Lecture notes for CMU's course on Linear Programming & Semidefinite Programming

Anupam Gupta, Ryan O'Donnell, and the scribes of 15-859E

November 5, 2013

Contents

Lecture 1

LPs: Algebraic View*

1.1 Introduction to Linear Programming

Linear programs began to get a lot of attention in 1940's, when people were interested in minimizing costs of various systems while meeting different constraints. We care about them today because we can solve them efficiently and a very general class of problems can be expressed as LPs. A linear program has variables, linear constraints on those variables, and a linear objective function which we aim to maximize or minimize. This might look something like the following:

$$\begin{aligned} x_1 &\geq 0 \\ x_1 + x_2 &\leq 2 \\ x_1 - x_2 &\geq 1 \\ x_2 &\geq 2 \\ \min \quad 3x_1 &+ 2x_2 \end{aligned}$$

The "feasible region", the settings for x_1, x_2 that satisfy the above constraints, look like this:
trict

Here is a simple example of a linear program called the *Diet problem*. There are n foods and m nutrients you need to have enough of in your diet. You'd like to spend the least money possible while getting enough of each nutrient. So, let a_{ij} denote the amount of nutrient i in each unit of food j, b_i be the minimum amount of nutrient i you need in your diet, and c_j be the cost of one unit of food j, and x_j be the variable representing the amount of food j you are solving to buy. These constraints are written as:

$$\sum_j a_{ij} x_j \geq b_j$$

$$x_j \geq 0$$

*Lecturer: Anupam Gupta. Scribe: Jamie Morgenstern.

And the objective function to minimize is the cost:

$$\min \sum_j c_j x_j$$

As you might notice, we suggestively chose a_{ij} notation for the coefficients of the constraints: usually, we do write LPs with matrices. To rewrite this in matrix form, let A be an $m \times n$ matrix with $A_{ij} = a_{ij}$, B be a $m \times 1$ column matrix with $B_i = b_i$, x be a $n \times 1$ column vector of variables, and c be the $n \times 1$ column vector such that $c_i = c_i$. Then, we write the constraints as

$$Ax \geq b$$
$$x \geq 0$$

and the objective as

$$\min c^T x$$

We will also find it useful to write A in two other ways: as a concatenation of its rows or its columns:

$$A = \begin{pmatrix} | & | & \dots & | \\ A_1 & A_2 & \dots & A_n \\ | & | & \dots & | \end{pmatrix} = \begin{pmatrix} — & a_1 & — \\ — & a_2 & — \\ — & \dots & — \\ — & a_m & — \end{pmatrix}$$

There are several forms in which people write their LPs. The minimization of $c^T x$ can be recast as a maximization of $-c^T x$. Similarly, one can recast upper bound constraints of the form

$$a_i x \geq b_i$$

to the equivalent lower bound constraints

$$-a_i x \leq b_i.$$

It is also easy to translate equality constraints into pairs of inequality constraints:

$$a_i x = b_i \iff a_i x \leq b_i \text{ and } a_i x \geq b_i$$

One can starting from inequality constraints and get to equality constraints (along with nonnegativity inequalities) as follows:

$$a_i x \leq b_i$$

becomes

$$a_i x + s_i = b_i,$$
$$s_i \geq 0.$$

where we have introduced new variables s_i, the "slack" variables.

Finally, if we have unconstrained variables x_i (which are allowed to be positive or negative), and we would like only non-negative variables, we could introduce two non-negative variables $x_i^+ \geq 0$ and $x_i^- \geq 0$ for each such unconstrained x_i, and replace each x_i by $x_i^+ - x_i^-$.

This allows us to move between various forms of LPs: the two most common forms of LPs are the *general form*, which is

$$\begin{aligned} &\min c^T x \\ &Ax \geq b \end{aligned}$$

and the *equational* (or *standard*) form, which is

$$\begin{aligned} &\min c^T x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

To go from the general form to the equational form, we need two things. First, we can add slack variables for each constraint $a_i x \geq b_i$ to get equality constraints $a_i x - s_i = b_i$. Second, since the general form doesn't require x to be positive, we can use the idea of replacing each x_i with $x_i^+ - x_i^-$, where $x_i^+, x_i^- \geq 0$. Can you see why these two LPs have the same feasible solution set? Going in the other direction is easier: just replacing $a_i x = b_i$ by two inequalities $a_i x \geq b_i$ and $a_i x \leq b_i$ gives us an LP in general form.

Note that given m constraints and n variables, you may end up with $O(m+n)$ constraints and $O(m+n)$ variables in this conversion process. Note that if $m \gg n$ this may not always be a desireable conversion to make.

Formally, a *feasible solution* is some $x \in \mathbb{R}^n$ such that x satisfies all constraints. We say that x is *optimal* if it maximizes the objective function subject to all constraints.

It is possible that LP's have either bounded feasible sets or unbounded feasible sets. In the case that the feasible set is unbounded, it may also be the case that the optimal value is also unbounded.

1.2 Fourier–Motzkin elimination

The first way we'll describe to solve LP's is known as the *Fourier–Motzkin elimination* algorithm. Consider an LP in general form:

$$\begin{aligned} &\min c^T x \\ &Ax \geq b \end{aligned}$$

Let us rewrite it using one additional variable in this slightly modified, but equivalent form:

$$\begin{aligned} &\min x_{n+1} \\ &Ax \geq b \\ &c^T x \leq x_{n+1} \end{aligned}$$

Now we will eliminate variables in the following way. For variable x_1, arrange the constraints we have into three groups: those where x_1 has positive coefficients (let the indices for these constraints belong to set $P \subseteq [m]$), those where it has negative coefficients (let $N \subseteq [m]$ be the set of these indices), and those which don't involve x_1 at all (let $Z = [m] \setminus (P \cup N)$ be the indices for such constraints). Consider any constraints $a_i x \geq b_i$ for $i \in P$: we can divide out by the coefficient a_{i1} of x_1 for each one, leaving you with constraints of the form:

$$\begin{aligned} & x_1 + \frac{a_{i2}}{a_{i1}} x_2 + \cdots + \frac{a_{in}}{a_{i1}} \geq \frac{b_i}{a_{i1}} x_n \\ \iff & x_1 \geq \frac{b_i}{a_{i1}} - \left(\sum_{j=2}^{n} \frac{a_{ij}}{a_{i1}} x_j \right) \end{aligned}$$

Note that such constraints give us lower bounds for x_1. Now we do a similar operation for the constraints $a_i x \geq b$ for $i \in N$: remember that $a_{i1} < 0$ for these constraints, so we need to take care that dividing by a negative number changes the inequality direction, leaving you with constraints of the form:

$$\begin{aligned} & a_{i1} x_1 + a_{i2} x_2 + \cdots + a_{in} x_n \geq b_i \\ \iff & x_1 + \frac{a_{i2}}{a_{i1}} x_2 + \cdots + \frac{a_{in}}{a_{i1}} x_n \leq \frac{b_i}{a_{i1}} \\ \iff & x_1 \leq \frac{b_i}{a_{i1}} - \left(\sum_{j=2}^{n} \frac{a_{ij}}{a_{i1}} x_j \right) \end{aligned}$$

Now we create new constraints as follows: for each $i \in P$ and $i' \in N$, we get $blah_i \leq x_i$ and $x_{i'} \leq blah_{i'}$, so we get the new constraint $blah_i \leq blah_{i'}$. More formally, for each such pair $i \in P, i' \in N$, we get the constraint:

$$\frac{b_i}{a_{i1}} - \left(\sum_{j=2}^{n} \frac{a_{ij}}{a_{i1}} x_j \right) \leq \frac{b_{i'}}{a_{i'1}} - \left(\sum_{j=2}^{n} \frac{a_{i'j}}{a_{i'1}} x_j \right)$$

(All the constraints in Z just get copied over to this new LP.) It is easy to check the following lemma:

Lemma 1.1. *Given LP1 on k variables, suppose eliminating x_1 gives the new linear program LP2. Show that (a) if $(x_1 x_2 \cdots x_k)$ was feasible for LP1 then $(x_2 x_3 \cdots x_k)$ is feasible for LP2, and if $(x_2 \cdots x_k)$ is feasible for LP2 then there exists some value $x_i' \in \mathbb{R}$ such that $(x_1' x_2 x_3 \cdots x_k)$ is feasible for LP1.*

Note that we took the $|P| + |N|$ constraints, and replaced them with $|P| \cdot |N|$ constraints. Hence from the m constraints, we now have at most $m^2/4$ constraints, but one fewer variable. Continuing this process for each of the variables $x_2, x_3, \ldots, x_n$, we get at most m^{2^n} constraints. And when we have a single variable x_{n+1} remaining, each constraint is of the form $x_{n+1} \leq \textit{something}$, or $x_{n+1} \geq \textit{something else}$. These can be combined to get values ℓ, h, such that $x_{n+1} \in [\ell, h]$. (If $\ell > h$ then the LP is infeasible, and if there is no lower bound

then the LP is unbounded.) Now since the LP sought to minimize x_{n+1}, we get that the optimal value of the LP is $x_{n+1} = \ell$. Moreover, it is easy to proceed backwards through this process to get settings for the eliminated variables $x_1, x_2, \ldots, x_n$ such that $\sum_{j=1}^{n} c_j x_j = \ell$. (Exercise!)

> **Note:** Kevin asked what happens if, say, N was empty, and x_1 only had lower bound constraints (constraints in P). In that case there are no constraints in $P \times N$, and hence we would end up throwing away all the constraints in P and N. Indeed, this makes sense, since whatever the setting of variables $x_2, \ldots, x_n$, having np upper bound constraints on x_1 means we could set x_1 as large as needed to satisfy constraints involving x_1.

1.2.1 Gaussian Elimination

This is a good time to just mention Gaussian elimination (converting a matrix to an upper triangular matrix; this can be used to solve systems of linear equations $Ax = b$). If we just had a collection of equality constraints, the elimination could proceed by taking the first constraint $\sum_j a_{1j} x_j = b_1$, rewriting this as $x_1 = a_{11}^{-1}(b_1 - \sum_{j=2}^{n} a_{1j} x_j)$, and substituting this into the other constraints. This is pretty much what we do using Gaussian elimination.

Gaussian elimination can be done in strongly polynomial time, meaning that

- The number of operations done is polynomial in n and m, and
- The size of the numbers in the intermediate stages of the algorithm are $poly(n, m, \log |a_{ij}|)$ (i.e., the size of the input). Hence we can ensure the matrix entries don't grow too large during the algorithm.

It remains an interesting (and open) problem whether all of linear programming can be done in strongly polynomial time.

> **Note:** Remember that the *size* of numbers measures the *number of bits* used to represent the numbers. Hence, if the entries of the matrix are $a_{ij} \in \{0, 1\}$, then 2^n has size polynomial in the size of the input, but 2^{2^n} does not.
>
> Formally, let us define the size: for an integer k, define $\text{size}(k) = 1 + \lceil \log_2(|k| + 1) \rceil$; for a rational p/q (with p, q coprime, $q > 0$), define $\text{size}(p/q) = \text{size}(p) + \text{size}(q)$; for a matrix $R = (r_{ij})$ of rationals, define $\text{size}(M) = \sum_{i,j} \text{size}(r_{ij})$

1.3 Equational Form Solving

We've seen that Fourier–Motzkin gives us a solution in at most m^{2^n} time. Now let's consider a faster method for solving linear programs. For this section, assume our LP is in the equational form

$$\min\{c^T x \mid Ax = b, x \geq 0\}$$

Let us make two assumptions (without loss of generality). Firstly, we assume that $Ax = b$ has a solution (otherwise our LP is infeasible). And we can use Gaussian elimination to check if $Ax = b$ has a solution or not. Secondly, we assume that the rows of A are linearly independent (otherwise there is some constraint which is superfluous and we can throw it out).

With these assumptions, note that $rank(A) = m$, the number of constraints. Given a subset $B \subseteq [n]$, we define A_B to be the concatenation of the B columns of A. Similarly, we define x_B to be the column vector consisting of the variables $\{x_i \mid i \in B\}$. Suppose we had some subset B with $|B| = m$ such that A_B's columns were linearly independent. Then, A_B would have full rank, and thus be invertible, so

$$A_B x_B = b$$

would have a unique solution

$$x_B = A_B^{-1} b.$$

We can extend this x_B to all n variables (by setting $x_i = 0$ for all indices $i \notin B$): this vector x we call a *basic solution*. Note that a basic solution satisfies $Ax = b$, but it may not satisfy the non-negativity constraints. We say the basic solution is feasible (called a *basic feasible solution* or *BFS*) if $x_B \geq 0$, i.e., it also satisfies the non-negativity constraints.

So suppose we knew that the optimal value was achieved at a BFS, we could just try all $\binom{n}{m}$ subsets of columns B with $|B| = m$ which are linearly independent, and take the optimal one. However, we don't yet know this: we really have two questions at this point. Firstly, *what if there exists a solution to the LP, but no BFS exists*? And secondly, *what if the optimum is attained only at non-BFS points*? It turns out neither of these is a problem.

Fact 1.2. *Every linearly independent set B with $|B| = m$ gives exactly one basic solution and at most one BFS.*

Theorem 1.3. *For any LP in equational form, one of the following holds*

1. *The LP is infeasible*
2. *The LP has unbounded optimum*
3. *The LP has a BFS which is optimal*

Proof. Suppose our LP is feasible, and has a bounded optimum. Additionally, assume we have some x^* which is feasible (i.e., $Ax^* = b, x^* \geq 0$). Now we will show that there exists a BFS x with value $c^T x \leq c^T x^*$. Hence, for any feasible solution, there is a BFS with no higher value, and so there must exist an optimal BFS.

OK, so given x^*, pick a feasible solution $\tilde{x}$ among those that have $c^T x \leq c^T x^*$, and where $\tilde{x}$ has the fewest number of nonzero coordinates. Let

$$P = \{i \mid \tilde{x}_i > 0\}$$

be the set of corrdinates that are *strictly* positive. (Note that since all the other coordinates in $\tilde{x}$ are zero, we get $\sum_{j \in P} A_j \tilde{x}_j = \sum_j A_j \tilde{x}_j = b$, or $A_P \tilde{x}_P = b$.)

There are two cases. Case I is when the columns corresponding to the indices in P are linearly independent. I.e., the columns of A_P are linearly independent. Since A has full rank (and so contains m linearly independent columns), if needed we can add some more columns

from $[n] \setminus P$ to P, to get a set B with $|B| = m$ such that A_B's columns are also linearly independent, and so A_B is invertible. Now consider

$$A_B x_B = b.$$

There is a unique solution x_B to this equation (but we don't know if that satisfies $x_B \geq 0$.) No worries: we already know that $\tilde{x}_B$ is a solution to this equation, so it must be the unique solution. And since $\tilde{x}$ is feasible, it is the BFS corresponding to B.

In case II, suppose the columns of A_p are not linearly independent, so there exists some (not all zero) coefficients w_i such that

$$\sum_{j \in P} w_j A_j = 0 \qquad \text{or, equivalently} \qquad A_p w_P = 0.$$

By setting $w_j = 0$ for all $j \notin P$, we get a vector w which is itself non-zero, but where $Aw = 0$. Hence, if we consider the vector $y = \tilde{x} - \epsilon\, w$, note that

$$Ay = A(\tilde{x} - \epsilon w) = b - \epsilon\, 0 = b.$$

Moreover, since w is non-zero only in the coordinates in P, and x is strictly positive in those coordinates, then for small enough ϵ, we know that $y = \tilde{x} - \epsilon w \geq 0$. So y is also a feasible solution for small enough epsilon.

Suppose, fortuitously, $c^T w = 0$. Then $c^T y = c^T(\tilde{x} - \epsilon w) = c^T \tilde{x}$. We can assume that w has some positive entry, else we can negate all entries of w. So as we increase ϵ, we are decreasing some of the (positive) entries in y, without changing the objective function. And at some point we will make some entry zero, contradicting that $\tilde{x}$ had the fewest non-zeroes among all x such that $c^T x \leq c^T x^*$.

Now suppose $c^T w > 0$ (if $c^T w < 0$, we can just negate all entries of w to reduce to this case). Again, if there existed one positive w_j, we could do the same argument as above, and get a contradiction. But maybe $c^T w > 0$ and all of the entries of w are negative. (Now flipping the signs does not help.) But this is now easy: note that now $y = \tilde{x} - \epsilon w$ is non-negative and hence feasible for *all* $\epsilon \geq 0$. Moreover, the objective function value $c^T y = c^T \tilde{x} - \epsilon(c^T w)$ goes to $-\infty$ as $\epsilon \to \infty$. This contradicts the fact that the LP has a bounded optimum, and we are done! □

Note: Ankit asked a question about how Theorem 1.3 helped solve an equational form LP in time $\binom{n}{m}$. Specifically, his question was this: Theorem 1.3 says that if the LP is feasible and bounded, then the optimum is achieved at a BFS (and we could try all of them). But what if the LP was unbounded or infeasible? How could we detect those cases in the same amount of time? Here's one way.

To start, Fact 2.1 says any equational form LP that is feasible has a BFS. So if all the basic solutions are infeasible (i.e., there is no BFS), we can safely answer "Infeasible".

So now it suffices to differentiate between the bounded/unbounded subcases. So consider the BFS that achieves the lowest objective function value among all the BFSs (assuming the LP is a minimization LP). We know the optimal value is either this value (call it δ), or it is $-\infty$. Consider the LP obtained by adding in the new constraint $c^\top x = \delta - 1$. This is another equational form LP with $m + 1$ constraints, and we can use the above argument to decide its feasibility. If this new LP is feasible, the original LP had optimum value $latex - \infty$, else the optimum of the original LP was $latex\delta$.

Lecture 2

Geometry of LPs*

Last time we saw that, given a (minimizing) linear program in equational form, one of the following three possibilities is true:

1. The LP is infeasible.
2. The optimal value of the LP is $-\infty$ (i.e., the LP does not have a bounded optimum).
3. A basic feasible solution exists that achieves the optimal value.

2.1 Finding a basic feasible solution

Suppose we have an LP in equational form:

$$\min\{\, c^T x \mid Ax = b,\, x \geq 0 \,\}, \quad \text{where } A \text{ is an } m \times n \text{ matrix of rank } m.$$

Recall how we might attempt to find a BFS to this LP: Let $B \subseteq [n]$, with $|B| = m$, such that A_B (the set of columns of A corresponding to the elements of B) is linearly independent. (Such a set of columns exists because A has full rank.) Let $N = [n] \setminus B$ be the indices of the columns of A that are not in B. Since A_B is invertible, we can define a vector $x \in \mathbb{R}^n$ by

$$\begin{aligned} x_B &= A_B^{-1} b, \\ x_N &= 0. \end{aligned}$$

By construction, x is a basic solution. If x is also feasible (i.e., if $x \geq 0$), then it is a BFS.

Fact 2.1. *Every LP in equational form that is feasible has a BFS. (Note that this BFS may or may not be optimal.)*

Proof. Pick some feasible point $\tilde{x} \in \mathbb{R}^n$. (In particular, since $\tilde{x}$ is feasible, $\tilde{x} \geq 0$.) Let

$$P = \{\, j \mid \tilde{x}_j > 0 \,\}$$

*Lecturer: Anupam Gupta. Scribe: Brian Kell.

be the set of coordinates of $\tilde{x}$ that are nonzero. We consider two cases, depending on whether the columns of A_P are linearly independent.

Case 1. The columns of A_P are linearly independent. Then we may extend P to a basis B, i.e., a subset $P \subseteq B \subseteq [n]$ with $|B| = m$ such that the columns of A_B are also linearly independent. Let $N = [n] \setminus B$; then $\tilde{x}_N = 0$ (because $P \subseteq B$). In addition, since $A\tilde{x} = b$, we have $A_B\tilde{x}_B = b$, so $\tilde{x} = A_B^{-1}b$. So $\tilde{x}$ is a basic solution; since it is feasible by assumption, it is a BFS. (Note, by the way, that the equation $\tilde{x} = A_B^{-1}b$ means that $\tilde{x}$ is the unique solution to $Ax = b$ having $x_N = 0$.)

Case 2. The columns of A_P are linearly dependent. Let $N = [n] \setminus P$. Then, by the definition of linear dependence, there exists a nonzero vector $w \in \mathbb{R}^n$ with $w_N = 0$ such that $A_P w_P = 0$. For any $\lambda \in \mathbb{R}$, the vector $\tilde{x} + \lambda w$ satisfies $A(\tilde{x} + \lambda w) = b$, because

$$A(\tilde{x} + \lambda w) = A\tilde{x} + \lambda Aw = b + 0 = b.$$

Because $\tilde{x}_N = 0$ and $w_N = 0$, we have $(\tilde{x} + \lambda w)_N = 0$, so $\tilde{x} + \lambda w$ has no more nonzero entries than $\tilde{x}$ does. Since $\tilde{x}_P > 0$, for sufficiently small $\epsilon > 0$ both $\tilde{x} + \epsilon w$ and $\tilde{x} - \epsilon w$ are feasible (i.e, $\tilde{x} \pm \epsilon w \geq 0$). Let $\eta = \sup\{\, \epsilon > 0 \mid \tilde{x} \pm \epsilon w \geq 0 \,\}$ be the largest such ϵ; then one of $\tilde{x} \pm \eta w$ has one more zero coordinate than $\tilde{x}$ does. We can repeat this until we find a feasible solution with no more than m nonzero coordinates, at which point Case 1 applies and we have found a BFS.

(Intuitively, for sufficiently small $\epsilon > 0$, one of $\tilde{x} \pm \epsilon w$ is moving toward a nonnegativity constraint, that is, toward the boundary of the nonnegative orthant. When ϵ becomes just large enough that the point $\tilde{x} \pm \epsilon w$ reaches the boundary of the nonnegative orthant, we have made one more coordinate of the point zero.) □

2.2 Geometric definitions

Definition 2.2. Given points $x, y \in \mathbb{R}^n$, a point $z \in \mathbb{R}^n$ is a *convex combination* of x and y if

$$z = \lambda x + (1 - \lambda)y$$

for some $\lambda \in [0, 1]$.

Definition 2.3. A set $X \subseteq \mathbb{R}^n$ is *convex* if the convex combination of any two points in X is also in X; that is, for all $x, y \in X$ and all $\lambda \in [0, 1]$, the point $\lambda x + (1 - \lambda)y$ is in X.

Definition 2.4. A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if for all points $x, y \in \mathbb{R}^n$ and all $\lambda \in [0, 1]$ we have

$$f\big(\lambda x + (1 - \lambda)y\big) \leq \lambda f(x) + (1 - \lambda)f(y).$$

Fact 2.5. *If $P \subseteq \mathbb{R}^n$ is a convex set and $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then, for any $t \in \mathbb{R}$, the set*

$$Q = \{\, x \in P \mid f(x) \leq t \,\}$$

is also convex.

Proof. For all $x_1, x_2 \in Q$ and all $\lambda \in [0,1]$, we have

$$f\big(\lambda x_1 + (1-\lambda)x_2\big) \leq \lambda f(x_1) + (1-\lambda) f(x_2) \leq \lambda t + (1-\lambda)t = t,$$

so $\lambda x_1 + (1-\lambda)x_2 \in Q$. □

Fact 2.6. *The intersection of two convex sets is convex.*

Proof. Let $P, Q \subseteq \mathbb{R}^n$ be convex sets, and let $x_1, x_2 \in P \cap Q$. Let $\lambda \in [0,1]$. Because $x_1, x_2 \in P$ and P is convex, we have $\lambda x_1 + (1-\lambda)x_2 \in P$; likewise, $\lambda x_1 + (1-\lambda)x_2 \in Q$. So $\lambda x_1 + (1-\lambda)x_2 \in P \cap Q$. □

Definition 2.7. A set $S \subseteq \mathbb{R}^n$ is a *subspace* if it is closed under addition and scalar multiplication.

Equivalently, S is a subspace if $S = \{\, x \in \mathbb{R}^n \mid Ax = 0 \,\}$ for some matrix A.

Definition 2.8. The *dimension* of a subspace $S \subseteq \mathbb{R}^n$, written $\dim(S)$, is the size of the largest linearly independent set of vectors contained in S.

Equivalently, $\dim(S) = n - \operatorname{rank}(A)$.

Definition 2.9. A set $S' \subseteq \mathbb{R}^n$ is an *affine subspace* if $S' = \{\, x_0 + y \mid y \in S \,\}$ for some subspace $S \subseteq \mathbb{R}^n$ and some vector $x_0 \in \mathbb{R}^n$. In this case the *dimension* of S', written $\dim(S')$, is defined to equal the dimension of S.

Equivalently, S' is an affine subspace if $S' = \{\, x \in \mathbb{R}^n \mid Ax = b \,\}$ for some matrix A and some vector b.

Definition 2.10. The *dimension* of a set $X \subseteq \mathbb{R}^n$, written $\dim(X)$, is the dimension of the minimal affine subspace that contains X.

Note that if S'_1 and S'_2 are two affine subspaces both containing X, then their intersection $S'_1 \cap S'_2$ is an affine subspace containing X. Hence there is a unique minimal affine subspace that contains X, so $\dim(X)$ is well defined.

Equivalently, given $x_0 \in X$, the dimension of X is the largest number k for which there exist points $x_1, x_2, \ldots, x_k \in X$ such that the set $\{x_1 - x_0, x_2 - x_0, \ldots, x_k - x_0\}$ is linearly independent.

Note that the definition of the dimension of a set X agrees with the definition of the dimension of an affine subspace if X happens to be an affine subspace, and the definition of the dimension of an affine subspace S' agrees with the definition of the dimension of a subspace if S' happens to be a subspace.

Definition 2.11. A set $H \subseteq \mathbb{R}^n$ is a *hyperplane* if $H = \{\, x \in \mathbb{R}^n \mid a^T x = b \,\}$ for some nonzero $a \in \mathbb{R}^n$ and some $b \in \mathbb{R}$.

A hyperplane is an affine subspace of dimension $n - 1$.

Definition 2.12. A set $H' \subseteq \mathbb{R}^n$ is a (closed) *halfspace* if $H' = \{\, x \in \mathbb{R}^n \mid a^T \geq b \,\}$ for some nonzero $a \in \mathbb{R}^n$ and some $b \in \mathbb{R}$.

A hyperplane can be written as the intersection of two halfspaces:

$$\{\, x \in \mathbb{R}^n \mid a^T x = b \,\} = \{\, x \in \mathbb{R}^n \mid a^T x \geq b \,\} \cap \{\, x \in \mathbb{R}^n \mid -a^T x \geq -b \,\}.$$

Both hyperplanes and halfspaces are convex sets. Therefore the feasible region of an LP is convex, because it is the intersection of halfspaces and hyperplanes. The dimension of the feasible region of an LP in equational form, having n variables and m linearly independent constraints (equalities), is no greater than $n - m$, because it is contained in the intersection of m distinct hyperplanes, each of which is an affine subspace of dimension $n - 1$. (The dimension of the feasible region may be less than $n - m$, because of the nonnegativity constraints, for instance.)

For example, the region in $\mathbb{R}^3$ defined by

$$\begin{aligned} x_1 + x_2 + x_3 &= 1, \\ x &\geq 0 \end{aligned}$$

is a 2-dimensional triangle; here, $n - m = 3 - 1 = 2$. (Note, however, that if the constraint were $x_1 + x_2 + x_3 = 0$, the region would have dimension 0.)

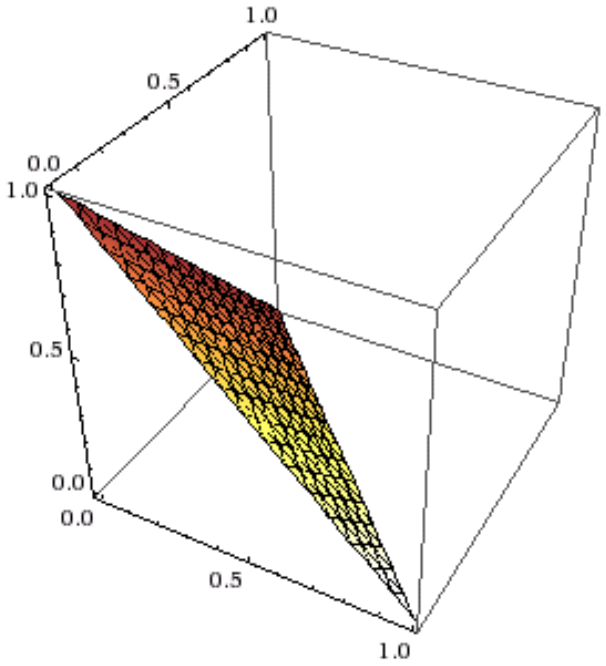


Figure 2.1: The region $\{\, x \in \mathbb{R}^3 \mid x_1 + x_2 + x_3 = 1,\ x \geq 0 \,\}$.

Definition 2.13. A *polyhedron* in $\mathbb{R}^n$ is the intersection of finitely many halfspaces.

For example, feasible regions of LPs are polyhedra.

Definition 2.14. A *polytope* is a bounded polyhedron, that is, a polyhedron P for which there exists $B \in \mathbb{R}^+$ such that $\|x\|_2 \leq B$ for all $x \in P$.

Both polyhedra and polytopes are convex.

Definition 2.15. Given a polyhedron $P \subseteq \mathbb{R}^n$, a point $x \in P$ is a *vertex* of P if there exists $c \in \mathbb{R}^n$ such that $c^T x < c^T y$ for all $y \in P$, $y \neq x$.

Suppose $\hat{x}$ is a vertex of a polyhedron $P \subseteq \mathbb{R}^n$. Let c be as in the definition above. Take $K = c^T \hat{x}$. Then for all $y \in P$ we have $c^T y \geq K$, so the polyhedron P is contained in the halfspace $\{ x \in \mathbb{R}^n \mid c^T x \geq K \}$, i.e., P lies entirely on one side of the hyperplane $\{ x \in \mathbb{R}^n \mid c^T x = K \}$. Furthermore, the vertex $\hat{x}$ is the *unique* minimizer of the function $c^T z$ for $z \in P$.

Definition 2.16. Given a polyhedron $P \subseteq \mathbb{R}^n$, a point $x \in P$ is an *extreme point* of P if there do not exist points $u, v \neq x$ in P such that x is a convex combination of u and v.

In other words, x is an extreme point of P if, for all $u, v \in P$,

$$\big(x = \lambda u + (1-\lambda)v \text{ for some } \lambda \in [0,1]\big) \quad \text{implies} \quad u = v = x.$$

2.3 Equivalence of vertices, extreme points, and basic feasible solutions

In fact, vertices and extreme points are the same thing, and for an LP the vertices (i.e., extreme points) of its feasible region are precisely its basic feasible solutions. This is shown in the following theorem.

Theorem 2.17. *Consider an LP in equational form, i.e.,* $\min\{ c^T x \mid Ax = b,\, x \geq 0 \}$*, and let* K *be its feasible region. Then the following are equivalent:*

1. *The point* x *is a vertex of* K*.*
2. *The point* x *is an extreme point of* K*.*
3. *The point* x *is a BFS of the LP.*

Proof. *(1)* $\Rightarrow$ *(2)*. Let x be a vertex of K. Then there exists $c \in \mathbb{R}^n$ such that $c^T x < c^T y$ for all $y \in K$, $y \neq x$. Suppose for the sake of contradiction that x is not an extreme point of K, so there exist some points $u, w \in K$ with $u, w \neq x$ and some $\lambda \in [0,1]$ such that $x = \lambda u + (1-\lambda)w$. Then

$$c^T x = \lambda c^T u + (1-\lambda) c^T w < \lambda c^T x + (1-\lambda) c^T x = c^T x,$$

which is a contradiction. Hence x is an extreme point of K.

(2) $\Rightarrow$ *(3)*. Let x be an extreme point of K. (In particular, x is a feasible solution for the LP, so $x \geq 0$.) Let $P = \{ j \mid x_j > 0 \}$ be the set of nonzero coordinates of x. We consider two cases, depending on whether A_P (the set of columns of A corresponding to P) is linearly independent.

Case 1. The columns of A_P are linearly independent. Then x is a BFS. (This is the same as in the proof of Fact 2.1: Extend P to a basis B, and let $N = [n] \setminus B$; then $x_B = A_B^{-1}b$ and $x_N = 0$.)

Case 2. The columns of A_P are linearly dependent. Then there exists a nonzero vector w_P such that $A_P w_P = 0$. Let $N = [n] \setminus P$ and take $w_N = 0$. Then $Aw = A_P w_P + A_N w_N = 0$. Now consider the points $y^+ = x + \lambda w$ and $y^- = x - \lambda w$, where $\lambda > 0$ is sufficiently small so that $y^+, y^- \geq 0$. Then

$$Ay^+ = A(x + \lambda w) = Ax + \lambda Aw = b + 0 = b,$$
$$Ay^- = A(x - \lambda w) = Ax - \lambda Aw = b - 0 = b,$$

so y^+ and y^- are feasible, i.e., $y^+, y^- \in K$. But $x = (y^+ + y^-)/2$ is a convex combination of y^+ and y^-, which contradicts the assumption that x is an extreme point of K. So Case 2 is impossible.

(3) $\Rightarrow$ (1). Suppose x is a BFS for the LP. We aim to show that x is a vertex of K, that is, there exists $c \in \mathbb{R}^n$ such that $c^T x < c^T y$ for all $y \in P$, $y \neq x$. Since x is a BFS, there exists a set $B \subseteq [n]$ with $|B| = m$ such that the columns of A_B are linearly independent, $A_B x_B = b$, and $x_N = 0$ (where $N = [n] \setminus B$). For $j = 1, \ldots, n$, define

$$c_j = \begin{cases} 0, & \text{if } j \in B; \\ 1, & \text{if } j \in N. \end{cases}$$

Note that $c^T x = c_B^T x_B + c_N^T x_N = 0^T x_B + c_N^T 0 = 0$. For $y \in K$, we have $y \geq 0$ (since y is feasible), and clearly $c \geq 0$, so $c^T y \geq 0 = c^T x$. Furthermore, if $c^T y = 0$, then y_j must be 0 for all $j \in N$, so $A_B y_B = b = A_B x_B$. Multiplying on the left by A_B^{-1} gives $y_B = A_B^{-1} b = x_B$. So x is the unique point in K for which $c^T x = 0$. Hence x is a vertex of K. □

Definition 2.18. A polyhedron is *pointed* if it contains at least one vertex.

Note that a polyhedron contains a (bi-infinite) line if there exist vectors $u, d \in \mathbb{R}^n$ such that $u + \lambda d \in K$ for all $\lambda \in \mathbb{R}$.

Theorem 2.19. *Let $K \subseteq \mathbb{R}^n$ be a polyhedron. Then K is pointed if and only if K does not contain a (bi-infinite) line.*

Note that the feasible region of an LP with nonnegativity constraints, such as an LP in equational form, cannot contain a line. So this theorem shows (again) that every LP in equational form that is feasible has a BFS (Fact 2.1).

