**CSE 521: Design and Analysis of Algorithms I** **Fall 2022**

# Lecture 17: Linear Programming and the Ellipsoid Method

*Lecturer: Haotian Jiang* *11/29/22*

**Disclaimer**: *These notes have not been subjected to the usual scrutiny reserved for formal publications.*

In this lecture, we introduce Linear Programming and algorithms for solving Linear Programs (LPs). The extraordinary success of the theory of LPs lies in the fact that they can be solved efficiently. Our particular focus will be on the ellipsoid method (developed by Shor [6], Yudin and Nemirovski [9]) which was used to give the first polynomial time algorithm for solving LPs in a breakthrough work by Kachiyan [5].

## 17.1 Linear Programming

The easiest class of optimization/feasibility problems examples are linear systems of equations, which can be solved efficiently by matrix inversion. Given a (full rank) matrix $A$, to solve $Ax = b$, it is enough to compute $A^{-1}b$. When the matrix $A$ does not have full rank, standard linear algebraic manipulations, such as Gaussian ellimination, can be applied.

In this lecture, we study the optimization of a linear cost (or objective) function over a linear system of inequalities, a problem widely known as Linear Programming. LPs can be used to model problems in a wide range of disciplines from engineering to nutrition and to the stock market. At some point it was estimated that half of all computational tasks in the world correspond to solving linear programs.

A general LP can be formalized, in *canonical form*, as follows:

$$\begin{aligned} \min_x \ & \langle c, x\rangle \\ \text{s.t. } & Ax \le b, \end{aligned} \tag{17.1}$$

where $x \in \mathbb{R}^n$ is represents the variables, $c \in \mathbb{R}^n$ defines the objective function, and $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ define the constraints. Often it is not uncommon to see LPs given in the following *standard form*:

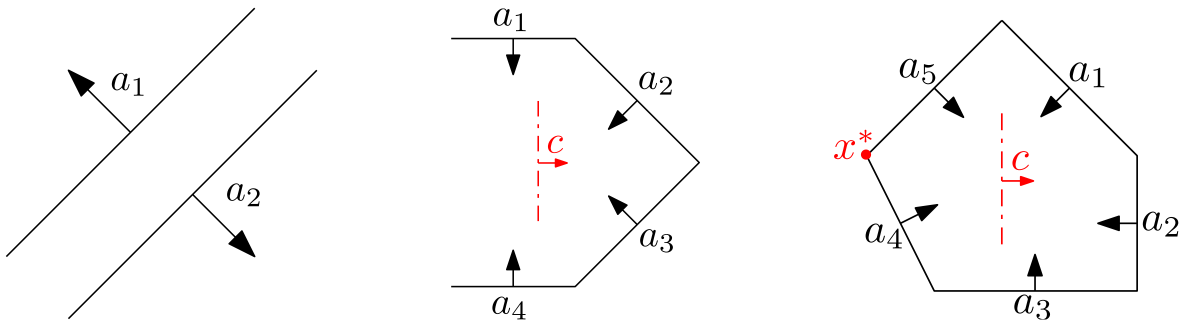
$$\begin{aligned} \min_x \ & \langle c, x\rangle \\ \text{s.t. } & Ax = b, \\ & x \ge 0. \end{aligned} \tag{17.2}$$

The two forms of LPs in (17.1) and (17.2) are fairly general and are actually equivalent. One can model various types of constraints interchangeably using the canonical and standard forms. For example, an inequality constraint $\langle a_1, x\rangle \ge b_1$ can be written as the inequality constraint $\langle -a_1, x\rangle \le -b_1$, and subsequently in standard form $\langle -a_1, x\rangle + s_1 = -b_1$ and $s_1 \ge 0$ by introducing the additional variable $s_1$. On the other hand, an equality constraint $\langle a_1, x\rangle = b_1$ can be equivalently written as two inequality constraints $\langle a_1, x\rangle \le b_1$ and $\langle -a_1, x\rangle \le -b_1$. The readers are invited to go down this line of argument to convince themselves that (17.1) and (17.2) are actually equivalent, and to think about how the number of constraints and variables change when going from one form to the other.

