# Linear Programming & Halfplane Intersection

## Example:

A company produces tables and chairs.
The profit for a chair is $2, and for a table $4.
Machine group A needs 4 hours to produce a chair
and 6 " " " " table
Machine group B needs 2 hours to produce a chair
and 6 hours " " " table
Per day there are at most 120 working hours for group A
and at most 72 " " " " B

How many chairs and tables should the company produce per day in order to maximize the profit?

Variables: $c_A$: # chairs produced on machine group A
$c_B$: # " " " " " B
$t_A$: # tables " " " " A
$t_B$: # " " " " " B

Constraints: $4 \cdot c_A + 6 \cdot t_A \leq 120$
$2 \cdot c_B + 6 \cdot t_B \leq 72$

Objective function (profit):

Maximize $2 \cdot (c_A + c_B) + 4 \cdot (t_A + t_B)$

## In general:

Variables: $x_1, \dots, x_d$

Constraints:

$$\begin{aligned} h_1&: \quad a_{11}x_1 + \dots + a_{1d}x_d \leq b_1 \\ h_2&: \quad a_{21}x_1 + \dots + a_{2d}x_d \leq b_2 \\ &\vdots \qquad\qquad \vdots \\ h_n&: \quad a_{n1}x_1 + \dots + a_{nd}x_d \leq b_n \end{aligned}$$

Objective function:

$$\text{Maximize} \quad f_{\vec{c}}(\vec{x}) := c_1x_1 + c_2x_2 + \dots + c_dx_d$$

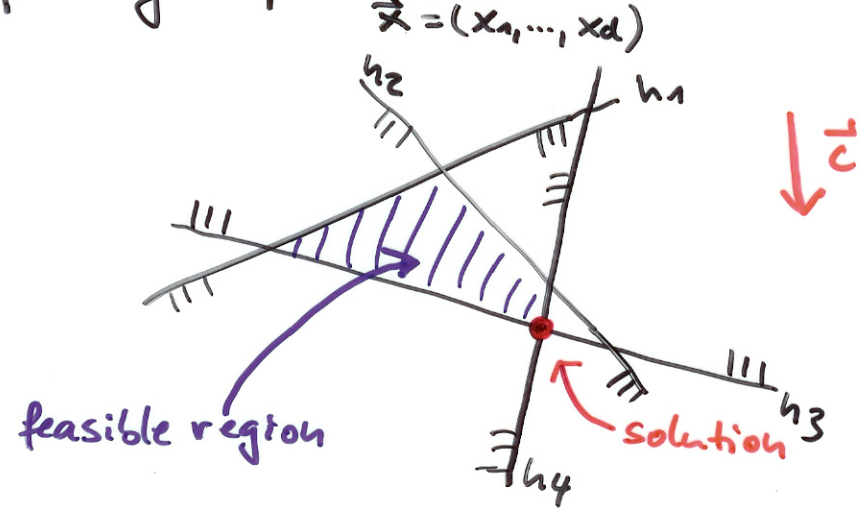
**|| Linear program in $d$ variables with $n$ constraints ||**

- Each constraint $h_i$ is a half-space in $\mathbb{R}^d$
- Set of points in $\mathbb{R}^d$ satisfying all constraints:

  $\bigcap_{i=1}^{n} h_i$ feasible region of the Linear program

- Maximizing $f_{\vec{c}}(\vec{x})$ ; $\vec{c} := (c_1, \dots, c_d)$ ;

  $\hat{=}$ finding a point that is extreme in direction $\vec{c}$




- In the following: Restrict ourselves to $\mathbb{R}^2$, i.e., $d = 2$

# Subproblem: Half-Plane Intersection

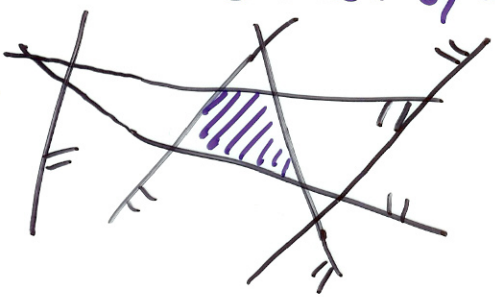
**Given:** Set $H = \{h_1, h_2, \ldots, h_n\}$ of halfplanes

