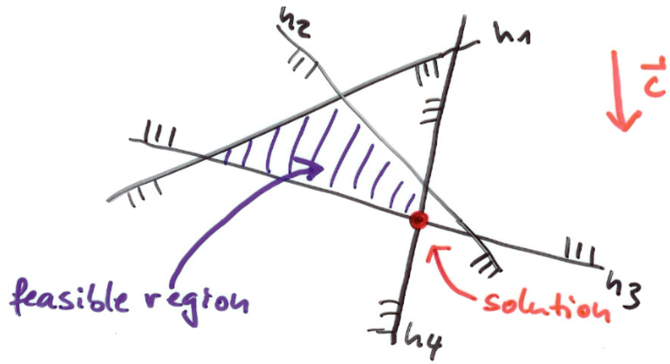
Computational Geometry



Linear Programming & Halfplane Intersection

Michael T. Goodrich

with slides by Carola Wenk, Tulane Univ.

Optimization Problem

- 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 hours for a table. Machine group B needs 2 hours to produce a chair, and 6 hours for a table.
- Per day there are at most 120 working hours for group A, and 72 working ours for group 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: # chairs produced on machine group B

t_A: # tables produced on machine group A

t_B: # tables produced on machine group B

Constraints: $4\, c_A + 6\, t_A \leq 120$

$2\, c_B + 6\, t_B \leq 72$

Objective function (profit): Maximize $2(c_A + c_B) + 4(t_A + t_B)$

Linear Program (LP)

Variables: $x_1, \ldots, x_d$

Constraints:
h_1: $a_{11}\, x_1 + \ldots + a_{1d}\, x_d \leq b_1$
h_2: $a_{21}\, x_1 + \ldots + a_{2d}\, x_d \leq b_2$
$\vdots \qquad\qquad \vdots$
h_n: $a_{n1}\, x_1 + \ldots + a_{nd}\, x_d \leq b_n$

Objective function:
Maximize $f_{\vec{c}}(\vec{x}) \coloneqq c_1\, x_1 + c_2\, x_2 + \ldots + c_d\, x_d$

$$\overbrace{\begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nd} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \leq \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}}^{A \cdot \vec{x} \leq \vec{b}}$$

$$f_{\vec{c}}(\vec{x}) = \underbrace{\begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ \vdots \\ c_d \end{pmatrix}}_{\vec{x} \cdot \vec{c}}$$

Linear Program in *d* variables with *n* constraints.

Linear Program (LP)

Variables: $x_1, \ldots, x_d$

Constraints: h_1: $a_{11}\, x_1 + \ldots + a_{1d}\, x_d \leq b_1$

h_2: $a_{21}\, x_1 + \ldots + a_{2d}\, x_d \leq b_2$

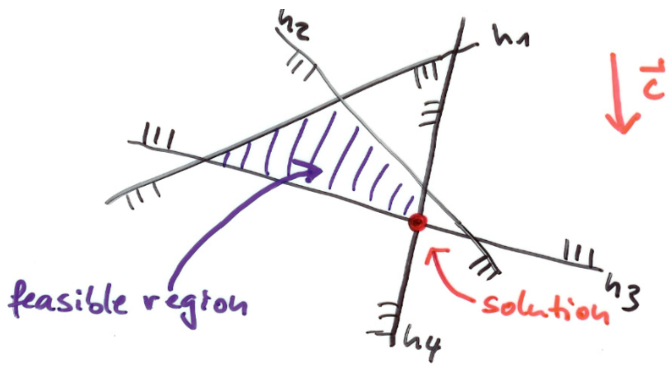
$\vdots$ $\qquad\qquad \vdots$

h_n: $a_{n1}\, x_1 + \ldots + a_{nd}\, x_d \leq b_n$

$$\overbrace{\begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nd} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \leq \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}}^{A \cdot \vec{x} \leq \vec{b}}$$

Objective function: Maximize $f_{\vec{c}}(\vec{x}) \coloneqq c_1\, x_1 + c_2\, x_2 + \ldots + c_d\, x_d$

- Each constraint h_i is a half-space in $\mathbf{R}^d$.
- Set of points in $\mathbf{R}^d$ satisfying all constraints: $\bigcap_{i=1}^{n} h_i$ feasible region of the LP
- Maximizing $f_{\vec{c}}(\vec{x}) = \vec{x} \cdot \vec{c}$ $\Leftrightarrow$ Finding a point $\vec{x}$ that is extreme in direction $\vec{c}$



