Linear Programming (LP):
- Fundamental optimization problem in $\mathbb{R}^d$
- Given a set of $n$ linear constraints (halfspaces) $H = \{h_1, \ldots, h_n\}$
  \[ h_i: a_{i1}x_1 + \ldots + a_{id}x_d \leq b_i \]

- Given a linear objective function
  \[ f(\mathbf{x}) = c_1x_1 + \ldots + c_dx_d = \mathbf{c}^T\mathbf{x} \]

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

\[
\begin{align*}
\text{maximize:} & \quad c^T x \\
\text{subject to:} & \quad Ax \leq b
\end{align*}
\]

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.

![Diagram](image)
History:

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 $O(n^{1.5})$ vertices

- Karp - Not known to be NP-hard

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

Karmarkar - Interior-Point Methods
(1984) - Move through polytope's
  interior
- (Weakly) polynomial
- Practical
LP in constant-dimensional space
- Assume - \( n \) is large
  \( d \) is a constant
- We'll present a (randomized) algorithm with (expected) running time \( O(d!n^n) = O(n) \)

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 \( h_1, \ldots, h_{n-1} \)
  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 \( v_{n-1} \notin h_n \) 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.

\[ v_{n-1} \] not feasible \( \Rightarrow \) below \( h_n \)

\[ v_n \] if not on \( h_n \) \( \Rightarrow \) above \( h_n \)

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

By convexity, \( p \in \text{feasible polytope} \)

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

\( \Rightarrow p \) is better solution than \( v_n \)

\( \times \) contradiction!

How to update?

- Intersect \( h_1, ..., h_{n-1} \) with \( h_n \) - \( O(d \cdot n) \)

- This yields a \( (d-1) \text{ dim} \) polytope

- Project \( c \) onto \( l_n \rightarrow c' \)

- Solve this \( (d-1) \text{ dim LP} \) recursively (If \( d = 1 \), solve by brute force \( O(n) \))

(See latex notes for details)
Running time? Pretty bad - $\Theta(n^d)$
- Let $W_\ell(n)$ be worst-case complexity for $n$ halfspaces in dim $d$

- Recurrence:

$$W_\ell(n) = W_\ell(n-1) + d + d \cdot n + W_{\ell-1}(n-1)$$

**Claim:** $W_\ell(n) = O(n^d)$

**Sketch:** Very similar recurrence:

$$W'_\ell(n) = W'_\ell(n-1) + W'_{\ell-1}(n)$$

Note similarity with binomial coeffs:

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

It is well known that $\binom{n}{d} = O(n^d)$
Applies to $W'$ as well.

**How to fix this?**

Easy! Randomize the choice of $h_n$

Why?

$$W_\ell(n) = W_\ell(n-1) + d + d \cdot n + W_{\ell-1}(n-1)$$

This solves to $O(n^d)$

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 \(- O(n)\)
2. Find initial subset \( \{ h_1, \ldots, h_d \} \) that provide
   initial optimum \( v_d \) (or return “unbounded”)
   \(- O(d \cdot n) \) (see text)\n3. 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 \( (v_{n-1} \notin h_n) \) return \( v_{n-1} \) \(- O(d)\)
5. else, project \( \{ h_1, \ldots, h_{n-1} \} + c \) onto \( h_n \) \(- O(dn)\)
   the bounding hyperplane for \( h_n \).
   Solve recursively, letting \( v_n \) be result. Return \( v_n \)

Expected Case Running Time:
- The running depends on the (random) choice of \( h_n \)
- 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 \((h_1, \ldots, h_d \text{ aren't})\)

**Recurrence:**

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

- **Recursively compute** \(V_{n-1}\)
- **Test if** \(v_{n-1} \in h_n\)
- **If not**
- **Project** \(h_1, \ldots, h_{n-1}\) onto \(h_n\)
- **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?

![Diagram](Image)
Observation: The optimum is determined by $d$ halfspaces (assuming general position)

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

$\Rightarrow p_n = \frac{d}{n}$  
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: $T_d(n) \leq \gamma_d \cdot d! \cdot n$, 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. $\leq \gamma_d \cdot d! \cdot (n-1) + d + \frac{d}{n} \left( d \cdot n + \gamma_{d-1} (d-1)! \cdot n \right)$

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

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

\text{want:} \\
\leq \gamma_d d! \cdot n

Suffices to select \( \gamma_d \) such that

\[ d + d^2 + \gamma_{d-1} d! - \gamma_d d! \leq 0 \]

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

We can satisfy this by setting:

\[
\begin{align*}
\gamma_1 & \leq 1 \\
\gamma_d & \leq \frac{d + d^2}{d!} + \gamma_{d-1}
\end{align*}
\]

\[ \Rightarrow \gamma_d \text{ is a constant depending on } d \]

**Summary:**

- Randomized algorithm for LP
- Expected run time of LP is \( \mathcal{O}(d! n) = \mathcal{O}(n) \) (since we assume \( d \) is constant)
- Variation depends on random choices, not input
- (Seidel) Prob of running slower extremely small