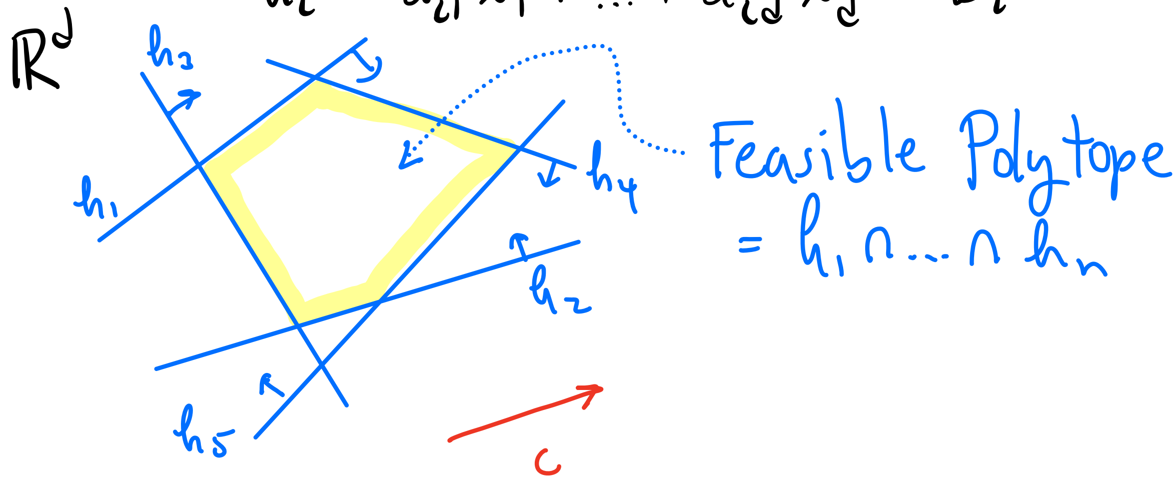
CMSC 754 - Computational Geometry
Lecture 7: Linear Programming

Linear Programming (LP):

- Fundamental optimization problem in $\mathbb{R}^d$
- Given a set of **n linear constraints** (halfspaces) $H = \{h_1, \dots, h_n\}$

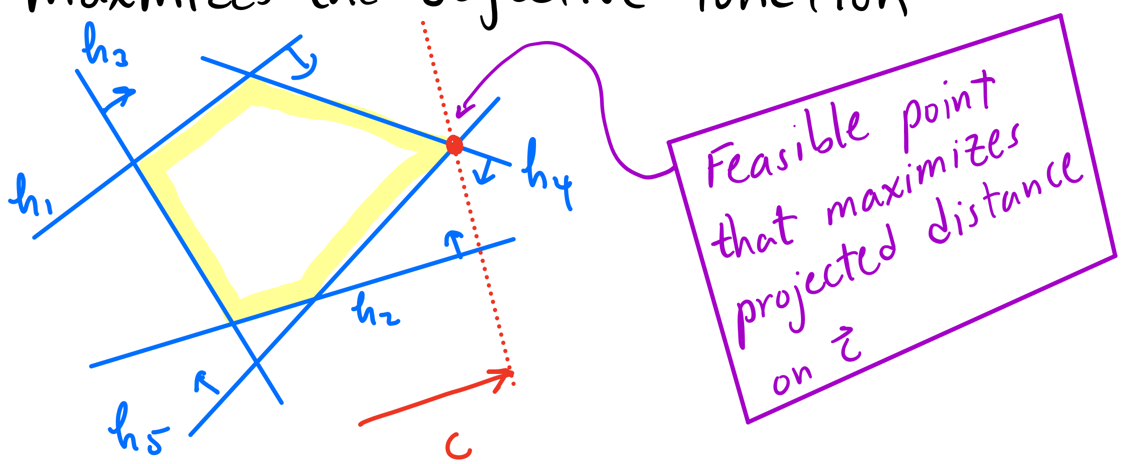
$$h_i : a_{i1}x_1 + \dots + a_{id}x_d \leq b_i$$



- Given a **linear objective function**

$$f(\bar{x}) = c_1x_1 + \dots + c_dx_d = c^T x$$

LP: **Find the vertex of the feasible polytope that maxmizes the objective function**



Matrix form:

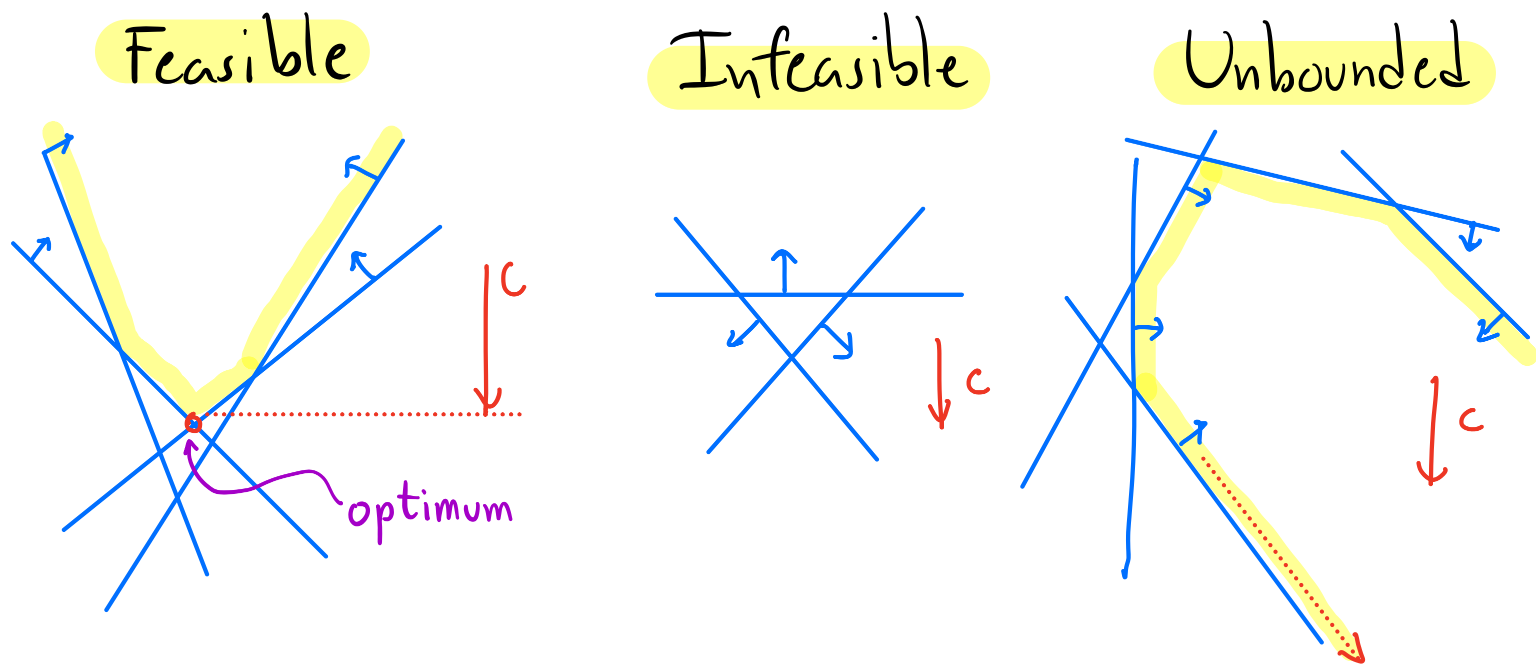
Given $c \in \mathbb{R}^d$ and $n \times d$ matrix A and $b \in \mathbb{R}^n$ find $x \in \mathbb{R}^d$ to:

$$\begin{aligned} \text{maximize: } & c^T x \\ \text{subject to: } & Ax \leq b \end{aligned}$$

i^{th} row of A corresponds to h_i

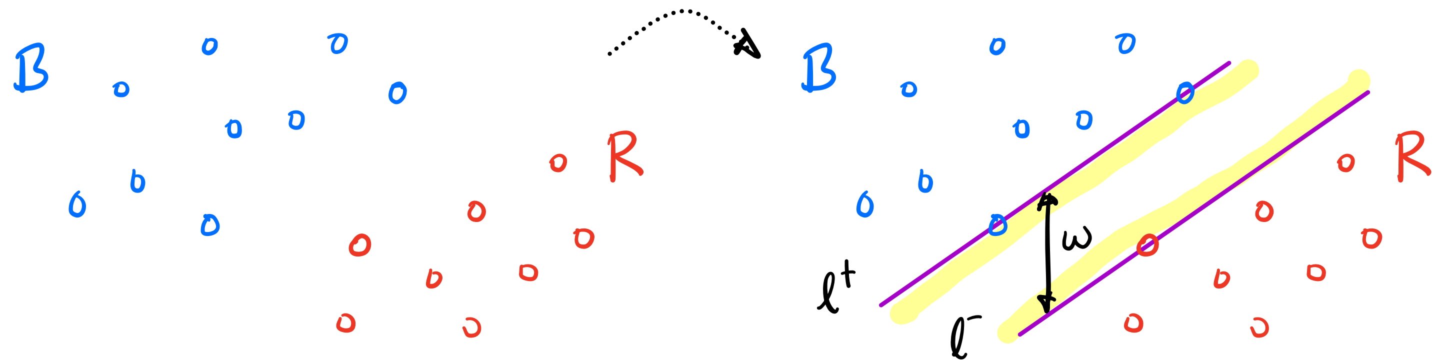
3 Possible Outcomes:

- **Feasible:** An optimal pt exists (gen'l position: a unique vertex of feasible polytope)
- **Infeasible:** No solution because feasible polytope is empty
- **Unbounded:** No (finite) solution because feasible polytope is unbounded in direction of objective fn.



Example:

- Given two point sets B + R in $\mathbb{R}^2$ find lines of max. vertical distance with B above both + R below both



- **Lines**: $\ell^+ : y = e \cdot x + f^+$ $\quad \ell^- : y = e \cdot x + f^-$

- **Constraints**: $\forall p \in B,\ p_y \geq e \cdot p_x + f^+$ (above ℓ^+)
  $\forall p \in R,\ p_y \leq e \cdot p_x + f^-$ (below ℓ^-)

- **Objective**: maximize $w = f^+ - f^-$

LP in $\mathbb{R}^3$

Standard form: Find (e, f^+, f^-)

to maximize $f^+ - f^- \equiv \underbrace{(0,1,-1)}_{c} \cdot (e, f^+, f^-)$

subject to:

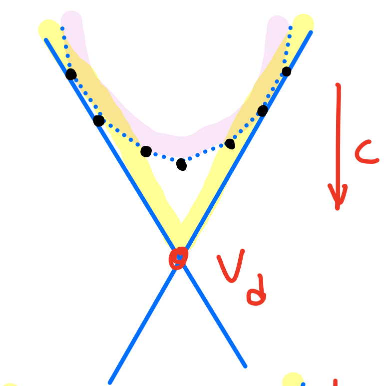
$$p_{ix} \cdot e + 1 \cdot f^+ + 0 \cdot f^- \leq p_{iy},\ \forall p_i \in B$$
$$-p_{jx} \cdot e + 0 \cdot f^+ - 1 \cdot f^- \leq -p_{jy},\ \forall p_j \in R$$

LP in constant-dimensional space

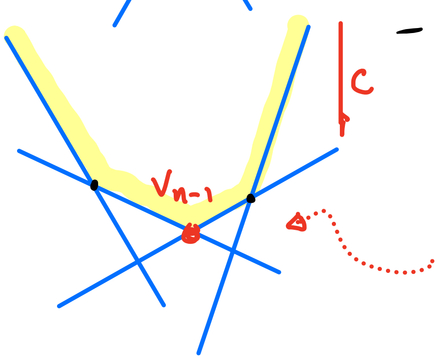
- Assume
  - n is large
  - d is a constant
- We'll present a (randomized) algorithm with (expected) running time $O(d!\,n) = O(n)$

Incremental Approach:

Overview:



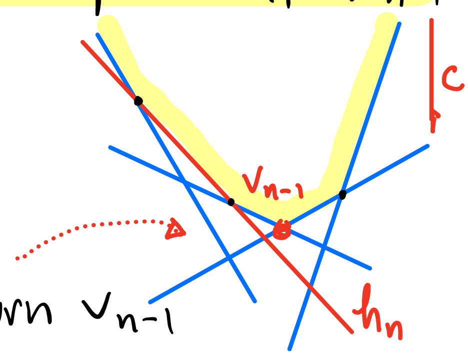
- Find d-halfspaces that define an initial vertex v_d (or report that LP is unbounded)
  - $\rightarrow O(dn)$ time (see our text)



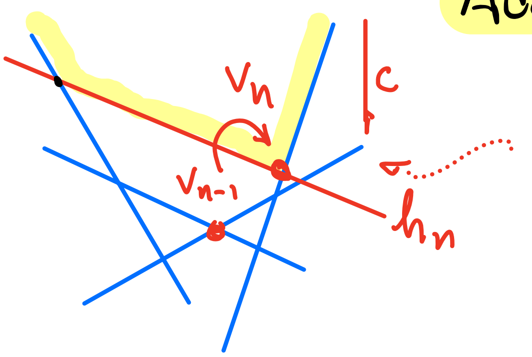
- Remove halfspace h_n and recursively compute LP on $n-1$ halfspaces $h_1 \dots h_{n-1}$

