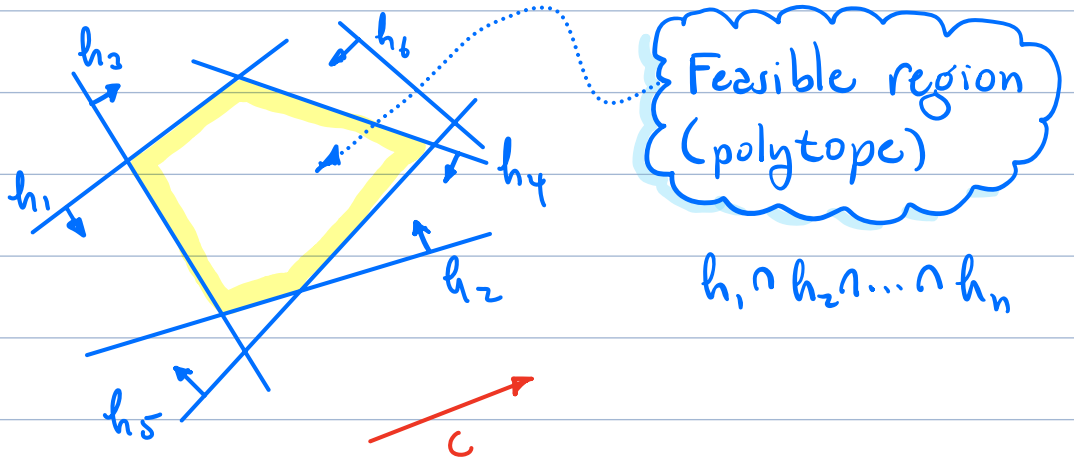
CMSC 754 - Computational Geometry
Lecture 7: Linear Programming

Linear Programming (LP):

- A fundamental optimization technique in $\mathbb{R}^d$
- Given a set of n linear constraints (halfspaces)

  $H = \{h_1, \ldots, h_n\}$, where $h_i : a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{id}x_d \leq b_i$

$\mathbb{R}^d$:

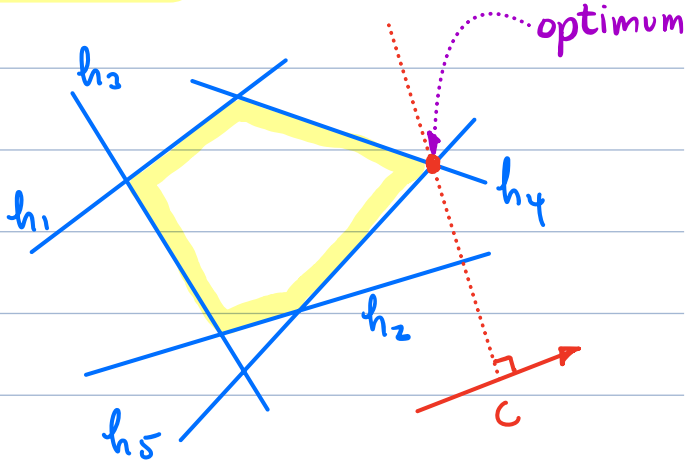


- also given a linear objective function

  $c(x) = c_1x_1 + \ldots + c_dx_d = c^Tx$ $\quad c, x \in \mathbb{R}^d$

Goal: Find the point of the feasible region that maximizes the objective function

(equiv, maximizes the orthogonal projection on the objective vector)



Matrix Form:

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

maximize: $c^T x$

subject to: $Ax \leq b$

i^{th} row of A corresponds to h_i

Three possible outcomes:

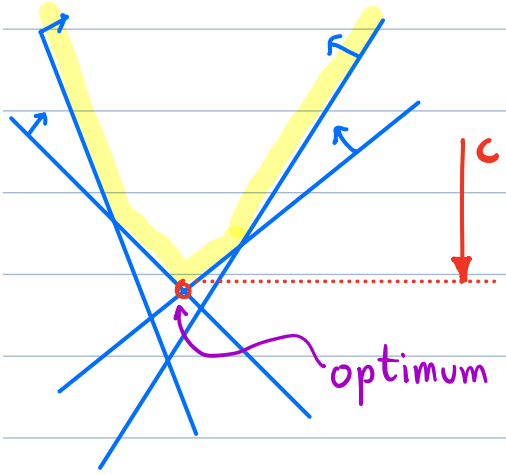
Assuming gen. pos. this is a vertex of feasible polytope

