Advanced Algorithms

Winter term 2019/20

Lecture 3. 2D Linear Programming via sweep-lines and randomization

Source: CG: A&A §4

Steven Chaplick & Alexander Wolff
Chair for Computer Science I
Maximizing Profit

You are the boss of a small company that produces two products, $P_1$ and $P_2$. If you produce $x_1$ units of $P_1$ and $x_2$ units of $P_2$, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$
Maximizing Profit

You are the boss of a small company that produces two products, $P_1$ and $P_2$. If you produce $x_1$ units of $P_1$ and $x_2$ units of $P_2$, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$

Your production runs on three machines $M_A$, $M_B$, and $M_C$ with the following capacities:

$$M_A : \quad 4x_1 + 11x_2 \leq 880$$
$$M_B : \quad x_1 + x_2 \leq 150$$
$$M_C : \quad x_2 \leq 60$$
Maximizing Profit

You are the boss of a small company that produces two products, $P_1$ and $P_2$. If you produce $x_1$ units of $P_1$ and $x_2$ units of $P_2$, your profit in € is

$$G(x_1, x_2) = 300x_1 + 500x_2$$

Your production runs on three machines $M_A$, $M_B$, and $M_C$ with the following capacities:

$$M_A : \quad 4x_1 + 11x_2 \leq 880$$
$$M_B : \quad x_1 + x_2 \leq 150$$
$$M_C : \quad x_2 \leq 60$$

Which choice of $(x_1, x_2)$ maximizes your profit?
The Answer

linear constraints:

\[ M_A : 4x_1 + 11x_2 \leq 880 \]
\[ M_B : x_1 + x_2 \leq 150 \]
\[ M_C : x_2 \leq 60 \]
The Answer

**Linear Constraints:**

- $M_A: 4x_1 + 11x_2 \leq 880$
- $M_B: x_1 + x_2 \leq 150$
- $M_C: x_2 \leq 60$
The Answer

\[ M_A : 4x_1 + 11x_2 \leq 880 \]
\[ M_B : x_1 + x_2 \leq 150 \]
\[ M_C : x_2 \leq 60 \]
The Answer

linear constraints:

\[ M_A : \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B : \quad x_1 + x_2 \leq 150 \]
\[ M_C : \quad x_2 \leq 60 \]
The Answer

linear constraints:

\[
\begin{align*}
M_A & : 4x_1 + 11x_2 \leq 880 \\
M_B & : x_1 + x_2 \leq 150 \\
M_C & : x_2 \leq 60
\end{align*}
\]

\[
\begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0
\end{align*}
\]
The Answer

linear constraints:

\[ \begin{align*}
M_A & : 4x_1 + 11x_2 \leq 880 \\
M_B & : x_1 + x_2 \leq 150 \\
M_C & : x_2 \leq 60
\end{align*} \]

\[ \begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0
\end{align*} \]
The Answer

linear constraints:

\[ M_A : \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B : \quad x_1 + x_2 \leq 150 \]
\[ M_C : \quad x_2 \leq 60 \]

\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

set of feasible solutions
The Answer

linear constraints:

\[ M_A : \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B : \quad x_1 + x_2 \leq 150 \]
\[ M_C : \quad x_2 \leq 60 \]

\[
\begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0
\end{align*}
\]

linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 \]
The Answer

linear constraints:

\[ M_A : 4x_1 + 11x_2 \leq 880 \]
\[ M_B : x_1 + x_2 \leq 150 \]
\[ M_C : x_2 \leq 60 \]

\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2) \]

set of feasible solutions
The Answer

**Linear constraints:**

- $M_A$: $4x_1 + 11x_2 \leq 880$
- $M_B$: $x_1 + x_2 \leq 150$
- $M_C$: $x_2 \leq 60$

- $x_1 \geq 0$
- $x_2 \geq 0$

**Linear objective function:**

$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2)$

- "Iso-profit line" (orthogonal to $(300, 500)$)
The Answer

**linear constraints:**

\[ M_A: \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B: \quad x_1 + x_2 \leq 150 \]
\[ M_C: \quad x_2 \leq 60 \]

\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

**linear objective fct.:**

\[ G(x_1, x_2) = 300x_1 + 500x_2 \]
\[ = (300, 500)(x_1, x_2) \]

set of feasible solutions

"iso-profit line" (orthogonal to \((300, 500)\))
The Answer

linear constraints:

\[ M_A : 4x_1 + 11x_2 \leq 880 \]
\[ M_B : x_1 + x_2 \leq 150 \]
\[ M_C : x_2 \leq 60 \]
\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 \]
\[ = (300, 500)(x_1, x_2) \]

"iso-profit line" (orthogonal to \((300, 500)\))
The Answer

**linear constraints:**

- $M_A: \ 4x_1 + 11x_2 \leq 880$
- $M_B: \ x_1 + x_2 \leq 150$
- $M_C: \ x_2 \leq 60$

$x_1 \geq 0$
$x_2 \geq 0$

**linear objective fct.:**

$G(x_1, x_2) = 300x_1 + 500x_2$

$= (300, 500)(x_1, x_2)$

"iso-profit line" (orthogonal to $(300, 500)$)
The Answer

**linear constraints:**

- **$M_A$:** $4x_1 + 11x_2 \leq 880$
- **$M_B$:** $x_1 + x_2 \leq 150$
- **$M_C$:** $x_2 \leq 60$

**Linear objective fct.:**

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2)$$

**set of feasible solutions**

**"iso-profit line"** (orthogonal to $(300, 500)$)
The Answer

**linear constraints:**

- $M_A : 4x_1 + 11x_2 \leq 880$
- $M_B : x_1 + x_2 \leq 150$
- $M_C : x_2 \leq 60$

**linear objective fct.:**

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2)$$

**set of feasible solutions**

**"iso-profit line" (orthogonal to $(300, 500)$)**

$$\partial M_A \cap \partial M_B =$$
The Answer

linear constraints:

- $M_A: 4x_1 + 11x_2 \leq 880$
- $M_B: x_1 + x_2 \leq 150$
- $M_C: x_2 \leq 60$

linear objective fct.:

$G(x_1, x_2) = 300x_1 + 500x_2$

$= (300, 500)(x_1, x_2)$

set of feasible solutions

$\partial M_A \cap \partial M_B = \left\{ \left( \frac{110}{40} \right) \right\}$

"iso-profit line" (orthogonal to $(300, 500)$)
The Answer

**Linear constraints:**

\[
\begin{align*}
M_A : & \quad 4x_1 + 11x_2 \leq 880 \\
M_B : & \quad x_1 + x_2 \leq 150 \\
M_C : & \quad x_2 \leq 60 \\
\end{align*}
\]

\[
\begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0 \\
\end{align*}
\]

