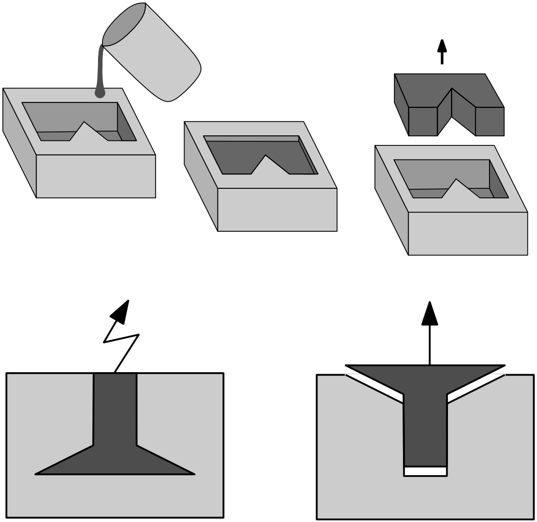
Linear Programming- Manufacturing with Molds

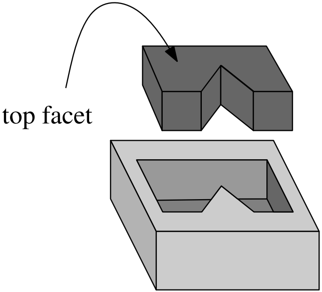
1395-2

Casting process



The Geometry of Casting

- top facet
- ordinary facet
  Every ordinary facet f has a corresponding facet in the mold, which we denote by $\hat{f}$.
- castable object



The Geometry of Casting

Question:

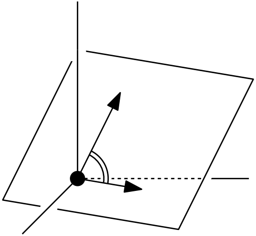
Given an object P, can it be removed from its mold by a single translation?
In other words, we want to decide whether a direction $\vec{d}$ exists such that P can be translated to infinity in direction $\vec{d}$ without intersecting the interior of the mold during the translation.

Observation:

Every ordinary facet f must move away from, or slide along, its corresponding facet $\hat{f}$ of the mold.

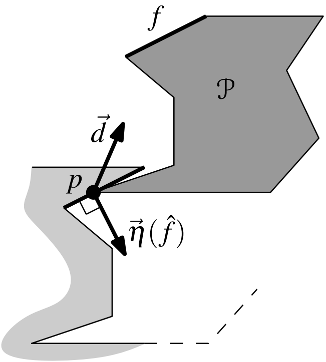
Angle between two vectors in 3d:

The angle of the vectors is the smaller of the two angles measured in the plane determined by them.

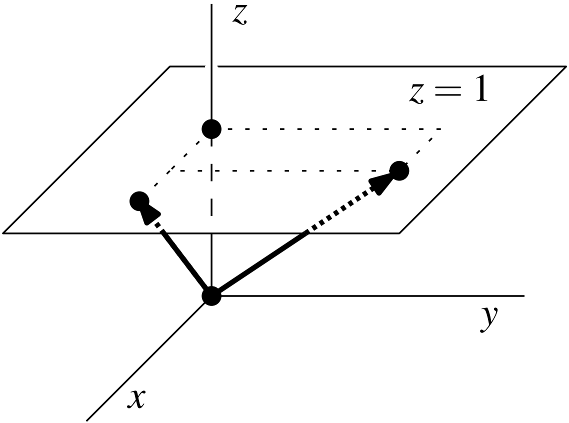


Lemma 4.1

P can be removed from its mold in direction d if and only if d makes an angle of at least $\pi/2$ with the outward normal of all ordinary facets of P.

