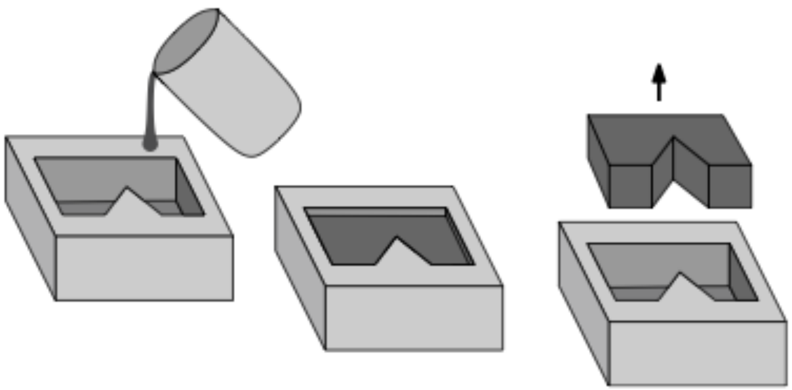
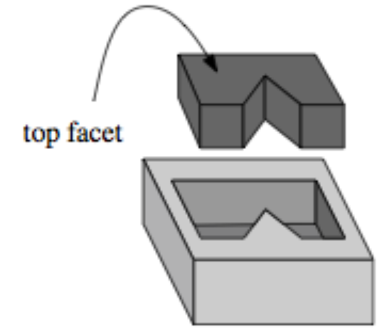
Manufacturing with molds

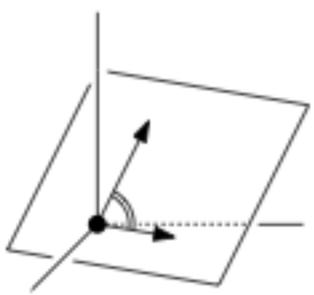


Computational Geometry
4 : Linear Programming
pages 63-93 (not sections 4.5 and 4.6)



The casting process

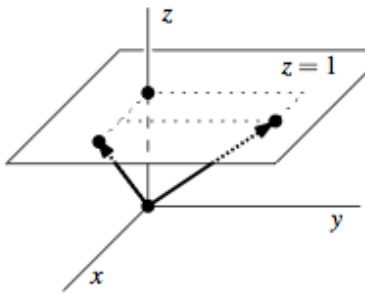
Geometry of casting



The object to be constructed is polyhedral.
Molds of one piece, not molds consisting of two or more pieces.
The object is removed from the mold by a single translation.

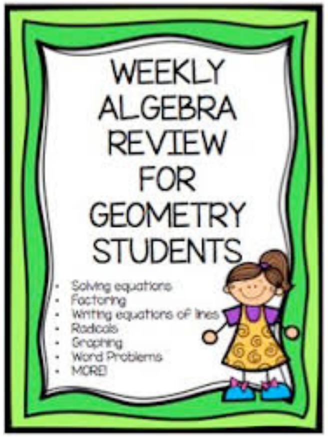
Lemma 4.1 *The polyhedron $\mathcal{P}$ can be removed from its mold by a translation in direction $\vec{d}$ if and only if $\vec{d}$ makes an angle of at least 90° with the outward normal of all ordinary facets of $\mathcal{P}$.*