  If infeasible $\rightarrow$ return

  else let v_{n-1} be opt



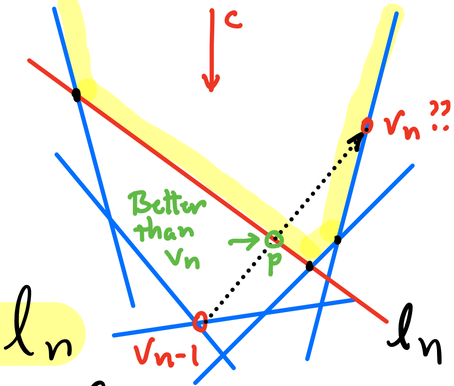
- Add back h_n
  - If ($v_{n-1} \in h_n$) return v_{n-1}
  - else ...



How to update opt. vertex?

Lemma: If v_{n-1} & h_n then new opt vertex (v_n) lies on the hyperplane bounding h_n.

Proof: Let ℓ_n be hyperplane bounding h_n. Assume c directed downwards.



v_{n-1} – not feasible $\Rightarrow$ below ℓ_n

v_n – if not on ℓ_n $\Rightarrow$ above ℓ_n

Let $p = \ell_n \cap \overline{v_{n-1} v_n}$

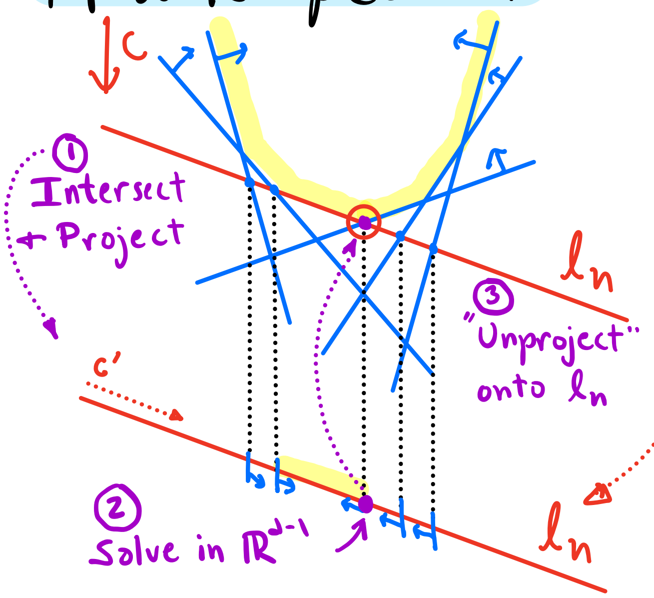
By convexity, $p \in$ feasible polytope

By linearity, obj. function gets progressively worse from $v_{n-1} \to v_n$

$\Rightarrow$ p is better solution than v_n

$\divideontimes$ contradiction!

How to update?



(See latex notes for details)

① Intersect $h_1, \dots, h_{n-1}$ with ℓ_n + project $\vec{c}$ [Yields an LP in $\mathbb{R}^{d-1}$ with $n-1$ constraints]

② Solve this $(d-1)$-dim LP recursively (If $d=1$, solve by brute force $O(n)$)

③ "Unproject" solution back onto ℓ_n

Running time? Pretty bad - $O(n^d)$

- Let $W_d(n)$ be worst-case complexity for n halfspaces in dim d
- Recurrence:

$$W_d(n) = W_d(n-1) + d + [dn + W_{d-1}(n-1)]$$

solve LP on $h_1 \dots h_{n-1}$

Test $v_{n-1} \in h_n$

Project onto ℓ_n

Solve LP on ℓ_n

Claim: $W_d(n) = O(n^d)$ ← Too slow!

How to fix this?

Easy! Randomize the choice of h_n

Why?

$$\boxed{W_d(n) = W_d(n-1) + d} + \boxed{dn + W_{d-1}(n-1)}$$

This solves to $O(n)$

Only applies if $v_{n-1} \notin h_n$

This rarely happens!