☺ **Feasible:** A (finite) optimal point exists

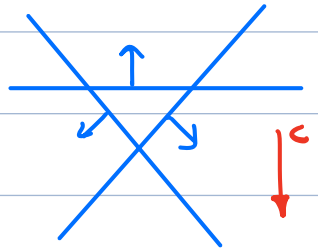
☹ **Infeasible:** No solution because feasible region is empty

☹ **Unbounded:** No optimum finite solution exists because feasible region is unbounded in direction of objective function

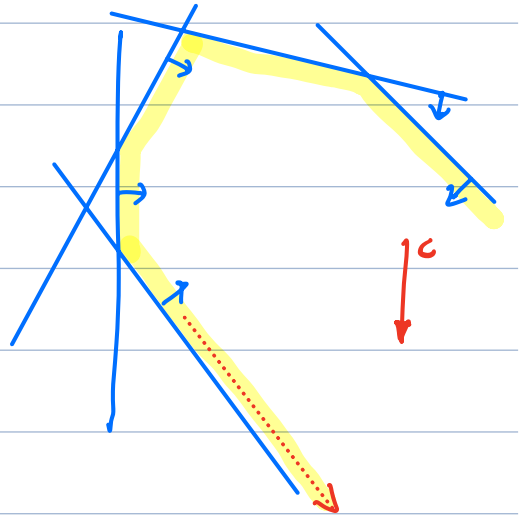
Feasible



Infeasible

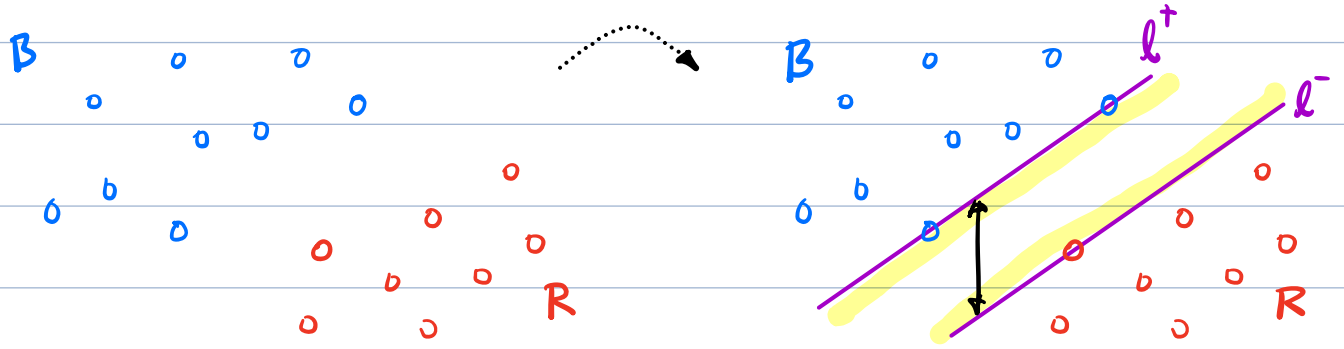


Unbounded



Example:

- Given two point sets, B & R, in $\mathbb{R}^2$, find two parallel lines ℓ^+ & ℓ^- that are maximally vertically separated, with ℓ^+ above ℓ^-, s.t. B lies on or above ℓ^+ & R lies on or below ℓ^-.



Formulated as an LP: **Points:** $b_i = (b_{ix}, b_{iy}) \in B$ $r_j = (r_{jx}, r_{jy}) \in R$

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

Constraints: for $i = 1, \ldots, |B|$ $b_{iy} \geq e \cdot b_{ix} + f^+$ (b_i above ℓ^+)

for $j = 1, \ldots, |R|$ $r_{iy} \leq e \cdot r_{ix} + f^-$ (r_j below ℓ^-)

$f^+ \geq f^-$ (ℓ^+ above ℓ^-)

Objective: maximize $f^+ - f^-$

LP Formulation:

Find $(e, f^+, f^-) \in \mathbb{R}^3$

maximizing: $f^+ - f^-$

subject to:

$b_{iy} \geq e \cdot b_{ix} + f^+$

$r_{jy} \leq e \cdot r_{jx} + f^-$

Matrix form:

$$x = \begin{bmatrix} e \\ f^+ \\ f^- \end{bmatrix} \qquad c = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}$$