Finding the translation direction …

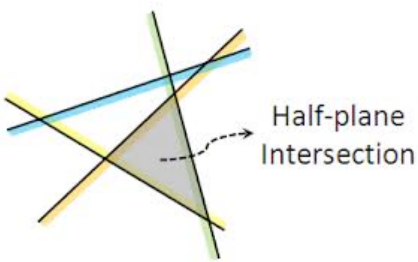


$$\vec{\eta}_x d_x + \vec{\eta}_y d_y + \vec{\eta}_z \leqslant 0.$$

**An ordinary facet induces a constraint.
It is an inequality that describes
a half-plane on the plane z = 1.**



**Given a set of half-planes,
find a point in their intersection
or decide that the intersection is empty.**

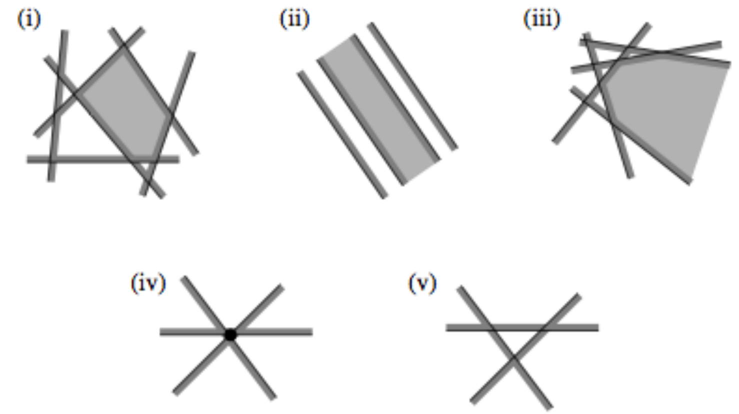


… is a purely geometric problem

Half-planes intersection

Let $H = \{h_1, h_2, \dots, h_n\}$ be a set of linear constraints in two variables, that is, constraints of the form

$$a_i x + b_i y \leqslant c_i,$$



This planar problem can be solved in expected linear time !

What is the meaning of expected ?
If we try all its facets as top facets, we derive the theorem :

Theorem 4.2 *Let $\mathcal{P}$ be a polyhedron with n facets. In $O(n^2)$ expected time and using $O(n)$ storage it can be decided whether $\mathcal{P}$ is castable. Moreover, if $\mathcal{P}$ is castable, a mold and a valid direction for removing $\mathcal{P}$ from it can be computed in the same amount of time.*

Divide and conquer algorithm

Algorithm INTERSECTHALFPLANES(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.
1. **if** card$(H) = 1$
2. **then** $C \leftarrow$ the unique half-plane $h \in H$
3. **else** Split H into sets H_1 and H_2 of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$.
4. $C_1 \leftarrow$INTERSECTHALFPLANES(H_1)
5. $C_2 \leftarrow$INTERSECTHALFPLANES(H_2)
6. $C \leftarrow$INTERSECTCONVEXREGIONS(C_1, C_2)

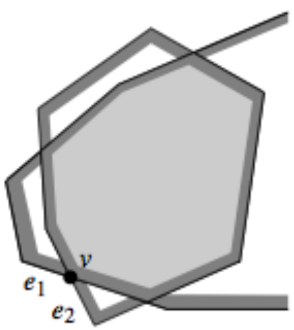
What remains is to describe the final procedure ?
But wait—didn't we see this problem before ?

Indeed, we can compute the intersection of two polygons in O(n log n+ k log n) !
Moreover, k < n !

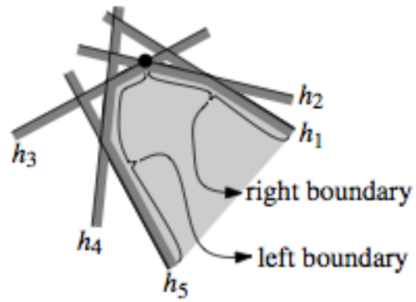
This gives the following recurrence for the total running time:

$$T(n) = \begin{cases} O(1), & \text{if } n = 1, \\ O(n \log n) + 2T(n/2), & \text{if } n > 1. \end{cases}$$

This recurrence solves to $T(n) = O(n \log^2 n)$.