2.4 Basic feasible solutions for general LPs

Note that we've defined basic feasible solutions for LPs in equational form, but not for general LPs. Before we do that, let us make an observation about equational LPs, and the number of *tight* constraints (i.e., those constraints that are satisifed at equality).

Consider an LP in equational form with n variables and m constraints, and let x be a BFS. Then x satisfies all m equality constraints of the form $a_i x = b_i$. Since $x_N = 0$, we see that x additionally satisfies at least $n - m$ nonnegativity constraints at equality. A constraint is said to be *tight* if it is satisfied at equality, so we have the following fact.

Fact 2.20. *If x is a BFS of an LP in equational form with n variables, then x has at least n tight constraints.*

We can use this idea to extend the definition of BFS to LPs that are not in equational form.

Definition 2.21. For a general LP with n variables, i.e., an LP of the form

$$\min\{\, c^T x \mid Ax \geq b,\, x \in \mathbb{R}^n \,\},$$

a point $x \in \mathbb{R}^n$ is a *basic feasible solution* if x is feasible and there exist some n linearly independent constraints that are tight (hold at equality) for x.

Proposition 2.22. *For an LP in equational form, this definition of BFS and the previous definition of BFS are equivalent.*

(You may want to prove this for yourself.) Using this definition, one can now reprove Theorem 2.17 for general LPs: i.e., show the equivalence between BFSs, vertices, and extreme points holds not just for LPs in equational form, but for general LPs. We can use this fact to find optimal solutions for LPs whose feasible regions are pointed polyhedra (and LPs in equational form are one special case of this).

Lecture 3

Basic Applications of LP*

Dantzig Presents LP George Dantzig developed Linear Programming during World War II and presented the ideas to a conference of eminent mathematicians and statisticians. Among the audience were Hotelling and von Neumann. In his book on the topic of LP, Dantzig recalls after finishing his speech asking for questions. Hotelling asks what the point of Dantzigs presentation has been pointing out the "world is not linear." Before Dantzig answers, von Neumann speaks up to say that if the axioms Dantzig has presented hold, then LP is an effective tool.

3.1 Max s-t Flow in a Directed Graph

Input: A di-graph:

$$G = (V, E)$$

Capacities:

$$\forall (u, v) \in E \quad c_{(u,v)} \geq 0$$

A source and a sink:

$$s, t \in V$$

Conservation of flow:

$$\text{flow into } (v \neq s, t) = \text{flow out of } v$$

A History of Flow The problem was originally studied by Tolstoy in the 1930's. Tolstoy was a mathematician in the Soviet Union studying how to optimally transport goods along the Soviet raliways from one city to another. In his formulation the vertices were cities and the edges were railroads connecting two cities. The capacity of each edge was the amount of goods the specific railroad could transport in a given day. The bottleneck was solely the capacities and not production or consumption on either end and there was no available storage at the intermediate cities.

*Lecturer: Ryan O'Donnell. Scribe: Will Devanny.

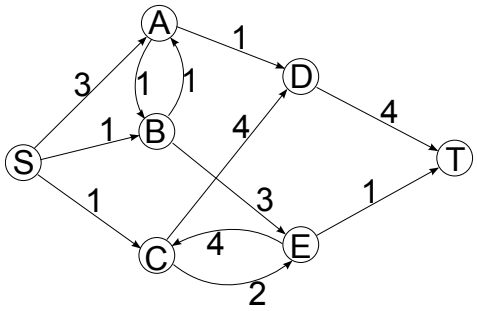


Figure 3.1: An input for the max s-t flow problem

The problem can be naturally set up as an LP by using a variable for the flow along each edge.

$$\begin{aligned} \max \quad & \sum_{v:(s,v)\in E} f_{(s,v)} - \sum_{v:(v,s)\in E} f_{(v,s)} && (3.1)\\ & \text{s.t.} \quad \forall (u,v) \in E \quad f_{(u,v)} \geq 0 \\ & \qquad\quad \forall (u,v) \in E \quad f_{(u,v)} \leq C_{(u,v)} \\ \forall v \neq s,t \quad & \sum_{u:(u,v)\in E} f_{(u,v)} = \sum_{w:(v,w)\in E} f_{(v,w)} \end{aligned}$$

We have to be careful with our objective function, Equation (3.1), to subtract any flow that might come back into the sink. In Figure 3.2, the results of a run of the LP on the example are shown.

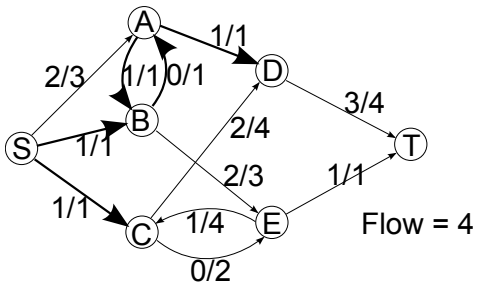


Figure 3.2: A solution to Figure 3.1

Remarks:

- Our solution has integral values.

  Theorem 3.1. *If the $C_{(u,v)}$ are integers then every b.f.s. optimal solution is integral.*

  This happens in general when the matrix A of the LP is unimodular.

- Is 4 the true optimal? Examine the cut created by $S = \{s, a\}$ and $T = V \setminus S$. The total capacity out of A is 4 and therefore $LPopt \leq 4$.

- Is that a coincidence? No.

  Theorem 3.2. *Max s-t flow = Min s-t cut in terms of the capacity graph.*

  This is an example of LP duality.

A History of Flow cont. Max flow was published in '54 again in the context of studying railroads by Ford and Fulkerson. They had heard about the problem from Ted Harris then working at Rand Corp. Harris orginally began studying flow in the USSR's railway system similar to Tolstoy years earlier. However Harris was looking at potential military applications of the min cut problem.

3.2 Max Perfect Matching in a Bipartite Graph

Dantzig studied max perfect matching during his time in the military. He had a group of people he wished to assign to an equal number of jobs. He knew a given person doing a given job would give the military some benefit. His goal was to give each person a job in such a way as to maximize the overall benefit. More formally we have a bipartite graph $G = (U \cup V, E)$ with some weight on the edges, $w_{u,v} \forall (u, v) \in E$. The weights are the value of person u doing job v.

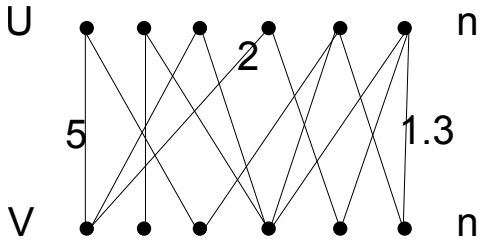


Figure 3.3: A bipartite graph on $2n$ vertices with associated weights

Our instinct in attacking this problem is to have a variable x_{uv} that is equal to 1 if we assign u job v and 0 if not:

$$\begin{aligned} \max \quad & \sum_{(u,v)\in E} w_{uv}x_{uv} \\ \text{s.t.} \quad & 0 \le x_{uv} \le 1 \\ & x_{uv} \in \mathbb{Z} \qquad (3.2) \\ \forall v \in V \quad & \sum_{u:(u,v)\in E} x_{uv} = 1 \qquad (3.3) \\ \forall u \in U \quad & \sum_{v:(u,v)\in E} x_{uv} = 1 \qquad (3.4) \end{aligned}$$

Unfortunately Equation (3.2) isn't a linear constraint. We need to use the LP relaxation.

3.2.1 LP Relaxation

To form an LP relaxation of and IP, we drop the IP constraints. This enables us to solve the program efficiently. In the current problem we would remove constraint (3.2).

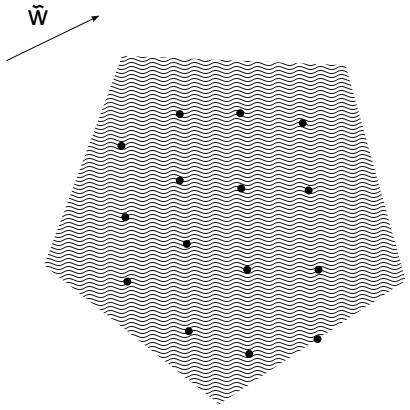


Figure 3.4: The feasible region of an LP with integer points inside

Remarks:

- The new LP is never unbounded, because we are inside the unit hypercube.
- If the LP is infeasible so is the original IP. Because if the feasible space of the LP is empty then it contains no integer points. The IP space inside of an LP can be seen in Figure 3.4.
- In general for relaxations, $Opt \le LPopt$. This holds even when the optimal value is infeasible ($-\infty \le c$).

For this problem, a lucky thing is true:

Theorem 3.3. *All extreme points are integral.*

Theorem 3.4 (Corrolary)**.** *If the LP is feasible so is the IP and IPopt = LPopt.*

Proof. By Contrapositive: If $\tilde{x}$ is feasible and non-integral then it is not an extreme point. $\tilde{x}$ not an extreme point means $\tilde{x} = \theta x^+ + (1-\theta)x^-$ for some $\theta \in [0,1]$.

Suppose we have a feasible and non-integral solution $\tilde{x}$. Then there is a non-integral edge. If we look at one of its end vertices, that vertex must have another non-integral edge incident to it because of Equation (3.3) and Equation (3.4). Similarly we can travel along this other edge to the its opposite vertex and find another non-integral edge. Because the graph is finite and bipartite by repeating this process we will eventually end up with an even length cycle of non-integral edges, C, as seen in Figure 3.5.

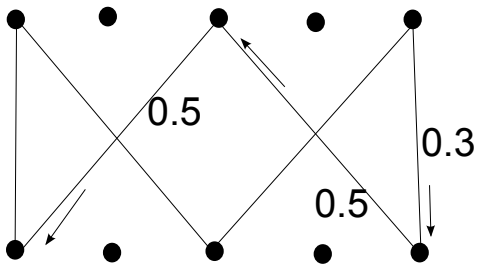


Figure 3.5: A non-integer cycle in a bipartite graph

Let $\epsilon = \min(\min_{(u,v)\in C} x_{uv}, \min_{(u,v)\in C} 1 - x_{uv})$. In words ϵ is the minimum distance from one of the weights on the cycle to an integer. Let x^+ be the same as $\tilde{x}$ but with ϵ added to the odd edges and $-\epsilon$ added to the even edges. Let x^- be the same as $\tilde{x}$ but with $-\epsilon$ added to the odd edges and $+\epsilon$ added to the even edges. We now have $\tilde{x} = \frac{1}{2}x^+ + \frac{1}{2}x^-$.

Iterate this process until we have all integer values. □

Does this respect the value of *Opt*?

$$\frac{obj(x^+) + obj(x^-)}{2} = obj(\tilde{x})$$

So $obj(\tilde{x})$ is the average of $obj(x^+)$ and $obj(x^-)$. Because $obj(\tilde{x}) = Opt$ and neither $obj(x^+)$ nor $obj(x^-)$ is greater than Opt, $obj(x^+)$ and $obj(x^-)$ must both be equal to Opt.

3.3 Minimum Vertex Cover

Input:
Undirected graph:

$$G = (V, E)$$

Vertex costs:

$$\forall v \in V \quad c_v \geq 0$$

Output:

$$S \subseteq V \text{s.t.} \forall (u, v) \in E u \in S \text{ or } v \in S \text{ with } \min \sum_{v \in S} c_{uv}$$

Remarks:

- The problem is NP-Hard. So we do not expect to find an LP to solve the problem perfectly.
- The greedy algorithm tends not to work too well.

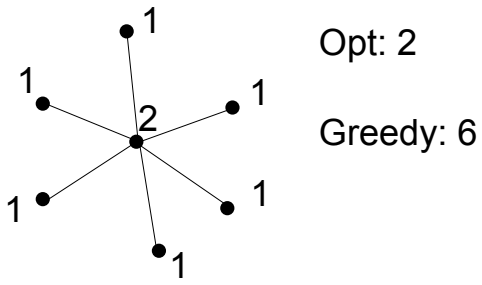


Figure 3.6: An example vertex cover problem

To phrase this as an IP, we will again use a variable x_v to be 1 if the vertex v is in the cover and 0 otherwise:

$$\begin{aligned} \min \quad & \sum_{v \in V} c_v x_v \\ \text{s.t.} \quad & \forall v \in V \quad 0 \leq x_v \leq 1 \\ & \forall v \in V \quad x_v \in \mathbb{Z} \\ & \forall (u, v) \in E \quad x_u + x_v \geq 1 \end{aligned} \tag{3.5}$$

To relax this IP we throw out Equation (3.5). This LP will give us a fractional cover of the vertices.

Remarks:

- LPopt $\leq$ IPopt
- The LP is bounded, because we are again inside the unit cube.
- The LP is feasible. We can set all the variables to 1 or to do slightly better $\frac{1}{2}$.

3.3.1 LP Rounding

The idea is to use the optimal fractional solution to obtain a nicer integral solution.

Given a feasible $\tilde{x}$. We can define $S = S_{\tilde{x}} = v \in V : \tilde{x}_v \geq \frac{1}{2}$.

Fact: S is always a vertex cover. In the LP solution $x_u + x_v \geq 1$ implies at least one if the x's is greater than $\frac{1}{2}$.

Fact:

$$Cost(S) \leq 2LPCost(\tilde{x})$$

$$LPCost(\tilde{x}) = \sum_{v \in S} c_v \tilde{x}_v \geq \sum_{v \in S} c_v \frac{1}{2} = \frac{1}{2} Cost(S)$$

Corrolary: Let x^* be an optimal LP solution. Then $Cost(S_{x^*} \leq 2LPCost(x^{\&}) = 2LPopt \leq 2IPopt)$.

Remarks:

- This is called a factor 2 approximation algorithm.
- No better approximation is known.
- If $\mathsf{P} \neq \mathsf{NP}$ then we can't do better than 1.36.
- If the Unique Games Conjecture is true then we can't do better than $2 - \epsilon$.
- Every extreme point is half integral $(0, \frac{1}{2}, 1)$.

3.4 Simplex Algorithm Intro

The simplex algorithm is not in P, not good in theory, and no longer considered the best in practice. Interior point methods anecdotally do better on larger data sets. The simplex algorithm is considered good in smoothed analysis, a combination of average and worst case.

Theorem 3.5. *Solving LPs poly-time reduces to testing LP feasibility.*

Proof. Consider an LP:

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

Suppose we can test feasibility of the LP in poly-time.

$$\begin{aligned} \text{Add constraint } & c^T x \geq 1000 && \text{Feasible? No} \\ & c^T x \geq 500 && \text{Feasible? Yes} \\ & c^T x \geq 750 && \text{Feasible? No} \\ & c^T x \geq 625 && \text{Feasible? Yes} \\ & \ldots && \text{(binary search)} \end{aligned}$$

- How do we pick the starting number? Number 4 on the first homework gives a way to upper bound the size of a feasible solution.
- How do we know when to stop? We can similarly estimate the granularity of the solution.

□

Lecture 4

Avis-Kaluzny and the Simplex Method*

Last time, we discussed some applications of Linear Programming, such as Max-Flow, Matching, and Vertex-Cover. The broad range of applications to Linear Programming means that we require efficient algorithms to solve it.

4.1 The Avis-Kaluzny Algorithm

The Avis-Kalunzy Algorithm is designed to find a basic feasible solution (BFS) of a given set of constraints. Its input: $A \in \mathbb{R}^{m \times n}$ and $b \in R^m$ such that

$$Ax \leq b$$
$$x \geq 0$$

Note that this algorithm requires the nonnegativity constraint to be present. The algorithm proceeds as follows:

4.1.1 The Algorithm

Terminology The set of equations obtained in step 1 is called the tableau or dictionary.
The set basic variables B is a set of variables, initially a set of slack variables, which has the property that each variable is alone on one side of each equation.
All other variables are called non-basic variables.
A basic variable is said to "exit the basis" when step 4 causes it to no longer be a basic variable, and a non-basic variable is said to "enter the basis" when step 4 causes it to become a basic variable.
Starting the algorithm:

*Lecturer: Ryan O'Donnell. Scribe: Aaron Snook.

Step 1: Slack Introduce slack variables $x_{n+1}...x_{n+m}$ for each of the inequalities in $\mathcal{A}$, turning them into equations. Obviously, but importantly, like the other variables, these slack variables also must be nonnegative.

Step 2: Try it Set all non-basic variables to 0, and see if this results in a BFS (does this cause the basic variables to be positive?) If it is, output the value of each variable under this assignment (or just the non-slack variables).

Step 3: B-rule Take the basic variable of minimum index that is negative in the assignment in step 2; call it x_b. On the other side of the selected equation, if all coefficients are negative, return this equation as evidence that the equation is INFEASIBLE. Otherwise, select the non-basic variable of minimum index that has a positive coefficient; call it x_n.

Step 4: Pivot Solve the selected equation for x_n in Step 3. This means that x_n enters the basis, and x_b exits the basis. For every instance of x_n in other equations, substitute the other side of the selected equation. Go to step 2.

4.1.2 Example

Suppose we were given the following set of equations:

$$\begin{aligned} x_1 + 2x_2 &\leq 6 \\ 2x_1 + x_2 &\leq 6 \\ 2x_1 + 2x_2 &\geq 7 \\ x_1, x_2 &\geq 0 \end{aligned}$$

After performing step 1, our tableau is:

$$\begin{aligned} x_3 &= -x_1 - 2x_2 + 6 \\ x_4 &= -2x_1 - x_2 + 6 \\ x_5 &= 2x_1 + 2x_2 - 7 \end{aligned}$$

We perform step 2: Set non-basic variables to 0. In this case, x_1 and x_2 are our non-basic variables. This means that

$$\begin{aligned} x_3 &= 6 \\ x_4 &= 6 \\ x_5 &= -7 \end{aligned}$$

This is not feasible as $x_5 < 0$. x_5 is the lowest-indexed variable below 0, so we proceed to step 3. We select the equation $x_5 = 2x_1 + 2x_2 - 7$ and note x_1 has a positive coefficient, so we select it for the pivot step.

We perform step 4 and solve for x_1: $x_1 = \frac{1}{2}x_5 - x_2 + \frac{7}{2}$. Go to step 2.
Our tableau is currently:

$$\begin{aligned} x_3 &= -\frac{1}{2}x_5 - x_2 + \frac{5}{2} \\ x_4 &= -x_5 + x_2 - 1 \\ x_1 &= \frac{1}{2}x_5 - x_2 + \frac{7}{2} \end{aligned}$$

We set non-basic variables to 0.

$$\begin{aligned} x_3 &= \frac{5}{2} \\ x_4 &= -1 \\ x_1 &= \frac{7}{2} \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

This is not a basic feasible solution. We continue, selecting x_2 to pivot about x_4:

$$\begin{aligned} x_3 &= -\frac{3}{2}x_5 - x_4 + \frac{3}{2} \\ x_2 &= x_5 + x_4 + 1 \\ x_1 &= -\frac{1}{2}x_5 - x_4 + \frac{5}{2} \end{aligned}$$

With the basic variables set to 0, we obtain

$$\begin{aligned} x_3 &= \frac{3}{2} \\ x_2 &= 1 \\ x_1 &= \frac{5}{2} \end{aligned}$$

Thus the solution $x_1 = \frac{5}{2}, x_2 = 1, x_3 = \frac{3}{2}, x_4 = 0, x_5 = 0$ works and we return $x_1 = \frac{5}{2}, x_2 = 1$
It can be verified that this is a solution of the original problem.

4.1.3 Correctness

Step 2 At each stage, a satisfying assignment to the tableau also satisfies the original tableau, as each tableau is a linear combination of previous tableaus.

Step 3 Suppose that you have an equation where the basic variable is negative when all non-basic variables are 0, and the coefficient of all non-basic variables is negative. This means that if the non-basic variables in the equation are non-negative, the basic variable is a sum of nonpositive terms, at least one of which is negative, and thus it is impossible to make the basic variable positive, thus making the entire system infeasible.
The most important thing to verify about this algorithm is that it actually terminates.

Theorem 4.1. *The Avis-Kaluzny algorithm terminates.*

Proof. First of all, note that there are only finitely many possible tableaus for any given set of constraints, as a set of basic variables uniquely determines a tableau given a set of initial constraints.

So if this algorithm does not terminate, there must be a cycle among the possible tableaus.

Suppose that we have a cycle in the set of possible tableaus. We will assume that x_{n+m} enters and leaves the basis during this cycle.

Justification Suppose that x_{n+m} does not enter and leave the basis during this cycle. Then it follows that the tableau formed by removing x_{n+m} in all equations (remove the equation containing x_{n+m} if x_{n+m} is a basic variable, and remove all instances of x_{n+m} from all equations otherwise) will also cycle, and we consider that tableau instead.

When x_{n+m} enters the basis, there must be an equation of the form

$$x_b = k + \sum_{a \in \{1..m+n\}} c_a x_a + c_{m+n} x_{m+n}$$

where c_i for all $i \in \{1..m+n\}$ is the coeffient of x_i in this equation, k is a constant, $c_i \leq 0$ for all $i < m+n$, $k < 0$, and x_b is the basis variable.
$k < 0$ because x_b must be negative when the non-basic variables are 0, and x_i for all $i < m+n$ must be nonpositive as otherwise they would be chosen over x_{m+n} to be the pivot.

This means that every solution to these equations with $x_1 ... x_{n+m-1} \geq 0$ has $x_{n+m} > 0$; otherwise x_b would be a sum of nonpositive values with at least one negative term.

When x_{n+m} leaves, if all non-basic variables are set to 0, as x_{n+m} is lexicographically last, all basic variables must be nonnegative on this assignment in order for x_{n+m} to be chosen. This is an assignment such that $x_1 ... x_{n+m-1} \geq 0$ but $x_{n+m} < 0$, contradicting the fact we proved when x_{n+m} entered the basis!
By contradiction, there is no cycle in the set of possible tableaus, and therefore the Avis-Kaluzny algorithm terminates.

□

4.2 The Simplex Algorithm

The Simplex algorithm is an algorithm that steps from a BFS to a better BFS, improving the quantity that we are trying to maximize (or minimize) at every step of the way.

Its input: $A \in \mathbb{R}^{m \times n}$, $b, c \in R^m$ such that

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

4.2.1 The algorithm

Step 1: Slack Introduce slack variables $x_{n+1}...x_{n+m}$ for each of the inequalities in $\mathcal{A}$, turning them into equations. This equations become your "extended" tableau, which is simply a tableau with the maximizing condition added on. You assume that you already have a BFS in the sense that you assume that if all non-basic variables are set to 0, then the basic variables will be non-negative.

Step 2: Improve In the expression to maximize, select a variable with a positive coefficient. This is called the improving variable. If this does not exist, then simply set all basic variables to

Step 3: Pivot Pivot on the equation that limits the improving variable the most (sets the lowest upper bound on it) by solving for the improving variable in that equation and then substituting for that variable in all other equations, including the equation for the quantity to be improved. Note that the upper bounds are conservative and assume that other variables are 0. If there is no limit on the improving variable, there is no limit to the quantity that needs to be maximized either.

4.2.2 Intuition

This algorithm can be viewed as "crawling along the edges" of the polytope representing the set of feasible solutions. In particular, this figure represents the crawl that we perform below.

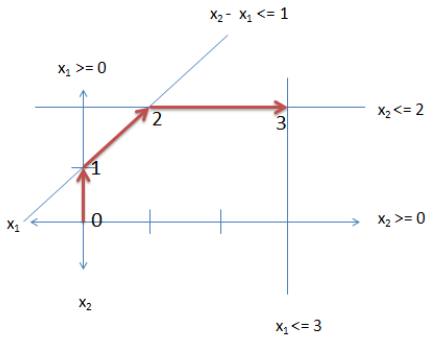


Figure 4.1: The traversal through the polytope in the Simplex example

4.2.3 Example

Consider the following equation:

$$\begin{aligned} & \max \quad z = x_1 + x_2 \\ \text{s.t} \quad & -x_1 + x_2 \leq 1 \\ & x_2 \leq 2 \\ & x_1 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

In step 1, we would make the following extended tableau:

$$\begin{aligned} \max \quad & z = x_1 + x_2 \\ \text{s.t} \quad & x_3 = 1 + x_1 - x_2 \\ & x_4 = 3 - x_1 \\ & x_5 = 2 - x_2 \end{aligned}$$

Note this state represents the BFS 0 in Figure 14.1. as when the non-basic variables are set to 0 $x_1 = x_2 = 0$.

In step 2, suppose that we choose x_2 as our improving variable. Note that $x_3 = 1 + x_1 - x_2$ limits x_2 to 1 as we assume x_1 is 0 for these purposes (we do not want to introduce an equation that causes a negative basic variable when non-basic variables are set to 0). $x_5 = 2 - x_2$ limits x_2 to 2 but this is not as strict of a bound. $x_4 = 3 - x_1$ does not relate to x_2.

In step 3, we pivot, so the tablau now looks like this:

$$\begin{aligned} \max \quad & z = 1 + 2x_1 - x_3 \\ \text{s.t} \quad & x_2 = 1 + x_1 - x_3 \\ & x_4 = 3 - x_1 \\ & x_5 = 1 - x_1 + x_3 \end{aligned}$$

Note this represents the BFS 1 in Figure 14.1, as when the non-basic variables are set to 0 $x_1 = 0$ and $x_2 = 1$.

Now, we must choose x_1 as our improving variable. x_1 is not limited by the x_2 equation, but is limited to 3 by the x_4 equation and 1 by the x_5 equation.
So we pivot on the x_5 equation to obtain the tableau

$$\begin{aligned} \max \quad & z = 3 - 2x_5 + x_3 \\ \text{s.t} \quad & x_2 = 2 - x_5 \\ & x_4 = 2 - x_3 + x_5 \\ & x_1 = 1 - x_5 + x_3 \end{aligned}$$

Note this represents the BFS 2 in Figure 14.1, as when the non-basic variables are set to 0 $x_1 = 1$ and $x_2 = 2$.

We choose x_3 as the improving variable. It is limited to 2 by the x_4 but is not limited by the x_1 equation.
So we pivot on the x_4 equation:

$$\begin{aligned} \max \quad & z = 5 - x_5 - x_4 \\ \text{s.t} \quad & x_2 = 2 - x_5 \\ & x_3 = 2 - x_4 + x_5 \\ & x_1 = 3 - x_4 \end{aligned}$$

Note this represents the BFS 3 in Figure 14.1, as when the non-basic variables are set to 0 $x_1 = 3$ and $x_2 = 2$. Notice this point is optimal.

There is no improving variable anymore, so we set $x_5 = 0$, $x_4 = 0$ and thus $x_3 = 2$, $x_2 = 2$, $x_1 = 3$, and so we return $x_1 = 3$ and $x_2 = 2$ as the optimal solution.

4.2.4 Issues

Unbounded? If this is the case, there will at some point be an improving variable that has no constraints on it.

No improvement? If you get an upper bound of 0 on a variable, your pivot will not increase the quality of the solution by making the non-basic variables 0. Simply continue on in this case.

It is possibile for the simplex algorithm to stall! (see the homework)

Also, there are a few choices that you have to make in this algorithm. In step 2, there are often several improving variables to choose. In step 3, how do you resolve ties between two equally constraining equations?

These choices actually affect some key runtime properties of the Simplex algorithm.

Pivoting Rule(choice of improving variable)	Can cycle?	Can take exponential time?
Largest Coefficient	Yes	Yes
Largest Increase	Yes	Yes
Steepest-Edge(most parralel to obj. vector)	Yes	Yes
Lexicographical Rules (Bland's rules)	No!	Yes
Least-Entered	Yes	Yes
Clairvoyant (we know best route to opt.)	Yes	?

Hirch's Conjecture stated that the edge-vertex graph of any polytope with n points in d-dimensional space has diameter at most n-d, which implies that a clairvoyant Simplex algorithm could reach the optimum in linear time. This was disproven in 2010 by Francisco Santos Leal, but it is still possible (and an open problem) whether or not the diameter of a polytope is polynomial in the number of points.

Lecture 5

LP Duality*

In Lecture #3 we saw the Max-flow Min-cut Theorem which stated that the maximum flow from a source to a sink through a graph is always equal to the minimum capacity which needs to be removed from the edges of the graph to disconnect the source and the sink. This theorem gave us a method to prove that a given flow is optimal; simply exhibit a cut with the same value.

This theorem for flows and cuts in a graph is a specific instance of the **LP Duality** Theorem which relates the optimal values of LP problems. Just like the Max-flow Min-cut Theorem, the LP Duality Theorem can also be used to prove that a solution to an LP problem is optimal.

5.1 Primals and Duals

Consider the following LP

$$\begin{aligned} P = \max(2x_1 + 3x_2) \\ \text{s.t.} \quad 4x_1 + 8x_2 &\leq 12 \\ 2x_1 + x_2 &\leq 3 \\ 3x_1 + 2x_2 &\leq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

In an attempt to solve P we can produce upper bounds on its optimal value.

- Since $2x_1 + 3x_2 \leq 4x_1 + 8x_2 \leq 12$, we know $\text{OPT}(P) \leq 12$.
- Since $2x_1 + 3x_2 \leq \frac{1}{2}(4x_1 + 8x_2) \leq 6$, we know $\text{OPT}(P) \leq 6$.
- Since $2x_1 + 3x_2 \leq \frac{1}{3}((4x_1 + 8x_2) + (2x_1 + x_2)) \leq 5$, we know $\text{OPT}(P) \leq 5$.

In each of these cases we take a positive linear combination of the constraints, looking for better and better bounds on the maximum possible value of $2x_1 + 3x_2$. We can formalize

*Lecturer: Anupam Gupta. Scribe: Timothy Wilson.

this, letting y_1, y_2, y_3 be the coefficients of our linear combination. Then we must have

$$\begin{aligned} 4y_1 + 2y_2 + 3y_2 &\geq 2 \\ 8y_1 + y_2 + 2y_3 &\geq 3 \\ y_1, y_2, y_3 &\geq 0 \\ \text{and we seek } \min(12y_1 + 3y_2 + 4y_3) & \end{aligned}$$

This too is an LP! We refer to this LP as the dual and the original LP as the primal. The actual choice of which problem is the primal and which is the dual is not important since the dual of the dual is equal to the primal.

We designed the dual to serve as a method of constructing an upperbound on the optimal value of the primal, so if y is a feasible solution for the dual and x is a feasible solution for the primal, then $2x_1 + 3x_2 \leq 12y_1 + 3y_2 + 4y_3$. If we can find two feasible solutions that make these equal, then we know we have found the optimal values of these LP.

In this case the feasible solutions $x_1 = \frac{1}{2}, x_2 = \frac{5}{4}$ and $y_1 = \frac{5}{16}, y_2 = 0, y_3 = \frac{1}{4}$ give the same value 4.75, which therefore must be the optimal value.

5.1.1 Generalization

In general, the primal LP

$$P = \max(c^\top x \mid Ax \leq b, x \geq 0, x \in \mathbb{R}^n)$$

corresponds to the dual LP,

$$D = \min(b^\top y \mid A^\top y \geq c, y \geq 0, y \in \mathbb{R}^m)$$

where A is an $m \times n$ matrix.

When there are equality constraints or variables that may be negative, the primal LP

$$\begin{aligned} & P = \max(c^\top x) \\ \text{s.t.} \quad & a_i x \leq b_i \text{ for } i \in I_1 \\ & a_i x = b_i \text{ for } i \in I_2 \\ & x_j \geq 0 \text{ for } j \in J_1 \\ & x_j \in \mathbb{R} \text{ for } j \in J2 \end{aligned}$$

corresponds to the dual LP

$$\begin{aligned} & D = \min(b^\top y) \\ \text{s.t.} \quad & y_i \geq 0 \text{ for } i \in I_1 \\ & y_i \in \mathbb{R} \text{ for } i \in I_2 \\ & A_j y \geq c_j \text{ for } j \in J_1 \\ & A_j y = c_j \text{ for } j \in J_2 \end{aligned}$$

5.2 The Duality Theorem

The Duality Theorem will show that the optimal values of the primal and dual will be equal (if they are finite). First we will prove our earlier assertion that the optimal solution of a dual program gives a bound on the optimal value of the primal program.

Theorem 5.1 (The Weak Duality Theorem)**.** *Let* $P = \max(c^\top x \mid Ax \leq b, x \geq 0, x \in \mathbb{R}^n)$*, and let* D *be its dual LP,* $\min(b^\top y \mid A^\top y \geq c, y \geq 0, y \in \mathbb{R}^m)$*. If* x *is a feasible solution for* P *and* y *is a feasible solution for* D*, then* $c^\top x \leq b^\top y$*.*

Proof.

$$\begin{aligned} c^\top x &= x^\top c \\ &\leq x^\top (A^\top y) && \text{(Since } y \text{ feasible for } D \text{ and } x \geq 0) \\ &= (Ax)^\top y \\ &\leq b^\top y && \text{(Since } x \text{ is feasible for } P \text{ and } y \geq 0) \quad \square \end{aligned}$$

From this we can conclude that if P is unbounded ($\mathrm{OPT}(P) = \infty$), then D is infeasible. Similarly, if D is unbounded ($\mathrm{OPT}(D) = -\infty$), then P is infeasible.

Therefore we have the following table of possibilities for the feasibility of P and D.

$P \backslash D$	Unbounded	Infeasible	Feasible
Unbounded	no	yes	no
Infeasible	yes	???	???
Feasible	no	???	???

The Duality Theorem allows us to fill in the remaining four places in this table.

Theorem 5.2 (Duality Theorem for LPs)**.** *If* P *and* D *are a primal-dual pair of LPs, then one of these four cases occurs:*

1. *Both are infeasible.*
2. P *is unbounded and* D *is infeasible.*
3. D *is unbounded and* P *is infeasible.*
4. *Both are feasible and there exist optimal solutions* x, y *to* P *and* D *such that* $c^\top x = b^\top y$*.*

We have already seen cases 2 and 3 as simple consequences of the Weak Duality Theorem. The first case can easily be seen to occur: a simple example takes A to be a $\mathbf{0}$ matrix, b to be strictly negative, and c to be strictly positive). Therefore the only remaining case of interest is case 4.

Geometric Proof. Let P be the program $\max(c^\top x \mid Ax \leq b, x \in \mathbb{R}^n)$ and D be dual program $\min(b^\top y \mid A^\top y = c, y \geq 0)$.

Suppose x^* is an optimal feasible solution for P. Let $a_i^\top x \leq b_i$ for $i \in I$ be all the constraints tight at x^*. We claim that the objective function vector c is contained in the cone $K = \{x \mid x = \sum_{i \in I} \lambda_i a_i, \lambda_i \geq 0\}$ generated by the vectors $\{a_i\}_{i \in I}$.