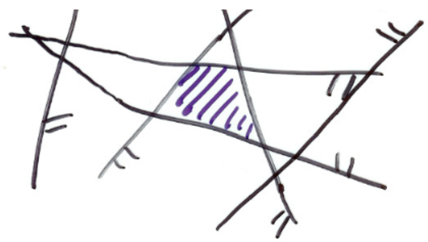
Subproblem: Half-Plane Intersection

Given: A set $H=\{h_1,\ldots, h_n\}$ of halfplanes

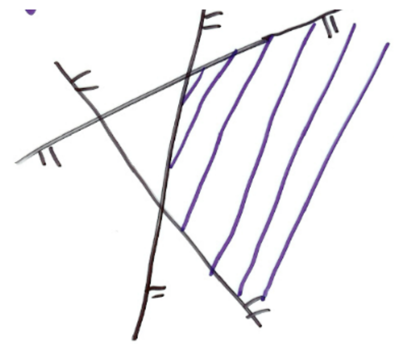
$h_i : a_i\, x + b_i\, y \le c_i$, $i=1..n$, with a_i , b_i , c_i constants.

Find: $\bigcap_{i=1}^{n} h_i$ set of points $\binom{x}{y} \in \boldsymbol{R}^2$ satisfying all *n* constraints at the same time

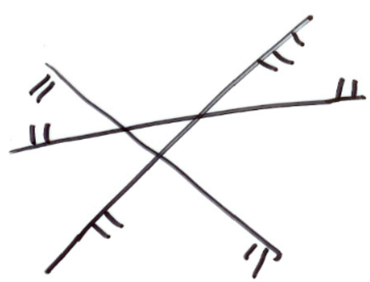
Convex polygonal region bounded by at most *n* edges