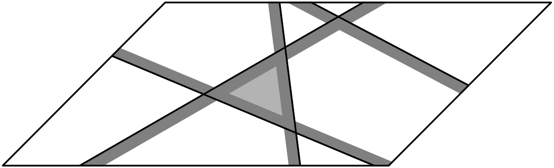


1-1 corresponding between direction and points



By Lemma 4.1

- $\vec{\eta} = (\vec{\eta}_x, \vec{\eta}_y, \vec{\eta}_z)$: outward normal of an ordinary facet
- $\vec{d} = (d_x, d_y, 1)$: removal direction
- $\angle(\vec{\eta}, \vec{d}) \geq \pi/2 \iff \vec{\eta}.\vec{d} \leq 0$
- $\vec{\eta}_x \times d_x + \vec{\eta}_y \times d_y + \vec{\eta}_z \leq 0$
- This inequality describe a half-plane in the plane $z = 1$.



- An object P is castable, for a fixed top facet $\iff$ the intersection of half-planes is non-empty.
- If the intersection problem is solvable in $\mathcal{O}(A)$ time, then the castability problem can be solved in $\mathcal{O}(An)$ time.
- We will solve the intersect5ion problem in $\mathcal{O}(n)$ expected time.
- **Theorem 4.2:** In $\mathcal{O}(n^2)$ expected time and using $\mathcal{O}(n)$ storage it can be decided whether a polyhydron with n facets is castable.

Half-plane intersection problem

Problem:

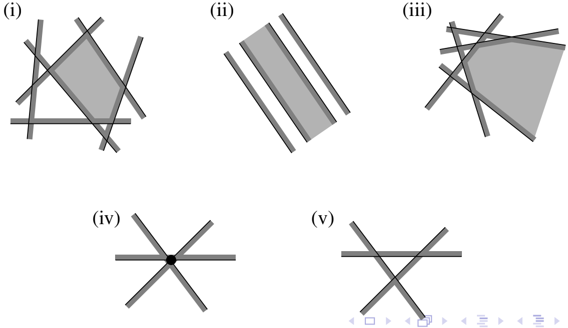
- Given a set $H = \{h_1, h_2, \ldots, h_n\}$ of half-planes, find the intersection of them.
- Given a set of n linear constraints in two variables, is there a point in the plane, find all points that satisfy all the constraints.

Note:

For casting problem, we do not need to find the intersection of half-planes. Here we consider a more general problem.

Observations:

- Since a half-plane is convex, the intersection of half-planes is convex.
- Since the intersection is convex, every half-plane bounding line can contribute at most one edge.
- A few examples of intersections of half-planes:



Computing the intersection of n half-planes

An straightforward divide-and-conquer algorithm:

Algorithm INTERSECTHALFPLANES(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.

```
if card(H) = 1
   then C ← the unique half-plane h ∈ H
   else Split H into sets H_1 and H_2 of size ⌈n/2⌉ and ⌊n/2⌋.
        C_1 ← INTERSECTHALFPLANES(H_1)
        C_2 ← INTERSECTHALFPLANES(H_2)
        C ← INTERSECTCONVEXREGIONS(C_1, C_2)
```

Time complexity:

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

$T(n) \in \mathcal{O}(n \log^2 n)$.

Computing the intersection of n half-planes

An straightforward divide-and-conquer algorithm:

Algorithm INTERSECTHALFPLANES(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.

```
if card(H) = 1
    then C ← the unique half-plane h ∈ H
    else Split H into sets H_1 and H_2 of size ⌈n/2⌉ and ⌊n/2⌋.
         C_1 ← INTERSECTHALFPLANES(H_1)
         C_2 ← INTERSECTHALFPLANES(H_2)
         C ← INTERSECTCONVEXREGIONS(C_1, C_2)
```

Time complexity:

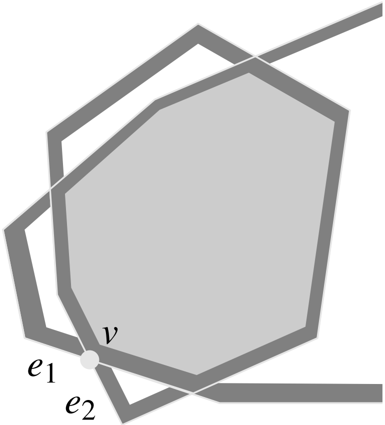
$$T(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1 \\ 2T(n/2) + \mathcal{O}(n \log n) & \text{if } n > 1 \end{cases}$$

$T(n) \in \mathcal{O}(n \log^2 n)$.