$$h_i: \; a_i x + b_i y \leq c_i \quad ; \quad i = 1..n$$

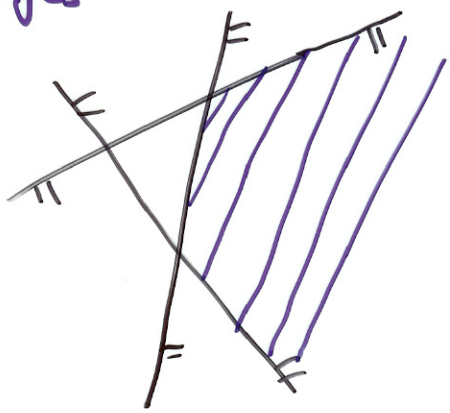
with $a_i, b_i, c_i$ constants.

**Find:** $\bigcap_{i=n}^{n} h_i$ — Set of all points $(x, y) \in \mathbb{R}^2$ satisfying all $n$ constraints at the same time

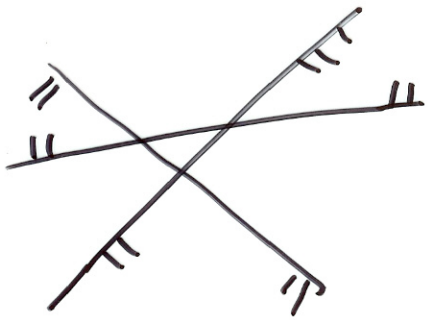
↑ convex polygonal region bounded by at most $n$ edges



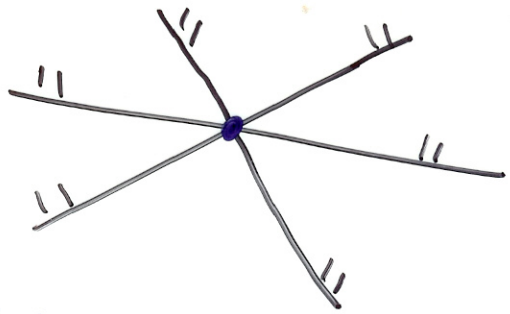
intersection bounded



intersection unbounded



intersection empty



intersection degenerated to a point

# Divide & conquer algorithm:

Algorithm Intersect_Halfplanes (H)

Input: A set H of $n$ half-planes in $\mathbb{R}^2$

Output: The convex polygonal region $C := \bigcap_{h \in H} h$

1. If $|H| = 1$ then
2. $\quad C := h \in H$
3. else Split H into sets $H_1$ and $H_2$ of size $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$
4. $\quad C_1 := $ Intersect_Halfplanes $(H_1)$
5. $\quad C_2 := $ Intersect_Halfplanes $(H_2)$
6. $\quad C := $ Intersect_Convex_Regions $(C_1, C_2)$

- Use Overlay of two planar subdivisions to implement Intersect_Convex_Regions in $O((n+k)\log n)$ time (careful to handle unbounded regions)

  ↑ Output complexity

  = # intersection points = # vertices in $C_1 \cap C_2 = n$

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

  $T(n) = O(n \log^2 n)$ running time

- We did not use the property that $C_1, C_2$ are convex

  $\rightsquigarrow$ Use plane sweep to develop an $O(n)$ implementation for Intersect_Convex_Regions

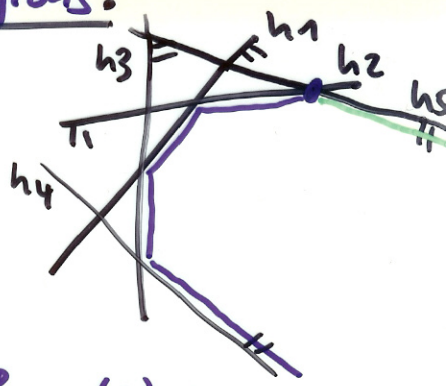
  $\rightsquigarrow$ This yields a runtime for Intersect_Halfplanes of