**Linear objective function:**

\[
G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1 \ x_2)
\]

"iso-profit line" (orthogonal to \((300 \ 500)\))

**Set of feasible solutions:**

\[
G(110, 40) =
\]

...
The Answer

linear constraints:

\[ M_A: 4x_1 + 11x_2 \leq 880 \]
\[ M_B: x_1 + x_2 \leq 150 \]
\[ M_C: x_2 \leq 60 \]

\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 \]
\[ = (300, 500) (x_1 \ x_2) \]

\[ G(110, 40) = 53,000 \]

set of feasible solutions

„iso-profit line“ (orthogonal to (300, 500))
The Answer

linear constraints:

\[
\begin{align*}
M_A : & \quad 4x_1 + 11x_2 \leq 880 \\
M_B : & \quad x_1 + x_2 \leq 150 \\
M_C : & \quad x_2 \leq 60
\end{align*}
\]

\[
\begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0
\end{align*}
\]

linear objective fct.:

\[
G(x_1, x_2) = 300x_1 + 500x_2
\]

\[
= (300, 500)(x_1 \ x_2)
\]

set of feasible solutions

iso-profit line“ (orthogonal to \((300, 500)\))
The Answer

**linear constraints:**

- $M_A : 4x_1 + 11x_2 \leq 880$
- $M_B : x_1 + x_2 \leq 150$
- $M_C : x_2 \leq 60$

**linear objective fct.:**

$$G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2)$$

**set of feasible solutions**

The optimal solution is $G(110, 40) = 53,000$

"iso-profit line" (orthogonal to $(300, 500)$)
The Answer

*linear constraints:* 
- \( MA: \ 4x_1 + 11x_2 \leq 880 \)
- \( MB: \ x_1 + x_2 \leq 150 \)
- \( MC: \ x_2 \leq 60 \)

*linear objective fct.:*
\[
G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1 \ x_2)
\]

---

set of feasible solutions

"iso-profit line" (orthogonal to \((300, 500)\))

\( G(110, 40) = 53,000 \)
The Answer

Linear constraints:

\[ \begin{align*}
M_A : & \quad 4x_1 + 11x_2 \leq 880 \\
M_B : & \quad x_1 + x_2 \leq 150 \\
M_C : & \quad x_2 \leq 60
\end{align*} \]

Linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2) \]

Set of feasible solutions:

"iso-profit line" (orthogonal to \((300, 500)\))

At \((110, 40)\), the profit is 53,000€.
The Answer

linear constraints:

\[ M_A : \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B : \quad x_1 + x_2 \leq 150 \]
\[ M_C : \quad x_2 \leq 60 \]

\[ x_1 \geq 0 \quad x_2 \geq 0 \]

linear objective fct.:

\[ G(x_1, x_2) = 300x_1 + 500x_2 \]
\[ = (300, 500)(x_1 \ x_2) \]
\[ G(110, 40) = 53,000 \]

"iso-profit line" (orthogonal to \((300 \ 500)\))
The Answer

Linear constraints:

\[ M_A : \quad 4x_1 + 11x_2 \leq 880 \]
\[ M_B : \quad x_1 + x_2 \leq 150 \]
\[ M_C : \quad x_2 \leq 60 \]

\[ x_1 \geq 0 \]
\[ x_2 \geq 0 \]

Linear objective function:

\[ G(x_1, x_2) = 300x_1 + 500x_2 = (300, 500)(x_1, x_2) \]

\[ G(110, 40) = 53,000 \]

"iso-profit line" (orthogonal to \((300, 500))\)
The Answer

**linear constraints:**

- \( M_A : \ 4x_1 + 11x_2 \leq 880 \)
- \( M_B : \ x_1 + x_2 \leq 150 \)
- \( M_C : \ x_2 \leq 60 \)

**Ax ≤ b**

\[
\begin{align*}
x_1 & \geq 0 \\
x_2 & \geq 0
\end{align*}
\]

**linear objective fct.:**

maximize \( c^T x \)

\[
G(x_1, x_2) = 300x_1 + 500x_2
\]

\[
= (300, 500)(x_1 \ x_2)
\]

\[
G(110, 40) = 53,000
\]

\( \Rightarrow \) maximum value of objective fct. given constraints

**set of feasible solutions**

**“iso-profit line’’ (orthogonal to \((300 \ 500)\))
The Answer

**linear constraints:**

- **$M_A$:** $4x_1 + 11x_2 \leq 880$
- **$M_B$:** $x_1 + x_2 \leq 150$
- **$M_C$:** $x_2 \leq 60$

$x_1 \geq 0$

$x_2 \geq 0$

**linear objective fct.:** maximize $c^T x$

$G(x_1, x_2) = 300x_1 + 500x_2$

$= (300, 500)(x_1 \quad x_2)$

$G(110, 40) = 53,000$

= maximum value of objective fct. given constraints

= $\max\{c^T x \mid Ax \leq b, x \geq 0\}$

"iso-profit line" (orthogonal to $(300 \quad 500))
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:

– Simplex [Dantzig ’47]
Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]
Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where $n$ and $d$ are large.
We consider $d = 2$. 
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

[“Book” application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar' 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
["Book" application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]

\[ \bigcap H = \emptyset \quad \bigcap H \text{ unbd. in dir. } c \]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:

- Simplex \ [Dantzig ‘47]
- Ellipsoid method \ [Khatchiyan ’79]
- Inner-point method \ [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.

[“Book” application: casting]

\[ \bigcap H = \varnothing \quad \bigcap H \text{ unbd. in dir. } c \]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig '47]
- Ellipsoid method [Khatchiyan '79]
- Inner-point method [Karmakar '84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research. ["Book" application: casting]

\[ \bigcap H = \emptyset \quad \bigcap H \text{ unbounded in dir. } c \quad \text{set of optima: segment} \quad \bigcap H \text{ bounded.} \]
Definition and Known Algorithms

Given a set $H$ of $n$ halfspaces in $\mathbb{R}^d$ and a direction $c$, find a point $x \in \bigcap H$ such that $cx$ is maximum (or minimum).

Many algorithms known, e.g.:
- Simplex [Dantzig ’47]
- Ellipsoid method [Khatchiyan ’79]
- Inner-point method [Karmakar’ 84]

Good for instances where $n$ and $d$ are large.

We consider $d = 2$.