Computing the intersection of n half-planes

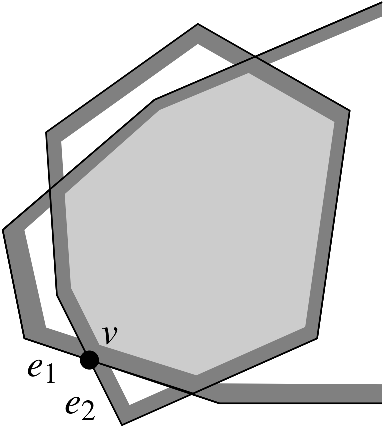
Question: Can we compute the intersection of two convex polygons in $o(n \log n)$ time?

Answer: Yes, we can.

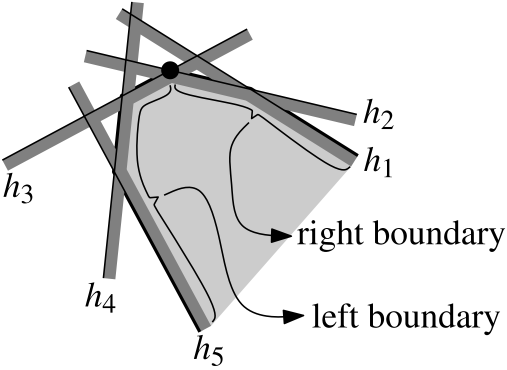


Computing the intersection of n half-planes

Question: Can we compute the intersection of two convex polygons in $o(n \log n)$ time?
Answer: Yes, we can.



Computing intersection of two convex polygons

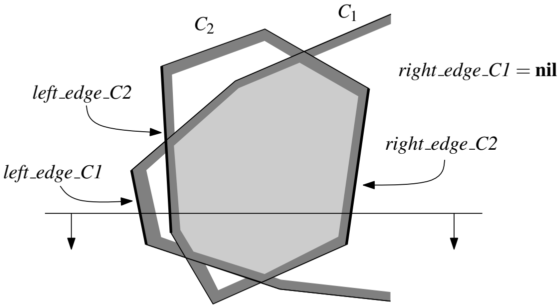


$$\mathcal{L}_{\text{left}}(C) = h_3, h_4, h_5$$

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

Computing intersection of two convex polygons

We use a plane sweep algorithm:
Note: At most 4 line segments intersect the sweep line.