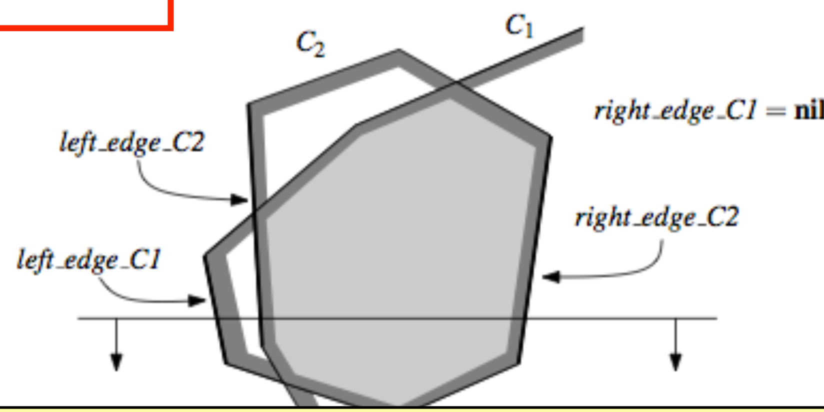


But our polygonal regions are convex !



$\mathcal{L}_{\mathrm{left}}(C) = h_3, h_4, h_5$

$\mathcal{L}_{\mathrm{right}}(C) = h_2, h_1$

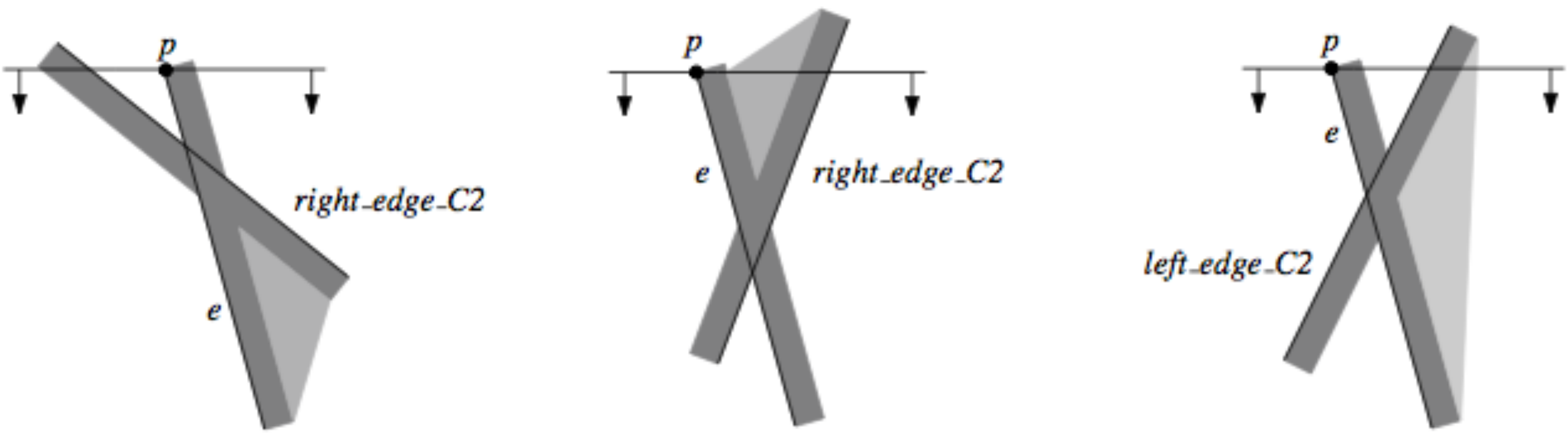


The new algorithm is a plane sweep algorithm:

we move a sweep line downward over the plane,
and we maintain the edges of C1 and C2 intersecting the sweep line.

Since C1 and C2 are convex, there are at most four such edges.
Hence, there is no need to store these edges in a complicated data structure.

Handling an event in the sweep algorithm



Half-planes intersection

This planar problem can be solved in expected linear time !