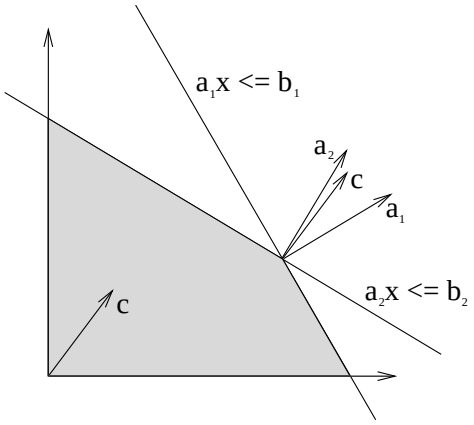


Figure 5.1: The objective vector lies in the cone spanned by the constraint vectors

Suppose for contradiction that c does not lie in this cone. Then there must exist a separating hyperplane between c and K: i.e., there exists a vector $d \in \mathbb{R}^n$ such that $a_i^\top d \leq 0$ for all $i \in I$, but $c^\top d > 0$. Now consider the point $z = x^* + \epsilon d$ for some tiny $\epsilon > 0$. Note the following:

- For small enough ϵ, the point z satisifes the constraints $Az \leq b$. Consider $a_j^\top z \leq b$ for $j \notin I$: since this constraint was not tight for x^*, we won't violate it if ϵ is small enough. And for $a_j^\top z \leq b$ with $j \in I$ we have $a_j^\top z = a_j^\top x^* + \epsilon a_j^\top d = b + \epsilon a_j^\top d \leq b$ since $\epsilon > 0$ and $a_j^\top d \leq 0$.
- The objective function value increases since $c^\top z = c^\top x^* + \epsilon c^\top d > c^\top x^*$.

This contradicts the fact that x^* was optimal.

Therefore the vector c lies within the cone made of the normals to the constraints, so c is a positive linear combination of these normals. Choose λ_i for $i \in I$ so that $c = \sum_{i \in I} \lambda_i a_i, \lambda \geq 0$ and set $\lambda_j = 0$ for $j \notin I$.

- We know $\lambda \geq 0$.
- $A^\top \lambda = \sum_{i \in [m]} \lambda_i a_i = \sum_{i \in I} \lambda_i a_i = c$.
- $b^\top \lambda = \sum_{i \in I} b_i \lambda_i = \sum_{i \in I} (a_i x^*) \lambda_i = \sum_{i \in I} \lambda_i a_i x^* = c^\top x^*$.

Therefore λ is a solution to the dual with $c^\top x^* = b^\top \lambda$, so by The Weak Duality Theorem, $\mathrm{OPT}(P) = \mathrm{OPT}(D)$. □

A somewhat more rigorous proof not relying on our geometric intuition that there should be a separating hyperplane between a cone and a vector not spanned by the cone relies on a lemma by Farkas that often comes in several forms. The forms we shall use are as follows

Theorem 5.3 (Farkas' Lemma (1894) - Form 1)**.** *Given $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, exactly one of the following statements is true.*

1. $\exists x \geq 0$ *such that* $Ax = b$.

2. $\exists y \in \mathbb{R}^m$ *such that* $y^\top A \geq 0$ *and* $y^\top b < 0$.

Theorem 5.4 (Farkas' Lemma - Form 2)**.** *Given* $A \in \mathbb{R}^{m \times n}$ *and* $b \in \mathbb{R}^m$*, exactly one of the following statements is true.*

1. $\exists x \in \mathbb{R}^n$ *such that* $Ax \geq b$.

2. $\exists y \geq 0, y^\top A = 0$*, and* $y^\top b > 0$.

Proofs. Left to the reader (homework 2). □

Duality Theorem using Farkas' Lemma. Let P be the program $\min(c^\top x \mid Ax \geq b, x \in \mathbb{R}^n)$ and D be dual program $\max(b^\top y \mid A^\top y = c, y \geq 0)$.

Suppose the the dual is feasible and its maximum value is δ. Let $P' = \{x \mid Ax \geq b, c^\top x \leq \delta\}$. If P' has a feasible solution, then P must also have a feasible solution with value at most δ. The LP P' is also equivalent to $\{x \mid Ax \geq b, -c^\top x \geq -\delta\}$.

Suppose for contradiction P' is infeasible. Then by Farkas' Lemma (Form 2) there exists $\begin{pmatrix} y \\ \lambda \end{pmatrix} \geq 0$ such that

$$(y^\top \ \lambda) \begin{pmatrix} A \\ -c^\top \end{pmatrix} = 0 \text{ and } (y^\top \ \lambda) \begin{pmatrix} b \\ -\delta \end{pmatrix} > 0$$

This implies $y^\top A - \lambda c^\top = 0$ and $y^\top b - \lambda\delta > 0$.

- If $\lambda = 0$, then $y^\top A = 0$ and $y^\top b > 0$. Choose $z \geq 0$ such that $A^\top z = c$ and $b^\top z = \delta$. Then for $\epsilon > 0$,

$$\begin{aligned} A^\top(z + \epsilon y) &= 0 \\ z + \epsilon y &\geq 0 && \text{(Since } y \geq 0\text{)} \\ b^\top(z + \epsilon y) &= \delta + \epsilon b^\top y \\ &> \delta \end{aligned}$$

  so $z + \epsilon y$ is a feasile solution of D with value greater than δ, a contradiction.

- Otherwise we can scale y and λ to make $\lambda = 1$ (since $y, \lambda \geq 0$), so $y^\top A = c^\top$ and $y^\top b > \delta$. This means y is a solution to D with value greater than δ, a contradiction.

Therefore P' is feasible, so P is feasible with value at most δ. By The Weak Duality Theorem, $\text{OPT}(P) = \delta = \text{OPT}(D)$. □

In the next couple of lectures, we will continue to explore duality, and its applications.

Lecture 6

Duality of LPs and Applications*

Last lecture we introduced duality of linear programs. We saw how to form duals, and proved both the weak and strong duality theorems. In this lecture we will see a few more theoretical results and then begin discussion of applications of duality.

6.1 More Duality Results

6.1.1 A Quick Review

Last time we saw that if the primal $(\mathcal{P})$ is

$$\begin{aligned} \max \quad & c^\top x \\ s.t. \quad & Ax \leq b \end{aligned}$$

then the dual $(\mathcal{D})$ is

$$\begin{aligned} \min \quad & b^\top y \\ s.t. \quad & A^\top y = c \\ & y \geq 0. \end{aligned}$$

This is just one form of the primal and dual and we saw that the transformation from one to the other is completely mechanical. The duality theorem tells us that if $(\mathcal{P})$ and $(\mathcal{D})$ are a primal-dual pair then we have one of the three possibilities

1. Both $(\mathcal{P})$ and $(\mathcal{D})$ are infeasible.
2. One is infeasible and the other is unbounded.
3. Both are feasible and if x^* and y^* are optimal solutions to $(\mathcal{P})$ and $(\mathcal{D})$ respectively, then $c^\top x^* = b^\top y^*$.

*Lecturer: Anupam Gupta. Scribe: Deepak Bal.

6.1.2 A Comment about Complexity

Note that the duality theorem (and equivalently, the Farkas Lemma) puts several problems related to LP feasibility and solvability in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$.

E.g., Consider the question of whether the equational form LP $Ax = b, x \geq 0$ is feasible. If the program is feasible, we may efficiently verify this by checking that a "certificate" point satisfies the equations. By taking this point to be a vertex and appealing to Hwk1 (Problem 4), we see that we may represent this certificate point in size polynomial in the size of the input. On the other hand, if the LP is infeasible, then Farkas Lemma (Form 1 from Lecture 5) says we can find a $y \in \mathbb{R}^m$ with $y^\top A \geq 0$ and $y^\top b < 0$. Again appealing to Homework 1, we may find a succinctly represented solution to this set of equations, thus providing a "certificate" for the infeasibility of the original LP.

We can similarly ask for whether the value of the LP $\max\{c^\top x \mid Ax \leq b\}$ is at least δ or not. Again, if we have n variables and m equations, we can convert this general-form LP into an equivalent equational form LP with $O(m+n)$ constraints and variables, and whose size is not much more. Now, if there is a solution with value at least δ, we can show a BFS x^* for this equivalent LP—this will have polynomial size, for the same reasons. And if there is no such solution of value δ or higher, there is a solution to the dual $\min\{b^\top y \mid A^\top y = c, y \geq 0\}$ of value strictly less than δ and we can give this dual solution. (Again this "proof" will be polynomial-sized.) Hence the decision problem "is the value of this maximization LP at least δ" is in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$.

6.1.3 Duality from Lagrange Multipliers

Suppose we have the problem $(\mathcal{P})$

$$\begin{aligned} \max \quad & c^\top x \\ s.t. \quad & Ax \leq b \end{aligned}$$

where as usual, the constraints are $a_i x \leq b_i$ for $i \in [m]$. Let $K = \{x \mid Ax \leq b\}$. Now consider the situation wher we are allowed to violate the constraints, but we penalize a violation of the ith constraint at a rate of $\lambda_i \geq 0$ per unit of violation. Let $\lambda = (\lambda_1 \ldots, \lambda_m)^\top$ and define

$$g(x, \lambda) := \; c^\top x + \sum_{i \in [m]} \lambda_i (b_i - a_i x).$$

Then we see that for each feasible $x \in K$, $\lambda \geq 0$, we get $g(x, \lambda) \geq c^\top x$. So now letting x be unconstrained we have that

$$g(\lambda) := \max_{x \in \mathbb{R}^n} \; g(x, \lambda) \geq OPT(\mathcal{P}).$$

In other words, for each λ, $g(\lambda)$ provides an upper bound on the optimal value of the LP. Naturally, we may ask for the best upper bound acheived in this way, *i.e.*,

$$g* = \min_{\lambda \geq 0} \; g(\lambda).$$

Putting together our definitions, we get

$$\begin{aligned} g* &= \min_{\lambda \geq 0} \max_{x} \left\{ c^\top x + \lambda^\top (b - Ax) \right\} \\ &= \min_{\lambda \geq 0} \left(b^\top \lambda + \max_{x} \left\{ (c^\top - \lambda^\top A)x \right\} \right) \end{aligned}$$

If $c^\top - \lambda^\top A$ has any non-zero entries, then the maximum over all x is ∞ which gives us a useless upper bound. Hence we really should only consider λ which satisfy $A^\top \lambda = c$. So all in all, this is

$$\begin{aligned} \min \quad & b^\top \lambda \\ s.t. \quad & A^\top \lambda = c \\ & \lambda \geq 0. \end{aligned}$$

which is the dual! So we see that the technique of Lagrange multipliers in this context is really just a form of duality. We will return to Lagrange multipliers later when dealing with more general convex optimization problems.

6.1.4 Complementary Slackness

Often times, the following theorem is very useful.

Theorem 6.1. *Suppose we have the primal dual pair $(\mathcal{P}), (\mathcal{D})$ from Section 6.1.1. If $(\mathcal{P}), (\mathcal{D})$ are both feasible with x^*, y^* feasible solutions, then following are equivalent*

1. *x^*, y^* are both optimal.*
2. *$c^\top x^* = b^\top y^*$.*
3. *$(y^*)^\top (Ax^* - b) = 0$*

In words, property 3 means that at optimality, either a dual variable is 0 or its corresponding inequality is tight (or both). Equivalently, for all constraints $i \in [m]$, if $y_i^* > 0$, then $a_i x = b_i$. Here we use the non-negativity of y^* and the fact that x^* is feasible.

Proof. 1 and 2 are equivalent by the duality theorem. We will prove 2 and 3 are equivalent. Suppose 2 holds. Then $c^\top x^* = (y^*)^\top b$ and on the other hand $c^\top x^* = (y^*)^\top A x^*$ since y^* is feasible. This holds if and only if $(y^*)^\top (Ax^* - b) = 0$ which is 3. □

6.2 Applications of Duality

In this section we will discuss two applications of duality. First the max-flow/min-cut theorem which was discussed in Lecture 3 without mention of duality. Then we will discuss König's Theorem on bipartite graphs.

6.2.1 Max-Flow = Min-Cut

In this problem, we are given a directed graph $G = (V, A)$ with two "special" vertices $s, t \in V$ called the source and sink. We are also given capacities c_e for all $e \in A$. The max-flow problem (or more formally, the max-(s,t)-flow problem) is to find an assignment of flows on the edges which obeys the capacities and maximizes the total amount of flow from s to t. For our purposes, we will formulate this differently than in Lecture 3.

Let P_{st} be the set of all paths from s to t. Note that P_{st} likely has size large comparaed to the number of nodes and arcs. Let f_p represent the flow assigned to path $p \in P_{st}$. Then the max-flow problem, which we will consider our primal problem $(\mathcal{P})$ is formulated as

$$\begin{aligned} \max \quad & \sum_{p \in P_{st}} f_p \\ s.t. \quad & \sum_{p \ni e} f_p \leq c_e \quad \forall e \in A \\ & f_p \geq 0 \quad \forall p \in P_{st}. \end{aligned}$$

Note in this formulation, there may be exponentially many variables, but according to earlier results, in any BFS there will be at most $|A|$ many non-zero variables. The dual formulation $(\mathcal{D})$ is then

$$\begin{aligned} \min \quad & \sum_{e \in A} c_e x_e \\ s.t. \quad & \sum_{e \in p} x_e \geq 1 \quad \forall p \in P_{st} \\ & x_e \geq 0 \quad \forall e \in A. \end{aligned}$$

We may think of x_e as the length of the edge e. Thus $c_e x_e$ represents the "volume" of the edge e. So this dual problem is saying, find a "volume-minimizing" assignment of lengths to the edges so that every s-t path has length at least 1. The duality theorem tells us that the max flow (optimal value for $(\mathcal{P})$) is equal to this value. But our goal is to show that max-flow is equal to the min-(s,t)-cut! So we'd better show that this dual value actually equals the min-(s,t)-cut (which we call the min-cut in the rest of the discussion, for brevity).

Soon, we will see that this dual actually has 0-1 BFS's. With this information it is obvious that $(\mathcal{D})$ will represent a minimum cut. Let us ignore this for now though, and prove the result with what we have.

For an s-t cut $(S, \overline{S})$, let $E(S, \overline{S})$ represent the edges crossing the cut and let $c(S, \overline{S})$ represent the sum of capacities of edges crossing the cut. Then for any (s,t)-cut $(S, \overline{S})$, we can let $x_e = 1$ for all $e \in E(S, \overline{S})$ and $x_e = 0$ for all others. Then this is clearly feasible for $(\mathcal{D})$. Consequently we have that

$$OPT(\mathcal{D}) \leq \text{min-}(s,t)\text{-cut}.$$

Now we must show the other, less trivial direction.

Theorem 6.2. *Suppose x is a solution of $(\mathcal{D})$ of value $c^\top x$. Then there exists an (s,t)-cut $(S, \overline{S})$ such that $c(S, \overline{S}) \leq c^\top x$.*

Proof. As above, we may interpret the x_e's as edge lengths. Let $d(v)$ be the shortest path distance from s to v for all $v \in V$ according to the lengths x_e. The x_e's are all non-negative so this is well defined. Note that $d(s) = 0$ and $d(t) \geq 1$ by the set of constraints in $(\mathcal{D})$.

Consider $\rho \in [0, 1)$. Let $S_\rho = \{v \in V \mid d(v) \leq \rho\}$. Then $(S_\rho, \overline{S}_\rho)$ is a feasible s-t cut in G. Now suppose ρ is chosen from $[0, 1)$ according to the uniform distribution. Then if we can show that

$$\mathbf{E}[c(S_\rho, \overline{S}_\rho)] \leq c^\top x$$

we will be done since this would imply that there exists a ρ with $c(S_\rho, \overline{S}_\rho) \leq c^\top x$. Note that

$$\mathbf{E}[c(S_\rho, \overline{S}_\rho)] = \sum_{e \in A} c_e \cdot \mathbf{Pr}[e \in E(S_\rho, \overline{S}_\rho)]$$

by linearilty of expectation. Let $e = (u, v)$ and let ρ^* be the smallest value so that $u \in S_{\rho^*}$. Then $\forall \rho \geq \rho^* + x_e$, $v \in S_\rho$. So $\mathbf{Pr}[u \in S_\rho,\ v \notin S_\rho] \leq x_e$, so

$$\mathbf{E}[c(S_\rho, \overline{S}_\rho)] = \sum_{e \in A} c_e \cdot \mathbf{Pr}[e \in E(S_\rho, \overline{S}_\rho)] \leq \sum_{e \in A} c_e \cdot x_e = c^\top x \qquad \square$$

So we have min-cut $\leq OPT(\mathcal{D})$, which proves that indeed max-flow is equal to min-cut by the duality theorem. In fact, we have proved that the polytope for $(\mathcal{D})$ is integral. Theorem 6.2 says that for any feasible solution x to the min-cut LP, and any cost vector c, there exists an integer s-t cut $(S_\alpha, \overline{S_\alpha})$ with cost at most $c^\top x$. Note that this s-t cut corresponds to an integer vector $y \in \mathbb{R}^{|A|}$ where $y_e = 1 \iff e \in E(S_\alpha, \overline{S}_\alpha)$ and $y_e = 0$ otherwise. This y is also feasible for the cut LP.

To see why the polyhedron K of $(\mathcal{D})$ is integer, consider any vertex x of K. By the definition of vertex, there is some cost function such that x is the unique minimizer for $\min\{c^\top x \mid x \in K\}$. But since $c^\top y \leq c^\top x$, and $y \in K$, it follows that $x = y$ and hence x is integral.

You may want to think about what information you can conclude about optimal flows/cuts using complementary slackness. E.g., we get that the paths carrying flow are all shortest paths according to the edge length x_e's: they all must have length 1. Similarly, if an edge has non-zero length according to the optimal dual solution, then it must be saturated in an optimal primal solution. (In fact, in *every* optimal primal solution.)

6.2.2 König's Theorem for Bipartite Graphs

Given an undirected bipartite graph $G = (U, V, E)$, a *matching* is a set of edges which do not intersect at any vertices. A *vertex cover* is a set of vertices S such that for all $e \in E$, $e \cap S \neq \emptyset$. Even though a vertex cover is covering edges, it is called a vertex cover because it is a set of vertices. To clarify, a vertex cover is a set of vertices: this is how one should keep from getting confused.

Theorem 6.3 (Kónig's Theorem)**.** *For bipartite graph $G = (U, V, E)$,*

$$\max\{|M| \ :\ M \text{ is a matching of } G\} = \min\{|S| \ :\ S \text{ is a vertex cover of } G\}$$

Proof. Let MM and MM_{LP} represent the cardinality of the maximum matching, the optimal value of the maximum matching LP relaxation. Similarly, let VC and VC_{LP} denote the cardinality of the minimum vertex cover, and the optimal value of the vertex cover LP relaxation respectively. So we have that MM_{LP} is given by

$$\begin{aligned} \max \quad & \sum_{(i,j)\in E} x_{ij} \\ s.t. \quad & \sum_{j:(i,j)\in E} x_{ij} \leq 1 \quad \forall i \in U \\ & \sum_{i:(i,j)\in E} x_{ij} \leq 1 \quad \forall j \in V \\ & x_{ij} \geq 0 \quad \forall (i,j) \in E. \end{aligned}$$

Then the dual is

$$\begin{aligned} \min \quad & \sum_{i\in U} y_i + \sum_{j\in V} z_j \\ s.t. \quad & y_i + z_j \geq 1 \quad \forall (i,j) \in E \\ & y_i,\, z_j \geq 0 \quad \forall (i,j) \in E. \end{aligned}$$

Adding in an integrality constraint to this gives us VC, since any vertex cover is feasible for this LP. Hence we define this dual to be VC_{LP}. So using the notations above to represent both the problem formulations and the optimum values, we now know, using duality theory that

$$MM \leq MM_{LP} = VC_{LP} \leq VC.$$

If we can show that the two inequalities are actually equalities, we would be done. In fact we will show that the BFS's of MM_{LP} and VC_{LP} are both integral.

Claim 6.4. *Any BFS of MM_{LP} is integral. Hence, $MM = MM_{LP}$ for bipartite graphs.*

Proof. We essentially did this in Lecture 3, except there we had equality constraints. So in that case, we always could find a cycle of fractional values. Here, this might not be the case. Suppose we have a fractional extreme point. If we find a cycle, proceed as in the Lecture 3 proof. We may only find a tree of fractional values. Similarly to the proof in Lecture 3, we alternately raise and lower the values by ϵ along a path from a leaf to a leaf on this tree. Choose ϵ small enough so that none of the constraints become violated after the adjustment. We can average these "complementary" solutions to contradict the extremity of the original point. □

Claim 6.5. *Any BFS of VC_{LP} is integral. Hence, $VC = VC_{LP}$ for bipartite graphs.*

Proof. Let y^* be an optimal solution to VC_{LP} chosen so that y^* has a maximum number of integer components. It does not make sense that $y*$ would have any component > 1, so assume all are ≤ 1. Let F be the set of fractional vertices. If $F = \emptyset$, we are done. WLOG,

suppose $F \cap U$ is larger than or the same size as $F \cap V$. Let $\epsilon = \min\{y_i^* \mid i \in F \cap U\}$. Then subtract ϵ from all the components in $F \cap U$ and add ϵ to all the components in $F \cap V$.

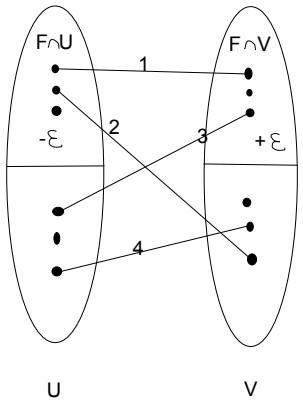


Figure 6.1: U and V

As seen in Figure 6.1, we need to check that constraints corresponding to edges of type 1 to 4 are still satisfied. Constraints of type 1 are not affected as epsilon is both added and subtracted. Constraints of type 4 are no affected at all and constraints of type 3 are trivially still satisfied. For constraints fo type 2, since the vertex in U was fractional and the vertex in V was not, the vertex in V must have had value 1! So subtracting ϵ from the U vertex will not violate the constraint. So we have a solution with objective function less than or equal to the original and with one less fractional component. This is a contradiction. Hence VC_{LP} has integer vertices and so $MM = MM_{LP} = VC_{LP} = VC$ and the theorem is proved. □

Putting it all together, we get that on bipartite graphs, the minimum cardinality vertex cover equals the maximum cardinality maximum matching. Note that this equality is false for general graphs (e.g., the 3-cycle shows a counterexample). □

An important aside: The proofs of Claims 6.4 and 6.5 show that the vertices of those LPs are integral: this fact is independent of what the objective function was. Indeed, such results immediately extend to weighted versions of the problems. E.g., we get that the *weighted* bipartite matching problem, where the edges have weights w_e, and the goal is to find the matching with the highest weight $\sum_{e \in M} w_e$, can be solved on bipartite graphs, just by finding a basic optimal solution to MM_{LP} with objective function $w^\top x$. Similarly, for the minimum weight vertex cover on bipartite graphs, we can seek to minimize $\sum_{i \in U} w_i y_i + \sum_{j \in V} w_j z_j$ subject to the constraints in VC_{LP}, and an optimal BFS gives us this min-weight vertex cover.

Another connection. Hall's theorem says that in a bipartite graph $G = (U, V, E)$, there is a matching M that matches all the vertices on the left (i.e. has cardinality $|U|$) if and only if every set $S \subseteq U$ on the left has at least $|S|$ neighbors on the right. König's theorem (which

shows that the size of the maximum matching in G is precisely the size of the minimum vertex cover of G) is equivalent to Hall's theorem. We leave the proof for the reader.

Lecture 7

Duality Applications (Part II)*

In this lecture, we'll look at applications of duality to three problems:

1. Finding *maximum spanning trees* (MST). We know that Kruskal's algorithm finds this, and we'll see a proof of optimality by constructing an LP for MST, and exhibiting a feasible dual solution whose cost is equal to the MST.

2. Finding *minimum cost arborescences*. We'll see an algorithm given independently by Edmonds, Chu & Liu, and Bock, which uses the dual to guide the algorithm, and to give a proof of the optimality of the solution.

3. Finally, we'll look at an LP formulation of *non-bipartite matchings*: this formulation is due to Edmonds, and we'll give a proof (due to Schrijver) that shows the integrality of all vertices of the perfect matching polytope we write down.

7.1 Maximum spanning tree

Given a graph $G = (V, E)$, and edge weights $w_e \geq 0$, the goal is to output a *spanning tree* of maximum weight. To design a linear program for this problem, we use variables $\{x_e\}_{e \in E}$.

Notation 7.1. For a set $S \subseteq V$, we denote by δS the set of edges leaving S. For $A \subseteq E$, define $x(A) = \sum_{e \in A} x_e$.

Consider the following LP.

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e \\ & \text{s.t.} \quad 1 \geq x_e \geq 0 \\ & \sum_{e \in E} x_i = n - 1 \\ & x(\delta S) \geq 1 \quad \forall S \neq \varnothing, V \end{aligned}$$

*Lecturer: Anupam Gupta. Scribe: Carol Wang.

Note: In class we left it as an exercise to see whether every vertex of this LP was integral. It is *not*: on the blog we later saw the following counterexample.

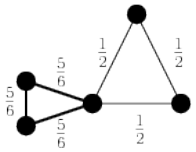


Figure 7.1

The maximum weight spanning tree has weight 2. However, the LP solution given here (with $x_e = 1/2$ on the thin edges, and $x_e = 5/6$ on the thick ones) has $w^\top x = 3 \cdot 5/6 = 2.5$. It also has $\sum_e x_e = 3 \cdot 1/2 + 3 \cdot 5/6 = |V| - 1$, and you can check it satisfies the cut condition. (In fact, the main gadget that allows us to show this LP has an "integrality gap" is to assign 1/2 to the edges of the thin triangle — much like for the naive non-bipartite matching LP you'll see later in this lecture.)

Well, we tried. Let's consider a slightly different LP. For $S \subseteq V$, let E_S denote all edges between vertices in S. (For simplicity, we will assume in this lecture that all the edge weights are non-negative.)

$$\begin{aligned} \max \quad & \sum_{e \in E} w_e x_e \\ \text{s.t.} \quad & x(E_S) \leq |S| - 1 \quad \forall S \subseteq V, |S| \geq 1 \\ & x_e \geq 0 \end{aligned} \tag{P}$$

Remark 7.2. Any spanning tree satisfies these constraints. Therefore, opt(P) is at least the weight of the maximum spanning tree.

Recall that Kruskal's algorithm starts with a forest consisting of all the vertices, and iteratively adds the heaviest edge which connects two trees.

Theorem 7.3. *There exists an integer optimum for the LP P, and Kruskal's algorithm finds it.*

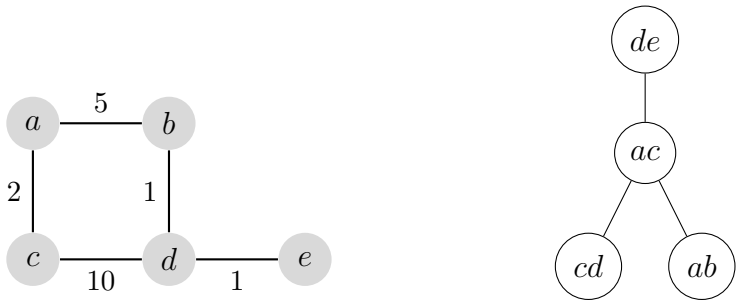
Proof. We will construct a dual solution such that its value is the value of the MST which Kruskal finds. Let's write down the dual.

Notation 7.4. For a set S, write $r(S) := |S| - 1$. (This is the size of a spanning tree on S.)

Then the dual of P is

$$\begin{aligned} \min \quad & \sum_{S \neq \varnothing} r(S) y_S \\ \text{s.t.} \quad & \sum_{S : e \in E_S} y_S \geq w_e \, \forall e \in E \\ & y_S \geq 0 \end{aligned} \tag{D}$$

That is, we should assign a value to each nonempty subset of vertices which gives "enough" weight to each edge.



(a) Graph G (b) Resulting tree

Figure 7.2: Example for Kruskal's algorithm

Primal solution

Kruskal: Pick edges $K = \{e_1, e_2, \ldots, e_{n-1}\}$ with $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_{n-1})$. Then a primal solution is given by

$$x_e = \begin{cases} 1 & e \in K \\ 0 & e \notin K \end{cases}.$$

The value of this solution is $\sum_e w_e x_e$, which is exactly the value of the Kruskal solution.

Dual solution

Suppose that we run Kruskal on our graph. We consider the sequence of components satisfied by the addition of each edge. This naturally induces a tree structure on the edges of the MST, where the parent of a subtree corresponding to some component C is the first edge added to the MST which leaves C.

For example, in Figure 7.2a, choosing the edges (c, d) and (a, b) satisfy the components $\{a, b\}$ and $\{c, d\}$, and adding (a, c) satisfies the entire component $\{a, b, c, d\}$. The final edge (d, e) then satisfies the entire tree.

We will consider the tree induced by Kruskal's algorithm.

Notation 7.5. Let $V(e_j)$ be the set of vertices spanned by the edges in the subtree rooted at e_j.

We will write $T(e_j)$ for this subtree.

Define a dual solution y_S by

$$y_S = \begin{cases} w_{e_j} - w_{\text{parent}(e_j)} & S = V(e_j) \text{ for some } j \\ 0 & \text{else} \end{cases}.$$

Example 7.6. For Figure 7.2b, we have $y_{\{c,d\}} = 10 - 2 = 8$, $y_{\{a,b\}} = 5 - 2 = 3$, $y_{\{a,b,c,d\}} = 2 - 1 = 1$, and $y_{\{a,b,c,d,e\}} = 1 - 0 = 1$.

We will show both that this solution is feasible, and that its value is exactly the value of the maximum spanning tree, proving the theorem.

Lemma 7.7. *y_S is feasible.*

Proof. Kruskal's algorithm is greedy, and the parent of any edge is added after that edge, so $y_S \geq 0$.

To show $\sum_{S:e\in E_S} y_S \geq w_e$ for every edge, fix an edge e. Consider the first time e lies in $T(e_j)$ for some j, and consider a path $p_1 = e_j, \ldots, p_k$ from e_j to the root of our tree. e lies in $V(p_i)$ for each i, and in particular, by nonnegativity,

$$\sum_{S:e\in E_S} y_S \geq \sum_i y_{V(p_i)} = w_{p_1} - w_{p_2} + \cdots + w_{p_k} = w_{p_1} = w_{e_j} \geq w_e,$$

where in the last step we used the fact that if w_e were greater than w_{e_j}, then we would have chosen e rather than e_j. □

Lemma 7.8.

$$\sum_{S\neq\varnothing} r(S)y_S = \sum_{e\in K} w_e.$$

(Recall K is the spanning tree output by Kruskal.)

Proof. We prove by (strong) induction the slightly stronger statement that

$$\sum_{S\subseteq V(e_j)} r(S)y_S = \sum_{e\in T(e_j)} w_e - r\big(V(e_j)\big)w_{\text{parent of } e_j}$$

for every $e_j \in K$.

We induct on the number of nodes in the subtree $T(e_j)$. In the base case, $T(e_j)$ is a leaf, so $|V(e_j)| = 2$ and the claim holds by our definition of y_S.

Therefore, suppose that the claim holds for subtrees of $\leq k$ nodes, and consider $T(e_j)$ of $k+1$ nodes.

Case 1. e_j has one child, e, in $T(e_j)$. Then $V(e_j) = V(e) \cup \{u\}$ for some vertex $u \notin V(e)$. In particular, $r(V(e)) = r(V(e_j)) - 1$. Then

$$\begin{aligned}\sum_{S\subseteq V(e_j)} r(S)y_S &= \left(\sum_{S\subseteq V(e)} r(S)y_S\right) + r(V(e_j))y_{V(e_j)} \\ &= \left(\sum_{e\in T(e)} w_e - r(V(e))w_{e_j}\right) + r(V(e_j))y_{V(e_j)},\end{aligned}$$

using the inductive hypothesis.

Since $r(V(e)) = r(V(e_j)) - 1$ and $y_{V(e_j)} = w_{e_j} - w_{\text{parent}(e_j)}$, the claim holds.

Case 2. e_j has two children e, e' in $T(e_j)$. Then $V(e_j) = V(e) \cup V(e')$, and $V(e) \cap V(e') = \varnothing$. In particular, $r(V(e_j)) = r(V(e)) + r(V(e')) + 1$. Applying the inductive hypothesis to $T(e)$ and $T(e')$, we can simplify as in Case 1.

Recall that $y_S = 0$ unless $S = V(e)$ for some $e \in K$.

□

Thus the maximum spanning tree LP has an integer optimum given by Kruskal's algorithm.

□

7.2 Minimum cost arborescence

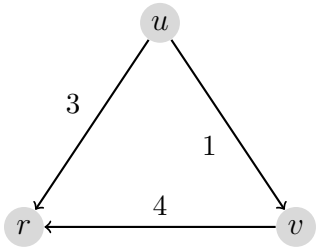
Think of these as spanning trees on *directed* graphs. Given a directed graph $G = (V, E)$ with a root vertex r, an *arborescence* of G is a subgraph $T = (V, E_T)$ such that:

1. Every node is connected to r, and there are no cycles even if we ignore directions.
2. Every node has a directed path to r.

Remark 7.9. One often sees this definition with directed paths from r to other nodes, but we will use this convention.

Remark 7.10. An arborescence may not exist for a given root r, but a certificate of infeasibility is a vertex with no path to r.

Note that it may also not be unique. Furthermore, the following example shows that adapting Prim's algorithm (greedily starting at the root) may not yield an optimal solution.



Notation 7.11. We write $\delta^+ S$ to denote the set of edges leaving S for any $S \subseteq V$.

We will assume $c_e \geq 0$ for every e. The primal LP is

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e \\ \text{s.t.} \quad & x(\delta^+ S) \geq 1 \qquad S \text{ valid} \\ & x_e \geq 0 \end{aligned} \tag{P}$$

and the dual is

$$\begin{aligned} \max \quad & \sum_{S \text{ valid}} y_S \\ \text{s.t.} \quad & \sum_{S : e \in \delta^+ S} y_S \leq c_e \qquad \forall e \\ & y_s \geq 0 \end{aligned} \tag{D}$$

where we will call $S \subseteq V$ valid if S is nonempty and $r \notin S$.

Algorithm for minimum cost arborescence

1. If zero-weight edges connect every $v \neq r$ to r, then we get a (integral) primal solution of value 0 (using depth first search or similar). A matching dual solution sets $y_S = 0$ for every S. In particular, this is optimal.

2. Otherwise, consider the graph restricted to zero-weight edges. Choose a maximal strongly connected component C of this subgraph. Then in the graph with all edges, there are no zero-weight edges out of C. Let $c^* = \min_{e \in \delta^+ C} c_e$. For each $e \in \delta^+ C$, define updated weights $c'_e = c_e - c^*$.