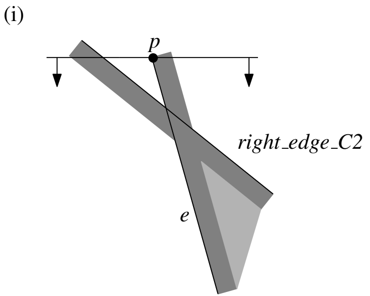


Computing intersection of two convex polygons

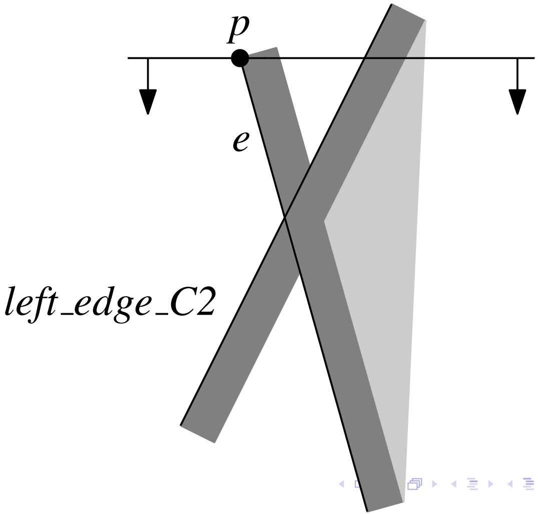
- At each event point some new edge e appears on the boundary.
- To handle the edge e: several cases: e belongs to $C1$ or to $C2$, and whether it is on the left or the right boundary.
- We consider the case when e is on the left boundary of $C1$.
  p: the upper endpoint of e.
  Three case (based on C):
  (1) the edge with p as upper endpoint,
  (2) the edge with $e \cap left\_edge\_C2$ as upper endpoint, and
  (3) the edge with $e \cap right\_edge\_C2$ as upper endpoint.

Computing intersection of two convex polygons

It performs the following actions.



Computing intersection of two convex polygons



Computing intersection of two convex polygons

Theorem 4.3

The intersection of two convex polygonal regions in the plane can be computed in $\mathcal{O}(n)$ time.

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

$T(n) \in \mathcal{O}(n \log n)$.

Corollary 4.4

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

Computing intersection of two convex polygons

Theorem 4.3

The intersection of two convex polygonal regions in the plane can be computed in $\mathcal{O}(n)$ time.

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

$T(n) \in \mathcal{O}(n \log n)$.

Corollary 4.4

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

Computing intersection of two convex polygons

Theorem 4.3

The intersection of two convex polygonal regions in the plane can be computed in $\mathcal{O}(n)$ time.

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

$T(n) \in \mathcal{O}(n \log n)$.

Corollary 4.4

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

Computing intersection of two convex polygons

Theorem 4.3

The intersection of two convex polygonal regions in the plane can be computed in $\mathcal{O}(n)$ time.

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

$T(n) \in \mathcal{O}(n \log n)$.

Corollary 4.4

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

Incremental Linear Programming

Previous Section:

- We computed the intersection of n half-planes in $\mathcal{O}(n \log n)$ time.
- The problem has $\Omega(n \log n)$ lower bound.
- So, we cannot solve the casting problem faster in this way.
- Note that, for casting problem we only need to know if the intersection is empty or not.

In this section:
An "expected" linear time algorithm presented to solve the problem using linear programming.

Linear optimization problem:

Maximize $c_1x_1 + c_2x_2 + \cdots + c_dx_d$ (objective function)
Subject to

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

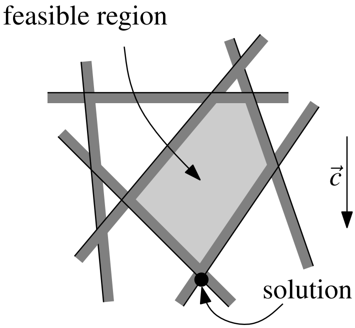
(constraints)

Linear optimization problem:

Objective function:

$f(x_1, \ldots, x_d) = c_1x_1 + c_2x_2 + \cdots + c_dx_d = \vec{c}.\vec{x}$.

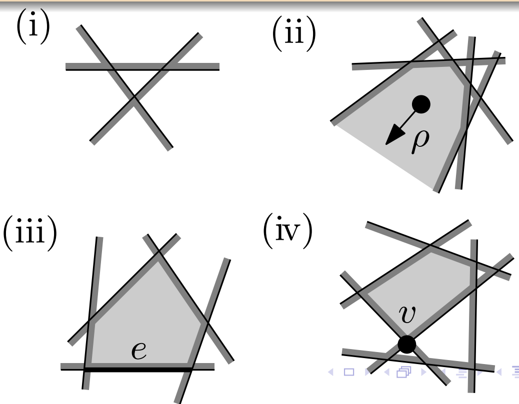
$\vec{c} = (c_1, \ldots, c_d)$, $\vec{x} = (x_1, \ldots, x_d)$.