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

  $T(n) = O(n \log n)$

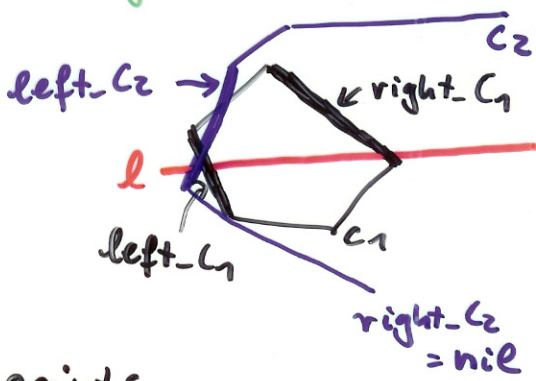
# Plane sweep for Intersect_Convex_Regions:

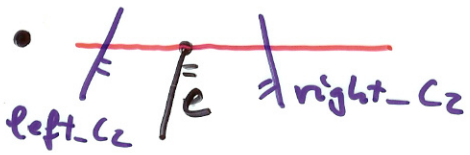
- Store convex polygonal region $C$ by sorted lists $\mathcal{L}_{left}(C)$, $\mathcal{L}_{right}(C)$ of halfplanes representing the left / right boundary of $C$

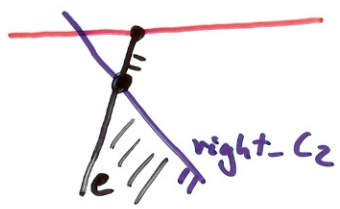


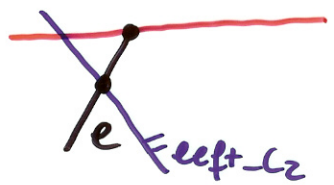

$\mathcal{L}_{left}(C) = h_2, h_1, h_3, h_4$
$\mathcal{L}_{right}(C) = h_5$

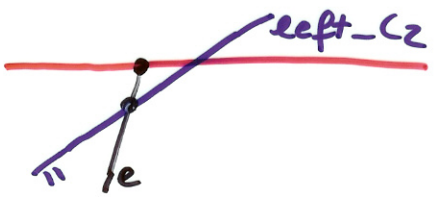
- Sweep from top to bottom, maintain edges left_$C_1$, right_$C_1$, left_$C_2$, right_$C_2$ intersecting the sweep line $\ell$




- Take vertices of $C_1$, $C_2$ as event points
- Event handling (let new edge $e$ be on left boundary of $C_1$):
  - 

→ add halfplane defining $e$ to $\mathcal{L}_{left}(C)$
  - 

→ add halfplane to $\mathcal{L}_{left}(C)$ and halfplane to $\mathcal{L}_{right}(C)$
  - 

or



Add appropriate halfplane to $\mathcal{L}_{left}(C)$

→ constant time event handling

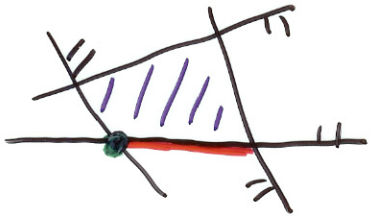
→ $O(n)$ time altogether

# Incremental Linear Programming

- Two-dimensional linear programming (LP) problem

  $(H, \vec{c})$ ; $H = \{h_1, \dots, h_n\}$

- Assume the LP is bounded (otherwise add constraints)
- Assume that there is a unique solution (if any)