   Run the algorithm recursively on G with C contracted to one vertex $\hat{C}$ and with the updated weights to get optimal primal/dual solutions x_e, y_S for the contracted graph. Inductively, x_e will be integral.

   Let A be an arborescence on C (of zero cost). Define primal/dual solutions $\hat{x}_e, \hat{y}_S$ for the uncontracted graph by

$$\hat{x}_e = \begin{cases} x_e & e \notin C \\ 1 & e \in A \\ 0 & e \in C \setminus A \end{cases} \qquad \hat{y}_S = \begin{cases} y_S & C \not\subseteq S \\ y_{\hat{C}} + c^* & S = C \\ y_{S \setminus C \cup \{\hat{C}\}} & C \subsetneq S \end{cases}.$$

Remark 7.12. If $\sum c'_e x_e = \sum y_S$ (i.e., if the primal and dual solutions mutually certify optimality), then $\sum c_e \hat{x}_e = \sum \hat{y}_S$. This holds since $\sum y_S + c^* = \sum \hat{y}_S$. Furthermore, in any minimal arborescence on the contracted graph, exactly one edge from $\delta^+ C$ will be chosen.

Lemma 7.13. *$\hat{x}_e$ and $\hat{y}_S$ are feasible.*

Proof. $\hat{x}_e$ is feasible because x_e is feasible (and clearly the arborescence A satisfies the conditions). To show $\hat{y}_S$ is feasible, we only need to check $\sum_{S : e \in \delta^+ S} \hat{y}_S \leq c_e$ for $e \in \delta^+ C$. It is easy to see that this holds by definition of $\hat{y}_S$, since $\sum y_S \leq c_e - c^*$. □

> **Note:** This LP we wrote for arborascences is very similar to the one we first wrote for spanning trees, but that one did not work, whereas this one does! Interesting. Indeed, this LP can be used to give an algorithm for MSTs on undirected graphs.
>
> Indeed, take an undirected graph and replace each undirected edge by two directed edges of the same weight, pointing in opposite directions. Now the max-weight arborescence in this digraph has the same weight as the maximum spanning tree in the original graph. So an LP that looks pretty similar to the failed undirected one (namely $\max\{w^\top x \mid x(\partial v) = 1, x(\partial S) \geq 1, x \geq 0\}$) on that specially constructed *directed* graph gives an arborescence that corresponds to an integral maximum spanning tree on the original undirected graph.

7.3 Minimum cost perfect matching

We saw how to do this in the bipartite case, but suppose we just have a general graph.

Here is the LP we were using before.

$$\begin{aligned} \min \quad & \sum_e c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta v} x_e = 1 \quad v \in V \\ & x_e \geq 0 \end{aligned}$$

This does not necessarily have integer vertices: consider an unweighted triangle. By assigning each edge 1/2, we get a solution of value 1.5, but the maximum integral matching has value 1.

But suppose we added the constraint $x(\delta S) \geq 1$ for every odd set S. Now our LP is

$$\begin{aligned} \min \quad & \sum_e c_e x_e \\ \text{s.t.} \quad & \sum_{e \in \delta v} x_e = 1 \quad v \in V \\ & x(\delta S) \geq 1 \quad 2 \nmid |S| \\ & x_e \geq 0 \end{aligned} \tag{P}$$

Note that in the second set of constraints, we can just consider S of size at least 3 and at most $|V|-3$: if $|S| = 1$ or $|V|-1$, then the first set of constraints already implies $x(\partial S) = 1$. So just focus on

$$\begin{aligned} x(\partial v) &= 1 && \forall v \in V \\ x(\partial S) &\geq 1 && \forall S \subset V, |S| \text{ odd }, 3 \leq |S| \leq |V| - 3 \\ x &\geq 0 \end{aligned}$$

Let us call this perfect matching polytope PM. We'll call the first set of equalities the *vertex constraints*, and the second set the *odd-set inequalities*.

Remark 7.14. The odd-set inequalities are satisfied by any perfect integral matching, because at least one vertex in an odd set must be matched to a vertex outside the set.

Theorem 7.15 (Edmonds)**.** *Every vertex of P is integral.*

Proof. This was proven on the course blog. For completeness, the proof is copied here.

Suppose not, and suppose there exists graphs for which there is a fractional vertex. Consider a minimal counterexample $G = (V, E)$ (minimizing the sum of $|V| + |E|$, say), and some vertex solution x that is not integral. Clearly, $|V|$ must be even, else it will not satisfy the odd set constraint for $S = V$. First, the claim is that G cannot have a vertex of degree 1, or be disconnected (else we'll get a smaller counterexample) or be just an even cycle (where we know this LP is indeed integral). Being connected implies that $|E| \geq |V| - 1$, and neither being a cycle nor having a degree-1 vertex implies that $|E| \neq |V|$. So $|E| > |V|$.

Recall there are $|E|$ variables. So any vertex/BFS is defined by $|E|$ tight constraints. If any of these tight constraints are the non-negativity constraints $x_e \geq 0$, then we could drop that edge e and get a smaller counterexample. And since at most $|V|$ tight constraints come from the vertex constraints. So at least one odd-set constraint should be tight. Say this tight odd-set constraint is for the odd set $W \subseteq V$ with $|W| \geq 3$: i.e.,

$$x(\partial W) = 1$$

Now consider the two graphs G/W and $G/\overline{W}$ obtained by contracting W and $\overline{W}$ to a single new vertex respectively, and removing the edges lying within the contracted set. Since both W and $\overline{W}$ have at least 3 vertices, both are smaller graphs.

Now x naturally extends to feasible solutions y and z for these new graphs. E.g., to get y, set $y_e = x_e$ for all edges $e \in E \setminus \binom{W}{2}$. Note that if the set W got contracted to new vertex $\widehat{w}$ in G/W, then the fact that $x(\partial W) = 1$ implies that $y(\partial \widehat{w}) = 1$, and hence y is a feasible solution to the perfect matching polytope for graph G/W. Similarly, z is a feasible solution to the perfect matching polytope for graph $G/\overline{W}$.

By minimality of G, it follows that the perfect matching LP is integral for both G/W and $G/\overline{W}$: i.e., the vertices of the perfect matching polytope for these smaller graphs all correspond to perfect matchings. And that means that

$$y = \sum_i \lambda_i \cdot \chi_{M_i},$$

where χ_{M_i} is the natural vector representation of the perfect matching M_i in G/W, for values $\lambda_i \geq 0, \sum_i \lambda_i = 1$. Also, λ_i's can be taken to be rational, since y is rational, as are χ_{M_i}. Similarly, we have a rational convex combination

$$z = \sum_i \mu_i \cdot \chi_{N_i},$$

where N_i are perfect matchings in $G/\overline{W}$. Since λ_i, μ_i are rationals, we could have repeated the matchings and instead written

$$y = \frac{1}{k} \sum_i \chi_{M_i}$$
$$z = \frac{1}{k} \sum_i \chi_{N_i}$$

Finally, we claim that we can combine these to get

$$x = \frac{1}{k} \sum_i \chi_{O_i}$$

where O_i's are perfect matchings in G. How? Well, focus on edge $e = \{l, r\} \in \partial W$, with $l \in W$. Note that $y_e = z_e = x_e$. If we look at k matchings M_i in the sum for y: exactly x_e fraction of these matchings M_i – that is, kx_e matchings – contain e. Similarly, exactly kx_e

of the matchings N_i in the sum for x contain e. Now we can pair such matchings (which share a common edge in ∂W) up in the obvious way: apart from the edge e, such an M_i contains edges only within $\overline{W}$ and matches up all the vertices in $\overline{W}$ except vertex r, and N_i contains edges only within W and matches up all the vertices in $W \setminus \{l\}$. And e matches up $\{l, r\}$. Hence putting together these perfect matchings M_i and N_i in G/W and $G/\overline{W}$ gets us a perfect matching O_i for G.

So x can be written as a convex combination of perfect matchings of G. Hence, for x to be an extreme point (vertex) itself, it must be itself a perfect matching, and integral. This gives us the contradiction. □

Max-Weight Matchings

We didn't get to this, but suppose you want to write an LP whose vertices are precisely (integral) matchings in G, not just the perfect matchings. Here is the polytope Edmonds defined.

$$\begin{aligned} x(\partial v) &\leq 1 && \forall v \in V \\ \textstyle\sum_{e \in \binom{S}{2}} x_e &\leq \tfrac{|S|-1}{2} && \forall S \subset V, |S| \text{ odd} \\ x &\geq 0 \end{aligned}$$

Clearly, all matchings in G are feasible for this LP. Moreover, one can use the Perfect Matching Theorem above to show that every vertex of this polytope is also integral.

Lecture 8

The Ellipsoid Algorithm*

Recall from Lecture 6 that the duality theorem places the linear programming feasibility and solvability problems in $\mathsf{NP} \cap \text{co-}\mathsf{NP}$. In this class, we will see the ellipsoid algorithm, which was the first polynomial time algorithm for the LP feasibility problem; this places the LP solvability problem in P. The "Ellipsoid algorithm" was introduced by N. Shor in early 1970's as an iterative method for general convex optimization, and later applied by Khachiyan (1979) for linear programs.

8.1 Ellipsoids

In this section, we define an ellipsoid and note some of its useful properties for future use.

Definition 8.1. A (closed) *ball* $B(c, r)$ (in $\mathbb{R}^n$) centered at $c \in \mathbb{R}^n$ with radius r is the set

$$B(c, r) := \{x \in \mathbb{R}^n \,:\, x^T x \leq r^2\}.$$

The set $B(0, 1)$ is called the *unit ball*.

An ellipsoid is just an affine transformation of a ball.

Definition 8.2. An *ellipsoid* E centered at the origin is the image $L(B(0, 1))$ of the unit ball under an *invertible* linear transformation $L : \mathbb{R}^n \to \mathbb{R}^n$. An ellipsoid centered at a general point $c \in \mathbb{R}^n$ is just the translate $c + E$ of some ellipsoid E centered at 0.

We can write the above definition in a more explicit way as follows:

$$\begin{aligned} L(B(0,1)) &= \{Lx \,:\, x \in B(0,1)\} \\ &= \{y \,:\, L^{-1}y \in B(0,1)\} \\ &= \{y \,:\, (L^{-1}y)^T L^{-1}y \leq 1\} \\ &= \{y \,:\, y^T (LL^T)^{-1} y \leq 1\} \\ &= \{y \,:\, y^T Q^{-1} y \leq 1\} \end{aligned}$$

*Lecturer: Ryan O'Donnell. Scribe: Srivatsan Narayanan.

where $Q = LL^T$.

What can we say about the matrix $Q = LL^T$? From basic linear algebra, we recall from basic linear algebra that it is *positive definite*. We record this as fact below.

Fact 8.3. *For a symmetric matrix $Q \in \mathbb{R}^{n\times n}$, the following conditions are equivalent:*

1. *$Q = LL^T$ for some $L \in \mathbb{R}^{n\times n}$.*
2. *All the n eigenvalues of Q are nonnegative.*[1]

We say that Q is positive semi-definite *if any of the above conditions hold.*

We will add many more equivalent characterizations to this list later in the course. We will not prove the whole claim in this class; instead we verify just one of the directions to give a flavor.

Proof. (Of 1. $\implies$ 2.) We are given that $Q = LL^T$ for some $L \in \mathbb{R}^{n\times n}$. Suppose λ is an eigenvalue of Q with eigenvector $x \neq 0$; that is, $Qx = \lambda x$. Then

$$\lambda\|x\|^2 = \lambda x^T x = x^T(\lambda x) = x^T(Qx) = x^T LL^T x = (L^T x)^T(L^T x) = \|L^T x\|^2 \geq 0,$$

which shows that λ is real and nonnegative. □

Fact 8.4. *For a symmetric matrix $Q \in \mathbb{R}^{n\times n}$, the following conditions are equivalent:*

1. *$Q = LL^T$ for some* nonsingular[2] *matrix L.*
2. *All the n eigenvalues of Q are strictly positive.*

We say that Q is positive definite *if any of the above conditions hold.*

From the above claims, it is clear that an ellipsoid can equivalently be represented in terms of a positive definite matrix Q.

Definition 8.5. If $Q \in \mathbb{R}^{n\times n}$ is a positive definite matrix, then the ellipsoid associated with Q and centered at $c \in \mathbb{R}^n$ is

$$E(c, Q) := \{c + y \,:\, y^T Q^{-1} y \leq 1\} = \{y \,:\, (y-c)^T Q^{-1}(y-c) \leq 1\}.$$

Remark 8.6. The standard ball $B(0, r)$ is the ellipsoid $E(0, r^2 I)$. More generally, an "axial ellipsoid" with semiaxes $r_1, \ldots r_n$ is given by the ellipsoid $E\left(0, \begin{pmatrix} r_1^2 & & & 0 \\ & r_2^2 & & \\ 0 & & r_3^2 & \\ & & & \ddots \end{pmatrix}\right)$.

The final ingredient is the following fact about the volume of an ellipsoid. Denote by $\mathrm{vol}(A)$ the volume of a set $A \subseteq \mathbb{R}^n$.

[1] Recall that all eigenvalues of a real symmetric matrix are real.

[2] Nonsingular matrices are also known as *invertible*.

Fact 8.7. *If $A \subseteq \mathbb{R}^n$ and L is a linear transformation, then*

$$\mathrm{vol}(L(A)) = |\det L| \cdot \mathrm{vol}(A).$$

In particular, the volume of an ellipsoid $E(c, Q)$ is given by

$$\mathrm{vol}(E(c,Q)) = |\det L| \cdot \mathrm{vol}(B(0,1)) = \sqrt{\det Q} \cdot \mathrm{vol}(B(0,1)).$$

Thus we have related the volume of any ellipsoid to the volume of the unit ball in n dimension. Fortunately, the exact value of the constant of the proportionality, the volume of the unit n-ball, is irrelevant to us.[3]

8.2 The Ellipsoid Algorithm

The ellipsoid algorithm takes as input a convex set, and returns a point from the set provided it is nonempty. (If the set is empty, then we return "empty".) It is clear that this algorithm is useful for testing LP feasibility. Further, since the LP solvability problem reduces to the LP feasibility problem in polynomial time, this algorithm can also be used to solve linear programs as well.

Formally, the ellipsoid algorithm tests if a given convex set $K \subseteq \mathbb{R}^n$ is empty.

8.2.1 Requirements

Apart from the input set K, we assume that we are provided two additional parameters:

1. A number $R \in \mathbb{Q}$ $(R > 0)$ such that $K \subseteq B(0, R)$.
2. A rational $r > 0$ such that either $K = \emptyset$ or $K \supseteq B(c, r)$ for some point c. (Think of this requirement as basically stating that the feasible solution is not completely contained in some affine hyperplane.)

Now, how do we satisfy these two requirements while using the ellipsoid algorithm for solving LPs?

1. The first condition is easily handled. Given a linear program

   $$K = \{Ax = b \,:\, x \geq 0\},$$

   we can add in constraints of the form $-2^L \leq x \leq 2^L$ for each $i \in [n]$ for some $L = \mathrm{poly}(\langle A \rangle, \langle b \rangle, n)$ (without affecting the feasibility of the LP). Since this region is fully contained inside the ball of radius $\sqrt{n}2^L$, we can provide $R = n2^L$ as the parameter.

[3] For the curious: the volume of the unit n-ball has the exact expression $\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)}$, where $\Gamma(\cdot)$ is the "Gamma function". Asymptotically, this volume is $\tilde{\Theta}\left(\frac{C^n}{n^{n/2}}\right)$ for some absolute constant $C > 0$ (hiding $\mathrm{poly}(n)$ factors).

2. The second requirement is slightly tricky since the feasible region of the given LP is contained in the hyperplane $Ax = b$, and hence, as such, there does *not* exist $r > 0$ satisfying the requirements. In this case, suppose the convex set K is empty. Then the hyperplane $Ax = b$ is separated from the positive "orthant" (i.e., the region $x \geq 0$) by a finite width $r > 0$ (that is expressible in polynomially many bits). Then the idea is to draw a small tube around $Ax = b$, so that the system of equations becomes full dimensional.

   Slightly more precisely, there exists some $\epsilon > 0$ such that the new convex set $K' = \{-\epsilon \leq Ax - b \leq \epsilon \ : \ x \geq 0\}$ is empty provided K is empty. (See Figure 8.1 for an illustration.) On the other hand, if K is nonempty and $c \in K$, then the set K' contains a ball $B(c, r)$ for some finite $r > 0$. (See Figures 8.2.) Finally, we can show that the numbers ϵ and r are expressible in polynomially many bits. Thus we may provide r as the parameter.

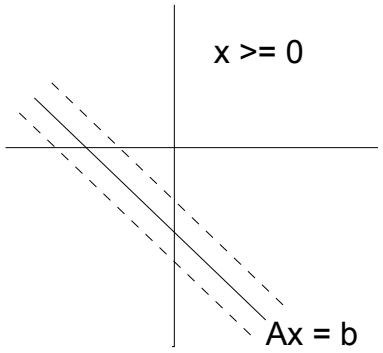


Figure 8.1: Handling the second requirement in the infeasible case. Note that there is a finite width $\epsilon > 0$ such that the region $-\epsilon \leq Ax - b \leq \epsilon$ does not intersect the positive orthant $x \geq 0$.

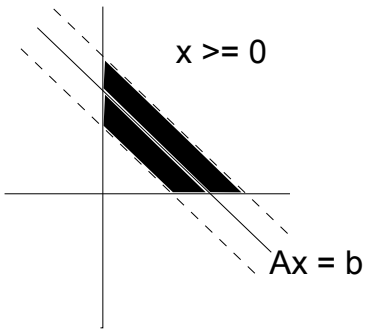


Figure 8.2: Handling the second requirement in the feasible case. Relaxing the LP to $-\epsilon \leq Ax - b \leq \epsilon$ makes the feasible set (the shaded region) full dimensional.

8.2.2 The algorithm

We now describe the ellipsoid algorithm. From now on, it is convenient to assume that the input LP has the form $Ax \leq b$ with $x \geq 0$.

If $n = 1$, then solve the LP directly and terminate; so without loss of generality, assume $n > 1$. The algorithm maintains an ellipsoid E that completely contains K (if K is nonempty). We initialize the algorithm with the ellipsoid $E(0, R^2 I)$ that was promised to satisfy this requirement.

1. Check if the center c is feasible (i.e., if $c \in K$). If so, then we are done.

2. Else, get a "separating hyperplane" through c; i.e., a hyperplane $a^T x = a^T c$ through c such that the set K is completely contained in the half-ellipsoid formed by the intersection of $a^T x \geq a^T c$ with the current ellipsoid. (To implement this step given an LP, we take a violated constraint $a^T x \leq \gamma$; then $a^T x = a^T c$ is a separating hyperplane through c.)

3. Take a smallest volume ellipsoid containing the half-ellipsoid which may contain K. Goto 1.

4. After $N = \text{poly}(n, \langle R \rangle, \langle r \rangle)$ iterations, stop and say $K = \emptyset$.

8.2.3 Analysis of the algorithm

Clearly, Step 1 of the algorithm is correct. Further, at every stage of the algorithm, we maintain the invariant that the convex set K is completely contained inside the current ellipsoid. Thus it only remains to show that if we terminate the algorithm after N steps and find no feasible point, then the set K is indeed empty.

Denote the ellipsoid at the end of iteration k by $E_k = E(c_k, Q_k)$ (the starting ellipsoid is $E_0 = E(0, R^2 I)$). We use the volume of ellipsoids E_k to track the number of iterations of the algorithm. The main claim is the following:

Theorem 8.8 (Volume reduction)**.** *For $k \geq 0$, we have*[4]*:*

$$\frac{\text{Vol}(E(c_{k+1}, Q_{k+1}))}{\text{Vol}(E(c_k, Q_k))} \leq e^{-\frac{1}{2(n+1)}}.$$

We prove this theorem in the next subsection.

Corollary 8.9. *1. After $2(n+1)$ steps, the volume goes down by a factor of $\frac{1}{e}$.*

2. Suppose $N > 2n(n+1)\ln(\frac{R}{r})$. Then after N iterations, the volume becomes smaller than $\text{Vol}(B(0, r))$.

[4]In class, we claimed the slightly stronger upper bound of $e^{-\frac{1}{2n}}$ with no proof. This only causes a constant factor increase in the number of iterations.

Proof. The first part is obvious. To see the second part, the volume of the ellipsoid at the end of N iterations is

$$\text{vol}(E_N) = \text{vol}(E_0)\exp\left(-\frac{N}{2(n+1)}\right) < \text{vol}(B(0,R))\exp\left(-n\ln\left(\frac{R}{r}\right)\right)$$

Rearranging,

$$\text{vol}(E_N) < R^n\text{vol}(B(0,1)) \times \frac{r^n}{R^n} = r^n\text{vol}(B(0,1)) = \text{vol}(B(0,r)) = \text{vol}(B(c,r)).$$

since all balls of the same radius have the same volume. It follows that $K \subseteq E_N$ has a volume strictly less than $\text{vol}(B(c,r))$. But this contradicts the second requirement of the algorithm, unless $K = \emptyset$. □

Thus if we did not find a feasible solution after N iterations, then we can safely terminate. Since N is polynomial in the length of the input, we have shown that this algorithm terminates in polynomial time.

But we are not done yet! Of course, we still need to prove Theorem 8.8. Also, to complete the description of the algorithm, we need to write down the smallest volume ellipsoid containing the half-ellipsoid that may have K. We do both of these in the next subsection.

8.2.4 The description of the half-ellipsoid: a simple case

We first deal with the simple case, where $E_0 = B(0,1)$ and the separating hyperplane is $a_0 = (-1,0,0,\ldots,0)$. Our goal is to find an ellipsoid E_1 that contains the region $E_0 \cap \{x : x_1 \geq 0\}$.

Lemma 8.10. *Define* $c_1 = (\frac{1}{n+1}, 0, 0, \ldots, 0)$, *and*

$$Q_1 = \frac{n^2}{n^2-1}\begin{pmatrix} 1-\frac{2}{n+1} & & & & \\ & 1 & & & 0 \\ & & 1 & & \\ & 0 & & \ddots & \\ & & & & 1 \end{pmatrix}$$

Then $E_1 = E(c_1, Q_1)$ *is the minimum volume ellipsoid containing the half-ball. Moreover,*

$$\frac{\text{vol}(E_1)}{\text{vol}(E_0)} \leq e^{-\frac{1}{2(n+1)}}.$$

Proof. In this notes, we only prove that the ellipsoid E_1 contains the desired half-ball and prove the bound on its volume. Although it is true that E_1 is the ellipsoid of *minimum volume*, we do not show that here. Note that this does not affect our algorithm or our analysis in any way.

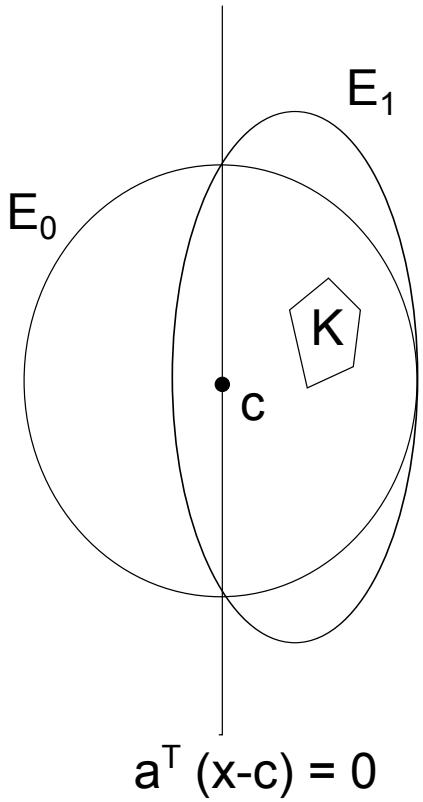


Figure 8.3: Ellipsoid E_1 covering the half-ellipsoid bounded by E_0 and the separating hyperplane $a^T(x-c)=0$.

Take any x in the half-ball; i.e., $\|x\| \le 1$ and $x_1 \ge 0$. Suppose $x = (x_1, \tilde{x})$ where $\tilde{x} = (x_2, x_3, \ldots, x_n)$. It is easy to verify that

$$Q_1^{-1} = \frac{n^2-1}{n^2}\begin{pmatrix} \frac{n+1}{n-1} & & & 0 \\ & 1 & & \\ 0 & & 1 & \\ & & & \ddots \end{pmatrix}.$$

Consider

$$
\begin{aligned}
x^T Q^{-1} x &= \frac{n^2-1}{n^2}\left(x_1 - \frac{1}{n+1}, \tilde{x}\right)^T \begin{bmatrix} \frac{n+1}{n-1} & & & \\ & 1 & & 0 \\ & & 1 & \\ & 0 & & \ddots \end{bmatrix} \left(x_1 - \frac{1}{n+1}, \tilde{x}\right) \\
&= \frac{n^2-1}{n^2}\frac{n+1}{n-1}\left(x_1 - \frac{1}{n+1}\right)^2 + \frac{(n^2-1)}{n^2}\|\tilde{x}\|^2 \\
&= \frac{1}{n^2}((n+1)x_1 - 1)^2 + \frac{(n^2-1)}{n^2}\|\tilde{x}\|^2 \\
&\overset{(a)}{\leq} \frac{1}{n^2}((n+1)x_1 - 1)^2 + \frac{(n^2-1)}{n^2}(1 - x_1^2) \\
&= \frac{(n+1)^2}{n^2}x_1^2 - 2\frac{n+1}{n^2}x_1 + \frac{1}{n^2} + \frac{n^2-1}{n^2} - \frac{n^2-1}{n^2}x_1^2 \\
&= \frac{2(n+1)}{n^2}(x_1^2 - x_1) + 1 \\
&\overset{(b)}{\leq} 1,
\end{aligned}
$$

where (a) follows from the fact that $x_1^2 + \|\tilde{x}\|^2 = \|x\|^2 \leq 1$, and (b) follows from the inequality $0 \leq x_1 \leq 1$. Therefore, the point x is inside the ellipsoid E_1.

The ratio of the volumes of the ellipsoids E_1 and E_0 is given by

$$
\frac{\mathrm{vol}(E_1)}{\mathrm{vol}(E_0)} = \sqrt{\det Q_1} = \sqrt{\left(\frac{n^2}{n^2-1}\right)^n \left(\frac{n-1}{n+1}\right)} = \sqrt{\left(\frac{n^2}{n^2-1}\right)^{n-1} \left(\frac{n}{n+1}\right)^2},
$$

after some rearrangement. Using the inequality $1 + z \leq e^z$ valid for all real z, we get

$$
\frac{\mathrm{vol}(E_1)}{\mathrm{vol}(E_0)} \leq \exp\left(\frac{n-1}{2} \cdot \frac{1}{n^2-1} - \frac{1}{n+1}\right) = \exp\left(-\frac{1}{2(n+1)}\right).
$$

□

8.2.5 The description of the ellipsoid: the general case

Suppose we have an ellipsoid $E_k = E(c_k, Q_k)$, and we have a separating hyperplane $a_k^T x = a_k^T c_k$ through the center c_k. Our goal is to compute the minimum volume ellipsoid E_{k+1} that contains the half-ellipsoid bounded by E_k and $a_k^T x \geq a_k^T c_k$.

By the definition of an ellipsoid, there exists some invertible affine transformation L^{-1} that takes E_k to $B(0,1)$ and a_k to $a = (-1, 0, 0, \ldots, 0)$. Thus we are back to the simple case in Subsection 8.2.4. Let E' be the ellipsoid just analyzed and take $E_{k+1} = LE'$. This clearly contains the half-ellipsoid. Further,

$$
\frac{\mathrm{vol}(E_{k+1})}{\mathrm{vol}(E_k)} = \frac{\mathrm{vol}(L(E'))}{\mathrm{vol}(L(B(0,1)))} = \frac{|\det L| \cdot \mathrm{vol}(E')}{|\det L| \cdot \mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(E')}{\mathrm{vol}(B(0,1))} \leq e^{-\frac{1}{2(n+1)}},
$$

by the previous analysis.

For implementation purposes, it is more desirable to describe the ellipsoid E_{k+1} more explicitly by computing the invertible transformation L. We will just state the final result without proof.

Claim 8.11. *The ellipsoid $E_{k+1} = (c_{k+1}, Q_{k+1})$ is given by: $c_{k+1} = c_k - \frac{1}{n+1}h$ and*

$$Q_{k+1} = \frac{n^2}{n^2-1}\left(Q_k - \frac{2}{n+1}hh^T\right).$$

where $h = \sqrt{a_k^T Q_k a_k}$.

Proof. Omitted. □

One final remark is in order about the correctness of our algorithm and its analysis. Note that the description of the half-ellipsoid relies on computing square roots. This makes the preceding analysis valid only for an idealized implementation assuming exact arithmetic. It is possible to handle this precision issues by keeping a good enough approximation of the real quantities using rational numbers. However, the full proof then becomes substantially more complicated. We will deal with this issue (partly) in the next lecture.

Lecture 9

More on Ellipsoid: Grötschel-Lovász-Schrijver theorems*

I ♡ László Lovász.

Ryan O'Donnell

We saw in the last lecture that the Ellipsoid Algorithm can solve the optimization problem

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

in time $\text{poly}(\langle A \rangle, \langle b \rangle, \langle c \rangle)$, where by $\langle z \rangle$ we mean the representation size of z. In proving this, we mostly disregarded numerical issues, such as how irrational numbers should be dealt with. In addition, we mostly stated the algorithm in terms of a general convex set K rather than just a polytope, but then we neglected the whole host of issues surrounding general convex sets. In this lecture, we will fill in the remaining details. Largely these are either numerical or conceptual issues that require great length to be treated formally. As a result, this lecture will be relatively informal; a more precise and complete treatment is provided by Grötschel, Lovász, and Shrijver in [GLS88].

9.1 LP runtime

One loose end that we would like to mention quickly first is that it seems we should be able to do without the dependence on $\langle b \rangle$ and $\langle c \rangle$ in the Ellipsoid runtime. In some sense, the "real complexity" of the optimization problem should only depend on the polytope, not the direction of optimization. This intuition was proven correct by É. Tardos in [Tar86], in which she showed that the Ellipsoid Algorithm can solve linear programs in $\text{poly}(\langle A \rangle)$ time, via a reduction to linear programs of $\text{poly}(\langle A \rangle)$ size. One consequence of this is that if, for example, A is a matrix whose entries are in $\{-1, 0, 1\}$, then solving the linear program can be done in strongly polynomial time. Matrices of this form do occur for many natural problems, max flow perhaps most prominent among them.

*Lecturer: Ryan O'Donnell. Scribe: John Wright.

9.2 Numerical details

We now treat the numerical issues that arise in the Ellipsoid Algorithm. First, we summarize the algorithm. Its task is to determine the feasibility (nonemptiness) of a convex set K in $\mathbb{R}^n$. It receives two inputs:

1. A radius $R \in \mathbb{Q}$ such that $K \subseteq B(0, R)$.
2. A positive $\epsilon \in \mathbb{Q}$.

It outputs one of two things:

1. A point $s \in K$.
2. An ellipsoid $E \supseteq K$ such that $\text{vol}(E) \leq \epsilon$.

The running time of the Ellipsoid Algorithm is $\text{poly}(n, \langle R \rangle, \langle \epsilon \rangle, \text{“}\langle K \rangle\text{”})$, where by “$\langle K \rangle$” is meant something to do with the size of K. When we're solving linear programs, this is $\langle A \rangle + \langle b \rangle + \langle c \rangle$; for more general convex programs, we'll return to this issue.

The algorithm is iterative. At each step, it maintains an ellipsoid $E(s, Q)$ with center s and matrix Q which contains K. The algorithm begins each step by testing whether s is in K. If so, then it outputs s. Otherwise, it (somehow) finds a vector a such that $a^\top x < a^\top s$ for all $x \in K$. This is the “separating hyperplane”. Using this, it performs an update $E(s, Q) \to E(s', Q')$ to a new ellipsoid with center s' and matrix Q', and then it starts this process all over again. The update rule for the new ellipsoid is:

$$s' = s - \frac{1}{n+1} \cdot \frac{1}{\sqrt{a^\top Q a}} \cdot Qa \tag{9.1}$$
$$Q' = \frac{n^2}{n^2-1}\left(Q - \frac{2}{n+1} \cdot \frac{Qaa^\top Q}{a^\top Q a}\right)$$

One important thing to note is the square root in the denominator of Equation (9.1). We will return to this shortly. The point of this update rule is to produce a new ellipsoid of significantly smaller volume which contains the half ellipsoid formed by intersecting $E(s, Q)$ with the set $\{x : a^\top x < a^\top s\}$. By the choice of a we know that this half ellipsoid contains K. The precise decrease in volume we proved in the last lecture was:

Theorem 9.1. $\text{vol}(E(s', Q'))/\text{vol}(E(s, Q)) \leq e^{-1/2n} \leq 1 - \frac{1}{3n}$.

Unfortunately, that square root means the possibility of irrational numbers, so that the Ellipsoid Algorithm couldn't update to $E(s', Q')$ even if it wanted to. To fix this, we'll need to show that if we perform approximate computations which are accurate to a sufficient degree, then everything comes out okay anyway. By necessity, this involves mucking around in some tedious numerical waters. To begin with, let N be the total number of iterations, which we will pin down later. Then we'll modify the Ellipsoid Algorithm so that, when doing computations, it rounds all numbers to precision 2^{-p}, where p is set to $100N$. This solves the irrational number problem, but introduces a new one: the rounding changes the

center of the updated ellipsoid, and this will cause it not to contain the half ellipsoid that it was supposed to cover. To compensate for this, we adjust the algorithm so that it "blows up" each ellipsoid by a factor of $(1 + \frac{1}{10n^2})$ in each dimension. This is sufficient to make the ellipsoids completely contain the exact ellipsoids they are approximating, but it again introduces a new problem, which is that we can no longer guarantee as strong a bound as in Theorem 9.1. Hopefully, we can still get a strong enough bound, similar to Theorem 9.1 but with, say, a $1 - \frac{1}{10n}$ rather than a $1 - \frac{1}{3n}$, and if this is the case then we can just set the number of iterations N to be something suitably large. In particular, taking

$$N := 10n\left(n\log(2R) + \log\left(\frac{1}{\epsilon}\right)\right) \tag{9.2}$$

is sufficient.

