Lecture 2

Convex Optimization

May 20, 2005
Lecturer: Nati Linial
Notes: Neva Cherniavsky

2.1 Introduction

In the field of algorithms, there are not many topics that are well-structured, especially when compared with a more established area like linear algebra. We tend to jump from one topic to another, and there are very few unifying principles. Linear and convex optimization is the exception to this rule. It is one of the only topics that has structure.

Convex programming is the more general version of linear programming, and to start we will discuss linear programming, since many of the same principles apply. Linear programming is the simplest general methodology for optimization. Optimization means there is some set Ω and a real function f defined on Ω, and we want the $\max_{x\in\Omega} f(x)$. But this goal is too general, and we can't say much about it; so we'll restrict the problem. For example, if Ω is an interval and f is continuous on Ω, we know what to do.

We let Ω be defined by a (finite) collection of linear equations and inequalities, and f linear. A direct application of this type of restriction comes up in a lot of situations. Furthermore, many other problems can be converted into a version of linear programming. A common technique is to take an NP-hard problem and relax it into a linear program, solve the LP, and get a reasonable approximation of the original problem.

2.2 Linear programming

Here is an example of a direct application of linear programming. Suppose you have a herd of cows and you need to feed them. There's some minimal amount of proteins, vitamins, and so on that each cow must eat. The food it can get these necessities from (hay, apples, sushi) has cost. So for example, in the table below we might have the data that 1 unit of hay has 3 units of protein and 2 units of vitamins.

	proteins	vitamins	minerals	$\cdots$
hay	3	2	$\cdots$	
apples				
sushi				
$\vdots$				

On the one hand, we want the cows to get what they need to be happy and healthy; on the other hand, we want to minimize cost. Suppose the minimal requirements are in a vector b so that if the cows required 5 units of proteins, 6 units of vitamins, etc, $b = (5, 6, \ldots)$. Call the matrix above that indicates how much of each necessity is provided by each food A. Write

$$xA \geq b$$

where x will be how much of each type of food we provide. Then we want to minimize the cost: given a cost vector c that indicates the price of one unit of each food type, $\min \langle c, x \rangle$.

There are lots of real world problems that map directly to a linear program - airline crew scheduling is one prominent example.

2.3 Discrete optimization

Suppose $x \in \{0, 1\}^n$ and we're given a flow problem (i.e. we're given a graph and vertices s and t). We could be looking for a collection of disjoint paths from s to t, or for the optimal flow from s to t. The latter falls directly into the LP framework, whereas the former does not. For the disjoint paths problem, we might write $\max \langle x, c \rangle$ subject to $Ax \leq b$, $x \in \{0, 1\}^n$. This is called an integer program and it's NP hard to solve exactly. We have lots and lots of optimization problems that map directly to integer programs, but it's a hard problem to solve. So we can relax it to a linear program by relaxing the constraint that $x \in \{0, 1\}$. Then we'll need to generate a solution to the integer program from the solution to the linear program. One way to do this is via randomized rounding; if the solution give 0.7, we round it to 1 with probability 0.7 and round it to 0 with probability 0.3.

One small example of how to show that this is close to the optimal solution of the integer program involved packing and covering. Packing and covering are two types of optimization that come up a lot. In packing, we want to get as many disjoint sets as possible to "fit" in some space; in covering we want to find as few sets as possible that cover the space. See Figure 1.1.

