First, let’s talk about the complexity of algorithm. How could we show that any simplex method is exponential? The only hope is by studying the graph (vertices, edges) of the feasible regions of LPs.

**Definition 1** Given a pointed polyhedron $Q$ and two vertices $v$ and $w$ of $Q$, $d_Q(v, w)$ is the smallest $k$ such that there are vertices $v_0 = v, v_1, \ldots, v_k = w$ of $Q$ with $[v_{j-1}, v_j]$ an edge of $Q$ for $1 \leq j \leq k$.

**Definition 2** The diameter of $Q$ is $\delta(Q) := \max\{d_Q(v, w) : v$ and $w$ are vertices of $Q\}$.

Finally:

**Definition 3** $\Delta_u(d, n) := \max\{\delta(Q) : Q$ is a pointed polyhedron in $\mathbb{R}^d$ with $n$ facets\};

$\Delta(d, n) := \max\{\delta(Q) : Q$ is a bounded polyhedron in $\mathbb{R}^d$ with $n$ facets\}.

In 1957, W. M. Hirsch conjectured $\Delta_u(d, n) \leq n - d$.

If $n \leq 2d$, $\Delta_u(d, n) = n - d$ seems best possible: since at least $n - d$ inequalities must be
exchanged. Klee and Walkup (‘67) showed that it is false (they constructed a particular counter example):
\[ \Delta_u(d, n) \geq n - d + \min\{\frac{d}{4}, \frac{n - d}{4}\}. \]

But the bounded case is still open (right or wrong). It is called the Hirsch conjecture:
\[ ?\Delta(d, n) \leq n - d. \]

Is it polynomial or exponential? The answer (Kalai and Kleitman, 1992): It is subexponential:
\[ \Delta_u(d, n) \leq (4d)^{\log_2 N} = n^{\log_2 d + 2}. \]

Log (this bound) is \( O(\log_2 n \cdot \log_2 d) \); Log (polynomial) is linear in \( \log_2 n, \log_2 d \); Log (exponential) is linear in \( n, d \). (We can see that the first one is between the other two, so it is superpolynomial, but subexponential.)

Is there a “simple” simplex method that is subexponential? Kalai gave a randomized simplex method whose expected number of steps is:
\[ \exp(K \sqrt{n \log_2 d}). \]

Here, \( K \) is an absolute constant. (For the TSP, \( n \) can be gigantic, so it may still look like an exponential, but it is just an upper bound.)

Now, let’s talk about the complexity of problems.

**Introduction to the ellipsoid method.** Let’s be more precise about “what is a polynomial-time algorithm”?

**Definition 4** An instance of an optimization problem, is a feasible set \( F \) and a cost function \( c : F \rightarrow \mathbb{R} \). The objective is to \( \min c \) over \( F \). An optimization problem is just a set of such instances.

The LP problem is the set of all instances where \( F \) is a polyhedron and \( c \) is a linear function.

An algorithm applies to a problem, and generates a solution for all its instances. Question: How long does the algorithm take? For LP, an instance is defined by \((A, b, c)\) (the data set). We’ll require the data to be integer-valued (or equivalently rational). We can write down an integer
\[ Z = \pm(Z_k2^k + Z_{k-1}2^{k-1} + \cdots + Z_02^0), \]
with each \( Z_j = 0 \) or \( 1 \), its binary representation in \( k \) bits, where \( k \) is about \( \lceil \log_2 |Z| \rceil \) (rounded up).

**Definition 5** \( \text{size}(Z) := \lceil \log(|Z| + 1) \rceil + 1, \) and \( \text{size}(A, b, c) := \sum_i \sum_j \text{size}(a_{ij}) + \sum_i \text{size}(b_i) + \sum_j \text{size}(c_j) = L. \)

This is \( O(\min \log_2 U) \), when \( U \) bounds all \( |a_{ij}|'s, |b_i|'s \) and \( |c_j|'s. \)

**Definition 6** A polynomial-time algorithm for a problem is one that, applied to any instance of that problem, gives a solution in a number of bit operations (+, −, ×, comparisons) that is bounded by a polynomial in its size.
Note, if an algorithm takes a polynomial number of arithmetical operations (+, -, \times, \div) on integers whose size remains polynomial in the size of the instance, this is a polynomial-time algorithm.

Is there a polynomial-time algorithm for LP? Yes. Khachiyan (1979, 1980) showed a polynomial-time algorithm: O(n^2L) iterations, each requiring O(n^2) arithmetical operations on integers of length O(L). He used the ellipsoid algorithm, which was developed by Yudin and Nemirovski (1976) and Shor (1977) for general convex programming.

**The ellipsoid algorithm**: This algorithm is applied to the feasibility form of LP: \( A^T x \leq b \).

Assume \( A \) is an \( m \times n \) matrix, so there are \( n \) inequalities in \( m \) unknowns.

A problem

\[
\min_{\bar{x}} \quad \bar{c}^T \bar{x} \\
A \bar{x} = \bar{b}, \\
\bar{x} \geq 0.
\]

where \( \bar{A} \) is an \( \bar{m} \times \bar{n} \) matrix, can be transferred into

\[
\begin{align*}
\bar{A} \bar{x} & \leq \bar{b}, \\
-\bar{A} \bar{x} & \leq -\bar{b}, \\
-1 \bar{x} & \leq 0, \\
\bar{c}^T - \bar{b}^T \bar{y} & \leq \bar{c}, \quad \text{(dual)} \\
\end{align*}
\]

So it is enough to be able to “solve” \( A^T x \leq b \).

Suppose we know,

\[
Q := \{ x \in \mathbb{R}^m, A^T x \leq b \} \subseteq B(0, R) = \{ x \in \mathbb{R}^m : \| x \| \leq R \},
\]

Figure 2: \( Q \) and \( B(\bar{x}, r) \).
and, if $Q$ is nonempty,

$$B(\hat{x}, r) \subseteq Q,$$

where $B(\hat{x}, r) = \{x \in \mathbb{R}^m : \|x - \hat{x}\| \leq r\}$.

for some (unknown!!) $\hat{x}$ and some $0 < r < R$.

What does the algorithm do? This algorithm generates a sequence of ellipsoids:

$$E_k = \{x \in \mathbb{R}^m : (x - x_k)^T B_k^{-1} (x - x_k) \leq 1\},$$

where $B_k \in \mathbb{R}^{m \times m}$ is symmetric and positive definite, i.e., $v^T B_k v > 0$ for all $v \neq 0$. So $E_0 = B(0, R) \supseteq Q$. At iteration $k$, either $x_k \in Q$ (great — stop! for we only need to find a feasible solution), or find a constraint, say $a_j^T x \leq b_j$, that is violated by $x_k$. Then find the minimum volume ellipsoid $E_{k+1}$ containing $\{x \in E_k : a_j^T x \leq a_j^T x_k\}$.

Figure 3: The iteration steps.
Key:

\[ \text{vol}(E_{k+1}) < \exp\left(-\frac{1}{2(m+1)}\right) \times \text{vol}(E_k). \]

This is the magic thing that makes everything work.