VERY important problem, e.g., in Operations Research.
[“Book” application: casting]

$\bigcap H$ bounded.
First Approach

- compute $\cap H$ iteratively
First Approach

- **compute** $\cap H$ **iteratively**

- **walk** $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time
First Approach

- compute \( \cap H \) iteratively
- walk \( \partial (\cap H) \), find vertex \( x \) w/ \( c_x \) maximum, \( O(n) \) time

**IntersectHalfplanes(\( H \))**

Let \( H = (h_1, \ldots, h_n) \)
\[ C \leftarrow h_1 \]
\[ \text{foreach } i \text{ from } 2 \text{ to } n \text{ do} \]
\[ C \leftarrow C \cap h_i \]
\[ \text{return } C \]
First Approach

- compute $\bigcap H$ iteratively
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

```plaintext
IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$

foreach $i$ from 2 to $n$ do
    $C \leftarrow C \cap h_i$
return $C$
```

Running time:
First Approach

- compute $\bigcap H$ iteratively
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

**IntersectHalfplanes**($H$)

Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
$\text{foreach } i \text{ from } 2 \text{ to } n \text{ do}$
$\quad C \leftarrow C \cap h_i$
return $C$

**Running time:** $T_{IH}(n) = n \cdot $ [Blank]

How??
First Approach

• compute $\cap H$ iteratively

• walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

```
IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$
C ← $h_1$
foreach $i$ from 2 to $n$ do
  C ← C $\cap$ $h_i$
return C
```

$T_{IH}(n) = n \cdot$ How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$
First Approach

- compute $\bigcap H$ iteratively
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

```plaintext
IntersectHalfplanes(H)

Let $H = (h_1, \ldots, h_n)$

$C \leftarrow h_1$

foreach $i$ from 2 to $n$ do

$C \leftarrow C \cap h_i$

return $C$
```

Running time: $T_{IH}(n) = n$. How??

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around $C$ to find $s_j, s_j' \in C$ intersecting $h_i$
First Approach

- compute $\cap H$ iteratively
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

IntersectHalfplanes($H$)

Let $H = (h_1, \ldots, h_n)$

$C \leftarrow h_1$

foreach $i$ from 2 to $n$ do

$C \leftarrow C \cap h_i$

return $C$

Running time: $T_{IH}(n) = n$. How??

C := chain of line segments $(s_1, \ldots, s_t)$

Walk around $C$ to find $s_j, s_j' \in C$ intersecting $h_i$

Update $C$
First Approach

- compute $\bigcap H$ iteratively
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

**IntersectHalfplanes($H$)**

Let $H = (h_1, \ldots, h_n)$

$C \leftarrow h_1$

foreach $i$ from 2 to $n$

$C \leftarrow C \cap h_i$

return $C$

Running time: $T_{IH}(n) = n \cdot O(n)$

$C :=$ chain of line segments $(s_1, \ldots, s_t)$

Walk around $C$ to find $s_j, s_j' \in C$ intersecting $h_i$

Update $C$
First Approach

• compute $\cap H$ iteratively

• walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

**IntersectHalfplanes($H$)**

Let $H = (h_1, \ldots, h_n)$

$C \leftarrow h_1$

foreach $i$ from 2 to $n$ do

$C \leftarrow C \cap h_i$

return $C$

$C := \text{chain of line segments } (s_1, \ldots, s_t)$

Walk around $C$ to find $s_j, s_j' \in C$ intersecting $h_i$

Update $C$

Running time: $T_{IH}(n) = n \cdot O(n)$

Total Time: $O(n^2)$ :(

First Approach

- compute $\cap H$ iteratively
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

```
IntersectHalfplanes(H)
Let $H = (h_1, \ldots, h_n)$
$C \leftarrow h_1$
foreach $i$ from 2 to $n$ do
  $C \leftarrow C \cap h_i$
return $C$
```

$T_{IH}(n) = n \cdot O(n)$

Total Time: $O(n^2)$

Exercise: Compute $C \cap h_i$ faster.
Second Approach

• compute $\cap H$ via divide and conquer

• walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time
Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

IntersectHalfplanes($H$)

```java
if |$H$| = 1 then
    $C \leftarrow h$, where $\{h\} = H$
else

return $C$
```
Second Approach

- compute $\cap H$ via divide and conquer
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

\[
\text{IntersectHalfplanes}(H) \ni \\
\text{if } |H| = 1 \text{ then} \\
\quad C \leftarrow h, \text{ where } \{h\} = H \\
\text{else} \\
\quad \text{split } H \text{ into sets } H_1 \text{ and } H_2 \text{ with } |H_1|, |H_2| \approx |H|/2 \\
\quad C_1 \leftarrow \text{IntersectHalfplanes}(H_1) \\
\quad C_2 \leftarrow \text{IntersectHalfplanes}(H_2) \\
\quad C \leftarrow \text{IntersectConvexRegions}(C_1, C_2) \\
\text{return } C
\]
Second Approach

- compute $\cap H$ via divide and conquer
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

IntersectHalfplanes($H$)

```plaintext
if $|H| = 1$ then
  $C \leftarrow h$, where $\{h\} = H$
else
  split $H$ into sets $H_1$ and $H_2$ with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow \text{IntersectHalfplanes}(H_1)$
  $C_2 \leftarrow \text{IntersectHalfplanes}(H_2)$
  $C \leftarrow \text{IntersectConvexRegions}(C_1, C_2)$
return $C$
```

Running time:

- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time
Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

IntersectHalfplanes($H$)

\[
\text{if } |H| = 1 \text{ then} \\
C \leftarrow h, \text{ where } \{h\} = H \\
\text{else} \\
\text{split } H \text{ into sets } H_1 \text{ and } H_2 \text{ with } |H_1|, |H_2| \approx |H|/2 \\
C_1 \leftarrow \text{IntersectHalfplanes}(H_1) \\
C_2 \leftarrow \text{IntersectHalfplanes}(H_2) \\
C \leftarrow \text{IntersectConvexRegions}(C_1, C_2) \\
\text{return } C
\]

Running time: $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$
Second Approach

- compute $\cap H$ via divide and conquer
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

**IntersectHalfplanes($H$)**

```plaintext
if $|H| = 1$ then
    $C \leftarrow h$, where $\{h\} = H$
else
    split $H$ into sets $H_1$ and $H_2$ with $|H_1|, |H_2| \approx |H|/2$
    $C_1 \leftarrow \text{IntersectHalfplanes}(H_1)$
    $C_2 \leftarrow \text{IntersectHalfplanes}(H_2)$
    $C \leftarrow \text{IntersectConvexRegions}(C_1, C_2)$
return $C$
```