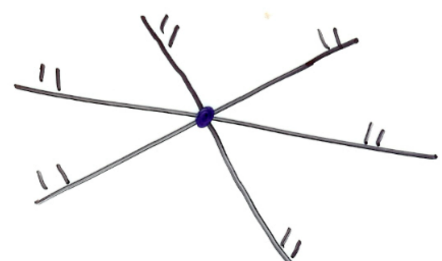
intersection bounded



intersection unbounded



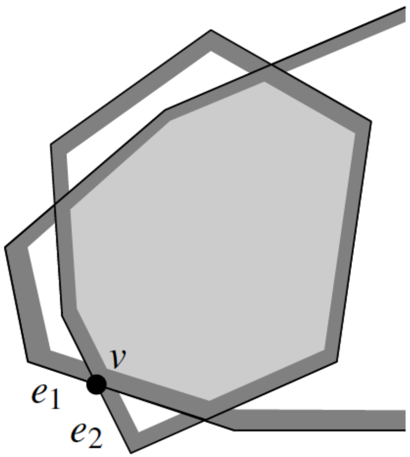
intersection empty



intersection degenerated to a point

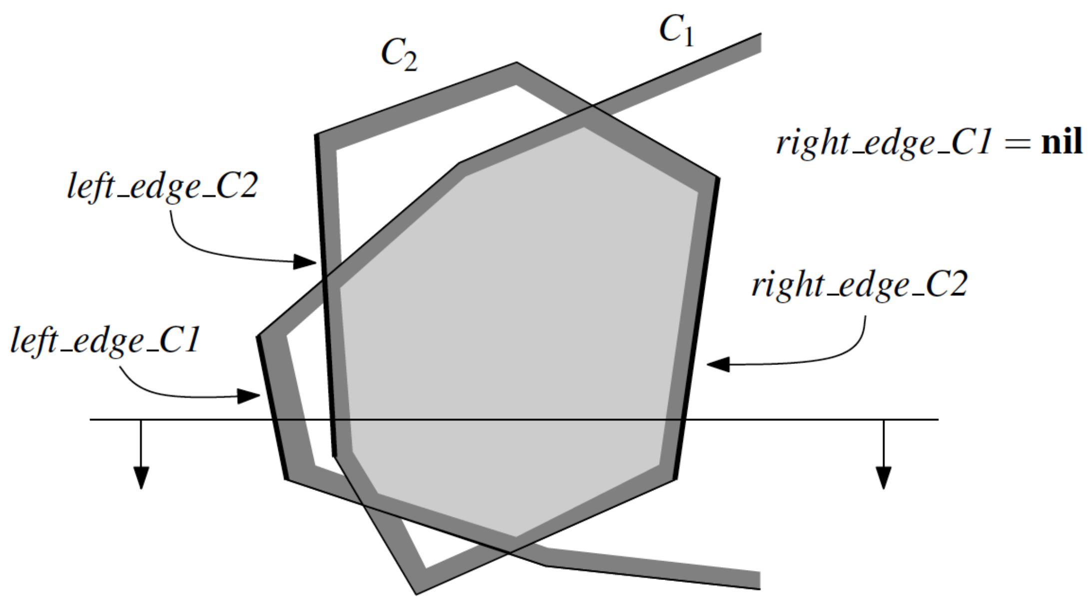
2D Divide & Conquer Algorithm

Algorithm Intersect_Halfplanes(H):
Input: A set H of n half-planes in $\boldsymbol{R}^2$
Output: The convex polygonal region $C := \bigcap_{h \in H} h_i$
if |H|=1 then $C := h \in H$
else
    Split H into sets H_1 and H_2 of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$
    C_1 := Intersect_Halfplanes(H_1)
    C_2 := Intersect_Halfplanes(H_2)
    C := Intersect_Convex_Regions(C_1, C_2)



2D Divide & Conquer Algorithm

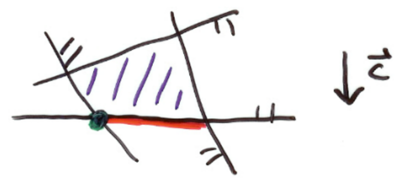
- Can implement Intersect_Convex_Regions using a sweep in $O(n)$ time: Just update constant-complexity interval intersecting the sweep line



- Runtime recurrence is $T(n)=2T(n/2)+n$, and hence the runtime is $O(n \log n)$

Incremental Linear Programming

- Two-dimensional linear programming (LP) problem: $H=\{h_1,\ldots, h_n\}$, $\vec{c}$
- 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

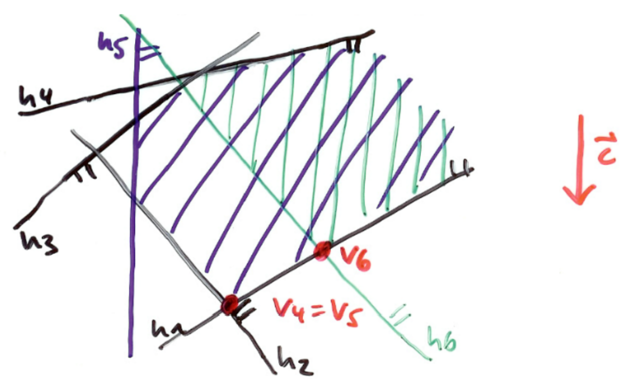
  $H_i = \{h_1, \ldots, h_i\}$

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

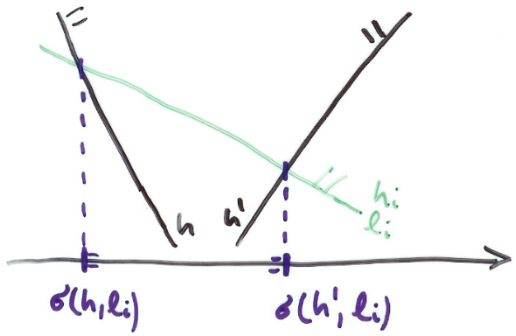
  v_i = unique optimal vertex for feasible region C_i, for $i \geq 2$

- Then: $C_1 \supseteq C_2 \supseteq \cdots \supseteq C_n = C$

  $\Rightarrow$ If $C_i = \emptyset$ for some i, then $C_j = \emptyset$ for all $j \geq i$
- Lemma: Let $2 \leq i \leq n$
  (i) If $v_{i-1} \in h_i$ then $v_i = v_{i-1}$
  (ii) If $v_{i-1} \notin h_i$ then $C_i = \emptyset$ or $v_i \in l_i :=$ line bounding h_i