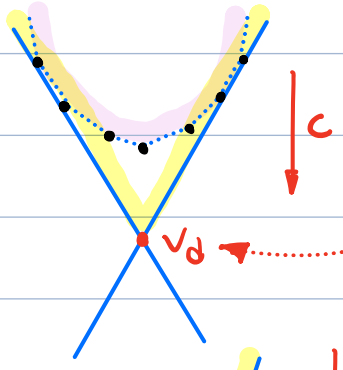
$$\begin{bmatrix} b_{1x} & 1 & 0 \\ \vdots & & \\ b_{mx} & 1 & 0 \\ -r_{1x} & 0 & -1 \\ \vdots & & \\ -r_{nx} & 0 & -1 \end{bmatrix} \begin{bmatrix} e \\ f^+ \\ f^- \end{bmatrix} \leq \begin{bmatrix} b_{1y} \\ \vdots \\ b_{my} \\ -r_{1y} \\ \vdots \\ -r_{ny} \end{bmatrix}$$

LP in spaces of constant dimension:

- Assume n is large (asymptotic quantity)
  - d is constant (we'll ignore even exponential depedence)
- We'll present a (randomized) algorithm with (expected) running time $O(d!\, n) = O(n)$.

Incremental Approach:

Initialization: Find d-halfspaces that define a bounded solution v_d (or report that LP is unbounded) $\rightarrow O(d\,n)$ time

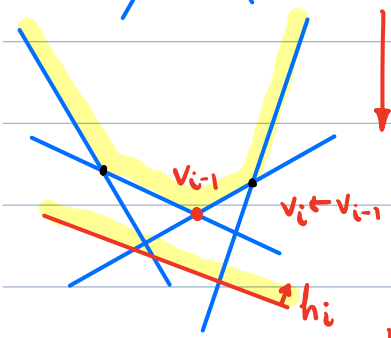


Incremental:

for $i \leftarrow d+1$ to n

- add halfspace h_i
- if $v_{i-1} \in h_i$

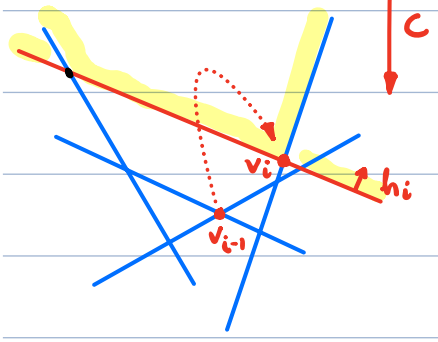
  set $v_i \leftarrow v_{i-1}$ (no change)



- else

  $v_i \leftarrow$ new optimal vertex

  (How? Later)

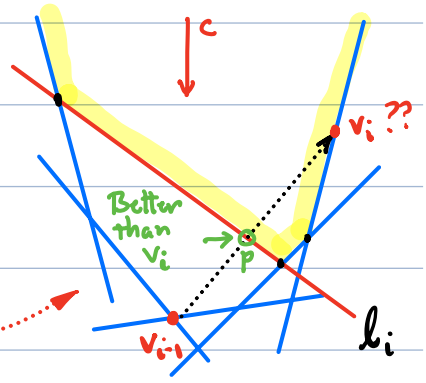


Lemma: If $v_{i-1} \notin h_i$, the new optimum v_i lies on h_i's bounding hyperplane (or LP is infeasible)

Proof: Let ℓ_i denote h_i's bounding hyperplane

- Assume (for illustration) c is directed downwards.
- v_{i-1} & h_i $\Rightarrow$ v_{i-1} is below ℓ_i
- Suppose to contrary v_i & ℓ_i
  - $\Rightarrow$ v_i lies above ℓ_i
  - $\Rightarrow$ Segment $\overline{v_{i-1}v_i}$ intersects ℓ_i
- Let $p = \overline{v_{i-1}v_i} \cap \ell_i$
- By convexity, p is feasible
- By linearity, objective fn. decreases from $v_{i-1} \rightarrow v_i$
  - $\Rightarrow$ p is a better solution than v_i

$\nrightarrow$ contradiction!