Randomized Incremental Algorithm

Input: $H = \{h_1, \dots, h_n\}$ constraint halfspaces in $\mathbb{R}^d$

$c \in \mathbb{R}^d$ objective vector

Output: Optimum vertex v or error $\begin{cases} \text{unbounded} \\ \text{or} \\ \text{infeasible} \end{cases}$

(1) If $(d = 1)$ solve LP by brute force - $O(n)$

(2) Find initial subset $\{h_1, \ldots, h_d\}$ that provide initial optimum v_d (or return "unbounded") - $O(d \cdot n)$ (see text)

(3) Randomly select halfspace from $\{h_{d+1}, \ldots, h_n\}$ - call it h_n. Recursively solve LP on remaining $n-1$ halfspaces $\rightarrow$ Let v_{n-1} be result

(4) If $(v_{n-1} \in h_n)$ return v_{n-1} $\rightarrow$ $O(d)$

(5) else, project $\{h_1, \ldots, h_{n-1}\} + c$ onto ℓ_n, $\rightarrow O(dn)$ the bounding hyperplane for h_n. Solve recursively, letting v_n be result. Return v_n

Expected Case Running Time:

- Running time depends on (random) choice, h_n

- Let $T_d(n)$ be the expected-case running time, over all choices of h_n.

- Let p_n = probability that $v_{n-1} \notin h_n$

- To simplify, assume all halfspaces chosen randomly ($h_1, \ldots, h_d$ aren't)

Recurrence:

$$T_d(n) = \begin{cases} 1 & \text{if } n = 1 \\ n & \text{if } d = 1 \\ T_d(n-1) + d + p_n\left(dn + T_{d-1}(n-1)\right) & \text{o.w.} \end{cases}$$

(3) Recursively compute v_{n-1}

(4) test if $v_{n-1} \in h_n$

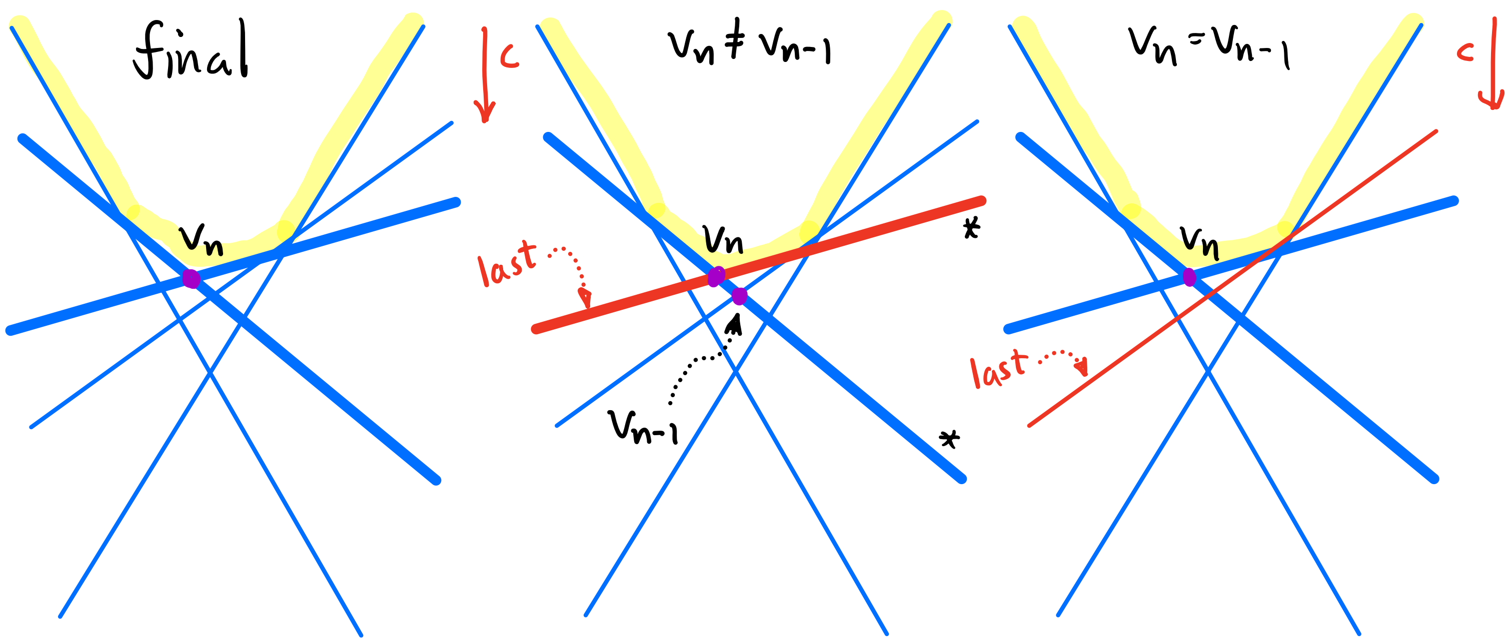
if not

(5) project $h_1, \dots h_{n-1}$ onto ℓ_n

(5) solve $d-1$ dim LP on projections

What is p_n? Backwards Analysis

- Let's consider the final configuration and ask – which halfspace came last and how does its choice affect things?