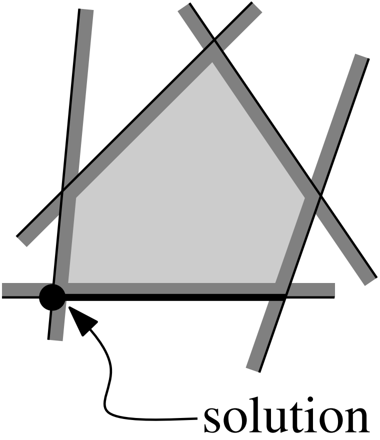
LP in the plane:

- H: n half-plane or in other words n linear constraints
- $f_{\vec{c}}(p)$: Objective function = $c_x p_x + c_y p_y$, $\vec{c} = (c_x, c_y)$, $p = (p_x, p_y)$.
- Goal: find $p \in \mathbb{R}^2$ s.t. $p \in \cap H$ and $f_{\vec{c}}(p)$ is maximized.

Cases:



Unique Answer:



Bound the feasible region: we add to half-plane to the set of half-planes

$$m_1 = \begin{cases} p_x \le M & \text{if } c_x > 0 \\ -p_x \le M & \text{Otherwise} \end{cases}$$

$$m_2 = \begin{cases} p_y \le M & \text{if } c_y > 0 \\ -p_y \le M & \text{Otherwise} \end{cases}$$

Notations:

- $(H, \vec{c})$: a linear program.
- $H_i = \{m_1, m_2, h_1, h_2, \ldots, h_i\}$.
- $C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \cdots \cap h_i$.
- Clearly $C_0 \supseteq C_1 \supseteq C_2 \supseteq \cdots C_n = C$.
- v_i: the optimal solution for C_i.
- If $C_i = \emptyset$, for some i, then the problem is infeasible (intersection is empty).

Linear Programming in $\mathbb{R}^2$

Bound the feasible region: we add to half-plane to the set of half-planes

$$m_1 = \begin{cases} p_x \leq M & \text{if } c_x > 0 \\ -p_x \leq M & \text{Otherwise} \end{cases}$$

$$m_2 = \begin{cases} p_y \leq M & \text{if } c_y > 0 \\ -p_y \leq M & \text{Otherwise} \end{cases}$$

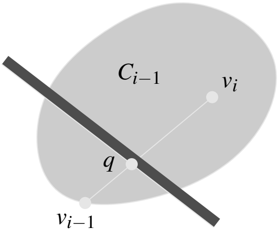
Notations:

- $(H, \vec{c})$: a linear program.
- $H_i = \{m_1, m_2, h_1, h_2, \ldots, h_i\}$.
- $C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \cdots \cap h_i$.
- Clearly $C_0 \supseteq C_1 \supseteq C_2 \supseteq \cdots C_n = C$.
- v_i: the optimal solution for C_i.
- If $C_i = \emptyset$, for some i, then the problem is infeasible (intersection is empty).

Lemma 4.5

We have
(i) If $v_{i-1} \in h_i$, then $v_i = v_{i-1}$.
(ii) If $v_{i-1} \notin h_i$, then either $C_i = \emptyset$ or $v_i \in \ell_i$, where ℓ_i is the line bounding h_i.

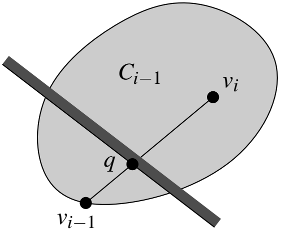
Proof.
(i): Clear, since $C_i \subseteq C_{i-1}$.
(ii):

