ and b) such that $a^T x_i = b$ and $a^T x_i \leq b$ for $j = 1, \cdots, i-1$. Any extreme solution will correspond to a new facet and to a horizon ridge. One of the two facets indicident to this horizon ridge is visible.

Step 2 Find all visible facets. Determine all horizon ridges. Delete all visible facets and all visible ridges.

This can be done by depth-first-search (DFS), since the visible facets and invisible facets are seperated by horizon ridges. In terms of running time, we charge the deletion time of the facets to when the facets were created.

Step 3 Construct all new facets. Each horizon ridge corresponds to a new facet containing the point x_i and the ridge (Figure 4).

Step 4 Each new facet contains d ridges. Generate all these new ridges. Every new ridge R is a sequence of $d-1$ points $a_1 < a_2 < \ldots < a_{d-1}$. Then match corresponding ridges using radix sort to construct the facet graph.

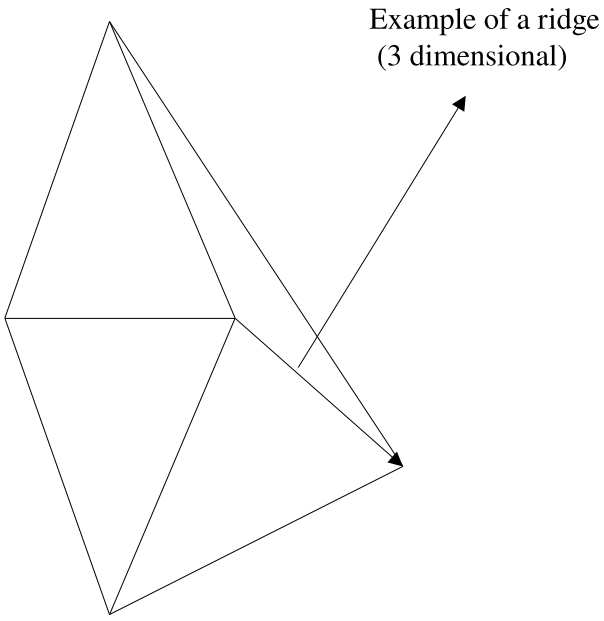

Figure 2: In 3-D, ridges are just edges.

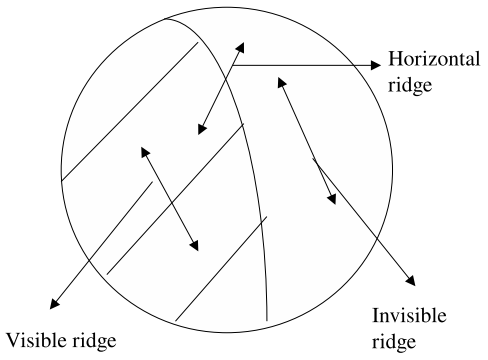

Figure 3: The visible ridges and the invisible ridges are seperated by horizon ridges.

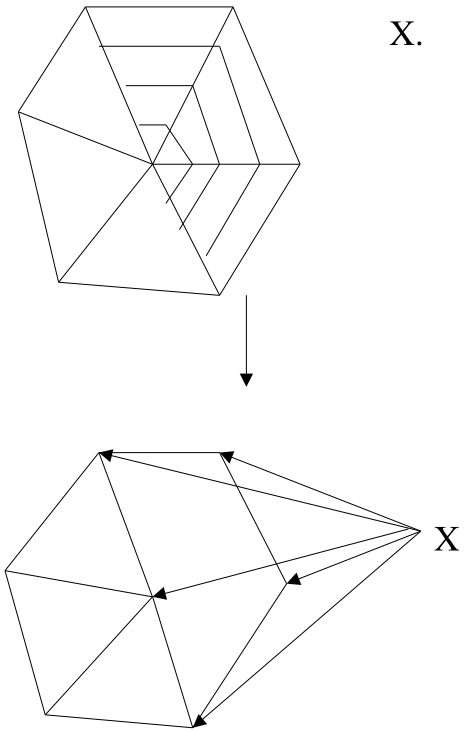

Figure 4: In the figure on the top, the shaded regions are visible facets. In the figure on the bottom, visible facets are removed and new facets are added.

The expected running time of Seidel's algorithm to compute the convex hull is $O(n^2 + n^{\lfloor d/2 \rfloor})$. Indeed the running time is

$$O\left(\sum_{i=d+2}^{n} (i + N_i)\right),$$

where N_i is the number of facets created at step i. One has that

$$E[N_i] = E[\text{facets of } P_i \text{ containing } x_i] \leq \frac{\binom{i-1}{d-1}}{\binom{i}{d}} O(i^{\lfloor d/2 \rfloor}) = \frac{d}{i} O(i^{\lfloor d/2 \rfloor}),$$

giving the required time bound.

References

[1] M. Goldwasser, "A survey of linear programming in randomized subexponential time", *ACM SIGACT News*, **26**, 96–104, 1995.

[2] J. Matousek, M. Sharir, and E. Welzl, "A subexponential bound for linear programming", *Algorithmica*, **16**, 498–516, 1996.

[3] R. Seidel, "Small-dimensional linear programming and convex hulls made easy", *Discrete & Computational Geometry*, **6**, 423–434, 1991.