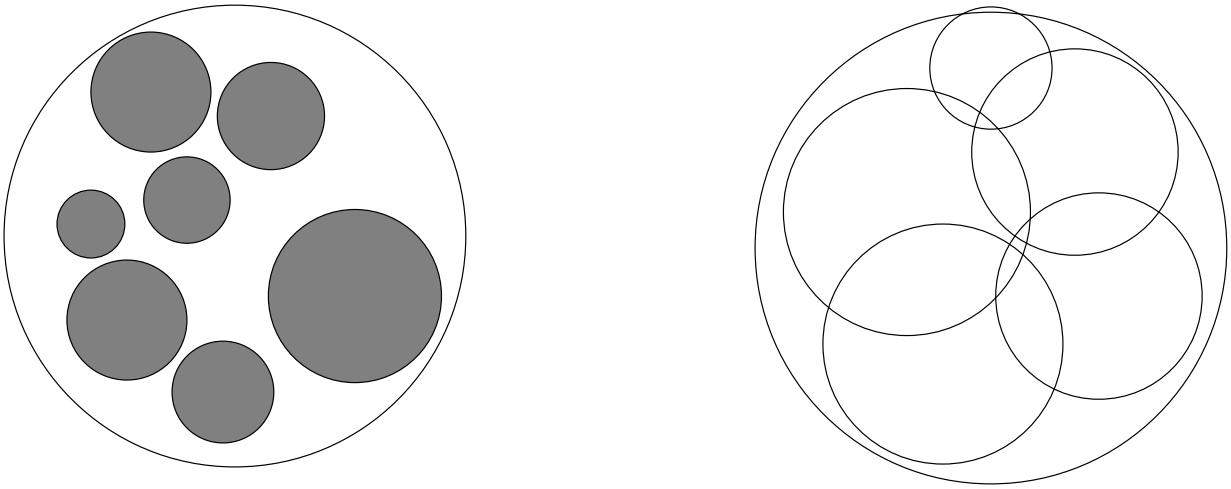

Figure 2.1: Packing versus covering

Packing is harder than covering, in that packing is hard to approximate, and covering is easy to approximate. Write the sets as $S_1, \ldots, S_k$, where each S_j is a 0-1 vector with a 1 in the ith spot if the ith element is in the set S_j. Let A be the matrix whose k rows are the 0-1 vectors $S_1, \ldots, S_k$. To solve covering, we want an $x \in \{0,1\}^k$ such that $xA \geq (1, 1, \ldots, 1) = \mathbf{1}$. This expresses that we've covered all the sets. To find the minimal covering, we write $\min \langle x, 1 \rangle$. This is an integer program and NP hard. So to solve it, we relax it to the linear program:

$$\begin{aligned} \min \langle x, 1 \rangle & \\ x &\geq 0 \\ xA &\geq \mathbf{1} \end{aligned}$$

1. Solve LP, call this solution x^*.
2. Select S_i with probability $x_i^* \cdot \log n$. Show that almost surely:
   - cost $\leq \log n \cdot$ opt.
   - It is a solution

We actually know this is tight; we can't get better than $\log n$ in polynomial time. But that theorem requires PCP theory.

2.4 Simplex algorithm

Ω is defined through a collection of linear equalities and inequalities: $\langle \alpha_i, x \rangle \geq \beta_i, \langle \gamma_j, x \rangle \geq \delta_j$ and so on. Such a set is a polyhedron. A polyhedron is an intersection of halfspaces; a polytope is a bounded polyhedron. Thus a linear program is the optimization of a linear function over a polytope or polyhedron.

In order to gain intuition into the Simplex algorithm, it's best to visualize in three or four dimensions. A plane is simply too primitive to gain understanding into what's going on. Think of what we're maximizing: $\{x | \langle \boldsymbol{\epsilon}, \boldsymbol{x} \rangle = a\}$. It's over a hypersphere. Typically, the maximizing point will be on the vertex of the polytope (or will be parallel to a face, so lie on the face). This gives us the following important observation: **The optimimum of an LP is always attained at a vertex of the relevant polyhedron**.