This lecture primarily works with LPs in the canonical form (17.1). The objective function can be viewed as a hyperplane in $\mathbb{R}^n$ with normal vector $c$. Further, one can express the constraint matrix $A$ as a series of row vectors:

$$A = \begin{bmatrix} a_1^{\mathrm{T}} \\ a_2^{\mathrm{T}} \\ \vdots \\ a_m^{\mathrm{T}} \end{bmatrix}.$$

When viewed from this perspective, it is easy to see the constraint set, or the *feasible set*, of an LP $\langle a_i, x\rangle \le b_i,\ i \in \{1, 2, \ldots, m\}$ as the intersection of finitely many halfspaces, an object known as a *polyhedron*. Depending on the shape of the polyhedron, the solution to an LP will always be one of the following three cases shown in Figure 17.2.



(a) Infeasible domain of $x$: No solution.

(b) Unbounded domain of $x$: $x^* = \pm\infty$

(c) Closed, bounded and compact domain of $x$: at least one vertex of the domain of $x$ is in the solution set $x^*$.

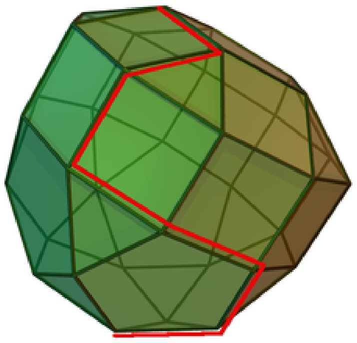
Figure 17.1: The possible domains of the solution variable $x$ for an LP, along with the solution.

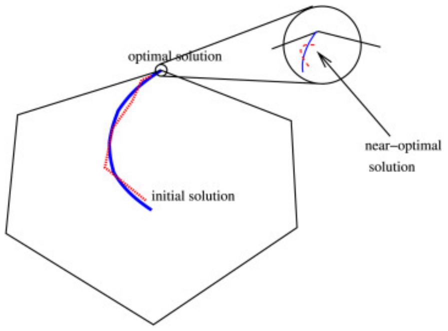
## 17.2 Algorithms for Solving Linear Programming

There are many well known algorithms for solving LPs. Three of the most well-known algorithms are the simplex method, the ellipsoid method (more generally, cutting plane methods), and the interior-point methods (IPMs). To make things simple, we assume throughout the rest of the lecture that the feasible set of the given LP is closed and bounded, i.e. it is as in Figure 17.1c. In this case, the LP has an optimal solution which corresponds to a vertex of its feasible region. There are standard, though sometimes technical, ways to detect and handle LPs which are of other two types.

The simplex method (see Figure 17.2a) is based on a very geometrically intuitive idea: start at a vertex of the feasible set $P = \{x \in \mathbb{R}^n : Ax \le b\}$ and move to an adjacent vertex with lower objective; continue until an optimal solution is reached. Theoretically, the worst case time complexity of the simplex method is exponential; however, the simplex method is very fast in practice, and toolboxes such as CPLEX and Gurobi can solve very large LPs (millions of solution variables) in a few seconds.

Interior point methods (see Figure 17.2b) solve an LP by starting at the interior of its feasible region and following what is known as a *central path*. It is beyond the scope of this course to define this object and this class of algorithms. For our purpose, it is sufficient to know that IPMs have both desirable theoretical complexity and fast practical efficiency. They are one of the most central class of methods for the recent breakthroughs in convex optimization. Roughly speaking, using an interior point method, one can solve a general LP with $n$ variables by solving $\tilde{O}(\sqrt{n})$ many systems of linear equalities.


(a) The simplex method for solving LPs. Picture credit to wikipedia.



(b) Interior-point methods for solving LPs. Picture credit to work by Mehlhorn and Saxena.

The first polynomial time algorithm for solving LPs is based on the ellipsoid method. This was the breakthrough result of Kachiyan [5], because of which the theory of LP becomes so prevalent and successful. We will discuss the ellipsoid method in detail in the next section.

**Technical Remarks.** While we have not mentioned it in class, it is important to spend a few paragraphs formalizing what it even means to solve LPs in *polynomial time*. As we will see below, this is a non-trivial consideration.