**Running time:**

$$T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$$
Second Approach

- compute $\cap H$ via divide and conquer
- walk $\partial (\cap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

```
IntersectHalfplanes(H)
if |H| = 1 then
    C ← h, where \{h\} = H
else
    split $H$ into sets $H_1$ and $H_2$ with $|H_1|, |H_2| \approx |H|/2$
    $C_1$ ← IntersectHalfplanes($H_1$)
    $C_2$ ← IntersectHalfplanes($H_2$)
    $C$ ← IntersectConvexRegions($C_1, C_2$)
return C
```

Running time: $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$

How complex can the new region be?
Second Approach

- compute $\bigcap H$ via divide and conquer
- walk $\partial (\bigcap H)$, find vertex $x$ w/ $cx$ maximum, $O(n)$ time

**IntersectHalfplanes$(H)$**

if $|H| = 1$ then 
  $C \leftarrow h$, where $\{h\} = H$
else 
  split $H$ into sets $H_1$ and $H_2$ with $|H_1|, |H_2| \approx |H|/2$
  $C_1 \leftarrow \text{IntersectHalfplanes}(H_1)$
  $C_2 \leftarrow \text{IntersectHalfplanes}(H_2)$
  $C \leftarrow \text{IntersectConvexRegions}(C_1, C_2)$
return $C$

**Running time:** $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$

How complex can the new region be?
Intersecting Convex Regions

$C_1$  \hspace{1cm}  $C_2$
Intersecting Convex Regions

\( C_1 \)

\( C_2 \)

\( \ell \)
Intersecting Convex Regions

How many segments on $\ell$?
Intersecting Convex Regions

How many segments on $\ell$?
Intersecting Convex Regions

How many segments on \( \ell \)?
Intersecting Convex Regions

How many segments on \( \mathcal{L} \)?
Intersecting Convex Regions

\( \ell \)

- \( \overline{\text{leftEdge}\, C_1} \)
- \( \overline{\text{left}\,(C_1)} \)
- \( \overline{\text{right}\,(C_1)} \)
- \( \overline{\text{rightEdge}\, C_1} \)
- \( \overline{\text{leftEdge}\, C_2} \)
- \( \overline{\text{right}\,(C_2)} \)
- \( \overline{\text{rightEdge}\, C_2} \)
- \( \overline{\text{Lleft}\,(C_2)} \)
- \( \overline{\text{Lright}\,(C_2)} \)

How many segments on \( \ell \)?

Is > 4 possible?
Intersecting Convex Regions

How many segments on $\ell$?

$\mathcal{L}_{\text{left}}(C_1)$

$\mathcal{L}_{\text{left}}(C_2)$

$\mathcal{L}_{\text{right}}(C_1)$

$\mathcal{L}_{\text{right}}(C_2)$

leftEdge$C_1$

rightEdge$C_1$

leftEdge$C_2$

rightEdge$C_2$

$C_1$

$C_2$

Is $> 4$ possible?

No!
Intersecting Convex Regions

How does this help us?

Is \( > 4 \) possible?

No!

How many segments on \( \ell \)?

\( \mathcal{L}_{\text{left}}(C_1) \)

leftEdge\( C_1 \)

leftEdge\( C_2 \)

rightEdge\( C_1 \)

rightEdge\( C_2 \)

\( \mathcal{L}_{\text{right}}(C_1) \)

\( \mathcal{L}_{\text{right}}(C_2) \)

\( C_1 \)

\( C_2 \)
Theorem. The intersection of two convex polygonal regions can be computed in linear time.
Sweep-Line Algorithm
Sweep-Line Algorithm

- $C_1$
- $C_2$

sweep line

events
Sweep-Line Algorithm
Sweep-Line Algorithm

The diagram illustrates two curves, $C_1$ and $C_2$, with event points and line segments connecting them. The event points are marked along the vertical axis.
Sweep-Line Algorithm
Sweep-Line Algorithm

next event?

C1

C2
Sweep-Line Algorithm

next event?

events

$C_1$

$C_2$
Sweep-Line Algorithm

C1

C2

events
Sweep-Line Algorithm

$C_1$

$C_2$

events
Sweep-Line Algorithm

$C_1$

$C_2$

events
Sweep-Line Algorithm

C_1

C_2
Sweep-Line Algorithm

$C_1$  $C_2$

events
Sweep-Line Algorithm
Sweep-Line Algorithm

next event?

$C_1$

$C_2$

events
Sweep-Line Algorithm

next event?

$C_1$

$C_2$

events
Sweep-Line Algorithm

\[ C_1 \]

\[ C_2 \]
Sweep-Line Algorithm

Done, since we have finished C!
Data Structures

1) event (-point) queue $Q$

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

$p \prec q \iff \text{def.}$

2) (sweep-line) status $T$
Data Structures

1) event (-point) queue $\mathcal{Q}$

$p \prec q \iff_{\text{def.}} y_p > y_q$

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $\mathcal{Q}$

\[ p \prec q \iff \text{def.} \quad y_p > y_q \]

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $\mathcal{Q}$

$p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

$p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

2) (sweep-line) status $T$
Data Structures

1) event (-point) queue $Q$

\[ p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

2) (sweep-line) status $T$
1) event (-point) queue $Q$

\[ p \prec q \iff y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in sorted order acc. to $\prec$

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

$p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

Store event pts in sorted order acc. to $\prec$ ... linear time?

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

\[ p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in \textit{sorted order} acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

2) (sweep-line) status $T$
1) event (-point) queue $Q$

$p \prec q \iff_{\text{def.}} y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$

Store event pts in *sorted order* acc. to $\prec$

nextEvent(): either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime?

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $\mathcal{Q}$

\[ p \prec q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in sorted order acc. to $\prec$

nextEvent(): either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments $\leq 4$ :)

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

\[ p \prec q \iff_{\text{def.}} y_p > y_q \text{ or } (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in \emph{sorted order} acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments $\leq 4$ :)

2) (sweep-line) status $\mathcal{T}$
Data Structures

1) event (-point) queue $Q$

\[ p \prec q \iff \text{def. } y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q) \]

Store event pts in sorted order acc. to $\prec$

nextEvent() : either, next point (by $\prec$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments $\leq 4$ :)

2) (sweep-line) status $\mathcal{T}$

Store the segments intersected by $\ell$ in left-to-right order.
Data Structures

1) event (-point) queue $Q$

$p ≺ q \iff_{\text{def.}} y_p > y_q \text{ or } (y_p = y_q \text{ and } x_p < x_q)$

Store event pts in \textit{sorted order} acc. to $≺$

nextEvent() : either, next point (by $≺$), or the intersection pt. of two active segments (below the sweep-line)

... runtime? $O(1)$, since num. active segments $\leq 4$ :)

