Linear Programming (LP):
- Fundamental optimization problem in $\mathbb{R}^d$
- Given a set of $n$ linear constraints (halfspaces) $H = \{h_1, \ldots, h_n\}$
  \[ h_i : a_i^T x_1 + \ldots + a_{i,d} x_d \leq b_i \]

$$\mathbb{R}^d$$

- Given a linear objective function
  \[ f(\vec{x}) = c_1 x_1 + \ldots + c_d x_d = \vec{c}^T \vec{x} \]

LP: Find the vertex of the feasible polytope that maximizes the objective function

Feasible point that maximizes projected distance on $\vec{c}$
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:

\[
\text{maximize: } c^T x \\
\text{subject to: } Ax \leq b
\]

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: $l^+: y = e \cdot x + f^+$, $l^-: y = e \cdot x + f^-$

- 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 $\omega = f^+ - f^-$, LP in $\mathbb{R}^3$

Standard form: Find $(e, f^+, f^-)$ to maximize $f^+ - f^- = (0, 1, -1) \cdot (e, f^+, f^-)$ 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)
  - $O(dn)$ time (see our text)
- Remove halfspace $h_n$ and recursively compute LP on $n-1$ halfspaces $h_1, \ldots, h_{n-1}$
  - If infeasible return
  - Else let $v_{n-1}$ be opt
- Add back $h_n$
  - If $(v_{n-1} \in h_n)$ return $v_{n-1}$
  - Else ...
Lemma: If \( v_{n-1} \neq h_n \) then new opt vertex \( (v_n) \) lies on the hyperplane bounding \( h_n \).

Proof: Let \( h_n \) be hyperplane bounding \( h_n \). Assume \( c \) directed downwards.

\[
\begin{align*}
\text{if } v_{n-1} & \text{ not feasible } \Rightarrow \text{ below } h_n \\
\text{if } v_n & \text{ if not on } h_n \Rightarrow \text{ above } h_n \\
\end{align*}
\]

Let \( p = h_n \cap \overline{v_{n-1}v_n} \)

By convexity, \( p \in \text{feasible polytope} \)
By linearity, obj. function gets progressively worse from \( v_{n-1} \rightarrow v_n \)

\( p \) is better solution than \( v_n \)
\( \times \) contradiction!

How to update?

1. Intersect \( h_1, \ldots, h_{n-1} \) with \( h_n \) + project \( c \) \[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 \( h_n \)

(See latex notes for details)
Running time? Pretty bad - \( O(n^d) \)
- Let \( W_d(n) \) be worst-case complexity for \( n \) halfspaces in dim \( d \)
- Recurrence:
  \[
  W_d(n) = W_d(n-1) + d + [dn + W_{d-1}(n-1)]
  \]
  
  Claim: \( W_d(n) = O(n^d) \)
  
  Too slow!

How to fix this?

Easy! Randomize the choice of \( h_n \)

Why?

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

This solves to \( O(n) \)

Only applies if \( v_{n-1} \& h_n \)

This rarely happens!

Randomized Incremental Algorithm

Input: \( H = \{ h_1, \ldots, h_n \} \) constraint halfspaces in \( \mathbb{R}^d \)
\( c \in \mathbb{R}^d \) objective vector

Output: Optimum vertex \( v \) or error \{ infeasible \}
(1) If \( d = 1 \) solve LP by brute force \(- \mathcal{O}(n)\)
(2) Find initial subset \( \{h_1, ..., h_d\} \) that provide
    initial optimum \( v_d \) (or return "unbounded")
    \(- \mathcal{O}(d \cdot n) \) (see text)
(3) Randomly select halfspace from \( \{h_{d+1}, ..., h_n\} \)
    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 \( v_{n-1} \) \( \rightarrow \mathcal{O}(d) \)
(5) else, project \( \{h_1, ..., h_{n-1}\} \) + \( c \) onto \( h_n \) \( \rightarrow \mathcal{O}(dn) \)
    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 \( p_n = \text{probability that } v_{n-1} \in h_n \)
- To simplify, assume all halfspaces
  chosen randomly (\( h_1, ..., h_d \) aren’t)
Recurrence:

\[ T_d(n) = \begin{cases} 
1 & \text{if } n = 1 \\
2 & \text{if } d = 1 \\
T_d(n-1) + d + p_n (dn + T_{d-1}(n-1)) & \text{otherwise}
\end{cases} \]

(3) Recursively compute \( V_{n-1} \)
(4) Test if \( V_{n-1} \in h_{n} \)
(5) If not, project \( h_1, \ldots, h_{n-1} \) onto \( h_n \)
(5) Solve \( d-1 \) dLinLP 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, \(v_{n-1} \cap h_n + v_n \neq v_{n-1}\)
- Otherwise, \(v_{n-1} \in h_n + v_n = v_{n-1}\)

\[ p_n = \frac{d}{n} \quad \text{If } n \gg d, \text{ } p_n \text{ very small} \]
\[ \text{+ bad case unlikely} \]

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 \(\gamma_d\) is a constant depending on dimension

Proof: Induction on \(n+d\)

\[ T_d(n) = T_d(n-1) + d + p_n (dn + T_{d-1}(n)) \]

by I.H. \(\leq \gamma_d d! (n-1) + d + \frac{d}{n} (d \cdot n + \gamma_{d-1} (d-1)! n) \)

+ def of \(p_n\)
\[ = \gamma_d \cdot d! \cdot (n-1) + d + (d^2 + \gamma_{d-1} d!) \]

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

want:
\[ \leq \gamma_d \cdot d! \cdot n \]

Suffices to select \( \gamma_d \) such that
\[ d + d^2 + \gamma_{d-1} d! - \gamma_d d! \leq 0 \]

\[ \Leftrightarrow d! \gamma_d \geq d + d^2 + \gamma_{d-1} d! \]

We can satisfy this by setting:
\[ \gamma_1 = 1 \]
\[ \gamma_d \leftarrow \frac{d + d^2}{d!} + \gamma_{d-1} \]

\[ \implies \gamma_d \text{ is a constant depending on } d \text{ and } n \]

Summary:
- Randomized algorithm for LP
- Expected run time of LP is \( O(d! \cdot n) = O(n) \)
  (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)

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

**Dantzig** - Simplex algorithm (1947)
   - Fast in practice
   - Exponential in worst case
   - Feasible polytope may have $O(n^{1.2})$ vertices
   - Karp - Not known to be NP-hard

**Khachiyan** - Ellipsoid Algorithm (1979)
   - (Weakly) polynomial time
     - Time depends on precision
   - Compute smaller and smaller ellipsoids containing optimum

**Karmarkar** - Interior-Point Methods (1984)
   - Move through polytope’s interior
   - (Weakly) polynomial
   - Practical

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