Towards this end, we need to introduce the notion of binary encoding length, or *bit complexity*, of an LP. Given in the form of (17.1), the input of the LP is essentially consisted of the objective vector $c \in \mathbb{R}^n$, the constraint matrix $A \in \mathbb{R}^{m\times n}$ and the RHS constraint vector $b \in \mathbb{R}^m$. Computers could not work with real numbers since it only uses a fixed number of bits, so we might as well assume that all entries of $A, b$ and $c$ are rational numbers. For each rational entry $a \in \mathbb{Q}$, we assume that $a$ is described as $a = p/q$ where $p, q$ are integers ($q \neq 0$ in particular), and $p$, $q$ are both represented in binary. Then the bit complexity of the entry $a$, denoted as $\langle a\rangle$, is defined to be the total number of binary bits needed to encode both $p$ and $q$. Similarly, $\langle A\rangle$, $\langle b\rangle$ and $\langle c\rangle$ are defined as the sum of the bit complexities of all their entries. We shall define $\langle \mathsf{LP}\rangle := \langle A\rangle + \langle b\rangle + \langle c\rangle$ as the total number of bits needed to encode the entire LP in form (17.1). With these definitions, we will prove the following theorem by Kachiyan [5].

**Theorem 17.1** (Ellipsoid method, [5]). *Given an LP in the form (17.1), the ellipsoid method finds the solution to the LP in time* $\mathsf{poly}(m, n, \langle \mathsf{LP}\rangle)$.

The type of algorithms as in Theorem 17.4 is known as *weakly-polynomial time algorithms*, since their runtime depends on the size of the input $\langle \mathsf{LP}\rangle$. One could hope for a stronger type of algorithms, known as *strong-polynomial time algorithms*, whose runtime can be upper bounded by $\mathsf{poly}(m, n)$ which is completely independent of the input size. The simplex method is roughly of this later type (in the sense of being independent of the input size) but its runtime is exponential in the worst case. It is one of the biggest open questions in the theory of LP to obtain a strongly-polynomial time algorithm for LP. This question is known as Smale's 9th question [7], listed by Steve Smale as one of the 18 most significant mathematics problems of the 21st century.

## 17.3 The Ellipsoid Method

We start with a few basic definitions. A subset $P \subseteq \mathbb{R}^n$ is *convex*, if for any two points $x, y \in P$, the line segment $\{\lambda x + (1-\lambda)y : \lambda \in [0,1]\}$ is contained in $P$. A hyperplane is of the form $\{x \in \mathbb{R}^n : \langle a, x\rangle = b\}$

where $a \in \mathbb{R}^n \setminus \{0\}$ and $b \in \mathbb{R}$, i.e. it contains all the points which satisfy a single linear equation. One of the most fundamental result about convex set is the following consequence of the Hahn-Banach separation theorem.

**Theorem 17.2** (Hyperplane Separation Theorem)**.** *Let $P \subseteq \mathbb{R}^n$ be a bounded and closed convex set. Then for any point $x \in \mathbb{R}^n$, either $x \in P$, or there exists a non-zero vector $v$ such that $\langle v, x\rangle < \min_{y\in P}\langle v, y\rangle$.*

In other words, a point $x$ either lies inside the compact (i.e., bounded and closed) convex set $P$, or it can be separated from $P$ by the hyperplane $H = \{z : \langle v, z\rangle = \langle v, x\rangle\}$ which passes through $x$ but does not pass through any point in $P$. We shall assume that there is a procedure to either decide $x \in P$ or output the vector $v$ for the separating hyperplane in Theorem 17.2. Such a procedure is called a *Separation Oracle*.

The ellipsoid method aims at solving the following problem of computing a point in a convex set.

**Problem 17.3.** *Given a convex set $P \subseteq \mathbb{R}^n$ such that $rB_2^n \subseteq P \subseteq RB_2^n$, where $0 < r < R$ and $B_2^n$ is the $n$-dimensional unit Euclidean ball, and a Separation Oracle for $P$, compute a point $x \in P$.*

Before proceeding to introduce how the ellipsoid method solves Problem 17.3, it is natural to ask: why does this problem has anything to do with LPs? As we will see in the next section, the question of solving LPs can be reduced to the problem of finding a point $x \in P$ where $P = \{x : Ax \leq b\}$ is a polyhedron. Moreover, the assumption that $rB_2^n \subseteq P \subseteq RB_2^n$ corresponds to the assumptions that the polyhedron $P$ is full-dimensional and non-empty, and bounded, in which case we can choose $-\mathsf{poly}(m, n, \langle \mathsf{LP}\rangle) \leq \log r \leq \log R \leq \mathsf{poly}(m, n, \langle \mathsf{LP}\rangle)$. This suffices to give a polynomial time algorithm for LP as long as we can prove the following theorem.