Why it's okay. We must ensure that the factor we have chosen to blow the ellipsoids up by does not blow them up too much. Doing this blow-up makes our approximating ellipsoid have $(1 + \frac{1}{10n^2})^n \approx (1 + \frac{1}{10n})$ times the volume of the exact ellipsoid. On the other hand, each exact ellipsoid has $(1 - \frac{1}{3n})$ times the volume of the previous ellipsoid. The net effect of these two opposing forces is that when we perform this rounding and blowing up, the updated ellipsoid has $(1 + \frac{1}{10n})(1 - \frac{1}{3n})$ times the volume of the original ellipsoid. This is no more than $(1 - \frac{1}{10n})$ times the volume of the original ellipsoid. The coefficient of 10 in Equation 9.2 is large enough so that the number of iterations yields a sufficiently small final ellipsoid.

There are some additional technical details that we need to attend to to ensure that everything still works after adding the rounding. For example,

Lemma 9.2. *In the exact version of the Ellipsoid Algorithm, let $B(s_k, Q_k)$ be the ellipsoid at step k, and let λ_k be the minimum eigenvalue of Q_k. Then $|s_k|, \|Q_k\|, 1/\lambda_k \leq R \cdot 2^{2k}$, for all k.*

In particular, all of these bounds are at most $R \cdot 2^{2N}$. This is the key fact used to show that the rounding errors do not get magnified too much when being multiplied by matrices. Specifically, we just need p to be large enough so that $2^{-p} \ll \frac{1}{R^2 \cdot 2^{2N}}$, and taking $p \approx \Theta(N)$ is sufficient.

9.3 Separation oracles

With that taken care of, we turn to applying the Ellipsoid Algorithm to more general convex sets. A difficulty that arises is that for a general convex set K, we may not have as convenient a description of it as we did in the case of sets of linear equations describing polytopes. Quickly glancing at the algorithm reveals that what we actually require from K is quite limited: we should be able to test whether $s \in K$ (so-called "membership testing"), and if it isn't, we should be able to find a separating hyperplane. This minimal interface is formalized below in the definition of a separation oracle.

Definition 9.3. A *strong separation oracle* for K, when given as input $s \in \mathbb{Q}^n$, either returns "$s \in K$", or $a \in \mathbb{Q}^n$ such that $a^\top x < a^\top s$ for all $x \in K$.

There's one more tiny condition that we enforce on these oracles, and it is that the separating hyperplane it returns should have a manageable bit complexity—polynomial in the relevant parameters. Without this constraint, one could design a malicious separation oracle which always returns separating hyperplanes of exponential bit complexity, and then there's nothing the Ellipsoid Algorithm can do, no matter how clever we are in designing it.

We've already constructed a separation oracle, implicitly, for polytopes in the Ellipsoid Algorithm for linear programs last lecture. Let's go through an example for a different type of convex set. Let K be the unit ball, i.e. $K = B(0, 1)$. Given $s \in \mathbb{Q}^n$, the separation oracle for K is

- If $\|s\|^2 \leq 1$, return "$s \in K$". Otherwise, return s.

That s is a good separating hyperplane is verified by the following simple calculation, which holds for $x \in K$:

$$s^\top x \leq \|s\| \cdot \|x\| \leq \|s\| < \|s\|^2 = s^\top s.$$

The first inequality is by Cauchy-Schwarz, the second is because $\|x\| \leq 1$, and the third is because $\|s\| > 1$.

Implementing the separation oracle is easy for the unit ball, but it may not be quite so easy for some less explicit convex set. Indeed, why would you want to run the Ellipsoid Algorithm on a general convex set? If you could compute the separation oracle, you might well be able to compute feasibility already. Well, we'd like to maximize a linear function over the set, the problem

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x \in K. \end{aligned}$$

Recall that to solve this using a feasibility tester such as the Ellipsoid Algorithm, we do a binary search over the possible optimum values γ, at each step of the search testing feasibility of the region $K' := K \cap \{x : c^\top x \geq \gamma\}$. Say that we were given SEP_K, a separation oracle for K. We would somehow also need to design a separation oracle for K'. Fortunately, this quite easy. Given $s \in \mathbb{Q}^n$, the separation oracle for K' is

- If $c^\top s < \gamma$, return $-c$. Otherwise, return $\text{SEP}_K(s)$.

9.4 General convex sets

This looks promising for optimizing a linear function over K, but when should we stop the binary search? With a linear program, we could stop in a fixed number of steps because we knew the optimal was a rational of bounded repesentation size. For a general K, on the other hand, it is possible, for instance, that the optimum is irrational, meaning that we have no hope of finding it. Examples of this aren't too hard to cook up. Consider

$$\begin{aligned} \max \quad & x + y \\ \text{s.t.} \quad & x^2 + 2y^2 \leq 1 \end{aligned}$$

The optimum is $\sqrt{3/2}$.

The issues aren't specific to the binary search either; the Ellipsoid Algorithm simply cannot test feasibility for general convex sets. The Ellipsoid Algorithm works by generating a series of smaller and smaller ellipsoids which contain K. In the case of linear programs, we were able to guarantee that either the associated polytope was infeasible, or it contained a ball of large enough radius. Thus, when the ellipsoids reach a certain small size, we can guarantee that their centers must be contained in the polytope, if it exists. But we cannot make a similar guarantee for a general convex set. In the extreme, the set may be a single point, in which case the Ellipsoid Algorithm can never hone in on K completely.

Finally, even if the set K starts out large, it is possible that the Ellipsoid Algorithm nonetheless has to determine feasibility of extremely small regions, due to the binary search. It could be that the convex set $K \cap \{c^\top x \geq \gamma\}$ is quite small. This is illustrated in Figure 9.1, where the binary search reduces the region the size of the feasible region.

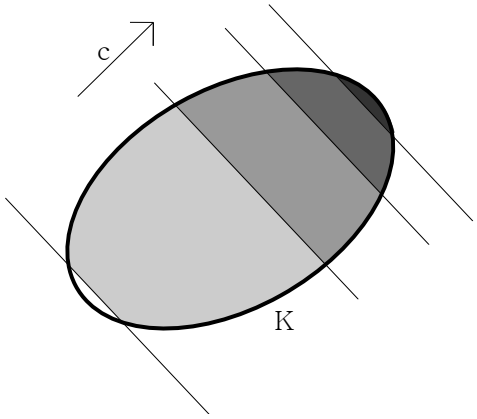


Figure 9.1: Binary search on the region K

At this point, we must either give up all hope of optimizing over general convex sets or define a relaxed notion of what it means to optimize (we'll opt for the latter). The relaxed notion of optimization involves demanding that the optimization only be approximate, in two separate ways. First, we saw that one of the two issues with optimizing over general K is that K may be too small. To rectify this, we introduce two approximate versions of K,

$$K^{+\epsilon} := \{y : \|y - x\| \leq \epsilon \quad \text{for some } x \in K\}$$

$$\text{and} \quad K^{-\epsilon} := \{x : B(x, \epsilon) \subseteq K\}.$$

Think of $K^{+\epsilon}$ as the set of points "almost in K", and $K^{-\epsilon}$ as the set of points "deep in K". Note that if K is not full dimensional, then $K^{-\epsilon}$ is empty. In our relaxed form of approximation, we will only require the algorithm to optimize with respect to these sets.

The other area where we must allow for some approximation is in the objective value of the solution. Instead of requiring that it be optimal, we require only that it be near-optimal. With these notions in place, we can now state the following key definition:

Definition 9.4. The task of *weak optimization* of

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & x \in K. \end{aligned}$$

is, when given a positive $\epsilon \in \mathbb{Q}$, to

- Either output "$K^{-\epsilon}$" is empty;
- Or, find a $y \in K^{+\epsilon}$ such that $c^\top x \leq c^\top y + \epsilon$ for all $x \in K^{-\epsilon}$.

As stated before, we may never find a point exactly in K, or a point which optimizes exactly in K. So we must settle for weak optimization. One benefit of this is that we will only need a "weak" separation oracle to carry out weak optimization.

Definition 9.5. A *weak separation oracle* for K, when given as input $s \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, asserts "$s \in K^{+\delta}$" if this is true, and otherwise returns $a \in \mathbb{Q}^n$ such that $\|a\|_\infty = 1$ and $a^\top x \leq a^\top s + \delta$ for all x in $K^{-\delta}$.

The constraint on the infinity norm of a is needed, because without it, setting $a := 0$ would always satisfy the approximate separating hyperplane constraint. Why can we now get away with using a weak separation oracle? Well, we're to have precision errors in the Ellipsoid Algorithm anyway, so a slight lack of precision here is ($\delta := 2^{-p}$) will not matter. With this in place, we are now ready to state the following general theorem:

Theorem 9.6. *Given R and a weak separation oracle for a convex $K \subseteq B(0, R)$, we can weakly optimize over K in time* $\text{poly}(n, \langle R \rangle, \langle \epsilon \rangle)$.

As a preview of things to come, we will use this to solve semidefinite programs in the next lecture.

9.5 Even more theorems

The remainder of this lecture will focus on a couple of the other results from [GLS88]. These are quite difficult to prove, so the discussion will be of a lighter nature.

9.5.1 Membership oracles

To start with, say you don't have a separation oracle for K, just a membership oracle. Recall that a membership oracle can test $s \in K$. This is a much more natural concept than a separation oracle, and it would be peachy if it were sufficient. Can we optimize over K, or even perform feasibility testing? Unfortunately, the answer is no. With only a membership oracle it's easy to see that an "adversarial set" $\tilde{K}$ can "evades" any points you query; you may never even "find" the set. However, say that in addition to a membership oracle, you are given a small ball that inhabits K: a point $s_0 \in \mathbb{Q}^n$ and a radius $r > 0$ such that $B(s_0, r) \subseteq K$. Then the following theorem says that this is sufficient to weakly optimize.

Theorem 9.7. *[YN76, GLS88] Given positive R, $r \in \mathbb{Q}$, a point $s_0 \in \mathbb{Q}^n$ such that $B(s_0, r) \subseteq K \subseteq B(s_0, R)$ and a weak membership oracle for K, then one can weakly optimize over K in time* $\text{poly}(n, \langle R \rangle, \langle r \rangle, \langle \epsilon \rangle, \langle s_0 \rangle)$.

(Here is the definition of a weak membership oracle: when given $s \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, it reports either that $s \in K^{+\delta}$ or $s \notin K^{-\delta}$.)

Ryan was not completely positive on how the proof of this goes, but here is what he thinks is the general outline. Basically, you want to use the weak membership oracle and the small ball of radius r to implement some sort of separation oracle. Then you can appeal to Theorem 9.6. Remember that a separation oracle, given a point s, does a membership test and, if that fails, returns a separating hyperplane. We have a membership oracle, so the membership test is easy, but what to do for the separating hyperplane is a little more complicated.

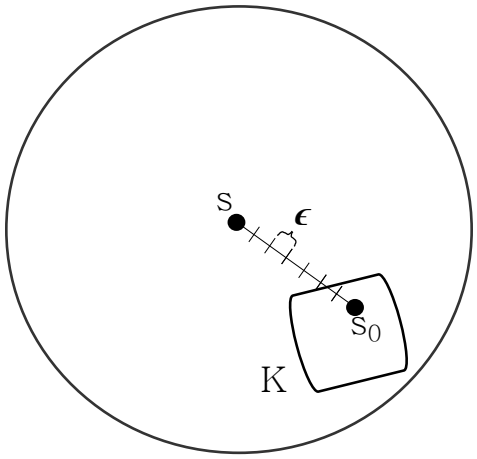


Figure 9.2: Crossing the boundary of K.

In this case, we know (roughly) that $s \notin K$, and we also that the point s_0 is in K. Thus, if we draw a line segment from s to s_0, there is a point on the line segment where the line segment crosses the boundary of K. If we sample points on the line at intervals of roughly $1/\epsilon$ from each other, as in Figure 9.2, we can find "adjacent" points on the line which lie on either side of the boundary of K. Now, if we test a lot of directions in a small ball around these points, we can find what is roughly the the tangent to the boundary, and we can use this direction as the separating hyperplane. The main problem with this is that it runs in time $\text{poly}(1/\epsilon)$ rather than $\text{poly}(\langle \epsilon \rangle)$, but there is a variant of the Ellipsoid Algorithm ("Shallow Cut") for which this is sufficient.

9.5.2 General convex functions

A natural question is whether the functions we optimize must be linear, or whether they can be more general. Indeed, the results hold for any *convex* function $f : \mathbb{R}^n \to \mathbb{R}$. Recall that a convex function is one in which for all points $x, y \in \mathbb{R}^n$ and $t \in [0, 1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

As always, we have to worry about what interface we have with f: an *oracle* for f, when given $x \in \mathbb{Q}^n$ and a positive $\delta \in \mathbb{Q}$, outputs a $y \in \mathbb{Q}$ such that $|f(x) - y| \leq \delta$. Furthermore, the oracle should run in time $\text{poly}(\langle x \rangle, \langle \delta \rangle)$. Given this, we have the following theorem:

Theorem 9.8. *Let f be a convex function. Given positive R, $r \in \mathbb{Q}$, a point $s_0 \in \mathbb{Q}^n$ such that $B(s_0, r) \subseteq K \subseteq B(s_0, R)$ and a weak membership oracle for K, then one can weakly optimize f over K in time* $\text{poly}(n, \langle R \rangle, \langle r \rangle, \langle \epsilon \rangle, \langle s_0 \rangle)$.

We can even give a short "proof" of this (omitting a few easy-to-work-out details), by converting it into the case covered by Theorem 9.7. First, note that the region $L := \{(x, \gamma) \in \mathbb{R}^{n+1} : f(x) \leq \gamma\}$ is convex. Then weakly optimizing f over K is equivalent to weakly optimizing the following program:

$$\begin{aligned} \max \quad & 0 \cdot x + (-1) \cdot \gamma \\ \text{s.t.} \quad & x \in K, \quad (x, \gamma) \in L, \end{aligned}$$

which is just optimizing a linear constraint over a convex region.

9.5.3 Solving large LPs

For our last topic, we'll cover the well-known case of solving linear programs with exponentially many constraints (and only a separation oracle). Why is this any different from what we have already covered? The biggest problem is that the linear program may not be full dimensional. For example, in the following figure, we see centers s_i drawing nearer to the region K and the ellipsoids growing taller and flatter, but they never actually find K.

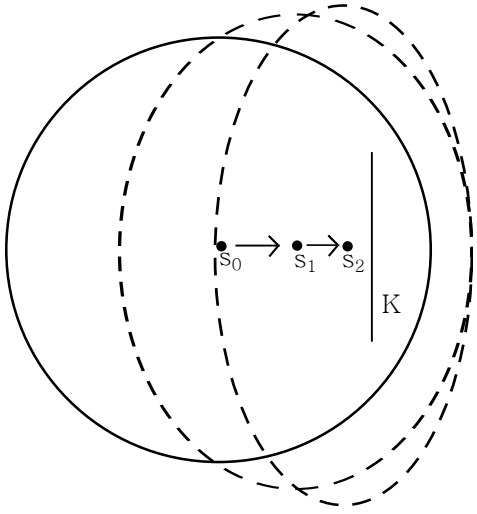


Figure 9.3: Ellipsoids closing in on a region K not of full dimension.

This difficulty is quite inherent and cannot be easily evaded. How could one try to resolve it? Well, the ellipsoids are getting really flat, so they seem to identify a lower dimensional subspace. So if we could identify this subspace and "jump" into it, we could continue testing

for feasibility of K in a subspace in which it is full-dimensional. This is indeed the idea behind the resolution of the problem. How to do this "subspace identification" is a difficult matter and it involves something called "simultaneous diophantine approximation". The scenario in simultaneous diophantine approximation is that there's a list of numbers, all almost expressible as integers over a certain bounded denominator, and the task is to find this denominator. This problem can be approximately solved by the so-called LLL algorithm, created by Lenstra, Lenstra, and Lovász. (The third major contribution in the area involving Lovász!)

The wonderful result of this is that we are able to get the following your-wildest-dreams-can-come-true theorem for large linear programs.

Theorem 9.9. *Let K be $\{x : Ax \leq b\} \subset \mathbb{R}^n$, with each inequality $a^\top x \leq b$ satisfying $\langle a \rangle, \langle b \rangle \leq l$. (The number of inequalities here must be finite.) Assume access to a strong separation oracle for K. Then in time* $\text{poly}(n, l)$*, you can:*

- *Perform feasibility/unboundedness testing.*
- *Perform exact optimization of primal and dual.*
- *Find an optimal vertex.*
- *etc.*

In essence, with this result, anything you can think of, you can do. The one downside is that in exchange for this, the proof is really hard.

Wrap-up. We've seen that the Ellipsoid Algorithm is capable of solving a diversity of convex optimization problems. In spite of its being relatively inefficient in practice, it has great theoretical value. Next lecture, we will apply the Ellipsoid Algorithm to solve semidefinite programs, an important class of optimization problems.

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1

Lecture 10

Semidefinite Programs and the Max-Cut Problem*

In this class we will finally introduce the content from the second half of the course title, Semidefinite Programs. We will first motivate the discussion of SDP with the Max-Cut problem, then we will formally define SDPs and introduce ways to solve them.

10.1 Max Cut

Let $G = (V, E)$ with edge weights $w_e > 0$ for all $e \in E$ where $\sum_{e \in E} w_e = 1$. Our goal is to maximize $\sum_{e \in \partial(A)} w_e$ over $A \subseteq V$, or equivalently maximize $\sum_{uv \in E} w_{uv} \mathbb{1}[A(u) \neq A(v)]$ over functions $A : V \to \{0, 1\}$.

Remark 10.1. Couple quick observations:

- $Opt = 1 \Leftrightarrow G$ is bipartite
- $Opt \geq \frac{1}{2}$

  Proof. (Algorithmic) Pick $A : V \to \{0, 1\}$ at random.

$$\begin{aligned}
\mathbf{E}\left[\text{cut val}\right] &= \mathbf{E}\left[\sum_{uv \in E} w_{uv} \mathbb{1}\left[A(u) \neq A(v)\right]\right] \\
&= \sum_{uv \in E} w_{uv} \mathbf{Pr}[A(u) \neq A(v)] \\
&= \sum_{uv \in E} w_{uv} \frac{1}{2} \\
&= \frac{1}{2}
\end{aligned}$$

□

*Lecturer: Ryan O'Donnell. Scribe: Franklin Ta.

- G complete $\Rightarrow Opt = \frac{1}{2} + \frac{1}{2(n-1)}$
- Max-cut is NP-hard.

Integer Programming Formulation Now lets look at the IP formulation for Max-Cut. For $\{x_v\}_{v\in V}, \{z_e\}_{e\in E} \in \{0,1\}$,

$$\begin{aligned} \max \quad & \sum_{uv\in E} w_{uv} z_{uv} \\ \text{s.t.} \quad & z_{uv} \leq x_u + x_v \\ & z_{uv} \leq 2 - (x_u + x_v) \end{aligned}$$

Where x encodes which partition the vertex is in, and z encodes whether the edge is cut. To see why this works, suppose that $x_u \neq x_v$, then $z_{uv} \leq 1, z_{uv} \leq 1$, so $z_{uv} = 1$. Otherwise, suppose $x_u = x_v$, then $z_{uv} \leq 2, z_{uv} \leq 0$, so $z_{uv} = 0$.

Linear Programming Relaxation To get the LP relaxation, just let x_v, $z_e \in [0,1]$. But unfortunately, this LP relaxation isn't very good. Set $x_v = \frac{1}{2}$ for all v, then $z_e = 1$ for all e, which makes $LPOpt = 1$ for all graph G.

Another Formulation Seeing that didn't work, let's try another formulation. For $y_v \in \mathbb{R}$,

$$\begin{aligned} \max & \sum_{uv\in E} w_{uv} \left(\frac{1}{2} - \frac{1}{2} y_u y_v\right) \\ \text{s.t.} \quad & y_v y_v = 1 \quad \forall v \in V \end{aligned}$$

Here we changed our indicator, y_v, to use 1 or -1 to encode which partition we are in. Note that if $y_u = y_v$, then $\left(\frac{1}{2} - \frac{1}{2} y_u y_v\right) = 0$, and if $y_u \neq y_v$, then $\left(\frac{1}{2} - \frac{1}{2} y_u y_v\right) = 1$.

SDP Relaxation[Delorme Poljak '90] Note that the previous formulation is still exactly Max-Cut (which is NP-hard) so we won't be able to solve it. So to relax it, we can allow $\vec{y}_v \in \mathbb{R}^d$: (this is the 'SDP')

$$\begin{aligned} \max & \sum_{uv\in E} w_{uv} \left(\frac{1}{2} - \frac{1}{2} \vec{y}_u \cdot \vec{y}_v\right) \\ \text{s.t.} \quad & \vec{y}_v \cdot \vec{y}_v = 1 \quad \forall v \in V \end{aligned}$$

To visualize what this SDP is doing, note that since $||\vec{y}_v|| = 1$, it is possible to embed $\{y_v\}_{v\in V}$ onto a unit sphere:

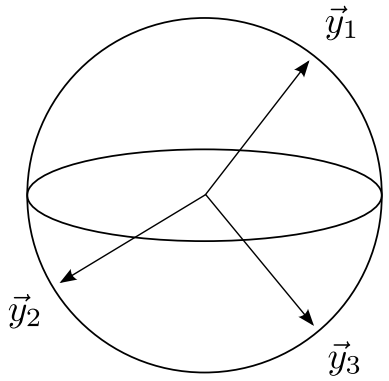


Figure 10.1: Vectors $\vec{y}_v$ embedded onto a unit sphere in $\mathbb{R}^d$.

Denote $\sigma_{uv} = \vec{y}_u \cdot \vec{y}_v = \cos(\angle(\vec{y}_u, \vec{y}_v))$. Then for $(u, v) \in E$, we want $\sigma_{uv} \approx -1$ as possible since this will maximize the sum. Translating to the figure above, we want to have vectors pointing away from each other as much as possible. Also note that if d is 1, the solution is exact, so $SDPOpt \geq Opt$. Also note that $d \leq n$ in general.

Example 10.2. Let's see some examples of how this SDP compares with Opt.

Let G be K_3. Clearly we can embed $\vec{y}_1, \vec{y}_2, \vec{y}_3$ 120 degrees apart in the unit circle in $\mathbb{R}^2$, so we get:

$$\begin{aligned} SDPOpt(G) &\geq \sum_{uv \in E} w_{uv} \left(\frac{1}{2} - \frac{1}{2} \vec{y}_u \vec{y}_v \right) \\ &= \sum_{i=1}^{3} \frac{1}{3} \left(\frac{1}{2} - \frac{1}{2} \cos\left(\frac{2\pi}{3} \right) \right) \\ &= \frac{3}{4} \end{aligned}$$

This bound can be shown to be tight, so $SDPOpt(G) = \frac{3}{4}$. It can also be shown that $Opt(G) = \frac{2}{3}$ and $LPOpt(G) = 1$. Thus $\frac{8}{9} SDPOpt(G) = Opt(G)$.

Another example is G being C_5. In this case we have $Opt(G) = .88\, SDPOpt(G)$

Remark 10.3. Ellipsoid algorithm can (weakly) solve SDP in polytime. So assume you can find optimal $\vec{y}_v$.

Randomized Rounding [Goemans Williamson '94] At this point, we know we can relax Max-Cut into a SDP, and we know we can solve SDPs, but we still need to somehow convert that back to a solution for Max-Cut. We can do this using Goemans-Williamson randomized rounding:

We want to cut the vectors $\{y_v\}_{v \in V}$ with a random hyperplane through zero where everything on one side of the hyperplane is in one partition, and everything on the other side of the hyperplane is in the other. We do this by choosing $\vec{g} \in \mathbb{R}^d \setminus \{0\}$ (where $\vec{g}$ is normal

to hyperplane) from any rotationally symmetric distribution. Then set $y_u = \text{sgn}(\vec{g} \cdot \vec{y}_u) \in \{-1, 1\}$.

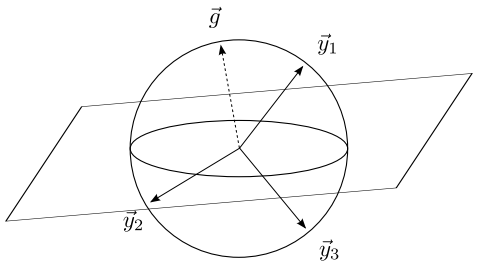


Figure 10.2: Vectors being separated by a hyperplane with normal $\vec{g}$.

Analysis of the rounding Fix $(u, v) \in E$. Then the probability that an edge (u, v) gets cut is the same as the probability that the hyperplane splits $\vec{y}_u$ and $\vec{y}_v$.

So consider just the 2-D plane containing $\vec{y}_u, \vec{y}_v$. Since the hyperplane was chosen from a rotationally symmetric distribution, the probability that the hyperplane cuts these two vectors is the same as the probability that a random diameter lies in between the angle θ of the two vectors.

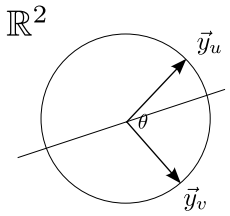


Figure 10.3: The plane of two vectors being cut by the hyperplane

Thus:

$$\begin{aligned}
\mathbf{Pr}[(u, v) \text{ gets cut}] &= \frac{\theta}{\pi} \\
&= \frac{\cos^{-1}(\vec{y}_u \cdot \vec{y}_v)}{\pi} \\
&= \frac{\cos^{-1}(\sigma_{uv})}{\pi} \\
\mathbf{E}[\text{cut val}] &= \sum_{uv \in E} w_{uv} \frac{\cos^{-1}(\sigma_{uv})}{\pi}
\end{aligned}$$

Now recall that $SDPOpt = \sum_{uv \in E} w_{uv}(\frac{1}{2} - \frac{1}{2}\sigma_{uv}) \geq Opt$.
So if we find α such that

$$\frac{\cos^{-1}(\sigma_{uv})}{\pi} \geq \alpha \left(\frac{1}{2} - \frac{1}{2}\sigma_{uv}\right) \quad \forall \sigma_{uv} \in [-1, 1]$$

then we can conclude $\mathbf{E}[\text{cut val}] \geq \alpha\, SDPOpt \geq \alpha Opt$

Plotting the above,

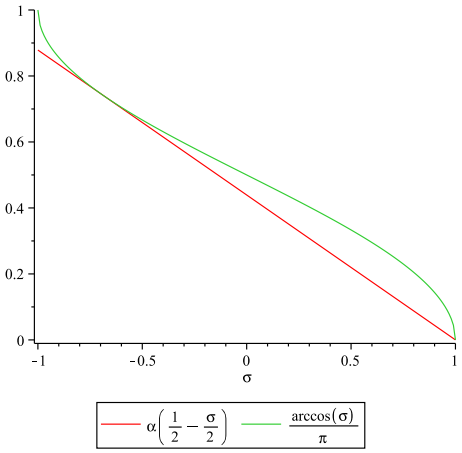


Figure 10.4: $\alpha\left(\frac{1}{2} - \frac{1}{2}\sigma\right)$ vs $\frac{\cos^{-1}(\sigma_{uv})}{\pi}$

we see that $\alpha = .87856...$ works.

Theorem 10.4. $\mathbf{E}[\textit{Goemans Williamson cut}] \geq .87856\, SDPOpt \geq .87856 Opt$

Note that this gives a polytime ".878-factor approximation algorithm for Max-Cut"

10.2 Semidefinite Programming

10.2.1 Positive Semidefinite Matrices

Theorem 10.5. *$S \in \mathbb{R}^{n\times n}$ symmetric is positive semidefinite (P.S.D.) iff equivalently*

1. *$S = Y^\top Y$ for some $Y \in \mathbb{R}^{d\times n}, d \leq n$.*
2. *S's eigenvalues are all greater than or equal to 0.*
3. *$x^\top Sx = \sum_{i,j} x_i x_j s_{ij} \geq 0$ for all $x \in \mathbb{R}^n$.*
4. *$S = LDL^\top$ where D diagonal, $D \geq 0$, L unit lower-triangular (i.e., $L = \begin{pmatrix} 1 & & & & 0 \\ * & 1 & & & \\ * & * & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ * & * & \dots & * & 1 \end{pmatrix}$).*
5. *There exist joint real random variables $Z_1, ..., Z_n$ such that $\mathbf{E}[Z_i Z_j] = s_{ij}$*
6. *$(S \in \mathbb{R}^{n\times n}) \in$ convex-hull of $\{xx^\top : x \in \mathbb{R}^n\}$*

Remark 10.6. 3 and 6 from Theorem 10.5 implies $\{S \in \mathbb{R}^{n^2} : S \text{ PSD}\}$ is convex

Remark 10.7. Recall the SDP of Max-Cut is:

$$\begin{aligned} \max \sum_{uv \in E} w_{uv} \left(\frac{1}{2} - \frac{1}{2} \sigma_{uv} \right) \\ \text{s.t.} \quad \sigma_{uv} = \vec{y}_u \cdot \vec{y}_v \\ \sigma_{vv} = 1 \quad \forall v \in V \end{aligned}$$

Thus,

$$\begin{aligned} \exists Y = \begin{pmatrix} | & | & & | \\ \vec{y}_{v_1} & \vec{y}_{v_2} & \cdots & \vec{y}_{v_n} \\ | & | & & | \end{pmatrix} \\ \text{s.t.} \quad (\sigma_{uv})_{u,v \in V} = YY^\top \end{aligned}$$

$\Leftrightarrow$ matrix $S = (\sigma_{uv})$ is PSD by Theorem 10.5.1.

Definition 10.8. A *semidefinite program* is an LP over n^2 variables σ_{ij} with extra constraints $S := (\sigma_{ij})$ is PSD. (This is really $\frac{n(n+1)}{2}$ variables since it is symmetric)

Theorem 10.9. *Omitting technical conditions SDP can be solved in polytime.*

10.2.2 Strong Separation Oracle for PSDness

Given symmetric matrix $S \in \mathbb{Q}^{n \times n}$, we want to assert S is PSD or find $x \in \mathbb{Q}^n$ s.t. $x^\top S x < 0$.

Idea: Compute smallest eigenvalue of S. If it is greater than or equal to 0, then S is PSD, otherwise we can use the corresponding eigenvector z to show that $z^\top S z < 0$.

Theorem 10.10. $\exists$ *Strongly polytime algorithm such that if S PSD, returns $S = LDL^\top$, and if S not PSD, returns x s.t. $x^\top S x < 0$. $(L, D, x \in \mathbb{Q})$*

Bonus: Since can compute $\sqrt{D}$ to any accuracy (square root each term in D) and $S = Y^\top Y$, we compute $Y = \sqrt{D} L^\top$. Where columns of Y are "vectors".

Proof. Do (symmetric) Gaussian Elimination on S:

$$S = \begin{pmatrix} * & * & \dots & * \\ * & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \dots & * \end{pmatrix}$$

Clear out first column with L_1 (which adds multiples of first row to other rows).

$$\underbrace{\begin{pmatrix} 1 & & & 0 \\ * & 1 & & \\ \vdots & & \ddots & \\ * & & & 1 \end{pmatrix}}_{L_1} S = \begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

Since symmetric, clear out first row with $L_1^\top$.

$$L_1 S L_1^\top = \begin{pmatrix} * & 0 & \dots & 0 \\ 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * \end{pmatrix}$$

Then clear out second row and column with L_2 and $L_2^\top$, and so on.

$$L_2 L_1 S L_1^\top L_2^\top = \begin{pmatrix} * & 0 & 0 & \dots & 0 \\ 0 & * & 0 & \dots & 0 \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \dots & * \end{pmatrix}$$

$$\vdots$$

$$\underbrace{L_n \dots L_2 L_1}_{L} S \underbrace{L_1^\top L_2^\top \dots L_n^\top}_{L^\top} = D$$

$$L S L^\top = D$$

$$S = L^{-1} D L^{-\top}$$

This will run to completion unless we hit the following cases:

If you just finished pivoting with a negative number, stop and output not PSD:

$$LSL^\top = \begin{pmatrix} * & 0 & 0 & \dots & 0 \\ 0 & -a & 0 & \dots & 0 \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \dots & * \end{pmatrix}$$

$$\begin{aligned} e_i^\top LSL^\top e_i &= e_i^\top \begin{pmatrix} * & 0 & 0 & \dots & 0 \\ 0 & -a & 0 & \dots & 0 \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \dots & * \end{pmatrix} e_i \\ &= -a \\ &< 0 \end{aligned}$$

Output $x = L^\top e_i$

If pivot is zero, and rows/cols of that pivot is not zero, stop and output not PSD:

$$LSL^\top = \begin{pmatrix} & & \\ 0 & \dots & b \\ \vdots & & \vdots \\ b & \dots & c \\ & & \end{pmatrix}$$

$$\begin{aligned} \begin{pmatrix} c & \dots & -b \end{pmatrix} LSL^\top \begin{pmatrix} c \\ \vdots \\ -b \end{pmatrix} &= \begin{pmatrix} c & \dots & -b \end{pmatrix} \begin{pmatrix} & & \\ 0 & \dots & b \\ \vdots & & \vdots \\ b & \dots & c \\ & & \end{pmatrix} \begin{pmatrix} c \\ \vdots \\ -b \end{pmatrix} \\ &= -cb^2 \\ &\leq 0 \end{aligned}$$

If $c \neq 0$, output $x = L^\top \begin{pmatrix} c \\ \vdots \\ -b \end{pmatrix}$, else output $x = L^\top \begin{pmatrix} 1 \\ \vdots \\ -b \end{pmatrix}$.

In all other cases we can run to completion and output $S = L^{-1} D L^{-\top}$

□

Lecture 11

The Lovász ϑ Function*

11.1 Perfect graphs

We begin with some background on perfect graphs. First, we define some quantities on graphs.

Definition 11.1. Given a graph G on n vertices, we define the following quantities:

1. The *clique number* of G, written as $\omega(G)$, is the size of the largest clique in G.
2. The *independence number* of G, written as $\alpha(G)$, is the size of the largest independent set in G.
3. The *chromatic number* of G, written as $\chi(G)$, is the minimum number of colors required to properly color G.
4. The *clique cover number* of G, written as $\bar{\chi}(G)$, is the size of the smallest clique cover in G, which is the minimum number of vertex disjoint cliques such that every vertex is in some clique.

Recall that the complement of a graph G, denoted $\bar{G}$, is the graph on the same vertices as G such that two vertices are connected in $\bar{G}$ if and only if they are not connected in G.

The following facts will be useful:

1. $\alpha(G) = \omega(\bar{G})$
2. $\chi(G) = \bar{\chi}(\bar{G})$
3. $\omega(G) \leq \chi(G)$
4. $\alpha(G) \leq \bar{\chi}(G)$
5. $\alpha(G)\chi(G) \geq n$

*Lecturer: Anupam Gupta. Scribe: David Witmer.

The last fact holds because each color class is an independent set.

Now we give the definition of a perfect graph, first stated by Berge.

Definition 11.2. A graph G is *perfect* if $\omega(G') = \chi(G')$ for all vertex-induced subgraphs G' of G.