take lexicographically smallest solution

- Incremental approach: Add one half-plane after the other

  $H_i := \{h_1, \dots, h_i\}$

  $C_i := h_1 \cap \dots \cap h_i$ ; $C := C_n = \bigcap_{h \in H} h$

  $v_i :=$ unique optimal vertex for feasible region $C_i$ ; $i \geq 2$

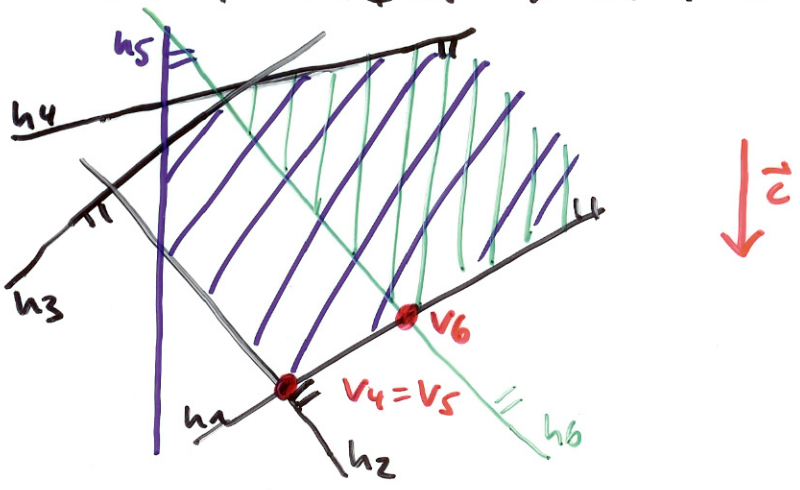
  $\rightsquigarrow C_1 \supseteq C_2 \supseteq \dots \supseteq C_n = C$

  $\rightsquigarrow$ If $C_i = \emptyset$ for some $i \Rightarrow C_j = \emptyset$ for all $j \geq i$

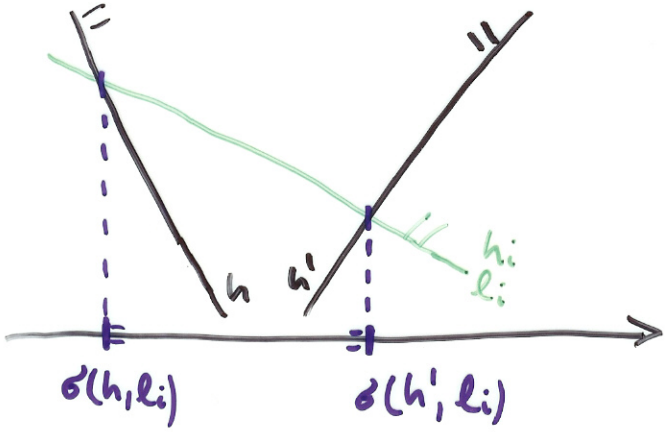
- **Lemma:** Let $2 \leq i \leq n$.

  (i) If $v_{i-1} \in h_i \Rightarrow v_i = v_{i-1}$

  (ii) If $v_{i-1} \notin h_i \Rightarrow C_i = \emptyset$ or $v_i \in l_i :=$ line bounding $h_i$



## Handle case (ii): $v_{i-1} \notin h_i$




Let $\overline{f_{\vec{c}}} : \mathbb{R} \to \mathbb{R}$

$x \longmapsto f_{\vec{c}}(x, \ell_i(x))$

restriction of $f_{\vec{c}}$ to $\ell_i$

Need to solve 1-dimensional LP:

Maximize $\overline{f_{\vec{c}}}(x)$

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

$\Rightarrow$ Feasible region is $[x_{left}, x_{right}]$ with

$$x_{left} := \max_{h \in H_{i-1}} \{\sigma(h, \ell_i) \mid \ell_i \cap h \text{ is bounded to the left}\}$$

$$x_{right} := \min_{h \in H_{i-1}} \{\sigma(h, \ell_i) \mid \ell_i \cap h \text{ is bounded to the right}\}$$

$\Rightarrow$ If $x_{left} > x_{right}$ the LP is infeasible

Otherwise either $x_{left}$ or $x_{right}$ is the optimum (depending on $\overline{f_{\vec{c}}}$)

$\Rightarrow$ We can compute a new optimal vertex $v_i$, or decide that the LP is infeasible, in $O(i)$ time.

## Algorithm 2D-Bounded-LP $(H, \vec{c})$:

Input: A two-dimensional LP $(H, \vec{c})$

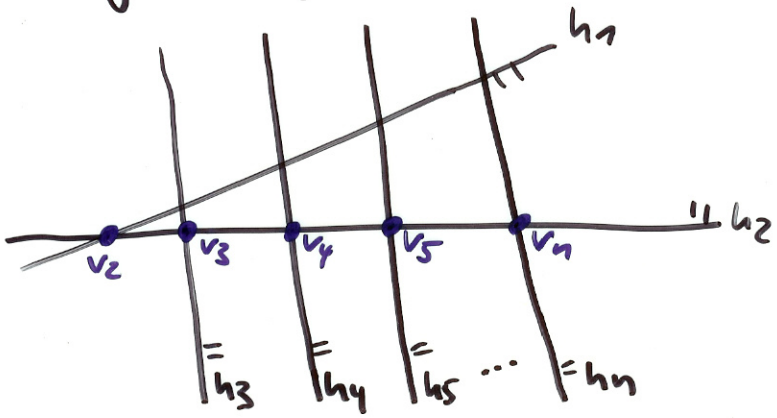
Output: Report if $(H, \vec{c})$ is infeasible.
Otherwise report the lexicographically smallest point that maximizes $f_{\vec{c}}$