**Theorem 17.4** (Ellipsoid Method)**.** *Let $\mathsf{SO}$ be the runtime of the Separation Oracle in Problem 17.3. Then there is an algorithm that solves Problem 17.3 in $O(n^2 \log(R/r) \cdot \mathsf{SO} + n^4 \log(R/r))$ time.*

**Remark 17.5.** *The ellipsoid method in Theorem 17.4 falls under a broader class of convex optimization algorithms known as the cutting plane methods. The best known runtime for Problem 17.3 is $O(n \log(R/r) \cdot \mathsf{SO} + n^3 \log(R/r))$ which was given by Jiang, Lee, Song and Wong in [4]. These type of methods are extremely useful theoretically, and have significant impact for combinatorial optimization as presented in the seminal work of Grötschel, Lovász, and Schrijver [2, 3]. Despite their outstanding performance in theory, the ellipsoid method and more generally cutting plane methods are not efficient in practice. The readers are advised to turn to either the simplex method or the interior-point methods if they would like to solve LPs in practice.*

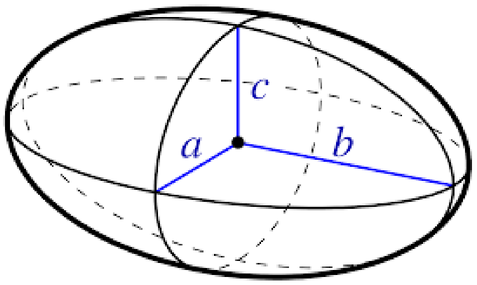
Now we turn to the presentation of the ellipsoid method and the proof of Theorem 17.4. Before we do so, let us first recall the definition of an ellipsoid. An ellipsoid is a convex set of the form

$$\begin{aligned}\mathcal{E} &:= \{x \in \mathbb{R}^n : (x - x_0)^\top A^{-1}(x - x_0) \leq 1\} \\ &= \{x \in \mathbb{R}^n : \langle A^{-1/2}(x - x_0), A^{-1/2}(x - x_0)\rangle \leq 1\},\end{aligned}$$

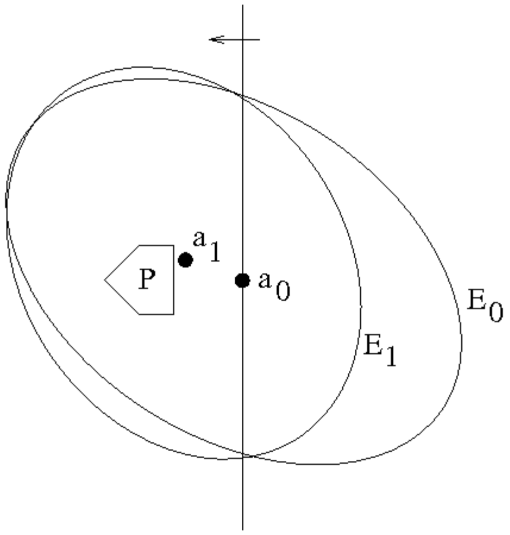
where $x_0 \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ is a PSD matrix, i.e. a symmetric matrix whose eigenvalues are all non-negative. Here we are using the notation of a quadratic form $y^\top My$ for a vector $y \in \mathbb{R}^n$ and a PSD matrix $M \in \mathbb{R}^{n\times n}$. The quadratic form $y^\top My$ is really just the inner product $\langle M^{1/2}y, M^{1/2}y\rangle$, which is the squared Euclidean length of the vector obtained by applying the linear transformation $M^{1/2}$ to the original vector $y$.

One can verify that $\mathcal{E}$ can also be written as $A^{1/2}B_2^n + x_0$. This is the convex set obtained by applying the linear transformation $A^{1/2}$ to the unit Euclidean ball $B_2^n$, and then shift it to be centered at $x_0$. From this, one can see that the ellipsoid $\mathcal{E}$ is centered at $x_0$, and its axes correspond to the eigenvectors of the matrix $A^{1/2}$ (see Figure 17.2a for an example).