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



Let $\overline{f_{\vec{c}}}: \boldsymbol{R} \rightarrow \boldsymbol{R}$

$x \mapsto f_{\vec{c}}(x, l_i(x))$

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

Need to solve 1-dimensional LP: Maximize $\overline{f_{\vec{c}}}$ subject to

$x \geq \sigma(h, l_i)$, $\quad h \in H_{i-1}$ and $l_i \cap h$ is bounded to the left

$x \leq \sigma(h, l_i)$, $\quad h \in H_{i-1}$ and $l_i \cap h$ is bounded to the right

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

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

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

⇒ If $x_{left} > x_{right}$ the LP is infeasible. Otherwise, either x_{left} or x_{right} is the optimum

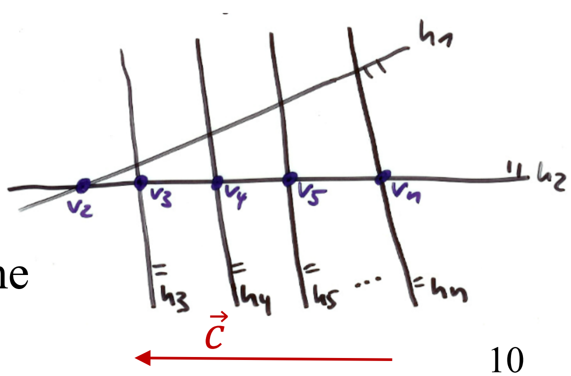
⇒ We can compute a new optimal vertex v_i, or decide that the LP is infeasible, in O(i) time.

Algorithm

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 $f_{\vec{c}}$
Let $h_1, \ldots, h_n$ be the half-planes of H
Let v_2 be the corner of C_2. // v_2 exists since we assume the LP is bounded
for i:=3 to n
O(1) { if $v_{i-1} \in h_i$ then $v_i \coloneqq v_{i-1}$
else
O(i) { $v_i \coloneqq$ point on l_i that maximizes $f_{\vec{c}}$ subject to the constraints in H_{i-1}
if such a point does not exist then
Report that the LP is infeasible, and return.
return v_n

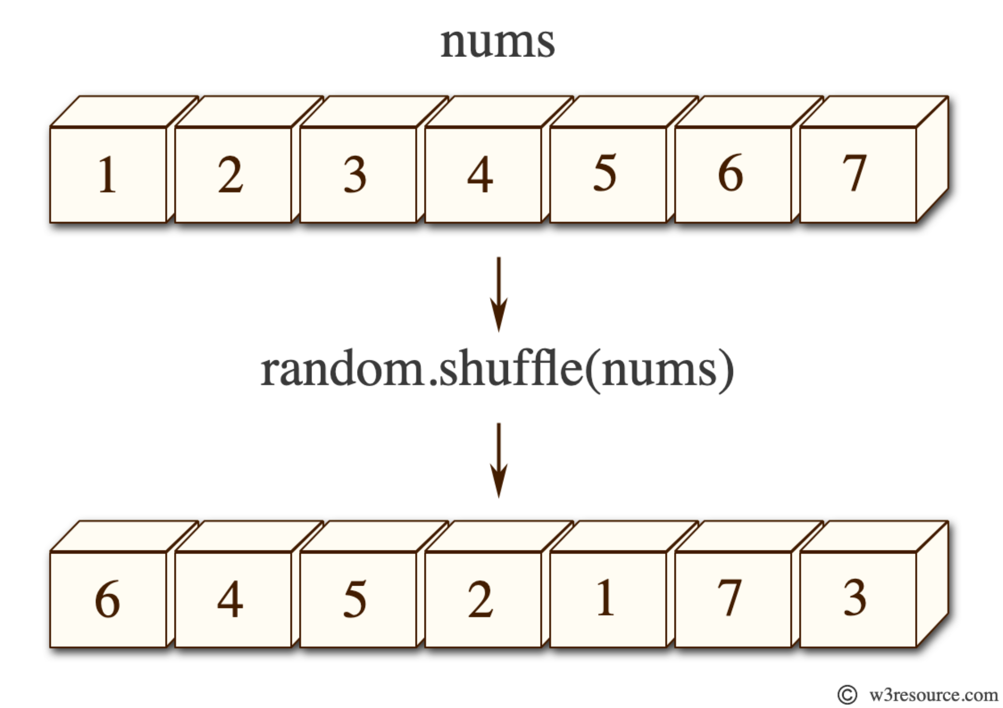
- Runtime: $O(\sum_{i=1}^{n} i) = O(n^2)$ Storage: O(n)