This is a nice fact; it provides us with a sort of discrete answer lying in a continuous domain. For an integer program, you can take the convex hull of possible solutions and run an LP on that, and often you can get an answer that is close to optimal. The intuition comes from geometry, but the way it works is via algebra. So we will need a kind of dictionary between geometry and algebra:

vertex	$\leftrightarrow$	basic feasible solution
Simplex moves from vertex to vertex	$\leftrightarrow$	move from feasible solution to another, better feasible solution

Simplex was created in 1949 and the author (Dantzig) thought it was so easy and obvious that someone else would come up with something better within a few years. But no one did for 30 years. In 1972 Klee and Minty showed that a large class of variants on Simplex can be exponential (time) on a pathological case. It is still open whether there exists a strongly polynomial algorithm for linear programming. There exist mildly exponential or subexponential algorithms that are similar to Simplex (Simplex-like), e.g. Kalai Kleitman.

Khachiyan came up with the first polynomial time algorithm for LP, based on ideas from the USSR. It is polynomial time in the input length, i.e. how many bits you need to encode A. It is not polynomial in

m and n; if it was, it would be "strongly" polynomial. In general, Simplex runs much more quickly than we would expect, and why this is the case is not understood. We could think of the polytope as a graph, with vertices corresponding to vertices and edges to edges, and ask questions about this graph: does it have a small diameter, i.e. a short path? If we had an oracle telling us where to go, could we guarantee that Simplex finished in polynomial time?

2.5 Convex optimization

Let Ω be a general convex set and the objective function be a linear function. We know a full dimensional polytope is the bounded intersection of finitely many half-spaces. This is equivalent to the convex hull of a finite set of points (vertices of the polytope). A convex set is more general. It is the (not necessarily finite) intersection of halfspaces; a ball, for example. We'll consider the following problem that can be shown equivalent to optimizing a linear function on a convex set: given a convex set P, decide whether $P = \emptyset$. I.e.:

$$\text{set } Q, \text{obj fn } f, \exists x \in Q | f(x) \geq c? \text{ Define } P = Q \cap \{u | f(u) \geq c\}, \text{ is } P = \emptyset? \tag{2.1}$$

Grötschel Lovász Schrijver created the right theoretical background within which we want to look at Khachiyan's algorithm. P is given by oracles. We need two types of oracles:

- Membership: is $x \in P$?
- Separation: If $x \notin P$, return a hyperplane H that separates P and x, i.e. H such that $x \in H^+, P \subseteq H^-$.

2.6 Ellipsoid algorithm

To solve the question 1.1, we use an ellipsoid.

1. Serious assumption: P is given via oracles.
2. P is included in a ball of radius $R = \mathcal{B}(0, R)$ (R can be very big, time depends on $\log r$) and if $P \neq \emptyset$, P contains a ball of radius $r > 0$ (r can be very small, time depends on $\log(1/r)$).

Here is the idea of the algorithm. Our invariant is we always have an ellipsoid that contains P. The ellipsoid is a centrally symmetric "egg". $E_0 = \mathcal{B}(0, R)$. $E_1, E_2, \ldots, E_i, \ldots$ are ellipsoids centered at point x_i. $E_i \supseteq P$. The main step of the algorithm is to test if $x_i \in P$. If so, report that $P \neq \emptyset$. If not, get the hyperplane H such that $x_j \in H^+$, $P \subseteq H^-$.

The polynomial running time depends on a fact from geometry. Let $E \subseteq \setminus^n$ be an n dimensional ellipsoid and let H be the hyperplane thgrough the center of E. Let $E^+ = H^+ \cap E$ be a half-ellipsoid. Then there exists an ellipsoid $E' \supset E^+$ and vol$(E') < (1 - \frac{1}{2n})vol(E)$. So the algorithm is to repeat the main step until we either find $x_j \in P$ or $volE_j$ is too small. If we know a lower bound on the size of P and an upper bound on the size of E_j, then we would know if it's impossible for E_j to contain P.

The hard part is the separation oracle. The Löwner John theorem says that every n dimensional convex body can be approximated well by an ellipsoid: $\forall K \exists E | nE \supset K \supset E$. An ellipsoid is an image of a ball

under a linear transform. You prove this using the fact that there is a linear transform that takes the ellipsoid back to the ball; and it’s enough to prove it for a unit ball.