2) (sweep-line) status $\mathcal{T}$

Store the segments intersected by $\ell$ in left-to-right order.
Also, maintain the new convex hull.
Second Approach: Halfplane Intersection

**Theorem.** The intersection of two convex polygonal regions can be computed in linear time.
Second Approach: Halfplane Intersection

**Theorem.** The intersection of two convex polygonal regions can be computed in linear time.

```plaintext
IntersectHalfplanes(H)

if |H| = 1 then C ← h, where \{h\} = H
else
    split H into sets H₁ and H₂ with |H₁|, |H₂| ≈ |H|/2
    C₁ ← IntersectHalfplanes(H₁)
    C₂ ← IntersectHalfplanes(H₂)
    C ← IntersectConvexRegions(C₁, C₂)

return C
```

**Running time:**

\[ T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n) \]
Theorem. The intersection of two convex polygonal regions can be computed in linear time.

$$\text{IntersectHalfplanes}(H)$$

\[
\begin{align*}
\text{if } |H| = 1 \text{ then } C &\leftarrow h, \text{ where } \{h\} = H \\
\text{else} \\
\quad \text{split } H \text{ into sets } H_1 \text{ and } H_2 \text{ with } |H_1|, |H_2| \approx |H|/2 \\
\quad C_1 &\leftarrow \text{IntersectHalfplanes}(H_1) \\
\quad C_2 &\leftarrow \text{IntersectHalfplanes}(H_2) \\
\quad C &\leftarrow \text{IntersectConvexRegions}(C_1, C_2)
\end{align*}
\]

return $C$

Running time: $T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n)$

Corollary. The intersection of $n$ half planes can be computed in $O(n \log n)$ time.
Second Approach: Halfplane Intersection

**Theorem.** The intersection of two convex polygonal regions can be computed in linear time.

\[
\text{IntersectHalfplanes}(H)
\begin{align*}
\text{if } |H| &= 1 \text{ then } C \leftarrow h, \text{ where } \{h\} = H \\
\text{else} \\
\quad &\text{split } H \text{ into sets } H_1 \text{ and } H_2 \text{ with } |H_1|, |H_2| \approx |H|/2 \\
\quad &\quad C_1 \leftarrow \text{IntersectHalfplanes}(H_1) \\
\quad &\quad C_2 \leftarrow \text{IntersectHalfplanes}(H_2) \\
\quad &\quad C \leftarrow \text{IntersectConvexRegions}(C_1, C_2)
\end{align*}
\]

**Running time:** \[ T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n) \]

**Corollary.** The intersection of \( n \) half planes can be computed in \( O(n \log n) \) time.

Can we do better?
A Small Trick: Make Solution Unique

\[ \cap H = \emptyset \]
\[ \cap H \text{ unbd. in dir. } c \]
\[ \cap H \text{ bounded.} \]
A Small Trick: Make Solution Unique

\[ \bigcap H = \emptyset \quad \bigcap H \text{ unbd. in dir. } c \quad \bigcap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)
A Small Trick: Make Solution Unique

\[ \cap H = \emptyset \quad \cap H \text{ unbounded in dir. } c \quad \cap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)
A Small Trick: Make Solution Unique

\[ \bigcap H = \emptyset \]
\[ \bigcap H \text{ unbd. in dir. } c \]
\[ \bigcap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)
A Small Trick: Make Solution Unique

- \( \cap H = \emptyset \)
- \( \cap H \) unbd. in dir. \( c \)
- \( \cap H \) bounded.

• Add two bounding halfplanes \( m_1 \) and \( m_2 \)
A Small Trick: Make Solution Unique

\[ \cap H = \emptyset \]

\[ \cap H \text{ unbd. in dir. } c \]

\[ \cap H \text{ bounded.} \]

• Add two bounding halfplanes \( m_1 \) and \( m_2 \)

\[
m_1 = \begin{cases} 
x \leq M & \text{if } c_x > 0, \\
x \geq M & \text{otherwise,} 
\end{cases}
\]

for some sufficiently large \( M \)
A Small Trick: Make Solution Unique

\[ \bigcap H = \emptyset \]

\[ \bigcap H \text{ unbd. in dir. } c \]

\[ \bigcap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)

\[
m_1 = \begin{cases} x \leq M & \text{if } c_x > 0, \\ x \geq M & \text{otherwise,} \end{cases} \text{ for some sufficiently large } M
\]

\[
m_2 = \begin{cases} y \leq M & \text{if } c_y > 0, \\ y \geq M & \text{otherwise.} \end{cases}
\]
A Small Trick: Make Solution Unique

\[ \bigcap H = \emptyset \]
\[ \bigcap H \text{ unbd. in dir. } c \]
\[ \bigcap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)

\[ m_1 = \begin{cases} 
  x \leq M & \text{if } c_x > 0, \\
  x \geq M & \text{otherwise,} 
\end{cases} \]

\[ m_2 = \begin{cases} 
  y \leq M & \text{if } c_y > 0, \\
  y \geq M & \text{otherwise.} 
\end{cases} \]

for some sufficiently large \( M \)

Idea: \( M \) based on obj.fct. \( c \).
see §4.5 of CG: A&A for more on unbounded LPs.
A Small Trick: Make Solution Unique

\[ \bigcap H = \emptyset \quad \bigcap H \text{ unbd. in dir. } c \quad \bigcap H \text{ bounded.} \]

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)

\[
m_1 = \begin{cases} 
  x \leq M & \text{if } c_x > 0, \\
  x \geq M & \text{otherwise,}
\end{cases}
\]

\[
m_2 = \begin{cases} 
  y \leq M & \text{if } c_y > 0, \\
  y \geq M & \text{otherwise.}
\end{cases}
\]

- Take the lexicographically largest solution.

Idea: \( M \) based on obj.fct. \( c \).
see §4.5 of CG: A&A for more on unbounded LPs.
A Small Trick: Make Solution Unique

- Add two bounding halfplanes $m_1$ and $m_2$

$$m_1 = \begin{cases} 
  x \leq M & \text{if } cx > 0, \\
  x \geq M & \text{otherwise,}
\end{cases}$$

for some sufficiently large $M$

$$m_2 = \begin{cases} 
  y \leq M & \text{if } cy > 0, \\
  y \geq M & \text{otherwise.}
\end{cases}$$

- Take the lexicographically largest solution.

Idea: $M$ based on obj.fct. $c$. see §4.5 of CG: A&A for more on unbounded LPs.
A Small Trick: Make Solution Unique

\( \cap H = \emptyset \quad \cap H \text{ unbd. in dir. } c \quad \cap H \text{ bounded.} \)

- Add two bounding halfplanes \( m_1 \) and \( m_2 \)

\[
m_1 = \begin{cases} 
  x \leq M & \text{if } c_x > 0, \\
  x \geq M & \text{otherwise,}
\end{cases}
\]

for some sufficiently large \( M \)

\[
m_2 = \begin{cases} 
  y \leq M & \text{if } c_y > 0, \\
  y \geq M & \text{otherwise.}
\end{cases}
\]

- Take the lexicographically largest solution.