The ellipsoid method operates as in Algorithm 1. We start with the ellipsoid $\mathcal{E}^{(0)} = RB_2^n$ which by assumption contains the convex set $P$. In each iteration, the algorithm checks the center $x^{(k)}$ of the current



(a) Example of a 3-dimensional ellipsoid with axes $a$, $b$ and $c$. Picure credit to wikipedia commons.



(b) The ellipsoid method for finding a point in $P$. Notations: $E_0 = \mathcal{E}^{(0)}$, $E_1 = \mathcal{E}^{(1)}$, $a_0 = x^{(0)}$, $a_1 = x^{(1)}$. Picture credit to Michel Goemans.

ellipsoid $\mathcal{E}^{(k)}$ (which is always maintained so that $P \subseteq \mathcal{E}^{(k)}$) using the Separation Oracle. If $x^{(k)} \in P$, then we are done since we have found a point in $P$; otherwise, the Separation Oracle will return a vector $v^{(k)}$ such that the entire convex set $P$ lies inside the halfspace $H^{(k)} := \{x \in \mathbb{R}^n : \langle v^{(k)}, x\rangle \geq \langle v^{(k)}, x^{(k)}\rangle\}$. Then we know that the convex set $P$ must satisfy $P \subseteq \mathcal{E}^{(k)} \cap H^{(k)}$. So the algorithm proceeds to construct a new ellipsoid $\mathcal{E}^{(k+1)} \supseteq \mathcal{E}^{(k)} \cap H^{(k)}$, where $\mathcal{E}^{(k+1)}$ is given by specifying its center $x^{(k+1)}$ and the matrix $A^{(k+1)}$ as in Algorithm 1. One iteration of the ellipsoid method is also illustrated in Figure 17.2b.

**Algorithm 1**

```
1:  procedure ELLIPSOID(SO, $P$)
2:      $x^{(0)} \leftarrow 0$ and $A^{(0)} \leftarrow R^2 \cdot I$
3:      Initial ellipsoid $\mathcal{E}^{(0)} := \{x \in \mathbb{R}^n : (x - x^{(0)})^\top (A^{(0)})^{-1}(x - x^{(0)}) \leq 1\}$,    ▷ $\mathcal{E}^{(0)} = R \cdot B_2^n$
4:      for $k = 0, 1, \cdots$ do
5:          Call the Separation Oracle at point $x^{(k)}$
6:          if $x^{(k)} \in P$ then return $x^{(k)}$    ▷ Found a point in $P$
7:          else
8:              Let $v^{(k)}$ be the output of the Separation Oracle
9:              Update $x^{(k+1)} \leftarrow x^{(k)} + \frac{1}{n+1} \cdot \frac{A^{(k)} v^{(k)}}{\sqrt{(v^{(k)})^\top A^{(k)} v^{(k)}}}$
10:             Update $A^{(k+1)} \leftarrow \frac{n^2}{n^2-1}(A^{(k)} + \frac{2}{n+1} \cdot \frac{A^{(k)} v^{(k)} (v^{(k)})^\top A^{(k)}}{(v^{(k)})^\top A^{(k)} v^{(k)}})$    ▷ Volume decreases by Lemma 17.6
11:         end if
12:     end for
13: end procedure
```

The central properties of the new ellipsoid $\mathcal{E}^{(k+1)}$ are the following.

**Lemma 17.6** (Properties of Ellipsoid Method)**.** *The ellipsoid $\mathcal{E}^{(k+1)}$ which is defined as $\{x \in \mathbb{R}^n : (x - x^{(k+1)})^\top (A^{(k+1)})^{-1}(x - x^{(k+1)}) \leq 1\}$ satisfies the following properties:*

1. *$\mathcal{E}^{(k+1)} \supseteq \mathcal{E}^{(k)} \cap H^{(k)}$, where $H^{(k)} := \{x \in \mathbb{R}^n : \langle v^{(k)}, x\rangle \geq \langle v^{(k)}, x^{(k)}\rangle\}$,*

*2.* $\mathrm{vol}(\mathcal{E}^{(k+1)}) \leq e^{-\frac{1}{2n}} \cdot \mathrm{vol}(\mathcal{E}^{(k)})$.