In this example, each step takes O(i) time



Randomized Linear Programming

- Perform incremental linear programming, but add the half-planes in random order
- Needs a way for creating a random permutation…



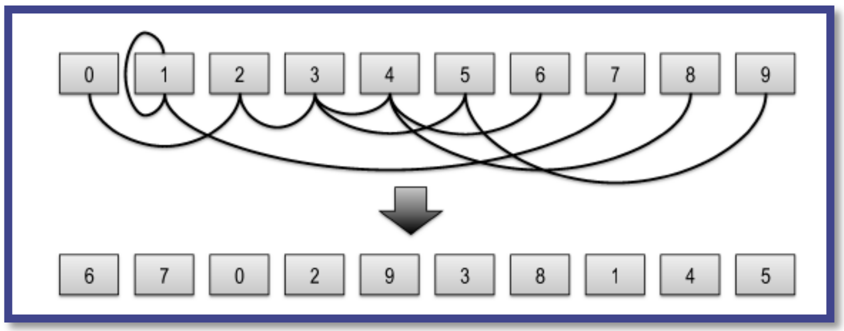
Fisher-Yates Shuffling

- There is a linear-time algorithm, known as the Fisher-Yates algorithm, which always succeeds.

Algorithm FisherYates(X):

  Input: An array, X, of n elements, indexed from position 0 to $n-1$
  Output: A permutation of X so that all permutations are equally likely

  for $k = n-1$ **downto** 1 **do**
    Let $j \leftarrow \text{random}(k+1)$ // j is a random integer in $[0, k]$
    Swap $X[k]$ and $X[j]$ // This may "swap" $X[k]$ with itself, if $j = k$
  return X



Analysis of Fisher-Yates

- This algorithm considers the items in the array one at time from the end and swaps each element with an element in the array from that point to the beginning.
- Notice that each element has an equal probability, of 1/n, of being chosen as the last element in the array X (including the element that starts out in that position).
- Applying this analysis recursively, we see that the output permutation has probability

$$\left(\frac{1}{n}\right)\cdot\left(\frac{1}{n-1}\right)\cdots\left(\frac{1}{2}\right)\cdot\left(\frac{1}{1}\right)=\frac{1}{n!}$$

- That is, each permutation is equally likely.

Randomized Incremental 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 $h_1, \ldots, h_n$ 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

Randomized Incremental Linear Programming

- Insertion order of the halfplanes determines the runtime
  $\rightarrow$ Varies between $O(n)$ and $O(n^2)$

$\Rightarrow$ Algorithm 2D_Randomized_Bounded_LP($H, \vec{c}$) inserts the halfplanes in random order

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

Proof: Random variable $X_i = \begin{cases} 1, v_{i-1} \notin h_i \\ 0, \quad otherwise \end{cases}$

- Total time spent in the else-part of the algorithm 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):

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

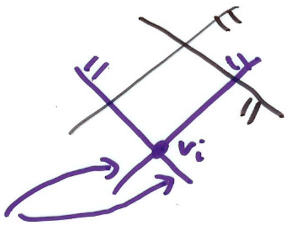
and $E(X_i) = P(X_i = 1) = P(v_{i-1} \notin h_i)$

Randomized Incremental Linear Programming

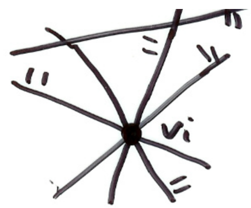
Apply backwards analysis to bound $E(X_i) = P(X_i = 1) = P(v_{i-1} \notin h_i)$:

- Fix $H_i = \{h_1, \ldots, h_i\}$; this determines C_i
- Analyze what happened in the last step when h_i was added
- P(Had to compute new optimal vertex when adding h_i)
  = P(Optimal vertex changes when we remove a halfplane from C_i)

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



$\frac{2}{i}$



0

2 out of i halfplanes defining v_i

- Hence $E(X_i) \leq \frac{2}{i}$
- Therefore the total expected runtime is
  $\sum_{i=1}^{n} O(i) \cdot \frac{2}{i} = O(n)$

□