\( \Rightarrow \) Set of solutions is either empty or a uniquely defined pt.

Idea: \( M \) based on obj.fct. \( c \).
see §4.5 of CG: A&A for more on unbounded LPs.
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just *one* (optimal) point!
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just *one* (optimal) point!

\[
2DBoundedLP(H, c, m_1, m_2)
\]

\[
H_0 = \{m_1, m_2\}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

\[
\text{return } v_n
\]
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just *one* (optimal) point!

\[
2D\text{BoundedLP}(H, c, m_1, m_2)
\]

\[
\begin{align*}
H_0 &= \{m_1, m_2\} \\
\nu_0 &\leftarrow \text{corner of } m_1 \cap m_2 \\
\text{for } i &\leftarrow 1 \text{ to } n \text{ do} \\
&\quad \text{if } \nu_{i-1} \in h_i \text{ then} \\
\text{return } \nu_n
\end{align*}
\]
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

$2DBoundedLP(H, c, m_1, m_2)$

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

for $i \leftarrow 1$ to $n$ do
  if $v_{i-1} \in h_i$ then
    $v_i \leftarrow$
  else
    $v_i \leftarrow$

$H_i = H_{i-1} \cup \{h_i\}$

return $v_n$
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

\[
2DBoundedLP(H, c, m_1, m_2)
\]

\[
\begin{align*}
H_0 &= \{m_1, m_2\} \\
v_0 &\leftarrow \text{corner of } m_1 \cap m_2 \\
\text{for } i &\leftarrow 1 \text{ to } n \text{ do} \\
&\quad \text{if } v_{i-1} \in h_i \text{ then} \\
&\quad \quad v_i \leftarrow v_{i-1} \\
&\quad \text{else} \\
&\quad \quad v_i \leftarrow \ldots \\
&\text{ } \\
&H_i = H_{i-1} \cup \{h_i\} \\
\text{return } v_n
\end{align*}
\]
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

```
2DBoundedLP(H, c, m₁, m₂)

H₀ = \{m₁, m₂\}
v₀ ← corner of m₁ \cap m₂
for i ← 1 to n do
    if vᵢ₋₁ ∈ hᵢ then
        vᵢ ← vᵢ₋₁
    else
        vᵢ ← 1DBoundedLP(π∂hᵢ(Hᵢ₋₁), π∂hᵢ(c))
    if vᵢ = nil then
        return nil
    Hi = Hᵢ₋₁ ∪ \{hᵢ\}
return vₙ
```
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

2DBoundedLP($H, c, m_1, m_2$)

\[
H_0 = \{m_1, m_2\}
\]
\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]
\[\text{for } i \leftarrow 1 \text{ to } n \text{ do}\]
\[
\quad \text{if } v_{i-1} \in h_i \text{ then}\n\quad \quad v_i \leftarrow v_{i-1}
\]
\[
\text{else}
\quad v_i \leftarrow 1\text{DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))
\]
\[
\quad \text{if } v_i = \text{nil } \text{ then}\n\quad \quad \text{return nil}
\]
\[
\quad H_i = H_{i-1} \cup \{h_i\}
\]
\[\text{return } v_n\]
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

\[
\text{2DBoundedLP}(H, c, m_1, m_2)
\]

\[
H_0 = \{m_1, m_2\}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do}
\]

\[
\begin{align*}
\quad & \text{if } v_{i-1} \in h_i \text{ then} \\
& \quad v_i \leftarrow v_{i-1}
\end{align*}
\]

\[
\begin{align*}
\quad & \text{else} \\
& \quad v_i \leftarrow \text{1DBoundedLP}\left(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)\right)
\end{align*}
\]

\[
\begin{align*}
\quad & \text{if } v_i = \text{nil} \text{ then} \\
& \quad \text{return } \text{nil}
\end{align*}
\]

\[
H_i = H_{i-1} \cup \{h_i\}
\]

\[
\text{return } v_n
\]
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

\[\text{2DBoundedLP}(H, c, m_1, m_2)\]

\[
H_0 = \{m_1, m_2\}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

for \(i \leftarrow 1\) to \(n\) do

\[
\begin{cases}
\text{if } v_{i-1} \in h_i \text{ then} & v_i \leftarrow v_{i-1} \\
\text{else} & v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))
\end{cases}
\]

if \(v_i = \text{nil}\) then

\[
\text{return } \text{nil}
\]

\[H_i = H_{i-1} \cup \{h_i\}\]

return \(v_n\)
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

2DBoundedLP($H, c, m_1, m_2$)

$$H_0 = \{m_1, m_2\}$$
$$v_0 \leftarrow \text{corner of } m_1 \cap m_2$$

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

$v_i \leftarrow v_{i-1}$

else

$v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$

if $v_i = \text{nil}$ then

return nil

return $v_n$
Incremental Approach

Idea: Don’t compute $\bigcap H$, but just one (optimal) point!

2DBoundedLP($H, c, m_1, m_2$)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow \text{corner of } m_1 \cap m_2$

for $i \leftarrow 1$ to $n$ do
  if $v_{i-1} \in h_i$ then
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$
    if $v_i = \text{nil}$ then
      return $\text{nil}$
    $H_i = H_{i-1} \cup \{h_i\}$
  return $v_n$
Incremental Approach

Idea: Don’t compute ∩ H, but just one (optimal) point!