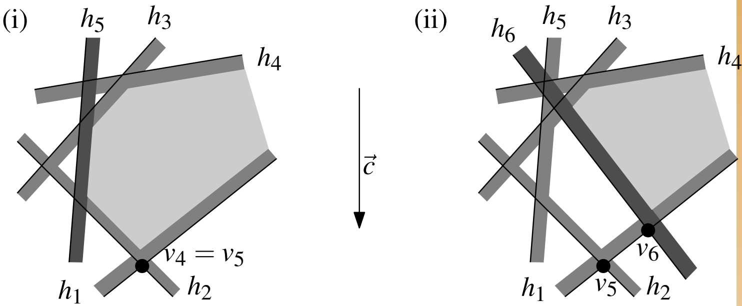


Lemma 4.5

We have
(i) If $v_{i-1} \in h_i$, then $v_i = v_{i-1}$.
(ii) If $v_{i-1} \notin h_i$, then either $C_i = \emptyset$ or $v_i \in \ell_i$, where ℓ_i is the line bounding h_i.

Proof.
(i): Clear, since $C_i \subseteq C_{i-1}$.
(ii):

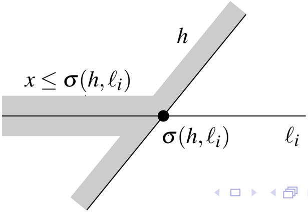




(i)
h_5
h_3
h_4
$v_4 = v_5$
h_1
h_2
$\vec{c}$
(ii)
h_6
h_5
h_3
h_4
v_6
h_1
v_5
h_2

New Simpler problem:

- Find the point p on ℓ_i that maximizes $f_{\vec{c}}(p)$, subject to the constraints $p \in h$, for $h \in H_{i-1}$.
- Maximize $f_{\vec{c}}(x)$
  subject to
  $x \geq \sigma(h, \ell_i)$, $h \in H_{i-1}$ and $\ell_i \cap h$ is bounded to the left
  $x \leq \sigma(h, \ell_i)$, $h \in H_{i-1}$ and $\ell_i \cap h$ is bounded to the right



1-dimensional linear program:

- Maximize $f_{\vec{c}}(x)$
  subject to
  $x \geq \sigma(h, \ell_i)$, $h \in H_{i-1}$ and $\ell_i \cap h$ is bounded to the left
  $x \leq \sigma(h, \ell_i)$, $h \in H_{i-1}$ and $\ell_i \cap h$ is bounded to the right
- $x_{left} = \max_{h \in H_{i-1}} \{\sigma(h, \ell_i) : \ell_i \cap h$ is bounded to the left$\}$
  $x_{right} = \min_{h \in H_{i-1}} \{\sigma(h, \ell_i) :$
  $\ell_i \cap h$ is bounded to the right$\}$

Lemma 4.6

A 1-dimensional linear program can be solved in linear time. Hence, if case (ii) of Lemma 4.5 arises, then we can compute v_i, in $\mathcal{O}(i)$ time.

Linear Programming in $\mathbb{R}^2$
incremental algorithm

2DBOUNDEDLP Algorithm:

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

Lemma 4.7

Algorithm 2DBOUNDEDLP computes the solution to a bounded linear program with n constraints and two variables in $\mathcal{O}(n^2)$ time and linear storage.

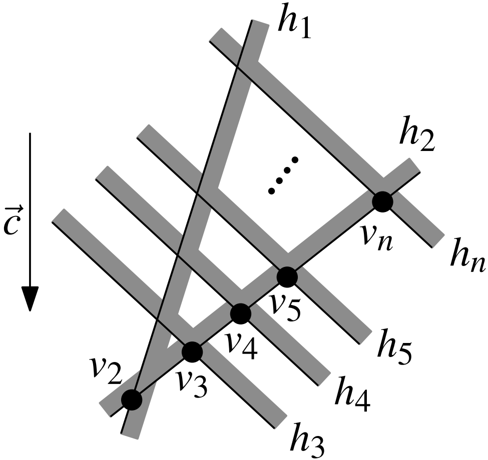
2DBoundedLP Algorithm:

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

Lemma 4.7

Algorithm 2DBoundedLP computes the solution to a bounded linear program with n constraints and two variables in $\mathcal{O}(n^2)$ time and linear storage.