What is the meaning of expected ?
If we try all its facets as top facets, we derive the theorem :

Theorem 4.3 *The intersection of two convex polygonal regions in the plane can be computed in $O(n)$ time.*

This theorem shows that we can do the merge step in INTERSECTHALF-PLANES in linear time. Hence, the recurrence for the running time of the algorithm becomes

$$T(n) = \begin{cases} O(1), & \text{if } n = 1, \\ O(n) + 2T(n/2), & \text{if } n > 1, \end{cases}$$

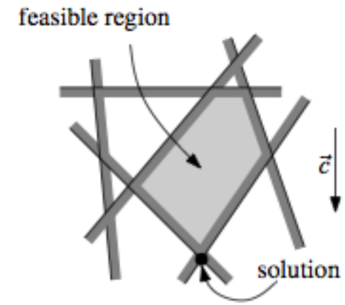
leading to the following result:

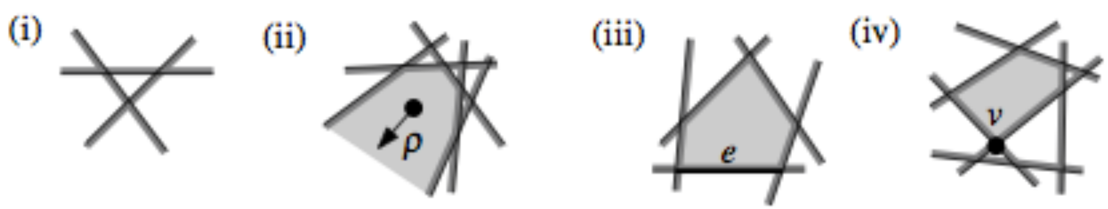
Corollary 4.4 *The common intersection of a set of n half-planes in the plane can be computed in $O(n \log n)$ time and linear storage.*

Incremental linear programming

Maximize $c_1x_1 + c_2x_2 + \cdots + c_dx_d$

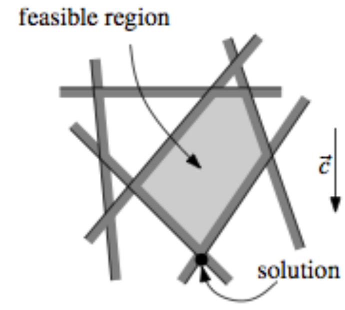
Subject to
$$\begin{aligned}
a_{1,1}x_1 + \cdots + a_{1,d}x_d &\leqslant b_1\\
a_{2,1}x_1 + \cdots + a_{2,d}x_d &\leqslant b_2\\
&\vdots\\
a_{n,1}x_1 + \cdots + a_{n,d}x_d &\leqslant b_n
\end{aligned}$$





Incremental bounded linear programming

$$\begin{array}{ll} \text{Maximize} & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\ \text{Subject to} & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leqslant b_1 \\ & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leqslant b_2 \\ & \vdots \\ & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leqslant b_n \end{array}$$



$$m_1 := \begin{cases} p_x \leqslant M & \text{if } c_x > 0 \\ -p_x \leqslant M & \text{otherwise} \end{cases}$$

$$m_2 := \begin{cases} p_y \leqslant M & \text{if } c_y > 0 \\ -p_y \leqslant M & \text{otherwise} \end{cases}$$

Let $(H, \vec{c})$ be a linear program. We number the half-planes $h_1, h_2, \ldots, h_n$. Let H_i be the set of the first i constraints, together with the special constraints m_1 and m_2, and let C_i be the feasible region defined by these constraints:

$$\begin{aligned} H_i &:= \{m_1, m_2, h_1, h_2 \ldots, h_i\}, \\ C_i &:= m_1 \cap m_2 \cap h_1 \cap h_2 \cap \cdots \cap h_i. \end{aligned}$$

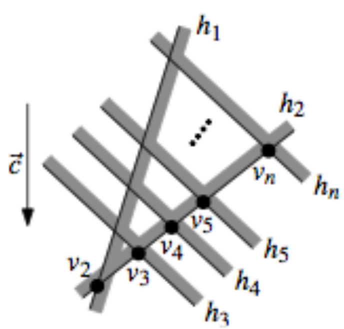
Algorithm 2DBOUNDEDLP($H, \vec{c}, m_1, m_2$)

Input. A linear program ($H \cup \{m_1, m_2\}, \vec{c}$), where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.