The formal proof of Lemma 17.6 is mainly technical calculations, and is not too important for this lecture. The main intuition is to first prove the lemma assuming that $\mathcal{E}^{(k)} = B_2^n$ and $v^{(k)} = e_1$ (the first cooridnate vector). In this special case, one can see that since $\mathcal{E}^{(k+1)}$ has to contain the half-ball $B_2^n \cap \{x : x_1 \geq 0\}$, the amount of distance that the center of $\mathcal{E}^{(k+1)}$ can be "pushed away" from the center of $\mathcal{E}^{(k)}$ (to achieve a smallest possible volume) is roughly $1/n$. From here, one can show that the volume decrease from $\mathcal{E}^{(k)}$ to $\mathcal{E}^{(k+1)}$ is only a factor of $e^{-\frac{1}{2n}} \approx 1 - \frac{1}{2n}$ instead of a constant (which is what one might expect, since we are cutting the ellipsoid $\mathcal{E}^{(k)}$ in half through its center!).

Now since an arbitrary ellipsoid is a linear transformation of $B_2^n$, the same volume bound still applies after performing the linear transformation. For interested readers, we refer to Section 2.2 of the excellent monograph by Bubeck [1] for a proof of Lemma 17.6. What is important for us to understand is the implication of the two statements in Lemma 17.6.

The first statement of Lemma 17.6 guarantees the correctness of the ellipsoid method. In particular, it guarantees that the ellipsoid $\mathcal{E}^{(k)}$ always contains $P$ until the center $x^{(k)} \in P$ at which point the algorithm is done. The second statement of Lemma 17.6 is the key to the convergence of the ellipsoid method. In particular, it immediately implies the following lemma.

**Lemma 17.7** (Convergence of Ellipsoid Method)**.** *The ellipsoid method solves Problem 17.3 in $O(n^2 \log(R/r))$ iterations.*

*Proof.* Since we have $rB_2^n \subseteq P$ and the algorithm always maintains $P \subseteq \mathcal{E}^{(k)}$, it follows that $\mathrm{vol}(\mathcal{E}^{(k)}) \geq \mathrm{vol}(rB_2^n) = \omega_n r^n$, where $\omega_n := \mathrm{vol}(B_2^n)$ is the volume of the unit Euclidean ball. Since we start with $\mathcal{E}^{(0)} = RB_2^n$ with volume $\mathrm{vol}(\mathcal{E}^{(0)}) = \omega_n R^n$, by Lemma 17.6, the ellipsoid in the $k$th iteration must satisfy

$$\omega_n r^n \leq \mathrm{vol}(\mathcal{E}^{(k)}) \leq (e^{-\frac{1}{2n}})^k \cdot \mathcal{E}^{(0)} \leq e^{-\frac{k}{2n}} \omega_n R^n.$$

The above implies that $k \leq 2n^2 \log(R/r)$. □

In other words, the ellipsoid method cannot run for too many iterations since otherwise the volume of $\mathcal{E}^{(k)}$ becomes so small that it cannot contain the convex set $P$. Now Theorem 17.4 essentially follows Lemma 17.7 by additionally noticing that each iteration of the ellipsoid method can be computed in time $O(n^2)$ by doing matrix-vector products.

## 17.4 Solving LPs Using the Ellipsoid Method

Great! Now that we have seen how the ellipsoid method allows us to compute a point $x$ in a convex set $P$ using a Separation Oracle, i.e. Problem 17.3, we are now ready to recover Kachiyan's breakthrough of solving LPs in polynomial time (modulo some technicalities which are not too important for the lecture).

For simplicity, let us first assume that the LP in (17.1) is bounded and non-degenerate, i.e. its feasible set can be empty, but is full-dimensional whenever it is non-empty. We will briefly touch upon how to remove these assumptions at the very end of this section.