Obs: The optimum is determined by d halfspaces (assuming gen'l position)

- If h_n is any of these, $v_{n-1} \notin h_n$ + $v_n \neq v_{n-1}$ ☹
- Otherwise, $v_{n-1} \in h_n$ + $v_n = v_{n-1}$ ☺

$\Rightarrow p_n = d/n$ If $n \gg d$, p_n very small + bad case unlikely

Why is it called "backwards"?

- We consider final config. and look backwards to our last random choice

Lemma: $T_d(n) \leq \gamma_d \, d! \, n$, where γ_d is a constant depending on dimension

Proof: Induction on $n + d$

simplify

$$T_d(n) = T_d(n-1) + d + p_n \left(dn + T_{d-1}(n) \right)$$

by I.H. + def of p_n

$$\leq \gamma_d \cdot d! \, (n-1) + d + \frac{d}{n} \left(d \cdot n + \gamma_{d-1} (d-1)! \, n \right)$$

$$= \gamma_d d!(n-1) + d + (d^2 + \gamma_{d-1} d!)$$

$$= \gamma_d d! \, n + (d + d^2 + \gamma_{d-1} d! - \gamma_d d!)$$

want:

$$\le \gamma_d d! \, n$$

Suffices to select γ_d such that

$$d + d^2 + \gamma_{d-1} d! - \gamma_d d! \le 0$$

$$\iff d! \gamma_d \ge d + d^2 + \gamma_{d-1} d!$$

We can satisfy this by setting:

$$\boxed{\begin{aligned} &\gamma_1 \leftarrow 1 \\ &\gamma_d \leftarrow \frac{d+d^2}{d!} + \gamma_{d-1} \end{aligned}}$$

$\Rightarrow \gamma_d$ is a constant depending on dim

$\square$

Summary:

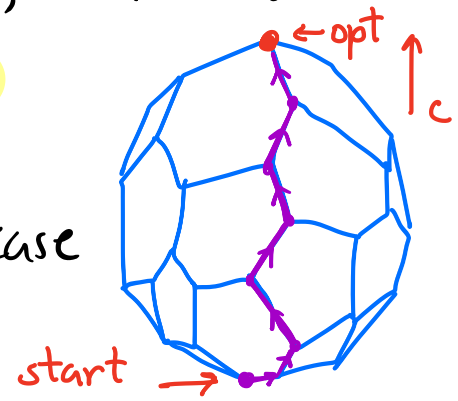
- Randomized algorithm for LP
- Expected run time of LP is $O(d!\, n) = O(n)$

  (since we assume d is constant)
- Variation depends on random choices, not input
- (Seidel) Prob of running slower extremely small

A Bit of History (Optional)

1940's: Used in operations research (Econ, Business)
Kantorovich, Dantzig, von Neuman

Dantzig (1947) – **Simplex algorithm**

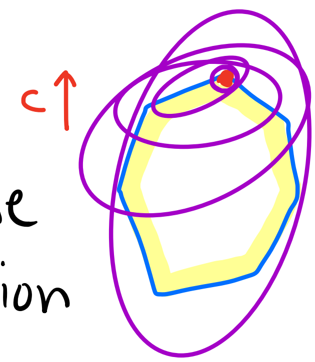


- fast in practice
- exponential in worst case

feasible polytope may have $O(n^{\lfloor d/2 \rfloor})$ vertices

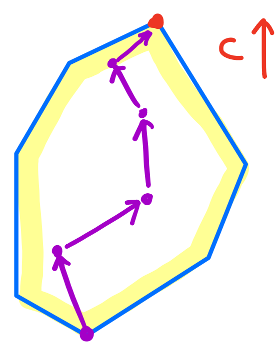
- Karp – Not known to be NP-hard

Khachiyan (1979) – **Ellipsoid Algorithm**



- (Weakly) polynomial time
  - ↪ Time depends on precision
- Compute smaller + smaller ellipsoids containing optimum

Karmarkar (1984) – **Interior-Point Methods**



- Move through polytope's interior
- (Weakly) polynomial
- Practical

Open – Is there a poly. time (purely combinatorial) algorithm?