Example 11.3. Consider the 5-cycle C_5, shown in Figure 11.1. C_5 is its own complement, so have the following values for the quantities defined above:

$$\omega(C_5) = 2$$
$$\alpha(C_5) = 2$$
$$\chi(C_5) = 3$$
$$\bar{\chi}(C_5) = 3$$

C_5 is the smallest non-perfect graph.

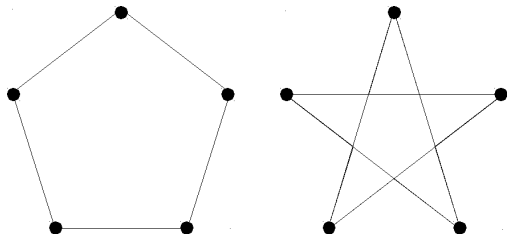


Figure 11.1: C_5 and $\bar{C_5}$. Note that C_5 is isomorphic to $\bar{C_5}$.

Bipartite graphs For any bipartite graph G, $\omega(G) = 2$ and $\chi(G) = 2$. Let $\mathrm{VC}(G)$ be the size of the minimum vertex cover of G. Then $\alpha(G) = n - \mathrm{VC}(G)$. By König's Theorem, this is equal to $n - \mathrm{MM}(G)$, where $\mathrm{MM}(G)$ is equal to the size of the maximum matching in G. In general, α is a lower bound for $\bar{\chi}$, but in this case, the two are equal. To see this, consider a clique cover of G consisting of 2-cliques corresponding to each edge of a maximum matching and 1-cliques for all remaining vertices as shown in Figure 11.2. The number of vertices not covered by the edges of the maximum matching is $n - 2\mathrm{MM}(G)$, so the number of cliques in this cover is $\mathrm{MM}(G) + (n - 2\mathrm{MM}(G)) = n - \mathrm{MM}(G)$. Then it must be true that $\bar{\chi}(G) \leq n - \mathrm{MM}(G)$, which, in turn, implies that $\alpha(G) = \bar{\chi}(G)$. This shows that both bipartite graphs and their complements are perfect.

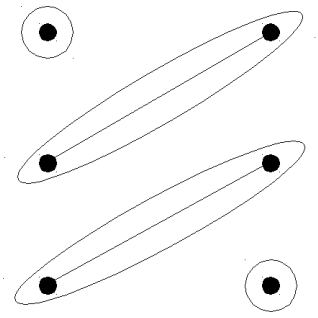


Figure 11.2: The clique cover corresponding to a matching of a bipartite graph.

Line graphs of bipartite graphs Recall that the line graph $L(G)$ of a graph G is the graph such that there is a vertex in $L(G)$ for each edge of G and two vertices of $L(G)$ are connected by an edge if and only their corresponding edges in G have a common endpoint. If G is bipartite, then $\omega(L(G))$ and $\chi(L(G))$ are both equal to the maximum degree of the vertices in G. In addition, $\alpha(L(G)) = \text{MM}(G)$ and $\bar{\chi}(L(G)) = \text{VC}(G)$. By König's Theorem, $\bar{\chi}(L(G)) = \text{MM}(G)$. Thus, line graphs of bipartite graphs and their complements are perfect.

Chordal graphs and interval graphs A chordal graph is a graph such that in every cycle of length four or more, there is an edge connecting two nonadjacent vertices of the cycle. Consider a set of intervals on the real line. The corresponding interval graph has a vertex for each interval and an edge between two vertices if the intersection of their intervals is nonempty. The set of interval graphs is a subset of the set of chordal graphs. An example of an interval graph is shown in Figure 11.3. Chordal graphs and their complements are perfect.

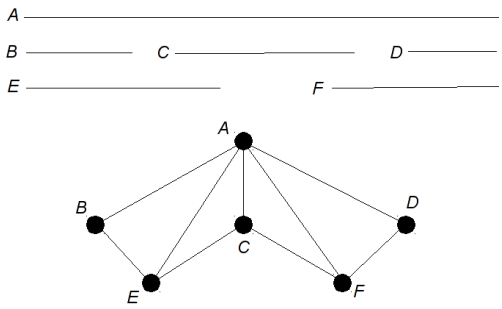


Figure 11.3: A set of intervals and the corresponding interval graph.

Comparability graphs Consider a partially ordered set P. The comparability graph of P is the graph with a vertex for each element of P and an edge between two elements if and only if their corresponding elements p and q in the partially ordered set are comparable ($p < q$ or $p > q$). Each clique in the comparability graph corresponds to a chain in the partially ordered set, and each independent set corresponds to an antichain. Let G be a

comparability graph. Then $\omega(G) = \chi(G)$. Consider the following coloring scheme: Choose a maximal antichain and color all of its elements one color. Remove these elements and continue inductively. Each time we remove a maximal antichain, the length of each maximal chain decreases by one, so $\omega(G)$ colors suffice. Since $\omega(G)$ is a lower bound for $\chi(G)$, we have equality. Also, $\alpha(G) = \bar{\chi}(G)$. Consider a maximal antichain. We can form a clique cover by taking the maximal chains containing the element of the antichain. Since $\alpha(G) \leq \bar{\chi}(G)$, the two quantities must be equal. Therefore, comparability graphs and their complements must be perfect.

For each of these classes of graphs, we see that their complements are also perfect. Lovász proved that this is true in general, a result known as the Weak Perfect Graph Theorem.

Theorem 11.4 (Weak Perfect Graph Theorem)**.** [Lov72] *If G is a perfect graph, then its complement is also a perfect graph.*

Fulkerson had previously reduced the problem to showing that if G is perfect than G' is perfect, where G' is the graph formed by taking some vertex v, making a copy v' adjacent to all of the same vertices as v, and connecting v and v' by an edge. This is what Lovász proved.

Recall that C_5 is not a perfect graph. More generally, it is true that any odd cycle of length greater than or equal to five is not perfect. Such a cycle is called an odd hole. An odd antihole is the complement of one of these cycles. A Berge graph is a graph that contains no odd holes and no odd antiholes. The Strong Perfect Graph Theorem states that a graph is perfect if and only if it is a Berge graph.

Theorem 11.5 (Strong Perfect Graph Theorem)**.** [CRST06] *A graph if and only if it is a Berge graph.*

Lovász gave an alternative characterization of perfect graphs:

Theorem 11.6. *A graph G is perfect if and only if for all induced subgraphs G', $\alpha(G')\omega(G') \geq n'$, where n' is the number of vertices in G'.*

Note that one direction is obvious: If G is perfect, then $\chi(G') = \omega(G')$, and it is always true that $\alpha(G')\chi(G') \geq n'$. Finally, it is also possible to check whether or not a graph is perfect in polynomial time.

Theorem 11.7. [CCL+05] *There exists a polynomial time algorithm to recognize perfect graphs.*

11.2 Computing α, ω, χ, and $\bar{\chi}$ for perfect graphs

Now we consider the problem of computing α, ω, χ, and $\bar{\chi}$ for perfect graphs. Assume we had a function $f(G)$ such that for all G, the following held:

$$\alpha(G) \leq f(G) \leq \bar{\chi}(G)$$

Then since $\alpha(G) = \bar{\chi}(G)$ for any perfect graph G, $f(G) = \alpha(G) = \bar{\chi}(G)$. If f were computable in polynomial time, we would be able to compute $\alpha(G)$ and $\bar{\chi}(G)$ for any perfect

graph G in polynomial time. We would be able to compute $\omega(G)$ and $\chi(G)$ by computing $f(\bar{G})$. We can make a first attempt at finding such an f using a linear program P.

$$\begin{aligned} \max & \sum_{i \in V} x_i \\ \text{s.t.} & \sum_{i \in C} x_i \leq 1 \quad \forall \text{ cliques } C \text{ in } G \\ & x_i \geq 0 \quad \forall i \in V \end{aligned} \tag{11.1}$$

Given a maximum independent set, setting $x_i = 1$ if i is in the set and $x_i = 0$ otherwise gives a feasible solution, so $\alpha(G) \leq \text{Opt}(P)$.

Consider the dual D:

$$\begin{aligned} \min & \sum_{\text{cliques } C} y_C \\ \text{s.t.} & \sum_{C \ni i} y_C \geq 1 \quad \forall i \in V \\ & y_C \geq 0 \quad \forall \text{ cliques } C \text{ in } G \end{aligned} \tag{11.2}$$

For a minimum clique cover, setting y_C to 1 if C is in the minimum clique cover ans $y_C = 0$ otherwise gives a feasible solution, so $\text{Opt}(D) \leq \bar{\chi}(G)$. This means that setting $f(G) := \text{Opt}(P) = \text{Opt}(D)$ satisfies $\alpha(G) \leq f(G) \leq \bar{\chi}(G)$ as desired.

However, we cannot solve these linear programs for general graphs. Consider the separation oracle that, given $x \in \mathbb{R}^{|V|}$ with $x \geq 0$ decides whether or not there exists some clique C such that $\sum_{i \in C} x_i \geq 1$. This solves the maximum weight clique problem, which is NP-hard. If we could solve P for general graphs, we would have such a separation oracle. This means that solving P must be NP-hard for general graphs. It is not clear now to solve D either, as it has an exponential number of variables.

Can we solve the P and D at least for perfect graphs? It is not clear even how to do that. So let's try using semidefinite programs.

11.3 The Lovász ϑ function

Lovász introduced a function ϑ satisfying $\alpha(G) \leq \vartheta(G) \leq \bar{\chi}(G)$ [Lov79]. We begin by developing an SDP relaxation for $\bar{\chi}(G)$. We assign a unit vector v_i to each vertex. If two vertices are in the same clique of the minimum clique cover, we would like their vectors to be the same. If two vertices are not in the same clique, we would like their vectors to be as far apart as possible. Note that when k vectors are as spread out as possible, the dot product of any pair of them is $-\frac{1}{k-1}$. This means that if we have a clique cover of size k, there is an assignment of unit vectors to vertices such that every vertex in a clique is mapped to the same vector and, if two vertices are not in the same clique, the dot product of their vectors is $-\frac{1}{k=1}$. This is shown for clique covers of size 2, 3, and 4 in Figure 11.4.

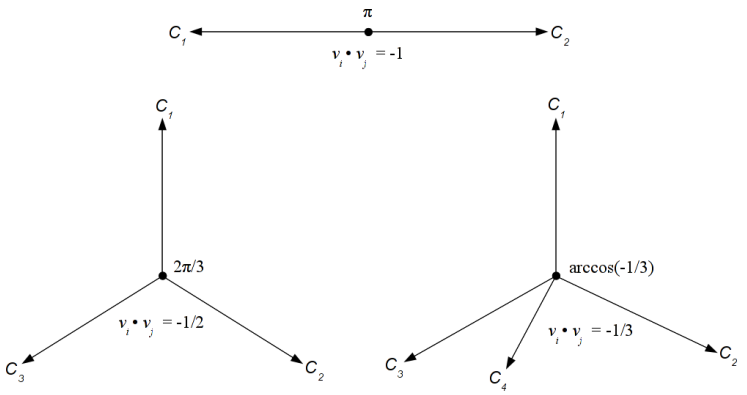


Figure 11.4: Assigning unit vectors to vertices such that the vertices in the same clique of the clique cover map to the same vector and vertices that are not in the same clique map to maximally separated vectors.

This suggests the following SDP relaxation:

$$\begin{aligned} &\min k \\ \text{s.t. } &\langle v_i, v_j \rangle = -\frac{1}{k-1} \quad i,j \in V, i \nsim j, i \neq j \\ &\langle v_i, v_i \rangle = 1 \qquad\qquad \forall i \in V \end{aligned} \tag{11.3}$$

where we use $i \sim j$ to denote that $(i,j) \in E(G)$, and $i \nsim j$ to denote that $(i,j) \notin E(G)$. We can now define the Lovász ϑ function.

Definition 11.8. Given $G = (V,E)$, $\vartheta(G)$ is the optimal value of the SDP in (11.3).

We can also write the following equivalent SDP:

$$\begin{aligned} &\min t \\ \text{s.t. } &\langle v_i, v_j \rangle = t \quad i,j \in V, i \nsim j, i \neq j \\ &\langle v_i, v_i \rangle = 1 \quad \forall i \in V \end{aligned}$$

In this case, the optimum is equal to $\frac{1}{1-\vartheta(G)}$. For a graph that is a clique, $-\frac{1}{k-1}$ and t both go to $-\infty$. Such graphs are not interesting to us, so this will not be a problem.

Theorem 11.9. $\alpha(G) \leq \vartheta(G) \leq \bar{\chi}(G)$

Proof. As described in the above discussion, any clique cover corresponds to a feasible solution of the SDP with an objective function value equal to the size of the clique cover. This implies that $\vartheta(G) \leq \bar{\chi}(G)$. It remains to show that $\alpha(G) \leq \vartheta(G)$. Suppose that $v_1, \ldots, v_s$ are the SDP solution vectors corresponding to a maximal independent set of size $s = \alpha(G)$ and let $v = \sum_{i=1}^{s} v_1$. Then $v^T v \geq 0$. It is also true that

$$v^T v = \left(\sum_{i=1}^{s} v_i \right)^T \left(\sum_{i=1}^{s} v_i \right) = \sum_{i=1}^{s} v_i^T v_i + \sum_{i \neq j} v_i^T v_j = s + \sum_{i \neq j} v_i^T v_j.$$

Then we have that $s + \sum_{i \neq j} v_i^T v_j \geq 0$. There are $s(s-1)$ terms in the sum, so, by averaging, there exist some distinct i and j such that

$$v_i^T v_j \geq -\frac{s}{s(s-1)} = -\frac{1}{s-1}.$$

Since $v_i^T v_j = -\frac{1}{\vartheta(G)-1}$ by the SDP constraints, $\alpha(G) = s \leq \vartheta(G)$. Therefore, we can conclude that $\alpha(G) \leq \vartheta(G) \leq \bar{\chi}(G)$. □

Note that $\vartheta(G)$ may not be rational for non-perfect graphs. For example, $\bar{\chi}(C_5) = 3$, $\alpha(C_5) = 2$, and $\vartheta(C_5) = \sqrt{5}$. So we cannot hope to get the exact optimum. However, we can solve the above semidefinite program to arbitrary accuracy using the ellipsoid algorithm, resulting in the following theorem.

Theorem 11.10. *For any $\epsilon > 0$, $\vartheta(G)$ can be computed to within ϵ error in time* $\text{poly}(n, \log \frac{1}{\epsilon})$.

The polynomial-time computability of the values of the parameters α, ω, χ, and $\bar{\chi}$ directly follows.

Corollary 11.11. *For any perfect graph G, $\alpha(G)$, $\omega(G)$, $\chi(G)$, and $\bar{\chi}(G)$ can be computed in polynomial time.*

11.3.1 Dual of the SDP

As an aside, the dual of the above SDP is the following SDP, with variables $B = (b_{ij})$:

$$\begin{aligned} \max \sum_{i,j \in V} b_{ij} & \\ \text{s.t. } b_{ij} &= 0 \quad i,j \in V, i \sim j \\ \sum_{i \in V} b_{ii} &= 1 \\ B &\succeq 0 \end{aligned} \tag{11.4}$$

We'll go through the process of deriving this dual program from the primal SDP in a future homework.

11.4 Non-perfect graphs and ϑ

We can also ask how closely ϑ approximated α for non-perfect graphs. Konyagin [Kon81] constructed a graph G such that $\alpha(G) = 2$ and $\vartheta(G) = \Omega(n^{1/3})$, which is the largest that $\vartheta(G)$ can be. Alon and Kahale generalized this result with the following theorem.

Theorem 11.12. [AK98] *If $\alpha(G) \leq k$, then $\vartheta(G) \leq Cn^{\frac{k-1}{k+1}}$ for some constant C.*

When α is not bounded, Feige showed the following result.

Theorem 11.13. [Fei97] *There exists a graph G such that $\alpha(G) = n^{o(1)}$ and $\vartheta(G) = n^{1-o(1)}$.*

Håstad's results for the hardness of approximating the clique problem [Hås99] also imply that such a graph must exist.

Kleinberg and Goemans showed that ϑ gives a 2-approximation for the size of the minimum vertex cover.

Theorem 11.14. [KG98] *For any graph G, $\frac{1}{2}\mathrm{VC}(G) \leq n - \vartheta(G) \leq \mathrm{VC}(G)$.*

This is not very useful for approximating $\mathrm{VC}(G)$, as the greedy algorithm for minimum vertex cover gives a 2-approximation. There are graphs for which this is tight, so we can do no better.

11.5 Finding cliques, independent sets, coloring, and clique covers of perfect graphs

Since we can compute α on perfect graphs, we can also find an independent set of size α in polynomial time. Consider the following algorithm: Remove a vertex from the graph. Calculate α for the resulting graph G'. If $\alpha(G') = \alpha(G) - 1$, put the removed vertex back; it must belong to the maximum independent set. Otherwise, leave it out. Repeat for the rest of the vertices in the new graph. The maximum independent set will remain at the end. We use the same method to find the maximum clique by noting that cliques are independent sets in the complement of the graph.

We next consider the problem if finding an optimal clique cover, which corresponds to an optimal coloring of the complement. In order to find an optimal clique cover, we need to find a maximum weight clique instead of a maximum size clique. To do this, we use a variant of the SDP in (11.4). Let w be non-negative vertex weights, which, without loss of generality, we assume to be integers. Then consider the following SDP:

$$\begin{aligned} \max \sum_{i,j \in V} \sqrt{w_i} b_{ij} \sqrt{w_j} & \\ \text{s.t. } b_{ij} &= 0 \quad i,j \in V, i \sim j \\ \sum_{i \in V} b_{ii} &= 1 \\ B &\succeq 0 \end{aligned} \tag{11.5}$$

We define $\vartheta(G, w)$ to be the optimal value of this SDP. Consider the graph G' formed by replacing each vertex i with a clique of size w_i such that two vertices in G' not in the same clique are adjacent if and only if the vertices in G corresponding to their cliques are adjacent. It is true that $\vartheta(G, w) = \vartheta(G')$. Let $\omega(G, w) = \omega(G'$ and $\chi(G, w) = \chi(G')$. Then $\omega(G, w) \leq \vartheta(\bar{G}, w) \leq \chi(G, w)$. Also, it is a fact that if G is perfect, then G' is perfect. In this case, $\omega(G, w) = \vartheta(\bar{G}, w) = \chi(G, w)$.. Therefore, by solving (11.5) and using self-reducibility as described above, we can find maximum weight cliques in perfect graphs.

We now give an algorithm to find a minimum clique cover, which corresponds to an optimal coloring of the complement.

Recall the primal-dual pair of linear programs P and D given above:

$$\begin{aligned} \max & \sum_{i \in V} x_i \\ \text{s.t.} & \sum_{i \in C} x_i \leq 1 \quad \forall \text{ cliques } C \text{ in } G \\ & x_i \geq 0 \quad \forall i \in V \end{aligned}$$

$$\begin{aligned} \min & \sum_{\text{cliques } C} y_C \\ \text{s.t.} & \sum_{C \ni i} y_C \geq 1 \quad \forall i \in V \\ & y_C \geq 0 \quad \forall \text{ cliques } C \text{ in } G \end{aligned}$$

These are the same as the linear programs (11.1) and (11.2) that we used in our initial attempt to find an f such that $\alpha(G) \leq f(G) \leq \bar{\chi}(G)$.

Step 1 Use the ellipsoid algorithm to solve the primal P. In order to do this, we need a separation oracle. Solving (11.5) to find the maximum weight of a clique gives us this separation oracle. The feasible region of P is a rational polyhedron, so we can find an optimal solution in polynomial time.

Step 2 Let $\mathcal{I} = \{C_1, C_2, \ldots, C_t\}$ be the set of polynomially-many cliques for which constraint were violated while running the ellipsoid algorithm in Step 1. Now consider the following linear program, which we will call $P_{\mathcal{I}}$:

$$\begin{aligned} \max & \sum_{i \in V} x_i \\ \text{s.t.} & \sum_{i \in C} x_i \leq 1 \quad \forall \text{ cliques } C \text{ in } \mathcal{I} \\ & x_i \geq 0 \quad \forall i \in V \end{aligned}$$

It is clear that $\text{Opt}(P_{\mathcal{I}}) \geq \text{Opt}(P)$. It cannot be true that $\text{Opt}(P_{\mathcal{I}}) > \text{Opt}(P)$. Otherwise, running the ellipsoid algorithm on the constraints in for the cliques in $\mathcal{I}$ would give $\text{Opt}(P) < \text{Opt}(P_{\mathcal{I}})$, which would contradict the correctness of the ellipsoid algorithm. So $\text{Opt}(P_{\mathcal{I}})$ must be equal to $\text{Opt}(P)$.

Step 3 Consider the dual of $P_{\mathcal{I}}$, which we will call $D_{\mathcal{I}}$. A feasible solution of $D_{\mathcal{I}}$ corresponds to a feasible solution of D in which all y_C such that $C \notin \mathcal{I}$ are set to 0. We know that $\text{Opt}(D_{\mathcal{I}}) = \text{Opt}(P_{\mathcal{I}})$, $\text{Opt}(P_{\mathcal{I}}) = \text{Opt}(P)$, and $\text{Opt}(P) = \text{Opt}(D)$, so $\text{Opt}(D_{\mathcal{I}}) = \text{Opt}(D)$. Now we can solve $D_{\mathcal{I}}$ in polynomial time to find an optimal solution of D. Call this solution y^*. Let C be some clique such that $y^*_C > 0$.

Step 4 By complementary slackness, if x^* is any optimal solution of P, then

$$\left(\sum_{i \in C} x_i^* - 1\right) y_C^* = 0.$$

Since $y_C^* > 0$, $\sum_{i \in C} x_i^* = 1$ for any optimal solution x^* of P. For any maximum independent set, let x be a solution to P such that $x_i = 1$ if i is in the set and $x_i = 0$ otherwise. For perfect graphs, $\alpha(G) = \text{Opt}(P) = \bar{\chi}(G)$, so x is an optimal solution to P. By the above, $\sum_{i \in C} x_i = 1$. This implies that all maximum independent sets of G contain exactly one vertex of C. Removing C from G therefore results in a graph G' such that $\alpha(G')$ is one less than $\alpha(G)$. Thus, recursing on G' gives us a clique cover of size $\bar{\chi}(G) = \alpha(G)$ as desired.

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1

Lecture 12

Semidefinite Duality*

In the past couple of lectures we have discussed semidefinite programs and some of their applications in solving computer science problems. In this lecture we introduce the concept of semidefinite duality and look at the relationship between a semidefinite program and its dual.

12.1 Semidefinite Matrices

Recall from Lecture 10 that a symmetric matrix A of size $n \times n$ is positive semidefinite (psd) if it meets any one of the following two criteria (or any of a host of others): the third criterion below is yet another useful way to characterize psd matrices.

1. $x^\top Ax \geq 0$ for all $x \in \mathbb{R}^n$.
2. All eigenvalues of A are non-negative.
3. $A = PDP^\top$ where D is the diagonal matrix $\text{diag}(\lambda_1, \lambda_2, \ldots \lambda_n)$ where the λ_i are the eigenvalues of A and P's columns are the eigenvectors of A. In D we note that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$.

   We define the function $\text{diag}(x_1, x_2, \ldots x_n)$ below.

$$\text{diag}(x_1, x_2, \ldots x_n) = \begin{pmatrix} x_1 & & & \\ & x_2 & & \\ & & \ddots & \\ & & & x_n \end{pmatrix}$$

   We note from this property that $A^{1/2} = PD^{1/2}P^\top$ where $D^{1/2} = \text{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots)$. From this, due to the orthonormality of P, it is clear that $A = A^{1/2}A^{1/2}$.

We would like to include additional notation to simplify future computations. Recall that $A \succeq 0$ denotes that A is positive semidefinite; let us define $A \preceq B$ to mean $B - A \succeq 0$.

*Lecturer: Anupam Gupta. Scribe: Alex Beutel.

Definition 12.1. Given symmetric matrices A, B we define $A \bullet B = Tr(A^\top B) = \sum_{ij} A_{ij}B_{ij}$.

We can think of A and B as vector of length n^2, then $A \bullet B$ is just the usual inner product between vectors. Note that if $x \in \mathbb{R}^n$, then $(xx^\top)$ is an $n \times n$ matrix, where $(xx^\top)_{ij} = x_i x_j$.

Fact 12.2. $x^\top Ax = \sum_{ij} x_i x_j A_{ij} = \sum_{ij} (xx^\top)_{ij} A_{ij} = (xx^\top) \bullet A$.

Based on these underlying principles of linear algebra and psd matrices, we can begin to derive some interesting facts that will be useful later in the lecture. The proofs were not all discussed in class, but are given here for completeness.

Fact 12.3. *For any two $n \times n$ matrices A, B, $Tr(AB) = Tr(BA)$.*

Proof. $Tr(AB) = \sum_i (AB)_{ii} = \sum_i \sum_j A_{ij}B_{ji} = \sum_j \sum_i B_{ji}A_{ij} = Tr(BA)$. □

Lemma 12.4. *For a symmetric $n \times n$ matrix A, A is psd if and only if $A \bullet B \geq 0$ for all psd B.*

Proof. One direction is easy: if A is not psd, then there exists $x \in \mathbb{R}^n$ for which $A \bullet (xx^\top) = x^\top Ax < 0$. But $xx^\top$ is psd, which shows that $A \bullet B < 0$ for some psd B.

In the other direction, let A, B be psd. We claim that C, defined by $C_{ij} = A_{ij}B_{ij}$, is also psd. (This matrix C is called the *Schur-Hadamard product* of A and B.) Then

$$\sum_{ij} A_{ij}B_{ij} = \sum_{ij} C_{ij} = \mathbf{1}^\top C \mathbf{1} \geq 0$$

by the definition of psd-ness of C. To see the claim: since $A \succeq 0$, there exist random variables $\{a_i\}_{i=1}^n$ such that $A_{ij} = E[a_i a_j]$. Similarly, let $B_{ij} = E[b_i b_j]$ for r.v.s $\{b_i\}$. Moreover, we can take the a's to be independent of the b's. So if we define the random variables $c_i = a_i b_i$, then

$$C_{ij} = E[a_i a_j]E[b_i b_j] = E[a_i a_j b_i b_j] = E[c_i c_j],$$

and we are done. (Note we used independence of the a's and b's to make the product of expectations the expectation of the product.) □

This clean random variable-based proof is from this blog post. One can also show the following claim. The linear-algebraic proof also gives an alternate proof of the above Lemma 12.4.

Lemma 12.5. *For $A \succ 0$ (i.e., it is positive definite), $A \bullet B > 0$ for all psd B, $B \neq 0$.*

Proof. Let's write A as $PDP^\top$ where P is orthonormal, and D is the diagonal matrix containing A's eigenvalues (which are all positive, because $A \succ 0$.

Let $\widehat{B} = P^\top BP$, and hence $B = P\widehat{B}P^\top$. Note that $\widehat{B}$ is psd: indeed, $x^\top \widehat{B}x = (Px)^\top B(Px) \geq 0$. So all of $\widehat{B}$'s diagonal entries are non-negative. Moreover, since $B \neq 0$, not all of $\widehat{B}$'s diagonal entries can be zero (else, by $\widehat{B}$'s psd-ness, it would be zero). Finally,

$$Tr(AB) = Tr((PDP^\top)(P\widehat{B}P^\top)) = Tr(PD\widehat{B}P^\top) = Tr(D\widehat{B}P^\top P) = Tr(D\widehat{B}) = \sum_i D_{ii}\widehat{B}_{ii}.$$

Since $D_{ii} > 0$ and $\widehat{B}_{ii} \geq 0$ for all i, and $\widehat{B}_{ii} > 0$ for some i, this sum is strictly positive. □

Lemma 12.6. *For psd matrices A, B, $A \bullet B = 0$ if and only if $AB = 0$.*

Proof. Clearly if $AB = 0$ then $A \bullet B = tr(A^\top B) = tr(AB) = 0$. For the other direction, we use the ideas (and notation) from Lemma 12.4. Again take the Schur-Hadamard product C defined by $C_{ij} = A_{ij}B_{ij}$. Then C is also psd, and hence $C_{ij} = E[c_i c_j]$ for random variables $\{c_i\}_{i=1}^n$. Then

$$A \bullet B = \sum_{ij} C_{ij} = \sum_{ij} E[c_i c_j] = E[\sum_{ij} c_i c_j] = E[(\sum_i c_i)^2].$$

If this quantity is zero, then the random variable $\sum_i c_i = \sum_i a_i b_i$ must be zero with probability 1. Now

$$(AB)_{ij} = \sum_k E[a_i a_k]E[b_k b_j] = \sum_k E[a_i b_j (a_k b_k)] = E[a_i b_j (\sum_k c_k)] = 0,$$

so AB is identically zero. □

12.2 Semidefinite Programs and their Duals

Given this understanding of psd matrices, we can now look at semidefinite programs (SDPs), and define their duals. Let us describe two common forms of writing SDPs. Consider symmetric matrices $A_1, A_2, \ldots A_m, C$, and reals $b_1, b_2, \ldots b_m$. The first form is the following one.

$$\begin{aligned} &\min C \bullet X \\ \text{s.t.} \quad &A_i \cdot X = b_i \quad i = 1 \ldots m \\ &X \succeq 0 \end{aligned} \tag{12.1}$$

Another common form for writing SDPs is the following.

$$\begin{aligned} &\max \sum_{i=1}^m b_i y_i = b^\top y \\ \text{s.t.} \quad &\sum_{i=1}^m A_i y_i \preceq C \end{aligned} \tag{12.2}$$

This of course means that $C - \sum A_i y_i \succeq 0$. If we set $S = C - \sum A_i y_i$ and thus $S \succeq 0$ then it is clear that this constraint can be rewritten as $\sum y_i A_i + S = C$ for $S \succeq 0$.

$$\begin{aligned} &\max b^\top y \\ \text{s.t.} \quad &\sum_{i=1}^m A_i y_i + S = C \\ &S \succeq 0 \end{aligned} \tag{12.3}$$

Given an SDP in the form (12.1), we can convert it into an SDP in the form (12.3), and vice versa—this requires about a page of basic linear algebra.

12.2.1 Examples of SDPs

The Max-Cut Problem

An example, which we've already seen, is the semidefinite program for the maxcut problem. Given a graph $G = (V, E)$, with edge weights w_{ij},

$$\begin{aligned} &\frac{1}{2}\max \sum_{(i,j)\in E} w_{ij}(1 - \langle v_i, v_j\rangle) \\ &\text{s.t.} \quad \langle v_i, v_i\rangle = 1 \quad \forall i \in V \end{aligned}$$

This is equivalent to

$$\begin{aligned} &\frac{1}{2}\max \sum_{(i,j)\in E} w_{ij}(1 - X_{ij}) \\ &\text{s.t.} \quad X_{ii} = 1 \quad \forall i \in V \\ &X \succeq 0 \end{aligned}$$

where we used the fact that $X \succeq 0$ iff there are vectors v_i such that $X_{ij} = \langle v_i, v_j\rangle$. For i, j such that $\{i, j\} \notin E$, define $w_{ij} = 0$, and for $\{i, j\} \in E$ define $w_{ji} = w_{ij}$; hence the objective function can now be written as

$$\frac{1}{4}\max \sum_{i,j\in V} w_{ij}(1 - X_{ij}).$$

(The extra factor of $1/2$ is because we count each edge $\{u, v\}$ twice now.) We can write this even more compactly, once we introduce the idea of the Laplacian matrix of the weighted graph.

$$L_{ij} = L(w)_{ij} = \begin{cases} \sum_k w_{ik} & \text{if } i = j, \\ -w_{ij} & \text{if } i \neq j. \end{cases}$$

Again, the objective function of the above SDP (ignoring the factor of $\frac{1}{4}$ for now) is

$$\begin{aligned} \sum_{i,j} w_{ij}(1 - X_{ij}) &= \sum_{i,j} w_{ij} - \sum_{i\neq j} w_{ij}X_{ij} \\ &= \sum_i \left(\sum_j w_{ij}\right) + \sum_{i\neq j} L_{ij}X_{ij} \\ &= \sum_i L_{ii}\, X_{ii} + \sum_{i\neq j} L_{ij}\, X_{ij} \\ &= L \bullet X \end{aligned}$$

Finally rewriting $X_{ii} = X \bullet (e_i e_i^\top)$, the SDP is

$$\begin{aligned} &\max \frac{1}{4} L \bullet X \\ &X \bullet (e_i e_i^T) = 1 \quad \forall i \\ &X \succeq 0 \end{aligned} \tag{12.4}$$

Note that this SDP is in the form (12.1).

Maximum Eigenvalue of Symmetric Matrices

Another simple example is using an SDP to find the maximum eigenvalue for a symmetric matrix A. Suppose A has eigenvalues $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_n$. Then the matrix $tI - A$ has eigenvalues $t-\lambda_1, t-\lambda_2, \ldots, t-\lambda_n$. Note that $tI-A$ is psd exactly when all these eigenvalues are non-negative, and this happens for values $t \geq \lambda_1$. This immediately gives us that

$$\lambda_1 = \min\{t \mid \text{s.t. } tI - A \succeq 0\} \tag{12.5}$$

We will use $\lambda_1(A) = \lambda_{\max}(A)$ to denote the maximum eigenvalue of the matrix A. Note that this SDP uses the form (12.3) given above.

> **Note:** John pointed out that one could also write the maximum eigenvalue computation as the following SDP:
>
> $$\begin{aligned} &\max A \bullet X \\ \text{s.t.} \quad &X \bullet I = 1 \\ &X \succeq 0 \end{aligned} \tag{12.6}$$
>
> Indeed, we will soon show that (12.6) is precisely the SDP dual of (12.5).

12.2.2 Weak Duality

Given the two SDP forms above, namely (12.1) and (12.3), let's first note that one can move purely syntactically from one to the other. Next, one can show that these form a primal-dual pair. Let us consider the minimization problem in (12.1) to be the primal, and the maximization problem (12.3) to be the dual form.

Theorem 12.7 (Weak Duality)**.** *If X is feasible for the primal SDP and (y, S) are feasible for the dual SDP, then $C \bullet X \geq b^\top y$.*

Proof. Suppose (y, S) and X are feasible, then:

$$\begin{aligned} C \bullet X &= (\sum y_i A_i + S) \bullet X && (12.7)\\ &= \sum y_i (A_i \bullet X) + (S \bullet X) && (12.8)\\ &= \sum y_i b_i + (S \bullet X) && (12.9) \end{aligned}$$

Since S and X are psd, Lemma 12.4 implies $S \bullet X \geq 0$. Therefore, $C \bullet X \geq b^\top y$. □

Note that the transformation between the primal SDP form and the dual form was syntactic, much like in the case of LPs. And we have weak duality, like LPs. However, in Section 12.3 we will see that strong duality does not always hold (there may be a gap between the primal and dual values), but will also give some natural conditions under which strong SDP duality does hold.