Output. If ($H \cup \{m_1, m_2\}, \vec{c}$) is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

1. Let v_0 be the corner of C_0.
2. Let $h_1, \ldots, h_n$ be the half-planes of H.
3. **for** $i \leftarrow 1$ **to** n
4.   **do if** $v_{i-1} \in h_i$
5.   **then** $v_i \leftarrow v_{i-1}$
6.   **else** $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
7.   **if** p does not exist
8.   **then** Report that the linear program is infeasible and quit.
9. **return** v_n

In more detail



Lemma 4.7 *Algorithm* 2DBOUNDEDLP *computes the solution to a bounded linear program with n constraints and two variables in $O(n^2)$ time and linear storage.*

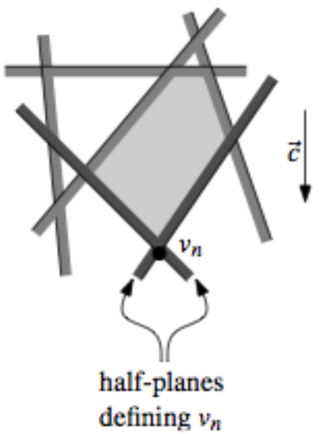
It is easy to see that the algorithm requires only linear storage. We add the half-planes one by one in n stages. The time spent in stage i is dominated by the time to solve a 1-dimensional linear program in line 6, which is $O(i)$. Hence, the total time needed is bounded by

$$\sum_{i=1}^{n} O(i) = O(n^2). \qquad \square$$

Randomized linear programming

Algorithm 2DRANDOMIZEDBOUNDEDLP($H, \vec{c}, m_1, m_2$)
Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.
Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

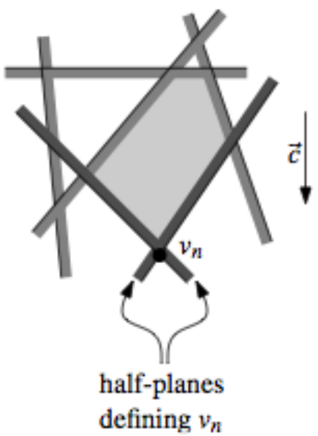
1. Let v_0 be the corner of C_0.
2. Compute a *random* permutation $h_1, \dots, h_n$ of the half-planes by calling RANDOMPERMUTATION($H[1 \cdots n]$).
3. **for** $i \leftarrow 1$ **to** n
4.   **do if** $v_{i-1} \in h_i$
5.   **then** $v_i \leftarrow v_{i-1}$
6.   **else** $v_i \leftarrow$the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints in H_{i-1}.
7.   **if** p does not exist
8.   **then** Report that the linear program is infeasible and quit.
9. **return** v_n



Lemma 4.8 *The 2-dimensional linear programming problem with n constraints can be solved in $O(n)$ randomized expected time using worst-case linear storage.*

Let X_i be a random variable, which is 1 if $v_{i-1} \notin h_i$, and 0 otherwise.

Expected complexity



$$\sum_{i=1}^{n} O(i) \cdot X_i.$$

Linearity of expectation

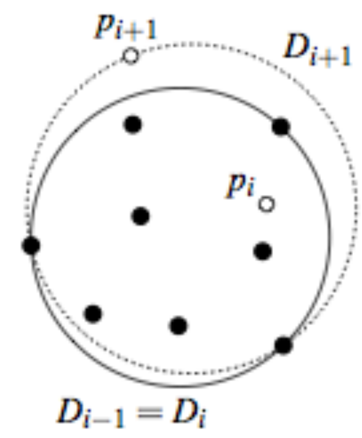
$$\mathrm{E}[\sum_{i=1}^{n} O(i) \cdot X_i] = \sum_{i=1}^{n} O(i) \cdot \boxed{\mathrm{E}[X_i].}$$