1. Let $h_1, \dots, h_n$ be the half-planes of $H$
2. Let $v_2$ be the corner of $C_2$ [$v_2$ exists since we assume the LP is bounded]
3. **for** $i := 3$ **to** $n$ **do**
4. **if** $v_{i-1} \in h_i$ **then**
5. $v_i := v_{i-1}$
6. **else** $v_i :=$ point on $l_i$ that maximizes $f_{\vec{c}}$
7. subject to the constraints in $H_{i-1}$
8. **if** such a point does not exist **then**
9. Report that the LP is infeasible
10. Quit
11. **return** $v_n$

## Analysis:

Runtime $\sum_{i=1}^{n} O(i) = O(n^2)$

Storage $O(n)$




Example which takes $O(i)$ time in every step

# Randomized Incremental Linear Programming

- Insertion order of halfplanes determines runtime
  → varies between $O(n)$ and $O(n^2)$

⇒ Design randomized algorithm

Algorithm 2D_Randomized_Bounded_LP $(H, \vec{c})$:

1. Compute a random permutation $h_1, \dots, h_n$ of the half-planes in $H$ by calling Random_Permutation$(H[1..n])$
2. 2D_Bounded_LP $(H, \vec{c})$

Assume the existence of a random number generator:

Random$(k)$: generates a random integer between 1 and integer $k$ in constant time

Algorithm Random_Permutation$(A)$:

Input: An array $A[1..n]$

Output: The array $A$ with the same elements, rearranged in a random order

```
1. for k := n downto 2 do
2.     rndindex := Random(k)
3.     Swap A[k] with A[rndindex]
```

**Theorem:** 2D_Randomized_Bounded_LP runs in $O(n)$ randomized expected time and $O(n)$ worst-case space

(depending only on random choices of the algorithm; not on the input)

**Proof:**

Random variable $X_i = \begin{cases} 1, & v_{i-1} \notin h_i \\ 0, & \text{else} \end{cases}$

Total time spent in lines 6-7 to solve 1-dimensional LPs, over all half-planes $h_1, \dots, h_n$:

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

Bound expected value (use linearity of expectation):

$$E\left(\sum_{i=1}^{n} O(i) \cdot X_i\right) = \sum_{i=1}^{n} O(i) \cdot E(X_i)$$

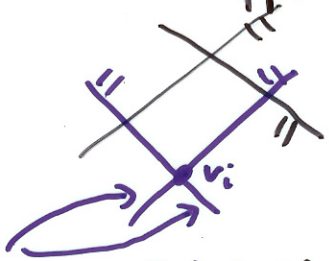
$E(X_i) = \text{Prob}(X_i) = \text{Prob}(v_{i-1} \notin h_i)$

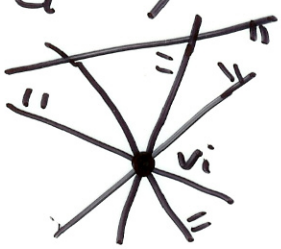
Apply **backwards analysis** to bound $E(X_i)$:

- Fix $H_i = \{h_1, \dots, h_i\}$ → determines $C_i$
- Analyze what happened in the last step when $h_i$ was added
- $\text{Prob}\left(\begin{array}{l}\text{Had to compute new optimal vertex} \\ \text{when adding } h_i\end{array}\right)$

$= \text{Prob}\left(\begin{array}{l}\text{Optimal vertex changes when we} \\ \text{remove a halfplane from } C_i\end{array}\right)$

$\leq \frac{2}{i}$




2 out of $i$ halfplanes defining $v_i$

$\Rightarrow E(X_i) \leq \frac{2}{i}$

$\Rightarrow$ Total expected runtime $\sum_{i=1}^{n} O(i) \cdot \frac{2}{i} = O(n)$