Lecture 22

Linear Programming II

22.1 Overview

In this lecture we describe a very nice algorithm due to Seidel for Linear Programming in low-dimensional spaces. We then discuss the general notion of Linear Programming Duality.

22.2 Seidel's LP algorithm

We now describe a linear-programming algorithm due to Raimund Seidel that solves the 2-dimensional (i.e., 2-variable) LP problem in $O(m)$ time (recall, m is the number of constraints), and more generally solves the d-dimensional LP problem in time $O(d!m)$.

Setup: We have d variables $x_1, \ldots, x_d$. We are given m linear constraints in these variables $\mathbf{a}_1 \cdot \mathbf{x} \leq b_1, \ldots, \mathbf{a}_m \cdot \mathbf{x} \leq b_m$ along with an objective $\mathbf{c} \cdot \mathbf{x}$ to maximize. (Using boldface to denote vectors.) Our goal is to find a solution $\mathbf{x}$ satisfying the constraints that maximizes the objective.

The idea: Here is the idea of Seidel's algorithm. Let's add in the constraints one at a time, and keep track of the optimal solution for the constraints so far. Suppose, for instance, we have found the optimal solution $\mathbf{x}^*$ for the first $m-1$ constraints (let's assume for now that the constraints so far do not allow for infinitely-large values of $\mathbf{c}\cdot\mathbf{x}$) and we now add in the mth constraint $\mathbf{a}_m \cdot \mathbf{x} \leq b_m$. There are two cases to consider:

Case 1: If $\mathbf{x}^*$ satisfies the constraint, then $\mathbf{x}^*$ is still optimal. Time to perform this test: $O(d)$.

Case 2: If $\mathbf{x}^*$ doesn't satisfy the constraint, then the new optimal point will be on the $(d-1)$-dimensional hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$, or else there is no feasible point.

Let's now focus on the case $d = 2$ and consider the time it takes to handle Case 2 above. With $d = 2$, the hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$ is just a line, and let's call one direction "right" and the other "left". We can now scan through all the other constraints, and for each one, compute its intersection point with this line and whether it is "facing" right or left (i.e., which side of that point satisfies the constraint). We find the rightmost intersection point of all the constraints facing to the right and the leftmost intersection point of all that are facing left. If they cross, then there is no solution. Otherwise, the solution is whichever endpoint gives a better value of $\mathbf{c} \cdot \mathbf{x}$ (if they give

the same value – i.e., the line $\mathbf{a}_m \cdot \mathbf{x} = b_m$ is perpendicular to $\mathbf{c}$ – then say let's take the rightmost point). The total time taken here is $O(m)$ since we have $m-1$ constraints to scan through and it takes $O(1)$ time to process each one.

Right now, this looks like an $O(m^2)$-time algorithm for $d = 2$, since we have potentially taken $O(m)$ time to add in a single new constraint if Case 2 occurs. But, suppose we add the constraints in *a random order*? What is the probability that constraint m goes to Case 2?

Notice that the optimal solution to all m constraints (assuming the LP is feasible and bounded) is at a corner of the feasible region, and this corner is defined by two constraints, namely the two sides of the polygon that meet at that point. If both of those two constraints have been seen already, then we are guaranteed to be in Case 1. So, if we are inserting constraints in a random order, the probability we are in Case 2 when we get to constraint m is at most $2/m$. This means that the *expected* cost of inserting the mth constraint is at most:

$$\mathbf{E}[\text{cost of inserting } m\text{th constraint}] \leq (1 - 2/m)O(1) + (2/m)O(m) = O(1).$$

This is sometimes called "backwards analysis" since what we are saying is that if we go backwards and pluck out a random constraint from the m we have, the chance it was one of the constraints that mattered was at most $2/m$.

So, Seidel's algorithms is as follows. Place the constraints in a random order and insert them one at a time, keeping track of the best solution so far as above. We just showed that the expected cost of the ith insert is $O(1)$ (or if you prefer, we showed $T(m) = O(1) + T(m-1)$ where $T(i)$ is the expected cost of a problem with i constraints), so the overall expected cost is $O(m)$.

What if the LP is infeasible? There are two ways we can analyze this. One is that if the LP is infeasible, then it turns out this is determined by at most 3 constraints. So we get the same as above with $2/m$ replaced by $3/m$. Another way to analyze this is imagine we have a separate account we can use to pay for the event that we get to Case 2 and find that the LP is infeasible. Since that can only happen once in the entire process (once we determine the LP is infeasible, we stop), this just provides an additive $O(m)$ term. To put it another way, if the system is infeasible, then there will be two cases for the final constraint: (a) it was feasible until then, in which case we pay $O(m)$ out of the extra budget (but the above analysis applies to the the (feasible) first $m-1$ constraints), or (b) it was infeasible already in which case we already halted so we pay 0.