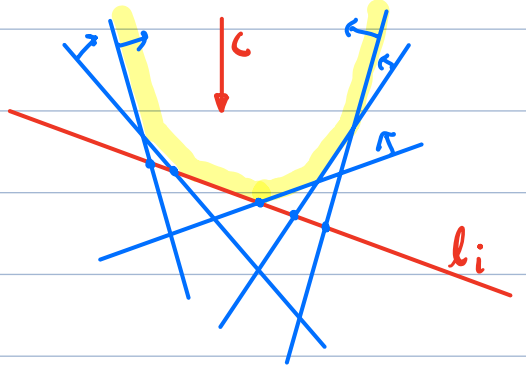


□

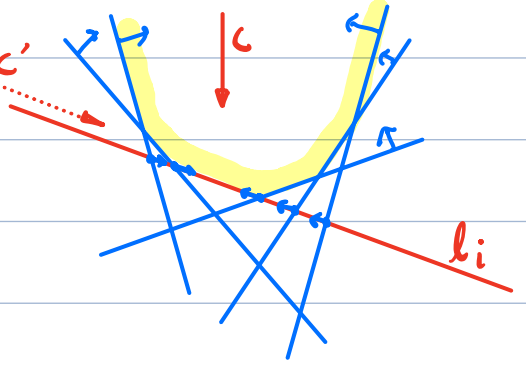
How to update optimal vertex?

- Given prior halfspaces $h_1, \ldots h_{i-1}$, objective vector c, and current hyperplane ℓ_i

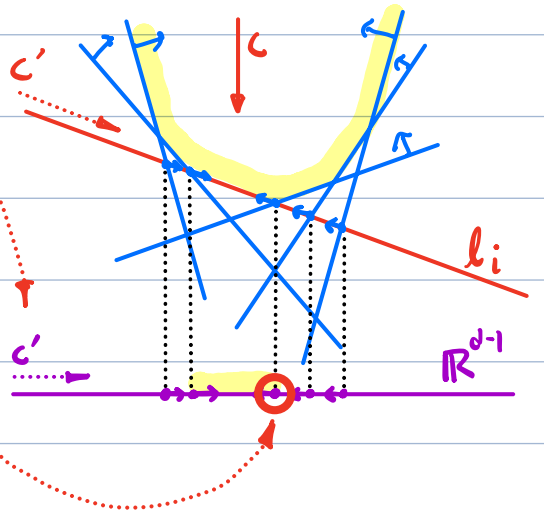
① Intersect $h_1, \ldots, h_{i-1}$ with ℓ_i



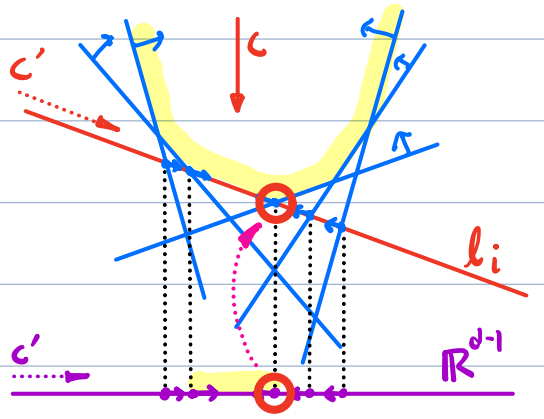
② $c' \leftarrow$ Project c onto ℓ_i



③ Project all this to $\mathbb{R}^{d-1}$
(Gauss elimination)



④ Recursively solve this (d-1)-dim LP on i-1 constraints



⑤ Project this back onto l_i

Running Time:

- Worst case - Bad! $O(n^d)$



- Expected case (Random insertion order)
  - Good! $O(n)$



Worst-Case Analysis:

- Let $W_d(n)$ = worst-case time for n constraints in $\mathbb{R}^d$

- View algorithm recursively, rather than incremental

- Recurrence:

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

- Solve LP on first $n-1$ constraints
- Test feasibility of v_{i-1}
- Project onto h_i
- Solve recursively

By induction (exercise) can show $W_d(n) \leq c \cdot n^d$

$\Rightarrow W_d(n) = O(n^d)$

Hint: Note the similarity with the Binomial Thm:

$$n^d \approx \binom{n}{d} = \binom{n-1}{d} + \binom{n-1}{d-1}$$

Good News - The worst case analysis is unnecessarily pessimistic $\rightarrow$ Randomize!