Since the half-planes are added in random order,
the probability that h_i is one of the special half-planes
is at most 2/i

Lemma 4.8 *The 2-dimensional linear programming problem with n constraints can be solved in $O(n)$ randomized expected time using worst-case linear storage.*

$$\sum_{i=1}^{n} O(i) \cdot \frac{2}{i} = O(n).$$

Smallest Enclosing Disk



Algorithm MINIDISC(P)
Input. A set P of n points in the plane.
Output. The smallest enclosing disc for P.
1. Compute a random permutation $p_1, \ldots, p_n$ of P.
2. Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
3. **for** $i \leftarrow 3$ **to** n
4.   **do if** $p_i \in D_{i-1}$
5.   **then** $D_i \leftarrow D_{i-1}$
6.   **else** $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \ldots, p_{i-1}\}, p_i$)
7. **return** D_n

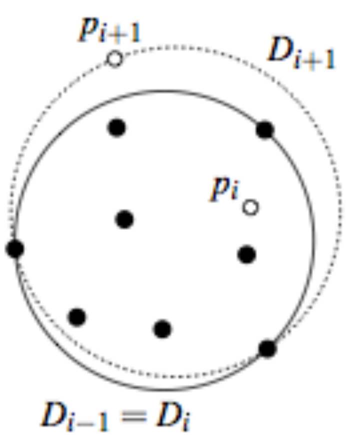
The simple randomized technique we used above turns out to be surprisingly powerful.

It can be applied not only to linear programming but to a variety of other optimization problems as well.

Smallest Enclosing Disk

MINIDISCWITHPOINT(P,q)
Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.
Output. The smallest enclosing disc for P with q on its boundary.

Compute a random permutation $p_1, \dots, p_n$ of P.
Let D_1 be the smallest disc with q and p_1 on its boundary.
for $j \leftarrow 2$ **to** n
  do if $p_j \in D_{j-1}$
    then $D_j \leftarrow D_{j-1}$
    else $D_j \leftarrow$ MINIDISCWITH2POINTS($\{p_1, \dots, p_{j-1}\}, p_j, q$)
return D_n

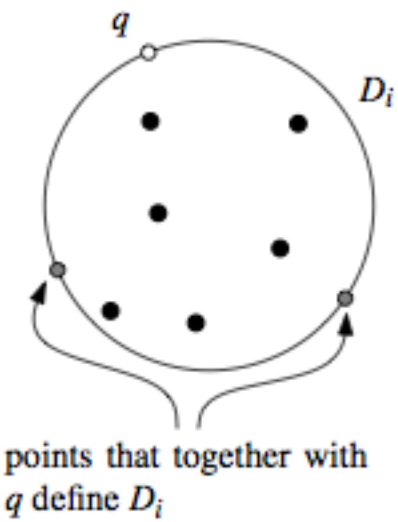


MINIDISCWITH2POINTS(P, q_1, q_2)
Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.
Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

Let D_0 be the smallest disc with q_1 and q_2 on its boundary.
for $k \leftarrow 1$ **to** n
  do if $p_k \in D_{k-1}$
    then $D_k \leftarrow D_{k-1}$
    else $D_k \leftarrow$ the disc with q_1, q_2, and p_k on its boundary
return D_n

Theorem 4.15 *The smallest enclosing disc for a set of n points in the plane can be computed in $O(n)$ expected time using worst-case linear storage.*

$$O(n)+\sum_{i=2}^{n} O(i)\frac{2}{i} = O(n).$$



One of the points on the boundary is q, so there are at most two points that cause the smallest enclosing circle to shrink.

Expected running time for miniDiskWithPoint !

Exercice 4

4.2 Consider the casting problem in the plane: we are given polygon $\mathcal{P}$ and a 2-dimensional mold for it. Describe a linear time algorithm that decides whether $\mathcal{P}$ can be removed from the mold by a single translation.