**From Feasibility to Optimization.** Our first task is to reduce the problem of LP in (17.1) to the problem of computing a point $x \in P$ for some polytope $P$. There are multiple ways to do this. For example, we can conside the polyhedron $P_0 := \{x \in \mathbb{R}^n : Ax \leq b\}$ given by the constraints in (17.1), and look at $P = P_0 \cap \{x \in \mathbb{R}^n : \langle c, x\rangle \leq \mathsf{OPT}\}$, where $\mathsf{OPT}$ is our guess of the optimum value of (17.1). Whenever the LP is bounded, then we can restrict our attention to $\mathsf{OPT}$ that is within the range of $[-e^{\mathsf{poly}(m,n,\langle \mathsf{LP} \rangle)}, e^{\mathsf{poly}(m,n,\langle \mathsf{LP} \rangle)}]$. Also the optimum LP value must only take discrete values with denominator at most $e^{\mathsf{poly}(m,n,\langle \mathsf{LP} \rangle)}$. So we can simply perform a binary search over this range of discrete values for the correct $\mathsf{OPT}$ value. Once we have guessed the correct optimum LP value, then solving LP is equivalent to finding $x \in P$. Note that this is not quite Problem 17.3, since this reduction does not yet allow us to assume the sandwiching condition $rB_2^n \subseteq P \subseteq RB_2^n$, but this will come shortly.

**Separation Oracle.** For a polyhedron $P = \{x \in \mathbb{R}^n : Ax \le b\}$, the Separation Oracle can be implemented as follows. When queried at a point $x^*$, check each constraint $\langle a_i, x^* \rangle \le b_i$. If all these constraints are satisfied, then $x^* \in P$; otherwise, say a violated constraint is $\langle a_i, x^* \rangle > b_i$, then we know that $-\langle a_i, x^* \rangle < -b_i \le -\langle a_i, y \rangle$ for all points $y \in P$. So the Separation Oracle can simply return the vector $-a_i$. In other words, any violated constraint could be used to give a separating hyperplane as required for implementing the Separation Oracle.

**Sandwiching Condition for $P$.** For our purposes, let us assume for simplicity that the polyhedron $P = \{x \in \mathbb{R}^n : Ax \le b\}$ from the previous step is bounded and non-degenerate, i.e. $P$ can be empty but whenever it is not empty it is full-dimensional.

Now under these assumptions, one can prove that the polytope $P$ is contained in a large ball $RB_2^n$ with radius $\log R = \mathsf{poly}(m, n, \langle \mathsf{LP} \rangle)$. If $P$ is non-empty, then $P$ also contains a small ball $rB_2^n$ of radius $\log r = -\mathsf{poly}(m, n, \langle \mathsf{LP} \rangle)$. Thus the ellipsoid method in Theorem 17.4 can be directly applied to the polytope $P$ using the Separation Oracle above. If the ellipsoid method fails to find a point in $P$ after $\mathsf{poly}(m, n, \langle \mathsf{LP} \rangle)$ iterations, then it certifies that the sandwiching condition for $P$ fails, which certifies that $P$ is empty.

**Removing Assumptions.** In the discussion above, we have assumed that the polyhedron $P$ in which we want to find a point is bounded and non-degenerate. These assumptions can be removed roughly as follows. For the non-degenerate assumption, if $P$ is degenerate, meaning that it is non-empty but is not full-dimensional, then directly running the ellipsoid method would be fruitless since $\text{vol}(P) = 0$. In this case, the correct idea is to find the affine subspace $W$ containing $P$ and run ellipsoid method on the subspace $W$.

The boundedness assumption can be removed using LP duality which will be discussed in a later lecture. Roughly speaking, an LP is unbounded if and only if its dual is infeasible (aka its constraints cannot all be satisfied). So checking the boundedness of an LP is equivalent to checking the non-emptyness of its dual, which we know how to handle.

There are several technicalities in the above discussions of removing the assumptions. We omit these technicalities since they are beyond the purpose of this course.

## 17.5 LPs with Exponential Number of Constraints

Sometimes when we cast a combinatorial optimization problem as an LP, the resulting LP has an exponential number of constraints. Although explicitly writing out such an LP would take exponential time, an amazing and surprising feature of the ellipsoid method is that it can be used to solve many of these LPs in polynomial time. The ingenious idea, due to Grötschel, Lovász, and Schrijver [2], is that the ellipsoid method only relies on a Separation Oracle of the constraint set of the LP, and do not need to know the LP explicitly[1].

One such example is the travelling salesman problem (TSP). More examples can be found in the excellent textbook on approximation algorithms by Williamson and Shmoys [8]. The TSP problem is the following: given a set $V$ of cities and distances between each pair of cities, what is the shortest possible tour that visits each city exactly once and returns to the original city? If we denote $c_{u,v}$ as the distance between cities

[1]As a comparison, both the simplex method and the interior-point methods require explicit knowledge of the LP.