- Insert the halfspaces in random order
- Why should this help?

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

This part solves to $O(d \cdot n) = O(n)$

This only applies when $v_{i-1} \notin h_i$

We'll show this rarely occurs in expectation when randomized.

Randomized Incremental Algorithm for LP

If ($d=1$) solve LP by brute force // $O(n)$
Find initial subset $h_1 \dots h_d$ that
  provides a (finite) feasible optimum
  v_d (or return "unbounded" if doesn't exist)
  (see text for details) // $O(d \cdot n)$
Permute remaining $n-d$ halfspaces randomly
for $i \leftarrow d+1$ to n
  if ($v_{i-1} \in h_i$) $v_i \leftarrow v_{i-1}$
  else $l_i \leftarrow$ hyperplane bounding h_i
    project $\{h_1, \dots, h_{i-1}\} + c$ onto l_i // $O(d \cdot i)$
    $v_i \leftarrow$ solve resulting $(d-1)$-dim LP
      recursively

Expected Running Time:

- Depends on random choices - h_i
- Let $T_d(i)$ denote expected running time for i halfspaces in $\mathbb{R}^d$
- Let p_i denote probability that $v_{i-1} \notin h_i$ (violation)
- For simplicity, assume all permutations of halfspaces are equally likely ($h_1, \dots, h_d$ are not, but $d \ll n$

Recurrence:

$$T_d(n) = \begin{cases} 1 & \text{if } n = d \quad \text{// basis case for } n \\ n & \text{if } d = 1 \quad \text{// basis case for } d \\ T_d(n-1) + d + p_n(dn + T_{d-1}(n-1)) & \text{o.w.} \end{cases}$$

- Recursively compute v_n
- Test $v_{n-1} \in h_n$
- if violation
- project onto h_n
- solve d-1 dim LP

To solve this – We need to know p_n

- Seems difficult! Depends on relationship between h_n + all prior halfspaces
- Cool trick – Backwards analysis



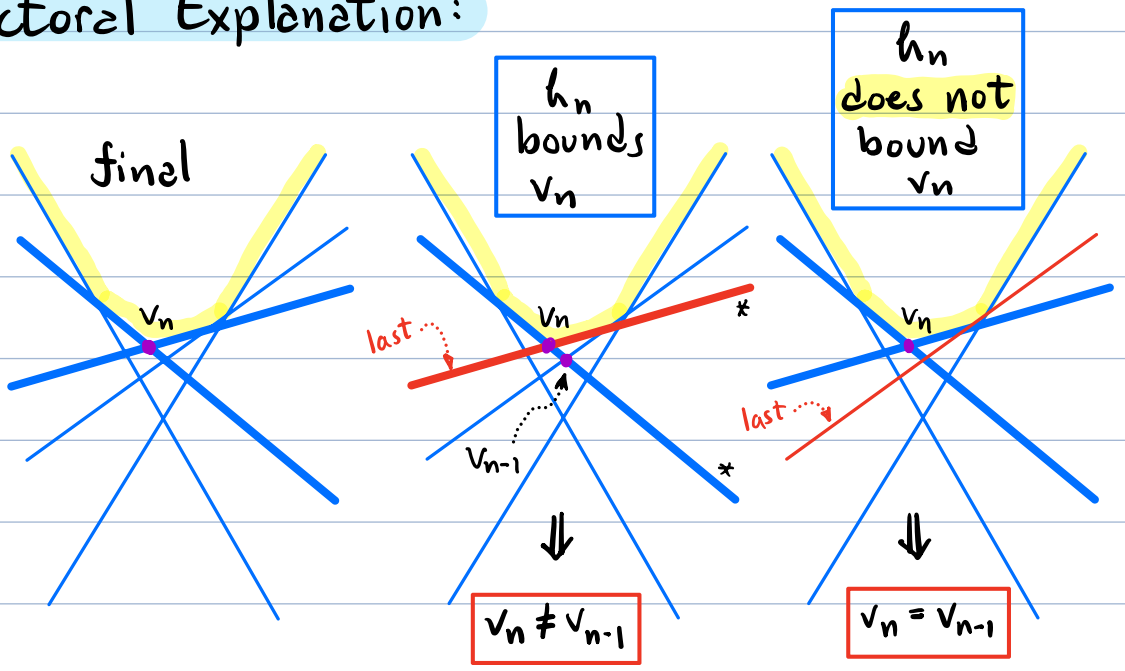
Idea:

- If permutation is random – All h_i equally likely to be last $\Rightarrow$ With prob $1/n$
- Let v_n be final optimum vertex
- v_n is intersection of d bounding hyperplanes
- Key: Last halfspace causes violation if and only if h_n is one of these

$\Rightarrow p_n = d/n$

Why? If h_n bounds v_n, then prior to its insertion, v_n did not exist $\Rightarrow$ v_{n-1} violates h_n

Pictoral Explanation:



Take-away Message: Recall $d = O(1)$; $p_n = d/n$

- As n grows, probability of violation decreases as $1/n$
- The expensive case (projection + recursion to $d-1$) becomes progressively less likely

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

p_n: d/n

Why is it called "Backwards" analysis?

- Rather than looking forwards from $n-1$ to n we look backwards from n to $n-1$
- No fundamental difference, but intuitively simpler

Expected Running Time:

Theorem: $T_d(n) = O(d!\,n) = O(n)$

Follows from...

Lemma: There exists a (dimension dependent) constant γ_d such that $T_d(n) \le \gamma_d\, d!\, n$

Proof: By induction on $n + d$

Recall:

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

Induction hypothesis + def of $p_n \Rightarrow$

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

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

($d(d-1)! = d!$; $\frac{n-1}{n} \le 1$)

$$\le \gamma_d\, d!\, n - \gamma_d d! + \left(d + d^2 + \gamma_{d-1}\cdot d!\right)$$

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

$$= \gamma_d\, d!\, n \quad \text{if} \quad d + d^2 + \gamma_{d-1}\, d! = \gamma_d\, d!$$

To satisfy this, let's define:

$$\gamma_1 = 1$$
$$\gamma_d = \frac{d+d^2}{d!} + \gamma_{d-1}$$

Since d is a constant, so is γ_d

We have:

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

Thus,

$$T_d(n) \leq \gamma_d\, d!\, n$$

□

Summary:

- Introduced Linear Programming (LP)
- Randomized incremental algorithm
  - $O(d!\, n)$ - Good assuming d is constant
    ≡ Solution has $O(1)$ degrees of freedom
- Note: Expectation is over algorithm's randomness
  - Does not depend on input!

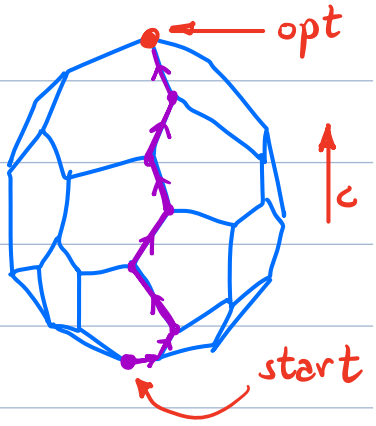
Short history of LP (optional)

1940's - Used in operations research (business)

- Kantorovich, Danzig, Von Neuman

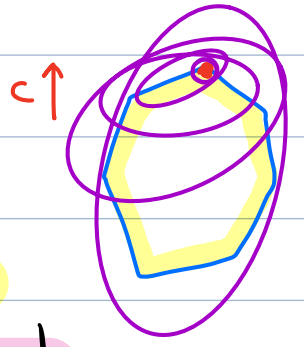
Danzig - Simplex algorithm (1947)

- Walk along edges of feasible polytope
- Simple, practical
- Exponential in worst case
- May have $n^{\lfloor d/2 \rfloor}$ vertices



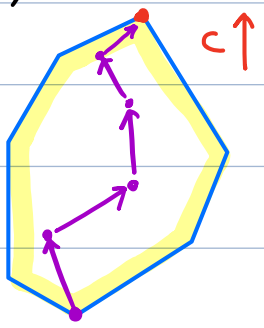
Khachiyan - Ellipsoid Algorithm (1979)

- Compute successively smaller ellipsoids containg optimum
- Weakly polynomial time
  - ↳ Polynomial in bit complexity
- Theoretically good, but impractical



Karmarkar - Interior-Point Algorithm (1984)

- Move toward optimum through polytopes' interior
- Also weakly polynomial but efficient in practice



Major open problem:

- Is there a strong polynomial alg for LP?
- That is, purely combinatorial, not numeric