2DBoundedLP\((H, c, m_1, m_2)\)

\[
\begin{align*}
H_0 &= \{m_1, m_2\} \\
v_0 &\leftarrow \text{corner of } m_1 \cap m_2 \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
&\quad \text{if } v_{i-1} \in h_i \text{ then} \\
&\quad \quad v_i \leftarrow v_{i-1} \\
&\quad \text{else} \\
&\quad \quad v_i \leftarrow \text{1DBoundedLP}\left(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)\right) \\
&\quad \quad \text{if } v_i = \text{nil} \text{ then} \\
&\quad \quad \quad \text{return nil} \\
&\quad \quad H_i = H_{i-1} \cup \{h_i\} \\
\text{return } v_n
\end{align*}
\]
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

\[2DBoundedLP(H, c, m_1, m_2)\]

\[
H_0 = \{m_1, m_2\}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

\[
v_i \leftarrow v_{i-1}
\]

else

\[
v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))
\]

if $v_i = \text{nil}$ then

\[
\text{return nil}
\]

\[
H_i = H_{i-1} \cup \{h_i\}
\]

return $v_n$
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

$$2DBoundedLP(H, c, m_1, m_2)$$

$$H_0 = \{m_1, m_2\}$$

$$v_0 \leftarrow \text{corner of } m_1 \cap m_2$$

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

$$v_i \leftarrow v_{i-1}$$

else

$$v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$$

if $v_i = \text{nil}$ then

return nil

$H_i = H_{i-1} \cup \{h_i\}$

return $v_n$
Incremental Approach

**Idea:** Don’t compute \( \cap H \), but just one (optimal) point!

\[
H_0 = \{ m_1, m_2 \}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

for \( i \leftarrow 1 \) to \( n \) do

if \( v_{i-1} \in h_i \) then

\[
v_i \leftarrow v_{i-1}
\]

else

\[
v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))
\]

if \( v_i = \text{nil} \) then

\[
\text{return nil}
\]

\[
H_i = H_{i-1} \cup \{ h_i \}
\]

return \( v_n \)
Incremental Approach

**Idea:** Don’t compute $\bigcap H$, but just one (optimal) point!

**2DBoundedLP**(\(H, c, m_1, m_2\))

- \(H_0 = \{m_1, m_2\}\)
- \(v_0 \leftarrow \text{corner of } m_1 \cap m_2\)
- For \(i \leftarrow 1\) to \(n\) do
  - If \(v_{i-1} \in h_i\) then
    - \(v_i \leftarrow v_{i-1}\)
  - Else
    - \(v_i \leftarrow 1\text{DBoundedLP} (\pi_{\partial h_i} (H_{i-1}), \pi_{\partial h_i} (c))\)
    - If \(v_i = \text{nil}\) then
      - Return nil
    - \(H_i = H_{i-1} \cup \{h_i\}\)
- Return \(v_n\)

**w-c running time:**
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

$2DBoundedLP(H, c, m_1, m_2)$

\[
H_0 = \{m_1, m_2\} \\
v_0 \leftarrow \text{corner of } m_1 \cap m_2 \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\quad \text{if } v_{i-1} \in h_i \text{ then} \\
\quad \quad \quad v_i \leftarrow v_{i-1} \\
\quad \text{else} \\
\quad \quad \quad v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)) \\
\quad \text{if } v_i = \text{nil} \text{ then} \\
\quad \quad \quad \text{return} \text{ nil} \\
\text{return } v_n \\
\]

$\partial h_i$

$\pi_{\partial h_i}(c)$

w-c running time:
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

$2DBoundedLP(H, c, m_1, m_2)$

\[
\begin{align*}
H_0 &= \{m_1, m_2\} \\
v_0 &\leftarrow \text{corner of } m_1 \cap m_2 \\
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
&\quad \text{if } v_{i-1} \in h_i \text{ then} \\
&\quad \quad v_i \leftarrow v_{i-1} \\
&\quad \text{else} \\
&\quad \quad v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)) \\
&\quad \quad \text{if } v_i = \text{nil} \text{ then} \\
&\quad \quad \quad \text{return nil} \\
&\quad \quad \text{H}_i = H_{i-1} \cup \{h_i\} \\
\text{return } v_n
\end{align*}
\]

w-c running time: $O(1)$
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

2DBoundedLP($H, c, m_1, m_2$)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

for $i \leftarrow 1$ to $n$ do
  if $v_{i-1} \in h_i$ then
    $v_i \leftarrow v_{i-1}$
  else
    $v_i \leftarrow 1$DBoundedLP($\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)$)
    if $v_i = \text{nil}$ then
      return $\text{nil}$
    $H_i = H_{i-1} \cup \{h_i\}$
  return $v_n$

w-c running time:

- $O(1)$
- $O(i)$

Idea: Don’t compute $\bigcap H$, but just one (optimal) point!
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

2DBoundedLP($H, c, m_1, m_2$)

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

$v_i \leftarrow v_{i-1}$

else

$v_i \leftarrow \text{1DBoundedLP}(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$

if $v_i = \text{nil}$ then

return $\text{nil}$

$H_i = H_{i-1} \cup \{h_i\}$

return $v_n$

w-c running time:

$T(n) = \sum_{i=1}^{n} O(i) = O(n)$
Incremental Approach

Idea: Don’t compute $\cap H$, but just one (optimal) point!

2DBoundedLP$(H, c, m_1, m_2)$

$H_0 = \{m_1, m_2\}$
$v_0 \leftarrow$ corner of $m_1 \cap m_2$

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

$v_i \leftarrow v_{i-1}$

else

$v_i \leftarrow 1$DBoundedLP$(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$

if $v_i = \text{nil}$ then

return $\text{nil}$

$H_i = H_{i-1} \cup \{h_i\}$

return $v_n$

w-c running time:

$T(n) = \sum_{i=1}^{n} O(i) = O(n^2)$
Incremental Approach

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

Randomized

\[
2DBoundedLP(H, c, m_1, m_2)
\]

\[
H_0 = \{m_1, m_2\}
\]

\[
v_0 \leftarrow \text{corner of } m_1 \cap m_2
\]

\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do}
\]

- if $v_{i-1} \in h_i$ then
  - $v_i \leftarrow v_{i-1}$
- else
  - $v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c))$

  - if $v_i = \text{nil}$ then
    - return $\text{nil}$
  - else
    - $H_i = H_{i-1} \cup \{h_i\}$

return $v_n$

\[
O(1)
\]

\[
O(i)
\]

**w-c running time:**

\[
T(n) = \sum_{i=1}^{n} O(i) = O(n^2)
\]
**Incremental Approach**

**Idea:** Don’t compute $\cap H$, but just one (optimal) point!

**Randomized**

\[
2DBoundedLP(H, c, m_1, m_2)
\]

compute random permutation of $H$

\[ H_0 = \{m_1, m_2\} \]

\[ v_0 \leftarrow \text{corner of } m_1 \cap m_2 \]

for $i \leftarrow 1$ to $n$ do

if $v_{i-1} \in h_i$ then

\[ v_i \leftarrow v_{i-1} \]

else

\[ v_i \leftarrow 1DBoundedLP(\pi_{\partial h_i}(H_{i-1}), \pi_{\partial h_i}(c)) \]

if $v_i = \text{nil}$ then

\[ \text{return nil} \]

\[ H_i = H_{i-1} \cup \{h_i\} \]

return $v_n$

\[
\begin{align*}
\text{w-c running time:} & \\
T(n) &= \sum_{i=1}^{n} O(i) = \\
&= O(n^2) \quad :-(
\end{align*}
\]
Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.
Result

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).
**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is $E[T_{2d}(n)] = \ldots$
**Result**

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 
1 & \text{if } v_{i-1} \notin h_i, \\
0 & \text{else.}
\end{cases}$ (indicator random variable).