12.2.3 General Cone Programs

Before we move on, let us actually place semidefinite duality (and LP duality) in a slightly broader context, that of duality in general cone programming. Suppose we consider a convex cone $K \in \mathbb{R}^n$ (i.e., it is convex, and for $x \in K$ and $\alpha \geq 0$, $\alpha x \in K$). We can now define the *dual cone* $K^* = \{y \mid x^\top y \geq 0 \ \forall x \in K\}$. E.g., here is an example of a cone in $\mathbb{R}^2$, and its dual cone.

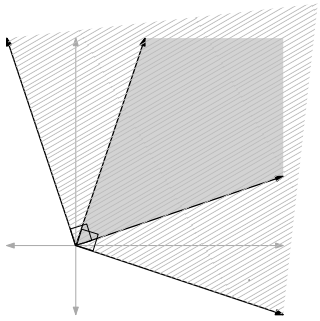


Figure 12.1: The cone K (in dark grey) and its dual cone K^* (shaded).

Moreover, here are some examples of K and the corresponding K^*.

$$\begin{aligned} K &= \mathbb{R}^n & K^* &= \{0\} \\ K &= \mathbb{R}^n_{\geq 0} & K^* &= R^n_{\geq 0} \\ K &= PSD_n & K^* &= PSD_n \end{aligned}$$

Let us write two optimization problems over cones, the primal and the dual. Given vectors $a_1, a_2, \ldots, a_m, c \in \mathbb{R}^n$ and scalars $b_1, b_2, \ldots, b_m \in \mathbb{R}$, the primal cone program (P') is

$$\begin{aligned} &\min c^\top x \\ \text{s.t.} \quad &a_i^\top x = b_i \quad i = 1 \ldots m \\ &x \in K \end{aligned}$$

The dual cone program (D') is written below:

$$\begin{aligned} &\max b^\top y \\ &\sum_{i=1}^m y_i a_i + s = c \\ &s \in K^*, y \in \mathbb{R}^m \end{aligned}$$

Claim 12.8 (Weak Duality for Cone Programs)**.** *If x is feasible for the primal (P') and (y, s) feasible for the dual (D'), then $c^\top x \geq b^\top y$.*

Proof. $c^\top x = (\sum y_i a_i + s)^\top x = \sum y_i a_i^\top + s^\top x \geq \sum y_i b_i + 0 = b^\top y$, where we use the fact that if $x \in K$ and $s \in K^*$ then $s^\top x \geq 0$. □

Now instantiating K with the suitable cones we can get LPs and SDPs: e.g., considering $K = \mathbb{R}^n_{\geq 0} = K^*$ gives us

$$\begin{array}{ll} \min c^\top x & \max b^\top y \\ \text{s.t.} \quad a_i^\top x = b_i \quad i = 1 \dots m & \sum_{i=1}^m y_i a_i + s = c \\ x \geq 0 & s \geq 0, y \in \mathbb{R}^m \end{array}$$

which is equivalent to the standard primal-dual pair of LPs

$$\begin{array}{ll} \min c^\top x & \max b^\top y \\ \text{s.t.} \quad a_i^\top x = b_i \quad i = 1 \dots m & \sum_{i=1}^m y_i a_i \leq c \\ x \geq 0 & y \in \mathbb{R}^m \end{array}$$

And setting $K = K^* = PSD_n$ gives us the primal-dual pair of SDPs (12.1) and (12.3).

12.2.4 Examples: The Maximum Eigenvalue Problem

For the maximum eigenvalue problem, we wrote the SDP (12.5). Since it of the "dual" form, we can convert it into the "primal" form in a purely mechanical fashion to get

$$\begin{array}{ll} \max A \bullet X \\ \text{s.t.} \quad X \bullet I = 1 \\ X \succeq 0 \end{array}$$

We did not cover this in lecture, but the dual can be reinterpreted further. recall that $X \succeq 0$ means we can find reals $p_i \geq 0$ and unit vectors $x_i \in \mathbb{R}^n$ such that $X = \sum_i p_i (x_i x_i^\top)$. By the fact that x_i's are unit vectors, $Tr(x_i x_i^\top) = 1$, and the trace of this matrix is then $\sum_i p_i$. But by our constraints, $X \bullet I = Tr(X) = 1$, so $\sum_i p_i = 1$.

Rewriting in this language, $\lambda_{\max}$ is the maximum of

$$\sum_i p_i \, (A \bullet x_i x_i^\top)$$

such that the x_i's are unit vectors, and $\sum_i p_i = 1$. But for any such solution, just choose the vector x_{i^*} among these that maximizes $A \bullet (x_{i^*} x_{i^*}^\top)$; that is at least as good as the average, right? Hence,

$$\lambda_{\max} = \max_{x \in \mathbb{R}^n : \|x\|_2 = 1} A \bullet (x x^\top) = \max_{x \in \mathbb{R}^n} \frac{x^\top A x}{x^\top x}$$

which is the standard variational definition of the maximum eigenvalue of A.

12.2.5 Examples: The Maxcut SDP Dual

Now let's revisit the maxcut SDP. We had last formulated the SDP as being of the form

$$\begin{array}{ll} \max \frac{1}{4} L \bullet X \\ X \bullet (e_i e_i^T) = 1 \quad \forall i \\ X \succeq 0 \end{array}$$

It is in "primal" form, so we can mechanically convert it into the "dual" form:

$$\begin{aligned} \min \frac{1}{4}\sum_{i=1}^{n} y_i \\ \text{s.t.} \quad \sum_i y_i(e_i e_i^\top) \succeq L \end{aligned}$$

For a vector $v \in \mathbb{R}^n$, we define the matrix $\mathrm{diag}(v)$ to be the diagonal matrix D with $D_{ii} = v_i$. Hence we can rewrite the dual SDP as

$$\begin{aligned} \min \tfrac{1}{4}\mathbf{1}^\top y \\ \text{s.t.} \quad \mathrm{diag}(y) - L \succeq 0 \end{aligned}$$

Let us write $y = t\mathbf{1} - u$ for some real $t \in \mathbb{R}$ and vector $u \in \mathbb{R}^n$ such that $\mathbf{1}^\top u = 0$: it must be the case that $\mathbf{1}^\top y = n \cdot t$. Moreover, $\mathrm{diag}(t\mathbf{1} - u) = tI - \mathrm{diag}(u)$, so the SDP is now

$$\begin{aligned} \min \tfrac{1}{4} n \cdot t \\ \text{s.t.} \quad tI - (L + \mathrm{diag}(u)) \succeq 0 \\ \mathbf{1}^\top u = 0. \end{aligned}$$

Hey, this looks like the maximum eigenvalue SDP from (12.5): indeed, we can finally write the SDP as

$$\frac{n}{4} \cdot \min_{u:\mathbf{1}^\top u = 0} \lambda_{\max}(L + \mathrm{diag}(u)) \tag{12.10}$$

What is this saying? We're taking the Laplacian of the graph, adding in some "correction" values u to the diagonal (which add up to zero) so as the make the maximum eigenvalue as small as possible. The optimal value of the dual SDP is this eigenvalue scaled up by $n/4$. (And since we will soon show that strong duality holds for this SDP, this is also the value of the max-cut SDP.) This is precisely the bound on max-cut that was studied by Delorme and Poljak [DP93].

In fact, by weak duality alone, any setting of the vector u would give us an upper bound on the max-cut SDP (and hence on the max-cut). For example, one setting of these correction values would be to take $u = 0$, we get that

$$maxcut(G) \leq SDP(G) \leq \frac{n}{4}\lambda_{\max}(L(G)), \tag{12.11}$$

where $L(G)$ is the Laplacian matrix of G. This bound was given even earlier by Mohar and Poljak [MP90].

Some Examples

Sometimes just the upper bound (12.11) is pretty good: e.g., for the case of cycles C_n, one can show that the zero vector is an optimal correction vector $u \in \mathbb{R}^n$, and hence the max-cut SDP value equals the $n/4\lambda_{\max}(L(C_n))$.

To see this, consider the function $f(u) = \frac{n}{4}\lambda_{\max}(L + \mathrm{diag}(u))$. This function is convex (see, e.g. [DP93]), and hence $f(\frac{1}{2}(u+u')) \leq \frac{1}{2}(f(u)+f(u'))$. Now if $f(u)$ is minimized for some non-zero vector u^* such that $\mathbf{1}^\top u = 0$. Then by the symmetry of the cycle, the vector $u^{(i)} = (u_i^*, u_{i+1}^*, \ldots, u_n^*, u_1^*, \ldots, u_{i-1}^*)^\top$ is also a minimizer. But look: each coordinate of $\sum_{i=1}^n u^{(i)}$ is itself just $\sum_i u_i^* = 0$. On the other hand, $f(0) = f(\frac{1}{n}\sum_i u^{(i)}) \leq \frac{1}{n}\sum_i f(u^{(i)}) = f(u)$ by the convexity of $f()$. Hence the zero vector is a minimizer for $f()$.

> **Note:** Among other graphs, Delorme and Poljak considered the gap between the integer max-cut and the SDP value for the cycle. The eigenvalues for $L(C_n)$ are $2(1-\cos(2\pi t/n)$ for $t = 0, 1, \ldots, n-1$. For $n = 2k$, the maximum eigenvalue is 4, and hence the max-cut dual (and primal) value is $\frac{n}{4} \cdot 4 = n$, which is precisely the max-cut value. The SDP gives us the right answer in this case.
>
> What about odd cycles? E.g., for $n = 3$, say, the maximum eigenvalue is $3/2$, which means the dual (=primal) equals $9/8$. For $n = 5$, $\lambda_{\max}$ is $\frac{1}{2}(5+\sqrt{5})$, and hence the SDP value is 4.52. This is ≈ 1.1306 times the actual integer optimum. (Note that the best current gap is $1/0.87856 \approx 1.1382$, so this is pretty close to the best possible.)

On the other hand, for the star graph, the presence of the correction vector makes a big difference. We'll see more of this in the next HW.

12.3 Strong Duality

Unfortunately, for SDPs, strong duality does not always hold. Consider the following example (from Lovász):

$$\begin{aligned} &\min y_1 \\ &\text{s.t.} \quad \begin{pmatrix} 0 & y_1 & 0 \\ y_1 & y_2 & 0 \\ 0 & 0 & y_1+1 \end{pmatrix} \succeq 0 \end{aligned} \tag{12.12}$$

Since the top-left entry is zero, SDP-ness forces the first row and column to be zero, which means $y_1 = 0$ in any feaasible solution. The feasible solutions are $(y_1 = 0, y_2 \geq 0)$. So the primal optimum is 0. Now to take the dual, we write it in the form (12.2):

$$y_1 \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} + y_2 \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \succeq \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{pmatrix}.$$

to get the dual:

$$\begin{aligned} &\max -X_{33} \\ &\text{s.t.} \quad X_{12} + X_{21} + X_{33} = 1 \\ &\qquad\quad X_{22} = 0 \\ &\qquad\quad X \succeq 0 \end{aligned}$$

Since $X_{22} = 0$ and $X \succeq 0$, we get $X_{12} = X_{21} = 0$. This forces $X_{33} = 1$, and the optimal value for this dual SDP is -1. Even in this basic example, strong duality does not

hold. So the strong duality theorem we present below will have to make some assumptions about the structure of the primal and dual, which is often called *regularization* or *constraint qualification*.

Before we move on, observe that the example is a fragile one: if one sets the top left entry of (12.12) to $\epsilon > 0$, suddenly the optimal primal value drops to -1. (Why?)

One more observation: consider the SDP below.

$$\begin{aligned} &\min x_1 \\ &\begin{pmatrix} x_1 & 1 \\ 1 & x_2 \end{pmatrix} \succeq 0 \end{aligned}$$

By PSD-ness we want $x_1 \geq 0$, $x_2 \geq 0$, and $x_1, x_2 \geq 1$. Hence for any $\epsilon > 0$, we can set $x_1 = \epsilon$ and $x_2 = 1/x_1$—the optimal value tends to zero, but this optimum value is never achieved. So in general we define the optimal value of SDPs using infimums and supremums (instead of just mins and maxs). Furthermore, it becomes a relevant question of whether the SDP achieves its optimal value or not, when this value is bounded. (This was not an issue with LPs: whenever the LP was feasible and its optimal value was bounded, there was a feasible point that achieved this value.)

12.3.1 The Strong Duality Theorem for SDPs

We say that a SDP is *strictly feasible* if it satisfies its positive semidefiniteness requirement strictly: i.e. with positive definiteness.

Theorem 12.9. *Assume both primal and dual have feasible solutions. Then $v_{\text{primal}} \geq v_{\text{dual}}$, where v_{primal} and v_{dual} are the optimal values to the primal and dual respectively. Moreover, if the primal has a strictly feasible solution (a solution x such that $x \succ 0$ or x is positive definite) then*

1. *The dual optimum is attained (which is not always the case for SDPs)*
2. *$v_{\text{primal}} = v_{\text{dual}}$*

Similarly, if the dual is strictly feasible, then the primal optimal value is achieved, and equals the dual optimal value. Hence, if both the primal and dual have strictly feasible solutions, then both v_{primal} and v_{dual} are attained.

> **Note:** For both the SDP examples we've considered (max-cut, and finding maximum eigenvalues), you should check that strictly feasible solutions exist for both primal and dual programs, and hence there is no duality gap.
>
> Strict feasibility is also a sufficient condition for avoiding a duality gap in more general convex programs: this is called the *Slater condition*. For more details see, e.g., the book by Boyd and Vanderberghe.

12.3.2 The Missing Proofs*

We did not get into details of the proof in lecture, but they are presented below for completeness. (The presentation is based on, and closely follows, that of Laci Lovász's notes.) We need a SDP version of the Farkas Lemma. First we present a homogeneous version, and use that to prove the general version.

Lemma 12.10. *Let A_i be symmetric matrices. Then $\sum_i y_i A_i \succ 0$ has no solution if and only if there exists $X \succeq 0$, $X \neq 0$ such that $A_i \bullet X = 0$ for all i.*

One direction of the proof (if $\sum_i y_i A_i \succ 0$ is infeasible, then such an X exists) is an easy application of the hyperplane separation theorem, and appears in Lovasz's notes. The other direction is easy: if there is such a X and $\sum_i y_i A_i \succ 0$ is feasible, then by Lemma 12.5 and the strict positive definiteness $(\sum_i y_i A_i) \bullet X > 0$, but all $A_i \bullet X = 0$, which is a contradiction.

We could ask for a similar theorem of alternatives for $\sum_i y_i A_i \succeq 0$: since we're assuming more, the "only if" direction goes through just the same. But the "if" direction fails, since we cannot infer a contradiction just from Lemma 12.4. And this will be an important issue in the proof of the duality theorem. Anyways, the Farkas Lemma also extends to the non-homogeneous case:

Lemma 12.11. *Let A_i, C be symmetric matrices. Then $\sum_i y_i A_i \succ C$ has no solution if and only if there exists $X \succeq 0$, $X \neq 0$ such that $A_i \bullet X = 0$ for all i and $C \bullet X \geq 0$.*

Proof. Again, if such X exists then we cannot have a solution. For the other direction, the constraint $\sum_i y_i A_i - C \succ 0$ is equivalent to $\sum_i y_i A_i + t(-C) \succ 0$ for $t > 0$ (because then we can divide through by t). To add this side constraint on t, let us define

$$A'_i := \begin{pmatrix} A_i & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad C' := \begin{pmatrix} -C & 0 \\ 0 & 1 \end{pmatrix}.$$

Lemma 12.10 says if $\sum_i y_i A'_i + tC' \succ 0$ is infeasible then there exists psd $X' \neq 0$, with $X' \bullet A'_i = 0$ and $X' \bullet C' = 0$. If X is the top $n \times n$ part of X', we get $X \bullet A_i = 0$ and $(-C) \bullet X + x_{n+1,n+1} = 0$. Moreover, from $X' \succeq 0$, we get $X \succeq 0$ and $x_{n+1,n+1} \geq 0$—which gives us $C \bullet X \geq 0$. Finally, to check $X \neq 0$: in case $X' \neq 0$ but $X = 0$, we must have had $x_{n+1,n+1} > 0$, but then $C' \bullet X \neq 0$. □

Now, here's the strong duality theorem: here the primal is

$$\min\{b^\top y \mid \sum_i y_i A_i \succeq C\}$$

and the dual is

$$\max\{C \bullet X \mid A_i \bullet X = b_i \forall i, X \succeq 0\}$$

Theorem 12.12. *Assume both primal and dual are feasible. If the primal is strictly feasible, then (a) the dual optimum is achieved, and (b) the primal and dual optimal values are the same.*

Proof. Since $b^\top y < opt_p$ and $\sum_i y_i A_i \succeq C$ is not feasible, we can define

$$A'_i := \begin{pmatrix} -b_i & 0 \\ 0 & A_i \end{pmatrix} \quad \text{and} \quad C' := \begin{pmatrix} -opt_p & 0 \\ 0 & C \end{pmatrix}$$

and use Lemma 12.11 to get psd $Y' \neq 0$ with $Y' \bullet A'_i = 0$ and $Y' \bullet C' \geq 0$. Say

$$Y' := \begin{pmatrix} y_0 & y \\ y & Y \end{pmatrix}$$

then $A_i \bullet Y = y_0\, b_i$ and $C \bullet Y \geq y_0\, opt_p$. By psd-ness, $y_0 \geq 0$. If $y_0 \neq 0$ then we can divide through by y_0 to get a feasible solution to the dual with value equal to opt_p.

What if $y_0 = 0$? This cannot happen. Indeed, then we get $A_i \bullet Y = 0$ and $C \bullet Y \geq 0$, which contradicts the *strict* feasibility of the primal. (Note that we are using the "if" direction of the Farkas Lemma here, whereas we used the "only if" direction in the previous step.) Again, we really need to assume strict feasibility, because there are examples otherwise. □

The notion of duality we've used here is Lagrangian duality. This is not the only notion possible, and in fact, there are papers that study other notions of duality that avoid this "duality gap" without constraint qualification. For example, see the paper of Ramana, Tüncel, and Wolkowicz (1997).

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1

Lecture 14

Canonical SDP Relaxation for CSPs*

14.1 Recalling the canonical LP relaxation

Last time, we talked about the canonical LP relaxation for a CSP. A CSP(Γ) is comprised of Γ, a collection of predicates R with label domain D. The canonical LP relaxation is comprised of two parts. Given an instance $\mathcal{C}$, with the domain of the variables being D, a constraint will be written as $C = (R, S) \in \mathcal{C}$. Then a solution to the LP relaxation contains two objects. First, for each $v \in V$, a probability distribution over labels for v. Formally, we have LP variables $(\mu_v[\ell])_{v \in V, \ell \in D}$ subject to

$$\forall v \in V, \qquad \sum_{\ell \in D} \mu_v[\ell] = 1$$
$$\forall \ell \in D, \quad \mu_v[\ell] \geq 0.$$

Second, for all $C = (R, S) \in \mathcal{C}$ we have a probability distribution λ_C over "local assignments" $S \to D$. These are similarly encoded with $\sum_C |D|^{|S|}$ many LP variables.

The objective function is

$$\max \sum_{C=(R,S)\in\mathcal{C}} w_c \mathop{\mathbf{Pr}}_{L\sim\lambda_C}[L(S) \text{ satisfies } R].$$

Finally, the thing that ties the μ's and the λ's together is the *consistent marginals condition* (a collection of linear equalities):

$$\forall C = (R, S) \in \mathcal{C} \quad \forall v \in S \quad \forall \ell \in D, \quad \mathop{\mathbf{Pr}}_{L\sim\lambda_C}[L(v) = \ell] = \mu_v[\ell].$$

We also showed that rounding the canonical LP relaxation of Max-SAT using plain randomized rounding achieved a $(1-1/e)$ approximation ratio. Recall that plain randomized rounding assigns to variables v in the following way:

$$F(v) = \begin{cases} 1 & : w.p. \quad \mu_v[1] \\ 0 & : w.p. \quad \mu_v[0] \end{cases}$$

*Lecturer: Ryan O'Donnell. Scribe: Jamie Morgenstern, Ryan O'Donnell.

The proof of this approximation factor looked at p_c, the probability a particular clause was satisfied by $L \sim \lambda_c$, and the probability that F satisfied that clause. In the last lecture it was shown that

$$\mathbf{Pr}[F \text{ satisfies } C] \geq 1 - \left(1 - \frac{p_c}{|S|}\right)^{|S|} \tag{14.1}$$

When $|S| \leq 2$, 14.1 $\geq (3/4)p_c$ which implies that this algorithm gets a 3/4-factor for clauses of length at most 2.

On the other hand, for clauses of length at least 2, the trivial random algorithm (assigning each variable to 1 or 0 with probability 1/2) satisfies 3/4 of clauses, yielding a 3/4 approximation. Can we get the best of both worlds, and combine the results for trivial random and plain randomized rounding of the LP relaxation to get a 3/4 approximation for Max-SAT?

The answer is yes, by combining the two assignment schemes. If we create our assignment F as

$$F(v) = \begin{cases} 1 & : w.p. \quad \text{avg}\{\mu_v[1], 1/2\} \\ 0 & : w.p. \quad \text{avg}\{\mu_v[0], 1/2\} \end{cases}$$

then F will satisfy 3/4 of clauses in expectation for Max-SAT. Showing this will be on the homework.

In fact it is possible to do better than a 3/4 approximation for various versions of Max-SAT. Below we give a laundry list of results proven using *SDPs* to improve this approximation ratio for Max-SAT.

$(\alpha_2(\beta), \beta)$ approximation for Max-2SAT, where	
$\alpha_2(\beta) \geq .940\beta$	[LLZ02]
and $\alpha_2(1-\epsilon) \geq 1 - O(\sqrt{\epsilon})$	[CMM07]
$(\frac{7}{8}\beta, \beta)$ for Max-3SAT,	[KZ97] (computer-assisted), [Zwi02] (computer-verified)
$(\frac{7}{8}, 1)$ for Max-4SAT	[HZ99]
$(.833\beta, \beta)$ for Max-SAT	[AW00]

It is reasonable to conjecture that there is a polynomial-time $(\frac{7}{8}\beta, \beta)$-approximation algorithm for Max-kSAT for any k.

14.2 Canonical CSP SDP relaxation

The SDP relaxation is similar to the LP relaxation, but with an important generalization. We will have exactly the same λ_C's for each constraint, and the same objective function. Rather than having the μ_v's, however, we'll have a *collection of joint real random variables* $(I_v[\ell])_{v \in V, \ell \in D}$. We will also have constraints which cause these random variables to hang together with the λ_C's in a gentlemanly fashion.

The random variables $I_v[\ell]$ will be called *pseudoindicator random variables*. We emphasize that they are jointly distributed. You should think of them as follows: there is a box, and

when you press a button on the side of the box ("make a draw"), out comes values for each of the $|V||D|$ random variables.

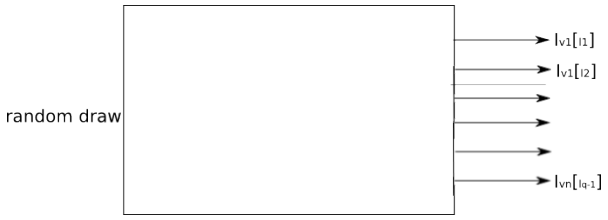


Figure 14.1: The pseudoindicator joint draw $I_v[l]$.

For now, never mind about how we actually represent these random variables or enforce conditions on them; we'll come to that later.

We'd love if it were the case that these pseudoindicator random variables were actual indicator random variables, corresponding to a genuine assignment $V \to D$. However, we can only enforce something weaker than that. Specifically, we will enforce the following two sets of conditions:

1. **Consistent first moments:**

$$\begin{aligned} \forall C = (R, S) \in \mathcal{C} \\ \forall v \in S \\ \forall \ell \in D \\ \mathop{\mathbf{Pr}}_{L \sim \lambda_C}[L(v) = \ell] = \mathbf{E}[I_v[\ell]] \end{aligned} \tag{14.2}$$

2. **Consistent second moments:**

$$\begin{aligned} \forall C = (R, S) \in \mathcal{C} \\ \forall v, v' \in S \\ \forall \ell, \ell' \in D \\ \mathop{\mathbf{Pr}}_{L \sim \lambda_c}[L(v) = \ell \wedge L(v') = \ell'] = \mathbf{E}\Big[I_v[\ell] \cdot I_{v'}[\ell']\Big] \end{aligned} \tag{14.3}$$

(We emphasize that v and v' need not be distinct, and ℓ and ℓ' need not be distinct.)

We also emphasize again that these pseudoindicator random variables are *n*ot independent, so the expected value of their product is not the product of their expected values.

We will show we can solve this optimally as an SDP (there are actually vectors "hiding inside the box"). Also, as explain more carefully in the next lecture, assuming the Unique

Games Conjecture the best polynomial-time approximation for *any* CSP is given by this SDP.

Now, a few remarks about this relaxation. First:

Remark 14.1. For all v, ℓ, $\mathbf{E}\Big[I_v[\ell]\Big] = \mathbf{E}\Big[I_v[l]^2\Big]$.

Proof. Consider any $C \ni v$. Apply (2) with $v = v'$, $\ell = \ell'$. Then, we have

$$\mathop{\mathbf{Pr}}_{L\sim\lambda_c}[L(v) = l \wedge L(v) = l] = \mathbf{E}\big[I_v[\ell]^2\big].$$

Of course also

$$\mathop{\mathbf{Pr}}_{L\sim\lambda_c}[L(v) = \ell \wedge L(v) = \ell] = \mathop{\mathbf{Pr}}_{L\sim\lambda_c}[L(v) = \ell].$$

Finally, apply (1) which says

$$\mathop{\mathbf{Pr}}_{L\sim\lambda_c}[L(v = \ell)] = \mathbf{E}\big[I_v[\ell]\big]$$

□

This is somewhat nice because $\mathbf{E}[I^2] = \mathbf{E}[I]$ is something satisfied by a genuinely 0-1-valued random variable I. In fact, our pseudoindicator random variables may take values outside the range $[0, 1]$. Still, they will at least satisfy the above.

Now, we will show that this "SDP relaxation" is in fact a relaxation (we still haven't explained why it's an SDP):

Theorem 14.2. $\mathrm{Opt}(\mathcal{C}) \le \mathrm{SDPOpt}(\mathcal{C})$

Proof. Let F be a legitimate (optimal) assignment. Then we can construct λ_C's and $I_v[\ell]$'s which achieve $\mathrm{Val}(F)$.

$$\lambda_C[L] = \begin{cases} 1 & : \text{ if } L \text{ is consistent with } F \\ 0 & : \text{ o/w} \end{cases}$$

Then, let $I_v[\ell]$ be the *constant* random variables

$$I_v[\ell] \equiv \begin{cases} 1 & : \text{ if } F(v) = \ell \\ 0 & : \text{ o/w} \end{cases}$$

It is easy to check that these λ_C's and $I_v[\ell]$'s satisfy the consistent first and second moment constraints and have SDP value equal to $\mathrm{Val}(F)$. □

Now, we show that the SDP relaxation is at least as tight as the LP relaxation.

Theorem 14.3. $\mathrm{SDPOpt}(\mathcal{C}) \le \mathrm{LPOpt}(\mathcal{C})$

Proof. Given an SDP solution $\mathcal{S}$ achieving SDPOpt, $\mathcal{S} = (\lambda_C, I_v[\ell])$, we must construct an LP solution and show its objective is no less than that of $\mathcal{S}$. Use the same λ_C's for the LP solution. Since the objective value depends only on the λ_C's, the objective value for the LP solution will be the same as the SDP value of $\mathcal{S}$. It remains to construct the distributions μ_v which are consistent with the λ_C's. Naturally, we set

$$\mu_v[\ell] = \mathbf{E}\Big[I_v[\ell]\Big].$$

Please note that this is indeed a probability distribution, because if we select any $C \ni v$ and apply (1), we get

$$\mathbf{E}\Big[I_v[\ell]\Big] = \Pr_{L\sim\lambda_C}[L(v) = \ell]$$

and the RHS numbers are coming from the genuine probability distribution $\lambda_C|_v$. The fact that the λ_C's and the μ_v's satisfy the LP's "consistent marginals" condition is equivalent to (1). □

Next, fix some $v \in V$. If the pseudoindicator random variables $(I_v[\ell])_{\ell\in D}$ were really legitimate constant random variables indicating a genuine assignment, we'd have $\sum_{\ell\in D} I_v[\ell] = 1$. In fact, this is true with probability 1 in any SDP solution:

Proposition 14.4. *Given a valid SDP solution, for any v, let $J = J_v = \sum_{\ell\in D} I_v[\ell]$. Then $J \equiv 1$.*

Proof. We will calculate the mean and variance of J. By linearity of expectation,

$$\mathbf{E}[J] = \sum_{\ell} \mathbf{E}[I_v[\ell]] = 1.$$

And,

$$\mathbf{E}[J^2] = \mathbf{E}\left[\left(\sum_{\ell} I_v[\ell]\right)\left(\sum_{\ell'} I_v[\ell']\right)\right]$$

By linearity of expectation, this is just

$$= \sum_{\ell,\ell'} \mathbf{E}\Big[I_v[\ell]\cdot I_v[\ell']\Big]$$

Choose any constraint $C \ni v$. By (2), we have

$$= \sum_{\ell,\ell'} \Pr_{L\sim\lambda_C}[L(v) = \ell \wedge L(v) = \ell']$$

Here every term with $\ell \neq \ell'$ is 0. So this reduces to

$$= \sum_{\ell} \Pr_{L\sim\lambda_c}[L(v) = \ell]$$

$$= 1$$

Then, computing the variance of J:

$$\mathbf{Var}[J] = \mathbf{E}[J^2] - \mathbf{E}[J]^2 = 0$$

Any random variable with zero variance is a constant random variable, with value equal to its mean. Thus, $J \equiv 1$. □

Theorem 14.5. *Condition (1) in the SDP is superfluous, in the sense that dropping it leads to an equivalent SDP (equivalent meaning that the optimum is the same for all instances).*

Proof. On the homework. □

Given the above theorem, we focus for a while on the optimization problem in which joint pseudoindicator random variables only need to satisfy (2). Let's now answer the big question: how is this optimization problem an SDP?

14.3 Why is it an SDP and how do we construct the pseudoindicators?

Let us define the numbers

$$\sigma_{(v,\ell),(v',\ell')} = \mathbf{E}\Big[I_v[\ell] \cdot I_{v'}[\ell'].\Big]$$

As this notation suggests, we will define a matrix Σ from these numbers. It will be an $N \times N$ matrix (for $N = |V||D|$), with rows and columns indexed by variable/label pairs:

$$\Sigma = (\sigma_{(v,\ell),(v',\ell')})$$

Now let us ask what the consistent second moments condition (2) is saying? The second moments constraint is satisfied if and only there exists a collection of N random variables (the pseudoindicators) whose second moment matrix is Σ. But, if you recall, this is equivalent definition #5 from Lecture 10 of PSD-ness of the matrix Σ. Thus our optimization problem — which has linear constraints on the variables λ_C and $\sigma_{(v,\ell),(v',\ell')}$, together with the condition that Σ is PSD — is indeed an SDP!

We still need to discuss how to actually "construct/sample from" pseudoindicator random variables ($I_v[\ell]$) corresponding to the Ellipsoid Algorithm's output Σ. It's much like in the beginning of the Goemans–Williamson algorithm: given Σ PSD, you compute (a very accurate approximation to) a matrix $U \in \mathbb{R}^{N \times N}$ such that $U^\top U = \Sigma$. The columns of U are vectors $\vec{y}_{v,\ell} \in \mathbb{R}^N$ such that $\vec{y}_{v,\ell} \cdot \vec{y}_{v',\ell'} = \sigma_{(v,\ell),(v',\ell')}$. How does this help?

The key idea is that **you can think of a vector as a random variable, and a collection of vectors as a collection of joint random variables**. How? A vector $\vec{y} \in \mathbb{R}^N$ defines a random variable Y as follows: to get a draw from Y, pick $i \in [N]$ uniformly at random and then output $Y = \vec{y}_i$. A collection of vectors

$$\vec{y}^{(1)}, \ldots, \vec{y}^{(d)}$$

defines a collection of jointly distributed random variables

$$Y^{(1)}, \ldots, Y^{(d)}$$

as follows: to get one draw from (all of) the $Y^{(j)}$'s, pick $i \in [N]$ uniformly at random and then output $Y^{(j)} = (\vec{y}^{(j)})_i$ for each $j \in [d]$. In this way, we can view the vectors that the SDP solver outputs (more precisely, the vectors gotten from the columns of the factorization $U^\top U = \Sigma$), namely

$$\vec{y}_{(v_1,\ell_1)}, \ldots, \vec{y}_{(v_n,\ell_q)},$$

as the collection of jointly distributed pseudoindicators,

$$Y_{v_1}[\ell_1], \ldots, Y_{v_n}[\ell_q].$$

Why does this work? The idea is that "**inner products are preserved**" (up to a trivial scaling):

Observation 14.6. Given vectors $\vec{y}, \vec{y}' \in \mathbb{R}^N$, the equivalent random variables Y, Y' satisfy:

$$\mathbf{E}[YY'] = \sum_{i=1}^{N} \frac{1}{N} \vec{y}_i \vec{y}'_i = \frac{1}{N} \vec{y} \cdot \vec{y}'$$

We'll make a slight definition to get rid of this annoying scaling factor:

Definition 14.7. We introduce the scaled inner product

$$\langle\langle \vec{y}, \vec{y}' \rangle\rangle := \frac{1}{N} \vec{y} \cdot \vec{y}'$$

Solving the SDP is equivalent to coming up with the pseudoindicator random variables, with this slight need to scale. Given $\vec{y}_{v,\ell}$ as in the original SDP, we define

$$\vec{z}_{v,\ell} = \sqrt{N} \vec{y}_{v,\ell}$$

Then,

$$\langle\langle \vec{z}_{v,\ell}, \vec{z}_{v',\ell'} \rangle\rangle = N \langle\langle \vec{y}_{v,\ell}, \vec{y}_{v',\ell'} \rangle\rangle = \vec{y}_{v,\ell} \cdot \vec{y}_{v',\ell'} = \sigma_{(v,\ell),(v',\ell')}$$

So actually, the joint random variables corresponding to this collection of vectors $\vec{z}_{v,l}$'s will be the pseudoindicator random variables.

14.4 Summary

There are several equivalent perspectives on the canonical SDP relaxation for a CSP.