A bad case:



Randomized algorithms

Use randomness to avoid bad cases.

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 $h_1, \ldots, h_n$ 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 $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Proof.
$$X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Total time} = \sum_{i=1}^{n} (\mathcal{O}(i) \times X_i)$$

$$\begin{aligned} E(\text{Total time}) &= E\left(\sum_{i=1}^{n} (\mathcal{O}(i) \times X_i)\right) \\ &= \sum_{i=1}^{n} (\mathcal{O}(i) \times E(X_i)). \end{aligned}$$

Lemma 4.8

The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Proof.

$$X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Total time} = \sum_{i=1}^{n} (\mathcal{O}(i) \times X_i)$$

$$\begin{aligned} E(\text{Total time}) &= E\left(\sum_{i=1}^{n} (\mathcal{O}(i) \times X_i)\right) \\ &= \sum_{i=1}^{n} (\mathcal{O}(i) \times E(X_i)). \end{aligned}$$

Lemma 4.8

The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Proof (Cont.). Any upper bound for $E(x_i)$?

$$\begin{aligned} E(X_i) &= Pr(v_{i-1} \notin h_i) \\ &\leq \frac{2}{i} \quad \text{(by backward analysis)} \end{aligned}$$

$$\begin{aligned} E(\text{Total time}) &= \sum_{i=1}^{n} (\mathcal{O}(i) \times E(X_i)) \\ &\leq \sum_{i=1}^{n} \left(\mathcal{O}(i) \times \frac{2}{i}\right) \in \mathcal{O}(n). \end{aligned}$$

Lemma 4.8

The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Proof (Cont.). Any upper bound for $E(x_i)$?

$$
\begin{aligned}
E(X_i) &= Pr(v_{i-1} \notin h_i) \\
&\leq \frac{2}{i} \quad \text{(by backward analysis)}
\end{aligned}
$$

$$
\begin{aligned}
E(\text{Total time}) &= \sum_{i=1}^{n} (\mathcal{O}(i) \times E(X_i)) \\
&\leq \sum_{i=1}^{n} \left(\mathcal{O}(i) \times \frac{2}{i} \right) \in \mathcal{O}(n).
\end{aligned}
$$

Lemma 4.8

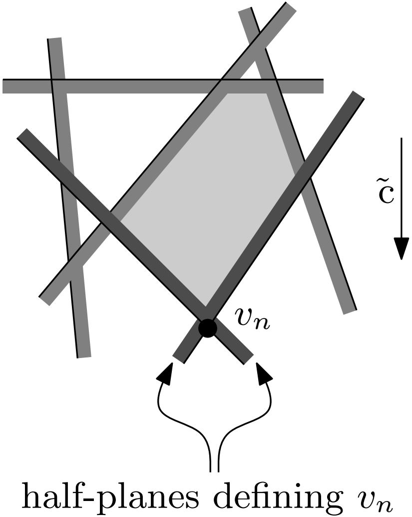
The 2-dimensional linear programming problem with n constraints can be solved in $\mathcal{O}(n)$ randomized expected time using worst-case linear storage.

Proof (Cont.). Any upper bound for $E(x_i)$?

$$\begin{aligned} E(X_i) &= Pr(v_{i-1} \notin h_i) \\ &\leq \frac{2}{i} \quad \text{(by backward analysis)} \end{aligned}$$

$$\begin{aligned} E(\text{Total time}) &= \sum_{i=1}^{n} (\mathcal{O}(i) \times E(X_i)) \\ &\leq \sum_{i=1}^{n} \left(\mathcal{O}(i) \times \frac{2}{i}\right) \in \mathcal{O}(n). \end{aligned}$$

Backward Analysis: Why $Pr(v_{i-1} \notin h_i) \leq \frac{2}{i}$?



Backward Analysis: Why $Pr(v_{i-1} \notin h_i) \leq \frac{2}{i}$?