$u, v \in V$, and $x_{u,v} \in \{0,1\}$ the indicator of whether we take the edge between cities $u$ and $v$, then the TSP problem can be captured by the following integer linear program:

$$\begin{aligned}
\min \quad & \sum_{u,v} x_{u,v} c_{u,v} \\
\text{s.t.} \quad & \sum_{u} x_{u,v} = 2 \qquad \forall v \in V, \\
& \sum_{u \in S, v \notin S} x_{u,v} \geq 2 \quad \forall S \subsetneq V, S \neq \emptyset, \\
& x_{u,v} \in \{0,1\} \qquad \forall u, v \in V.
\end{aligned}$$

The first set of constraints capture that for each city $v \in V$, exactly two edges adjacent to $v$ is taken (i.e. its corresponding $x$ value is 1) by the tour: one going into the city $v$, and one leaving $v$. The second set of constraints say that for any cut (recall this notion from the first lecture!) between $S$ and $V \setminus S$, at least two edges in the cut are taken – this ensures that the set of edges taken actually form a connected tour and visit every city. One can verify that this integer linear program exactly captures the TSP problem. But as the TSP problem is NP-Hard, one cannot solve the above integer linear program efficiently due to the integer constraints $x_{u,v} \in \{0,1\}$. So we need to look at the following LP relaxation[2] (known as the Held-Karp relaxation) where the integer constraints $x_{u,v} \in \{0,1\}$ are replaced by inequality constraints $x_{u,v} \in [0,1]$:

$$\begin{aligned}
\min \quad & \sum_{u,v} x_{u,v} c_{u,v} \\
\text{s.t.} \quad & \sum_{u} x_{u,v} = 2 \qquad \forall v \in V, \\
& \sum_{u \in S, v \notin S} x_{u,v} \geq 2 \quad \forall S \subsetneq V, S \neq \emptyset, \\
& x_{u,v} \in [0,1] \qquad \forall u, v \in V.
\end{aligned} \tag{Held-Karp LP}$$

Note that the Held-Karp LP has an exponential number of constraints, since there is an exponential number of choices of the set $S \subsetneq V$, $S \neq \emptyset$. However, we can still use the ellipsoid method to solve the Held-Karp LP in polynomial time as follows. We will always run the ellipsoid method on the affine subspace defined by $\sum_u x_{u,v} = 2$ for all $v \in V$, so the first set of constraints are always satisfied. Now all we need to show is how to implement a Separation Oracle for the set of inequality constraints of the Held-Karp LP, and as discussed earlier, this amounts to either assert that all inequality constraints are satisfied, or find a violated constraint when there is one.

Checking whether the constraints $x_{u,v} \in [0,1]$ are all satisfied $\forall u, v \in V$ is easy, since there are only $|V|^2$ of them. Now for the constraints $\sum_{u \in S, v \notin S} x_{u,v} \geq 2$ for all $S \subsetneq V, S \neq \emptyset$, the key idea is that we don't need to check all of them, we just need to find the set $S \subsetneq V, S \neq \emptyset$ achieving the minimum value of $\sum_{u \in S, v \notin S} x_{u,v}$ and see if its value is at least 2 or not. But this is just finding the minimum cut of the complete graph on $V$ with weight $x_{u,v}$ for each edge $(u,v)$, a problem we know how to solve efficiently back from the first lecture of this course! This efficient implementation of a Separation Oracle allows us to run the ellipsoid method without explicitly writing out the underlying exponential-sized LP. More details can be found in [2].

[2]Generalizations of this LP are used in many network design problems such as steiner tree, steiner forest, group steiner tree, etc.

## References

[1] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[2] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[3] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, 1988.

[4] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 944–953, 2020.

[5] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[6] Naum Z Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.

[7] Steve Smale. Mathematical problems for the next century. *The mathematical intelligencer*, 20(2):7–15, 1998.

[8] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[9] David B Yudin and Arkadii S Nemirovski. Evaluation of the information complexity of mathematical programming problems. *Ekonomika i Matematicheskie Metody*, 12:128–142, 1976.