What about unboundedness? One way we can deal with this is put everything inside a bounding box $-\lambda \leq x_i \leq \lambda$ (so, for instance, if all c_i are positive then the initial $\mathbf{x}^* = (\lambda, \ldots, \lambda)$) where we view λ symbolically as a limit quantity. For example, in 2-dimensions, if $\mathbf{c} = (0, 1)$ and we have a constraint like $2x_1 + x_2 \leq 8$, then we would see it is not satisfied by (λ, λ), intersect the contraint with the box and update to $\mathbf{x}^* = (4 - \lambda/2, \lambda)$.

So far we have shown that for $d = 2$, the expected running time of the algorithm is $O(m)$. For general values of d, there are two main changes. First, the probability that constraint m enters Case 2 is now d/m rather than $2/m$. Second, we need to compute the time to perform the update in Case 2. Notice, however, that this is a $(d-1)$-dimensional linear programming problem, and so we can use the same algorithm recursively, after we have spent $O(dm)$ time to project each of the $m-1$ constraints onto the $(d-1)$-dimensional hyperplane $\mathbf{a}_m \cdot \mathbf{x} = b_m$. Putting this together we have a recurrence for expected running time:

$$T(d, m) \;\leq\; T(d, m-1) + O(d) + \frac{d}{m}[O(dm) + T(d-1, m-1)].$$

This then solves to $T(d, m) = O(d!m)$.

22.3 Linear Programming Duality

Given a set of constraints like $\mathbf{a}_1 \cdot \mathbf{x} \leq b_1$ and $\mathbf{a}_2 \cdot \mathbf{x} \leq b_2$, notice that you can add them to create more constraints that have to hold, like $(\mathbf{a}_1 + \mathbf{a}_2) \cdot \mathbf{x} \leq b_1 + b_2$. In fact, any positive linear combination has to hold.

To get a feel of what this looks like geometrically, say we start with constraints $x_1 \leq 1$ and $x_2 \leq 1$. These imply $x_1 + x_2 \leq 2$, $x_1 + 2x_2 \leq 3$, etc. [draw picture] In fact, you can create any constraint running through the intersection point $(1, 1)$ that has the entire feasible region on one side by using different positive linear combinations of these inequalities.

Now, suppose you have a linear program in n variables (I've switched back from "d" to "n") with objective $\mathbf{c} \cdot \mathbf{x}$ to maximize. As we've discussed, unless the feasible region is unbounded (and let's assume for this entire discussion that the feasible region is bounded), the optimum point will occur at some vertex $\mathbf{x}^*$ of the feasible region, which is an intersection of n of the constraints, and have some value $v^* = \mathbf{c} \cdot \mathbf{x}^*$. An interesting fact is that just as in the simple example above, if you take the n constraints that define the vertex $\mathbf{x}^*$ and look at all positive linear combinations, you can again create any constraint you want going through $\mathbf{x}^*$ that has the feasible region on one side. In particular, it is possible to create the constraint $\mathbf{c} \cdot \mathbf{x} \leq v^*$ using some positive linear combination of them. [draw another picture]

Notice that if you do this – create the constraint $\mathbf{c} \cdot \mathbf{x} \leq v^*$ using some positive linear combination of the given constraints – then it gives a proof that $\mathbf{x}^*$ is optimal! (Since clearly you can't do better than v^*). In fact, it suggests we might try to *solve* for the optimal value you can get by trying to solve for a positive linear combination of the constraints that produces $\mathbf{c} \cdot \mathbf{x} \leq v$ for as small a value v as possible.

So, let's say the original LP was $A\mathbf{x} \leq \mathbf{b}$, where A is the matrix of coefficients a_{ij}. Then we can write what we are trying to do as a new linear program. In particular, we want variables $y_1, \ldots, y_m$ (these are the multipliers on each constraint) with $y_i \geq 0$ for all i, such that $\mathbf{y}^T A = \mathbf{c}$, and subject to that, we want to minimize $\mathbf{y}^T \mathbf{b}$ (which is what the number on the right-hand-side will be). Or, if you don't like matrix form, then as equations we want $y_1 a_{11} + y_2 a_{21} + \ldots + y_m a_{m1} = c_1$, etc. This is called the *dual* to the original LP. The dual has one variable for each constraint of the original (primal) LP and a constraint for each variable of the primal LP (plus non-negativity of the y's). This turns out to be powerful for a number of applications.

Formally, given the primal LP:

$$\begin{aligned} \text{maximize } & \mathbf{c}^T \mathbf{x} \\ \text{subject to } A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0}, \end{aligned}$$

the dual LP is:

$$\begin{aligned} \text{minimize } & \mathbf{y}^T \mathbf{b} \\ \text{subject to } \mathbf{y}^T A &\geq \mathbf{c}^T \\ \mathbf{y} &\geq \mathbf{0}, \end{aligned}$$

where we are using "$\geq$" for vectors to mean that the LHS is greater than or equal to the RHS in every coordinate, and similarly for "$\leq$".