Then the expected running time is

$$
E[T_{2d}(n)] = E[\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)]
$$
**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$
Result

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E\left[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)\right]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i)$$
**Result**

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \not\in h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

\[
E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\
= O(n) + \sum E[X_i] \cdot O(i) \\
= O(n) + \sum \Pr[X_i = 1] \cdot O(i)
\]

We fix the $i$ random halfplanes in $H_i$. 
Theorem. The 2D bounded LP problem can be solved in $O(n)$ expected time.

Proof. Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is
\[
E[T_{2d}(n)] = E[\sum_{i=1}^{n}(1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\
= O(n) + \sum E[X_i] \cdot O(i) \\
= O(n) + \sum \Pr[X_i = 1] \cdot O(i)
\]

We fix the $i$ random halfplanes in $H_i$.

$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}$. 
**Theorem.** The 2D bounded LP problem can be solved in \( O(n) \) expected time.

**Proof.** Let \( X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases} \) (indicator random variable).

Then the expected running time is

\[
E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\
= O(n) + \sum E[X_i] \cdot O(i) \\
= O(n) + \sum \Pr[X_i = 1] \cdot O(i)
\]

We fix the \( i \) random halfplanes in \( H_i \).

\( \Pr[X_i = 1] = \) probability that the optimal solution changes when \( h_i \) is added to \( H_{i-1} \).
The 2D bounded LP problem can be solved in $O(n)$ expected time.

Let $X_i = \begin{cases} 
1 & \text{if } v_{i-1} \notin h_i, \\
0 & \text{else.} 
\end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i)$$

We fix the $i$ random halfplanes in $H_i$.

$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}.$

$= \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.$
## Result

### Theorem.

The 2D bounded LP problem can be solved in $O(n)$ expected time.

### Proof.

Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E\left[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)\right]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i)$$

We fix the $i$ random halfplanes in $H_i$.

$\Pr[X_i = 1] =$ probability that the optimal solution changes when $h_i$ is added to $H_{i-1}$.

$= probability that the optimal solution changes when $h_i$ is removed from $H_i$.

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$. 


Result

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is
\[
E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] \\
= O(n) + \sum E[X_i] \cdot O(i) \\
= O(n) + \sum \Pr[X_i = 1] \cdot O(i)
\]

We fix the $i$ random halfplanes in $H_i$.

\[
\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}.
\]

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i$. 

\[
\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.
\]

$\leq 2/i.$
Result

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.**

Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \not\in h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i)$$

We fix the $i$ random halfplanes in $H_i$.

$$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}.$$  

$$= \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.$$  

$$\leq 2/i. \text{ This is independent of the choice of } H_i.$$  

i.e., when $v_i \in \partial h_i$ and $v_i \in \partial h_j$ for exactly one $j < i.$
**Result**

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i)$$

We fix the $i$ random halfplanes in $H_i$.

$$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}.$$  

$$= \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.$$  

$$\leq 2/i. \text{ This is independent of the choice of } H_i.$$
Result

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.** Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)] = O(n) + \sum E[X_i] \cdot O(i) = O(n) + \sum \Pr[X_i = 1] \cdot O(i) = O(n).$$

We fix the $i$ random halfplanes in $H_i$.

$\Pr[X_i = 1] =$ probability that the optimal solution changes when $h_i$ is added to $H_{i-1}$.

$= \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.$

$\leq \frac{2}{i}. \text{This is independent of the choice of } H_i.$
**Result**

**Theorem.** The 2D bounded LP problem can be solved in $O(n)$ expected time.

**Proof.**

Let $X_i = \begin{cases} 1 & \text{if } v_{i-1} \notin h_i, \\ 0 & \text{else.} \end{cases}$ (indicator random variable).

Then the expected running time is

$$E[T_{2d}(n)] = E[\sum_{i=1}^{n} (1 - X_i) \cdot O(1) + X_i \cdot O(i)]$$

$$= O(n) + \sum E[X_i] \cdot O(i)$$

$$= O(n) + \sum \Pr[X_i = 1] \cdot O(i) = O(n).$$

We fix the $i$ random halfplanes in $H_i$.

$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is added to } H_{i-1}.$

$\Pr[X_i = 1] = \text{probability that the optimal solution changes when } h_i \text{ is removed from } H_i.$

$\leq 2/i.$ This is independent of the choice of $H_i$.

Proof technique: Backward analysis!
Alt. for Intersecting Convex Regions

Use sweep-line alg. for \textbf{map overlay} (line-segment intersections)!

Running time $T_{MO}(n) =$

\textbf{CG: A & A §2}
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$, where $I = \#$ intersection points.
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$,
where $I = \#$ intersection points.
here: $I \leq \ldots$

CG: A & A §2
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$,

where $I = \#$ intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR
Alt. for Intersecting Convex Regions

Use sweep-line alg. for **map overlay** (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$,
where $I = \#$ intersection points.

*here:* $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$

CG: A & A §2
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$,

where $I = \#\text{ intersection points}$.

Here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$

$\leq 2T_{IH}(n/2) + O(n \log n)$

$\in$
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time \( T_{\text{MO}}(n) = O((n + I) \log n) \),
where \( I = \# \) intersection points.

Running time \( T_{\text{IH}}(n) = 2T_{\text{IH}}(n/2) + T_{\text{ICR}}(n) \)
\[ \leq 2T_{\text{IH}}(n/2) + O(n \log n) \]
\[ \in O(n \log^2 n) \]
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$,
where $I =$ # intersection points.

here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$
$\leq 2T_{IH}(n/2) + O(n \log n)$
$\in O(n \log^2 n)$

As this is more general, it is unsurprisingly worse ... *

* it can happen sometimes that general algorithms give optimal runtimes for special cases
Alt. for Intersecting Convex Regions

Use sweep-line alg. for map overlay (line-segment intersections)!

Running time $T_{MO}(n) = O((n + I) \log n)$, where $I = \#$ intersection points.

Here: $I \leq n \rightarrow O(n \log n)$ for ICR

Running time $T_{IH}(n) = 2T_{IH}(n/2) + T_{ICR}(n)$

$\leq 2T_{IH}(n/2) + O(n \log n)$

$\in O(n \log^2 n)$

As this is more general, it is unsurprisingly worse ... *

$\rightsquigarrow$ Better to use specialized algorithm for intersecting convex regions/polygons

* it can happen sometimes that general algorithms give optimal runtimes for special cases