- Pseudoindicator random variables which satisfy the first and second moment consistency constraints. This perspective is arguably best for understanding the SDP. for using an SDP solver

- Pseudoindicator random variables which just satisfy the consistent second moments constraints. This perspective is arguably best when constructing SDP solutions by hand.
- Vectors $(\vec{y}_{v,\ell})$ satisfying the first and second "moment" consistency constraints. This perspective is the one that's actually used computationally, on a computer.

There is one more equivalent perspective that we will see in the next lecture, which is arguably the best perspective for developing "SDP rounding algorithms":

- *Jointly Gaussian* pseudoindicator random variables which satisfy the consistent first and second moment constraints.

In the next lecture we will see how to make the pseudoindicators jointly Gaussian, and why this is good for rounding algorithms.

Lecture 16

The Multiplicative Weights Algorithm*

In the next couple lectures, we will devise modular, iterative algorithms for solving LPs and SDPs. "Multiplicative weights" is a retronym for the simple iterative rule underlying these algorithms; it is known by different names in the various fields where it was (re)discovered. Check out the survey by Arora, Hazan and Kale [AHK05]; our discussion will be based on their treatment. Due to its broad appeal, we will consider multiplicative weights in more generality than we need for solving LPs and SDPs. In this lecture, we'll introduce some strategies for playing a prediction game. We'll tweak the game to suit our optimization needs. Finally, we'll play the tweaked game with a strategy called Hedge.

16.1 Warmup: prediction with expert advice

The following sequential game is played between an omniscient Adversary and an Aggregator who is advised by N experts. Special cases of this game include predicting if it will rain tomorrow, or if the stock market will go up or down.

For $t = 1, \ldots, T$:

1. Each expert $i \in [N]$ advises either yes or no.
2. Aggregator predicts either yes or no.
3. Adversary, with knowledge of the expert advice and Aggregator's prediction, decides the yes/no outcome.
4. Aggregator observes the outcome and suffers if his prediction was incorrect.

Naturally, Aggregator wants to make as few mistakes as possible. Since the experts may be unhelpful and the outcomes may be capricious, Aggregator can hope only for a relative

*Lecturer: Anupam Gupta. Scribe: Shiva Kaul.

performance guarantee. In particular, Aggregator hopes to do as well as the best single expert in hindsight [1]. In order to do so, Aggregator must track which experts are helpful. We will consider a few tracking strategies. Almost every other aspect of the game - that advice is aggregated into a single value, that this value is binary, and even that the game is sequential - is not relevant; we will generalize or eliminate these aspects.

If there is a perfect expert, then an obvious strategy is to dismiss experts who aren't perfect. With the remaining experts, take a majority vote. Every time Aggregator makes a mistake, at least half of the remaining experts are dismissed, so Aggregator makes at most $\log_2 N$ mistakes. We can use the same strategy even when there isn't a perfect expert, if we restart after every expert has been eliminated. If the best expert has made M mistakes by time T, then Aggregator has restarted at most $M+1$ times, so it has made at most $(M+1)\log_2 N$ mistakes. This bound is rather poor since it depends multiplicatively on M.

16.1.1 Fewer mistakes with Weighted Majority

We may obtain an additive mistake bound by softening our strategy: instead of dismissing experts who erred, discount their advice. This leads to the Weighted Majority algorithm of Littlestone and Warmuth [LW89]. Assign each expert i a weight $w_i^{(1)}$ initialized to 1. Thereafter, for every t:

- Predict yes/no based on a weighted majority vote per $\vec{w}^{(t)} = (w_1^{(t)}, \ldots, w_N^{(t)})$
- After observing the outcome, for every mistaken expert i, set $w_i^{(t+1)} = w_i^{(t)}/2$

Theorem 16.1. *For any sequence of outcomes, duration T and expert i,*

$$\#\text{ of WM mistakes} \leq 2.41 \cdot (\#\text{ of } i\text{'s mistakes}) + \log_2 N$$

Proof. Let

$$\Phi^{(t)} = \sum_{i \in [N]} w_i^{(t)}$$

be a 'potential' function. Observe the following facts:

- By definition, $\Phi^{(1)} = N$
- Also by definition, $\frac{1}{2}^{\#\text{ of } i\text{'s mistakes}} \leq \Phi^{(T+1)}$
- At any τ when WM errs, at least half of the weight gets halved:

$$\Phi^{(\tau+1)} \leq \frac{3}{4}\Phi^{(\tau)}$$

This implies

$$\Phi^{(T+1)} \leq \frac{3}{4}^{\#\text{ of } i\text{'s mistakes}} \cdot \Phi^{(1)}$$

[1] The excess number of mistakes is called (external) regret.

Combining these facts yields

$$\frac{1}{2}^{\#\text{ of } i\text{'s mistakes}} \leq \frac{3}{4}^{\#\text{ of WM mistakes}} \cdot N$$

Taking logarithms of both sides,

$$-(\#\text{ of } i\text{'s mistakes}) \leq \log_2 N + \log_2(3/4) \cdot \#\text{ of WM mistakes}$$

so finally

$$\#\text{ of WM mistakes} \leq (1/\log_2(4/3)) \cdot (\#\text{ of } i\text{'s mistakes}) + \log_2 N$$

□

The unseemly leading constant is a consequence of our arbitrary choice to halve the weights. If we optimize ϵ in the update rule

$$w_i^{(t+1)} = w_i^{(t)}/(1+\epsilon)$$

then we may achieve

$$\#\text{ of WM mistakes} \leq 2(1+\epsilon) \cdot (\#\text{ of } i\text{'s mistakes}) + O(\log N/\epsilon).$$

16.2 Tweaking the game

We now modify the game with a view to solving LPs and SDPs. We perform these modifications individually in order to dispel some seductive misconceptions about the new game's meaning. The impervious (or impatient) reader may skip to the game description at the end of the section.

The first modification bakes weighting into the game.

For $t = 1, \ldots, T$:

1. Each expert $i \in [N]$ advises either yes or no.
2. Allocator picks some distribution $\vec{p}^{(t)} = (p_1^{(t)}, \ldots, p_N^{(t)})$ over the experts.
3. Adversary, with knowledge of the expert advice and $\vec{p}^{(t)}$, decides the yes/no outcome.
4. Allocator observes the outcome.
5. A single expert is sampled from $\vec{p}^{(t)}$ but isn't revealed to either Allocator or Adversary. Allocator suffers if this expert errs.

Let $m_i^{(t)}$ be 1 if expert i erred at t, and 0 otherwise. The new goal is to bound his total expected number of mistakes

$$\sum_t \vec{p}^{(t)} \cdot \vec{m}^{(t)} \tag{16.1}$$

in terms of the total number of mistakes made by any single expert

$$\sum_t m_i^{(t)}$$

Note the sampled expert isn't revealed to either party. By arguments posted on the blog [Gup11], Adversary may declare the entire sequence of outcomes in advance without losing any power. Eliminating the sequential nature of the game was on our agenda.

The attentive reader recalls that eliminating the aggregation step was also on our agenda. Yet this section has introduced a new aggregation step: randomly choosing a single expert rather than taking a deterministic weighted-majority vote. The truly important change was not randomized aggregation, but rather Allocator's new goal of minimizing 16.1. This quantity may be interpreted as the expected number of mistakes of a randomized Aggregator, but it is still well-defined even if there's no aggregation [2]. We consider $\vec{p}^{(t)}$ to be chosen deterministically; randomized aggregation may be layered on top.

Finally, we replace binary mistakes with continuous costs. Rather than returning a yes/no outcome which induces a mistake vector in $\{0,1\}^N$, Adversary returns a cost vector in $[-1,1]^N$. Negative cost may be interpreted as benefit. As we will see, $[-\rho,\rho]$ could work as well. Congruently, each expert advises some value in $[-1,1]$ rather than yes/no.

In summary, the game proceeds as follows.

For $t = 1, \ldots, T$:

1. Each expert $i \in [N]$ advises some value in $[-1,1]$.
2. Allocator picks some distribution $\vec{p}^{(t)} = (p_1^{(t)}, \ldots, p_N^{(t)})$ over the experts.
3. Adversary, with knowledge of the expert advice and $\vec{p}^{(t)}$, determines a cost vector $\vec{m}^{(t)} = (m_1^{(t)}, \ldots, m_N^{(t)}) \in [-1,1]^N$.
4. Allocator observes the cost vector and suffers $\vec{p}^{(t)} \cdot \vec{m}^{(t)}$.

16.3 Hedge and a useful corollary

We play the new game with the Hedge strategy of Freund and Schapire [FS97]. Its exponential update rule distinguishes it from Weighted Majority. Assign each expert i a weight $w_i^{(1)}$ initialized to 1. At each time t:

[2] It's also still useful. In the next lecture, the 'experts' correspond to individual constraints of an LP or SDP. Higher weight is given to constraints satisfied by thin margins. The convex combination of constraints is a single 'summary' constraint which emphasizes the difficult constraints. Reducing many constraints to a single summary constraint will be algorithmically useful.

- Pick the distribution $p_j^{(t)} = w_j^{(t)}/\Phi^{(t)}$
- After observing the cost vector, set $w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\epsilon \cdot m_i^{(t)})$

The following theorem may be interpreted as "the total expected cost of Hedge is not much worse than the total cost of any individual expert."

Theorem 16.2. *Suppose $\epsilon \leq 1$ and for $t \in [T]$, $\vec{p}^{(t)}$ is picked by Hedge. Then for any expert i,*

$$\sum_{t \leq T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} \leq \sum_{t \leq T} m_i^{(t)} + \frac{\ln N}{\epsilon} + \epsilon T$$

Proof. This proof also involves the potential function Φ. By definition,

- $\Phi^{(1)} = N$.
- $\Phi^{(T+1)} \geq w_i^{(T+1)} = \exp(-\epsilon \sum_{t \leq T} m_i^{(t)})$

Again by definition,

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_j w_j^{(t+1)} \\
&= \sum_j w_j^{(t)} \cdot \exp(-\epsilon m_j^{(t)})
\end{aligned}$$

The exponentiated term is in $[-1, 1]$. Since $e^x \leq 1 + x + x^2$ for $x \in [-1, 1]$,

$$\begin{aligned}
&\leq \sum_j w_j^{(t)}(1 - \epsilon m_j^{(t)} + \epsilon^2 (m_j^{(t)})^2) \\
&\leq \sum_j w_j^{(t)}(1 - \epsilon m_j^{(t)} + \epsilon^2) \\
&= \sum_j w_j^{(t)}(1 + \epsilon^2) - \sum_j w_j^{(t)} \cdot \epsilon \cdot m_j^{(t)} \\
&= \Phi^{(t)}(1 + \epsilon^2) - \epsilon \sum_j \Phi^{(t)} \cdot p_j^{(t)} \cdot m_j^{(t)} \\
&= \Phi^{(t)}(1 + \epsilon^2 - \epsilon(\vec{p}^{(t)} \cdot \vec{m}^{(t)})) \\
&\leq \Phi^{(t)} \cdot \exp(\epsilon^2 - \epsilon \vec{p}^{(t)} \cdot \vec{m}^{(t)})
\end{aligned}$$

Combining these statements yields

$$\exp(-\epsilon \sum_t m_i^{(t)}) \leq \Phi^{(T+1)} \leq \Phi^{(1)} \cdot \exp(\epsilon^2 T - \epsilon \sum_t \vec{p}^{(t)} \cdot \vec{m}^{(t)})$$

Taking (natural) logarithms,

$$-\epsilon \sum_t m_i^{(t)} \leq \ln \Phi^{(1)} + \epsilon^2 T - \epsilon \sum_t \vec{p}^{(t)} \cdot \vec{m}^{(t)})$$

The final result follows after some rearranging. □

In the next lecture, we will use an 'average cost' corollary of the previous result.

Corollary 16.3. *Suppose $\epsilon \leq 1$ and for $t \in [T]$, $p^{(t)}$ is picked by Hedge in response to cost vectors $\vec{m}^{(t)} \in [-\rho, \rho]^N$. If $T \geq (4\rho^2 \ln N)/\epsilon^2$, then for any expert i:*

$$\frac{1}{T} \sum_t \vec{p}^{(t)} \cdot \vec{m}^{(t)} \leq \frac{1}{T} \sum_t m_i^{(t)} + 2\epsilon$$

Its extension to cost vectors in $[-\rho, \rho]^N$ is simple: run Hedge on cost vectors normalized within $[-1, 1]$, and then scale up the bound.

16.3.1 Multiplicative weights

For completeness, we will mention the update rule which is most closely associated with the term 'multiplicative weights':

$$w_i^{(t+1)} = w_i^{(t)}(1 - \epsilon m_i^{(t)})$$

This update rule achieves a mistake bound of:

$$\sum_{t \leq T} \vec{p}^{(t)} \cdot \vec{m}^{(t)} \leq \sum_{t \leq T} m_i^{(t)} + \frac{\ln N}{\epsilon} + \epsilon \sum_t |m_i^{(t)}|$$

Since $\sum_t |m_i^{(t)}|$ may be smaller than T, this improves upon Hedge for benign cost vectors.

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1

Lecture 17

Solving LPs/SDPs using Multiplicative Weights*

In the last lecture we saw the Multiplicative Weights (MW) algorithm and how it could be used to effectively solve the experts problem in which we have many experts and wish to make predictions that are approximately as good as the predictions made by the best expert. In this lecture we will see how to apply the MW algorithm to efficiently approximate the optimal solution to LPs and SDPs.

17.1 Multiplicative Weights

Recall the following result from Lecture 16 about the "Hedge" algorithm:

Theorem 17.1. *Suppose the cost vectors are $\bar{m}^{(t)} \in [-1,1]^N$. Then for any $\epsilon \leq 1$, and for any T, the Hedge algorithm guarantees that for all $i \in [m]$,*

$$\sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon + \frac{\ln N}{\epsilon}$$

So the total cost paid by the algorithm is no more than an additive factor of $\epsilon + \frac{\ln N}{\epsilon}$ worse than the cost incurred by any individual component of the cost vector. Theorem 17.1 implies a similar result for the *average* cost incurred per round. (One can get a similar result for the MW algorithm, where instead of the update rule $w_i^{(t)} \leftarrow w_i^{(t)} \cdot \exp(-\epsilon m_i^{(t)})$, we used the rule $w_i^{(t)} \leftarrow w_i^{(t)} \cdot (1 - \epsilon m_i^{(t)})$.)

Corollary 17.2. *Suppose the cost vectors are $\bar{m}^{(t)} \in [-\rho, \rho]^N$. Then for any $\epsilon \leq \frac{1}{2}$, and for any $T \geq \frac{4 \ln N}{\epsilon^2} \rho^2$, the Hedge algorithm guarantees than for all $i \in [m]$*

$$\frac{1}{T} \sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

*Lecturer: Anupam Gupta. Scribe: Tim Wilson.

Note: We did not cover this in lecture, but one can show that if the cost vectors are in $[0, \rho]$, then using the MW algorithm, the setting $T \geq \frac{4 \ln N}{\epsilon^2}\rho$ suffices to get the same guarantee of

Lemma 17.3. *Suppose the cost vectors are $\bar{m}^{(t)} \in [0, \rho]^N$. Then for any $\epsilon \leq \frac{1}{2}$, and for any $T \geq \frac{4 \ln N}{\epsilon^2}\rho$, the MW algorithm guarantees than for all $i \in [m]$*

$$\frac{1}{T}\sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T}\sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

A proof of this can be found in the Arora, Hazan, and Kale survey [AHK05].

17.2 Solving LPs with Multiplicative Weights

We will use the MW algorithm to help solve LPs with m constraints of the form

$$\begin{aligned} \min\ & c^\top x \\ \text{s.t. } & Ax \geq b \\ & x \geq 0 \end{aligned}$$

Supposing that we know $c^\top x^* = \text{OPT}$ (by binary search), we will aim to find an ϵ-approximate solution $\widetilde{x}$ such that

$$\begin{aligned} c^\top \widetilde{x} &= \text{OPT} \\ A\widetilde{x} &\geq b - \epsilon \mathbf{1} \\ \widetilde{x} &\geq 0 \end{aligned}$$

or output "infeasible" if no solution exists. The runtime for this will be $O\left(\frac{\rho^2 \log m}{\epsilon^2}\right)$ where ρ is the "width" of the LP which will be defined shortly.

17.2.1 Simplifying the Constraints

Instead of searching for solutions $x \in \mathbb{R}^n$, we will package together the "easy" constraints into the simple convex region

$$K = \{x \in \mathbb{R}^n \mid x \geq 0, c^\top x = \text{OPT}\}$$

Now we wish to solve $Ax \geq b$ such that $x \in K$. Note that this is particularly easy to solve if $Ax \geq b$ is only one constraint, i.e., we are trying to determine whether $\exists x \in K$ such that $\alpha^\top x \geq \beta$ for some $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}$. For example, if $c \geq 0$ and

$$\max_i \alpha_i \frac{\text{OPT}}{c_i} \geq \beta$$

we can set $x = \frac{\text{OPT}}{c_i} e_i$ which will satisfy our constraints; else we could output `Infeasible`. For general c we are essentially reduced to solving an LP over two constraints, which while not as trivial as this, is still simple.

We will henceforth assume we have an oracle that given $\alpha \in \mathbb{R}^n, \beta \in \mathbb{R}$, and $K \subseteq \mathbb{R}^n$ either returns $x \in \mathbb{R}^n$ such that $\alpha^\top x \geq \beta$, or correctly asserts that there is no such x.

17.2.2 Using Multiplicative Weights

We will use this oracle that allows us to satisfy one constraint ($\alpha x \geq \beta$) for $k \in K$, along with the MW algorithm to get an algorithm satisfy all of the constraints $Ax \geq b$ for $x \in K$.

Each of the constraints $a_i^\dagger topx \geq b_i$ will be viewed as an "expert" for a total of m experts. Each round we will produce a vector $\bar{p}^{(t)}$ that will give us a convex combination of the constraints as follows

$$\underbrace{\bar{p}^{(t)} \cdot A}_{\alpha^{(t)}} x \geq \underbrace{\bar{p}^{(t)} \cdot b}_{\beta^{(t)}}$$

Using our oracle, we can determine whether $\alpha^{(t)} x \geq \beta^{(t)}$ has some solution $x^{(t)} \in K$, or if no such solution exists. Clearly if no solution exists, then $Ax \geq b$ is infeasible over K, so our LP is infeasible. (It's easy to see the contrapositive: if there were a solution to $Ax \geq b, x \in K$, then this vector x would also satisfy $\alpha^{(t)} x \geq \beta^{(t)}$; here we use the fact that $\bar{p}^{(t)} \geq 0$.) Moreover, the vector $\bar{p}^{(t)}$ serves as proof of this infeasibility.

Otherwise, we will set our cost vector so that $\bar{m}_i^{(t)} = a_i x^{(t)} - b_i$, update our weights and proceed with the next round. If we have not determined the LP to be infeasible after T rounds we will terminate and return the solution

$$\widetilde{x} = \frac{1}{T} \sum_{t \leq T} x^{(t)}$$

Why do we set our cost vectors this way? It almost seems like we should incur no cost when $a_i x^{(t)} - b_i \geq 0$ (i.e., when we satisfy this constraint), whereas we are incurring a higher cost the more we satisfy it. Well, the idea is whenever $a_i^{(t)} x - b_i$ is positive, we have oversatisfied the constraint. Giving a positive cost to this constraint causes us to reduce the weight of this constraint in this next round. This works analogously to the experts problem where an expert who is wrong (has high cost) is given less credence (less weight) in future rounds. Similarly, for any constraint in which $a_i^{(t)} x - b_i$ is negative, we have failed the constraint. Giving a negative cost to this constraint causes us to increase the weight of this constraint in the next round.

Initially we set all of our weights equal to express our ignorance; "all constraints are equally hard". Whenever we update our weights we reduce the weights of constraints we oversatisfied so we'll cover them less in future rounds. We increase the weights of constraints we didn't satisfy so we'll cover them more in future rounds. Our hope is that over time this will converge to a solution where we satisfy all constraints to a roughly equal extent.

17.2.3 Analyzing Multiplicative Weights

Supposing that we do not discover our LP is infeasible, how many rounds should we run and how good will our solution be? If we define

$$\rho = \max\{1, \max_{i, x \in K} \{|a_i^\top x - b_i|\}\}$$

to be the maximum magnitude of any cost assigned to a constraint, then we may immediately apply Corollary 17.2 to find that after $T \geq \frac{4 \ln n}{\epsilon^2} \rho^2$ rounds,

$$\frac{1}{T} \sum_{t \leq T} \bar{p}^{(t)} \cdot \bar{m}^{(t)} \leq \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} + \epsilon$$

where $\epsilon \leq \frac{1}{2}, \bar{m}^{(t)} = a_i^\top x^{(t)} - b_i \in [-\rho, \rho]^n$ for all $i \in [m]$, and each $x^{(i)} \in K$. Note that we do not actually need to find ρ; it suffices to keep track of $\rho_t = \max\{1, \max_{i,t' \leq t}\{|a_i^\top x^{(t')} - b_i|\}\}$, the maximum cost seen so far, and run until $T \geq \frac{4 \ln n}{\epsilon^2} \rho_T^2$.

What guarantee do we get? On the left hand side of this inequality we have

$$\begin{aligned} \bar{p}^{(t)} \cdot \bar{m}^{(t)} &= \bar{p}^{(t)} \cdot (Ax^{(t)} - b) \\ &= \bar{p}^{(t)} \cdot Ax^{(t)} - \bar{p}^{(t)} \cdot b \\ &\geq 0 \end{aligned}$$

where the final inequality holds due to our oracle's properties. Therefore the left hand side is at least 0. And on the right hand side we have

$$\begin{aligned} \frac{1}{T} \sum_{t \leq T} \bar{m}_i^{(t)} &= \frac{1}{T} \sum_{t \leq T} a_i^\top x^{(t)} - b_i \\ &= a_i^\top \left(\frac{1}{T} \sum_{t \leq T} x^{(t)} \right) - b_i \\ &= a_i^\top \widetilde{x} - b_i \end{aligned}$$

Combining this with our inequality for the right hand side we get

$$\begin{aligned} \forall i : \quad a_i^\top \widetilde{x} - b_i + \epsilon &\geq 0 \\ a_i^\top \widetilde{x} &\geq b_i - \epsilon \end{aligned}$$

Therefore we can obtain an ϵ-feasible solution to $Ax \geq b, x \in K$ in time $O\left(\frac{\log m}{\epsilon^2} \rho^2\right)$ time where $\rho = \max\{1, \max_{i, x \in K}\{|a_i^\top x - b_i|\}\}$ is the width of the LP.

17.2.4 Example: Minimum Set Cover

Recall the minimum fractional set cover problem with m sets $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$ and n elements U. The goal is to pick fractions of sets in order to cover each element to an extent of 1: i.e., to solve the following LP—

$$\begin{aligned} \min & \sum_S x_S \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1 \quad \forall e \\ & x_S \geq 0 \end{aligned}$$

Suppose we know OPT $= L \in [1, m]$, so $K = \{\sum_S x_S = L, x_S \geq 0\}$. We want to find $x \in K$ such that $\sum_{S \ni e} x_S \geq 1$ for all elements e. Our oracle, given some $\bar{p}$, must try to find $x \in K$ such that

$$\begin{aligned} & \sum_e \bar{p}_e \sum_{S \ni e} x_S \geq \sum_e \bar{p}_e \cdot 1 = 1 \\ \iff & \sum_S x_S \sum_{e \in S} \bar{p}_e \geq 1 \\ \iff & \sum_S x_S \cdot p(S) \geq 1 \end{aligned}$$

where $p(S)$ is the total weight of elements in S. This quantity is clearly maximized over K by concentrating on a set with the maximum weight and setting

$$x_S = \begin{cases} L & \text{for some } S \in \mathcal{F} \text{ maximizing } p(S) \\ 0 & \text{for all other } S \end{cases}$$

Note that the width of this LP is at most

$$\max_e \sum_{S \ni e} x_S - 1 \leq L - 1 \leq m - 1$$

How does the weight update step work? Initially we set $w_i^{(1)}$ for all constraints. Whenever a set is overcovered, we reduce the weight of that set so we don't try as hard to cover it in the next step. Whenever a set is undercovered we increase the weight of the set so we try harder to cover it in the next step. Now, after $4L^2 \ln n/\epsilon^2$ steps we will obtain an ϵ-approximate solution $\widetilde{x}$ such that

$$\begin{aligned} \sum_S \widetilde{x}_S &= L \\ \sum_{S \ni e} \widetilde{x}_S &\geq 1 - \epsilon \\ \widetilde{x} &\geq 0 \end{aligned}$$

Note that, in this case, the constraint matrix is completely nonnegative, and we can scale up our solution to get a feasible solution $\widehat{x} = \widetilde{x}/(1-\epsilon)$ so that

$$\begin{aligned} \sum_S \widehat{x}_S &= \frac{L}{1-\epsilon} \approx L(1+\epsilon) \\ \sum_{S \ni e} \widehat{x}_S &\geq 1 \\ \widetilde{x} &\geq 0 \end{aligned}$$

17.2.5 Comments

1. The scaling we used for minimum set cover to obtain a non-optimal, feasible solution can be applied to any LP where $b > \mathbf{1}\epsilon$—indeed, we could just multiply all the x values by $\max_i 1/(b_i - \epsilon)$. This is often useful, particularly when we're going to round this LP solution and incur further losses, and hence losing this factor may be insignificant.

2. If the constraint matrix A is all positive the problem is said to be a *covering problem* (we are just interested in putting enough weight on x to cover every constraint). If the constraint matrix is all negative—or equivalently, if we have $Ax \leq b$ with an all-positive matrix A—the problem is said to be a *packing problem* (we are packing as much weight into x as possible without violating any constraint). In either case, we can use a similar scaling trick to get a non-optimal, feasible solution.

   In this case we can reduce the run-time further. Assume we have a covering problem: $\min\{c^\top x \mid Ax \geq b, x \geq 0\}$. By scaling, we can transform this into a problem of the form
   $$\min\{c^\top x \mid Ax \geq 1, x \geq 0\}$$
   The uniform values of $b_i = 1$ allows us to set the cost vectors $m_i^{(t)} = a_i^\top x^{(t)}$ instead of $m_i^{(t)} = a_i^\top x^{(t)} - 1$; this translation does not change the algorithm. But the positive cost vectors allow us to use Lemma 17.3 to reduce the runtime from $O\left(\frac{\log m}{\epsilon^2}\rho^2\right)$ to $O\left(\frac{\log m}{\epsilon^2}\rho\right)$.

3. In general, the width of our LPs may not turn out to be as nice. For example, in the *weighted* minimum set cover problem
   $$\begin{aligned} \min & \sum_S c_S x_S \\ \text{s.t.} & \sum_{S \ni e} x_S \geq 1 \quad \forall e \\ & x_S \geq 0 \end{aligned}$$
   our optimum, and hence the width, can increase to as much as $m \cdot \frac{\max_S c_S}{\min_S c_S}$. An approach developed by Garg and Könemann [GK07] can be useful to solve the problems without the width penalty.

4. The MW algorithm does not need a perfect oracle. Being able to determine given $\alpha \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ if there is no $x \in K$ with $\alpha^\top x \geq \beta$, or else returning an $x \in K$ such that $\alpha^\top x \geq \beta - \epsilon'$ is sufficient for our purposes. This gives us solutions $\widetilde{x} \in K$ such that
   $$Ax \geq b - (\epsilon + \epsilon')\mathbf{1}.$$

5. There was exactly one point where we used the fact that our constraints were linear. That was concluding that
   $$\frac{1}{T}\sum_{t \leq T} a_i^\top x^{(t)} - b_i = a_i^\top \widetilde{x} - b_i$$

However, we can make a similar claim for any set of convex constraints as well: if we wanted to find $x \in K$ such that $f_i(x) \leq 0$ for $i \in [m]$, with the f_i's convex. Then as long as we could solve the oracle and find $x \in K$ with $\sum_i p_i^{(t)} f_i(x) \leq 0$ efficiently, the rest of the argument would go through. In particular, in the step where we used linearity, we could instead use

$$\frac{1}{T}\sum_{t \leq T} f_i(x^{(t)}) \leq f_i\left(\frac{1}{T}\sum_{t \leq T} x^{(t)}\right) = f_i(\widetilde{x}).$$

17.3 Solving SDPs with Multiplicative Weights

Suppose we now move to solving SDPs of the form

$$\begin{aligned} \min C \bullet X \\ \text{s.t. } A_i \bullet X \geq b_i \\ X \succeq 0 \end{aligned}$$

note that the first few constraints are linear constraints. It is only the psd-ness constraint that is non-linear—so we only need to modify our MW algorithm by absorbing the $X \succeq 0$ constraint into the oracle. It will be also convenient to require the constraint $\mathrm{tr}(X) = 1$ as well: usually we can guess the trace of the solution X. (If the trace of the solution we seek is not 1 but R, we can scale the problem by R to get unit trace.) Then the oracle we must implement is this:

> Let $K := \{X \mid X \succeq 0, \mathrm{tr}(X) = 1\}$. Given a symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\beta \in \mathbb{R}$, does there exist $X \in K$ such that $\mathbf{A} \bullet X \geq \beta$?

(Again, $\mathbf{A}, \beta$ will be obtained in the algorithm by setting $\mathbf{A}^{(i)} := p_i^{(t)} A_i$, and $\beta^{(i)} := p_i^{(t)} b_i$.) But we know from Lecture 12 that this is equivalent to asking whether the maximum eigenvalue of the symmetric matrix $\mathbf{A}$ is at least β. Indeed, if this is so, and if $\lambda_{\max}$ is the maximum eigenvalue of $\mathbf{A}$ with unit eigenvector x, then

$$\begin{aligned} \mathbf{A} \bullet (xx^\top) &= \mathrm{tr}(\mathbf{A}^\top xx^\top) \\ &= \mathrm{tr}(\mathbf{A}xx^\top) \\ &= \mathrm{tr}(\lambda_{\max} xx^\top) \\ &= \lambda_{\max} \end{aligned}$$

so our oracle should return $X = xx^\top$, else it should return `Infeasible`. Moreover, using the Observation #4 on the previous page, it suffices to return x such that $x^\top \mathbf{A} x \geq \lambda_{\max} - \epsilon$. How fast this can be done depends on the particular structure of the matrix $\mathbf{A}$; in the next section we see that for the max-cut problem, the matrix $\mathbf{A}$ itself is psd, and hence we can find such an x relatively quickly.

17.3.1 Example: Max Cut

This part is loosely based on the paper of Klein and Lu [KL96]. Recall the Max Cut SDP we derived in Lecture 12:

$$\begin{aligned} \max \frac{1}{4} L \bullet X \\ \text{s.t. } (e_i e_i^\top) \bullet X = 1 \quad \forall i \\ X \succeq 0 \end{aligned}$$

As usual, we will think of the edge weights as summing to 1: this means that $\mathrm{tr}(L) = \sum_i L_{ii} = -\sum_{i \neq j} L_{ij} = 1$. If we let $b = \mathrm{OPT}$ and scale X by $1/n$, we are looking for feasibility of the constraints:

$$\begin{aligned} \frac{n}{4b} L \bullet X \geq 1 \\ n(e_i e_i^\top) \bullet X = 1 \quad \forall i \\ X \succeq 0 \end{aligned}$$

Finally, if we take $K = \{X \mid X \succeq 0, \mathrm{tr}(X) = 1\}$, the above SDP is equivalent to finding $X \in K$ such that

$$\begin{aligned} \frac{n}{4b} L \bullet X \geq 1 \\ n(e_i e_i^\top) \bullet X \geq 1 \quad \forall i \end{aligned}$$

(This is because $\mathrm{tr}(X) = 1$ means $\sum_i X_{ii} = 1$. Since we have the constraints $n(e_i e_i^\top) \bullet X = nX_{ii} \geq 1$, this means $X_{ii} = 1/n$ for all i.) By the discussions of the previous section, our oracle will need to check whether there exists $X \in K$ such that $D^{(t)} \bullet X \geq 1$, where

$$D^{(t)} = p_0^{(t)} \frac{n}{4b} L + \sum_{i=1}^{n} p_i^{(t)} n(e_i e_i^\top).$$

And again, is is equivalent to checking whether $\lambda_{\max}(D^{(t)}) \geq 1$.

Implementing the oracle. It is useful to note that $D^{(t)}$ is positive semidefinite: indeed, it is the sum of the Laplacian (which is psd), and a bunch of matrices $e_i e_i^\top$ (which are psd).

> **Note:** In Homework #6, you will show that for any psd matrix D, the "power method" starting with a random unit vector can find $x \in K$ such that $D \bullet (xx^\top) \in [\lambda_{\max}(D)/(1+\epsilon), \lambda_{\max}(D)]$. The algorithm succeeds with high probability, and runs in time $O(\epsilon^{-1} m \log n)$ time, where m is the number of edges in G (and hence the number of non-zeroes in L).

So we can run this algorithm: if it answers with an x such that $D^{(t)} \bullet (xx^\top)$ is smaller than $1/(1+\epsilon)$, we answer saying $\lambda_{\max}(D^{(t)}) < 1$. Else we return the vector x: this has the property that $D^{(t)} \bullet (xx^\top) \geq 1/(1+\epsilon) \geq 1 - \epsilon$. Now, using the Observation #4 on the previous page, we know this will suffice to get a solution that has an $O(\epsilon)$ infeasibility.

Bounding the width. The width of our algorithm is the maximum possible magnitude of $D^{(t)} \bullet X$ for $X \in K$, i.e., the maximum possible eigenvalue of $D^{(t)}$. Since $D^{(t)}$ is positive

semidefinite all of its eigenvalues are non-negative. Moreover, $\mathrm{tr}(L) = 1$, and also $\mathrm{tr}(e_i e_i^\top) = 1$. So

$$\begin{aligned}\lambda_{\max}(D^{(t)}) &\leq \sum \lambda_i(D^{(t)}) = \mathrm{tr}(D^{(t)}) \\ &= \mathrm{tr}\left(p_0^{(t)}\frac{n}{4b}L + \sum_{i=1}^{n} p_i^{(t)} n(e_i e_i^\top)\right) \\ &= p_0^{(t)}\frac{n}{4b}\mathrm{tr}(L) + \sum_{i=1}^{n} p_i^{(t)}\, n\,\mathrm{tr}(e_i e_i^\top) \\ &= n(1 + 1/4b).\end{aligned}$$

Finally, the max-cut values we are interested in lie between $1/2$ (since the max-cut is at least half the edge-weight) and 1. So $b \in [1/2, 1]$, and the width is $O(n)$.

Running Time. Setting the width $\rho = O(n)$ gives us a runtime of

$$O\left(\frac{n^2 \log n}{\epsilon^2} T_{oracle}\right)$$

which we can reduce to

$$O\left(\frac{n \log n}{\epsilon^2} T_{oracle}\right)$$

using Lemma 17.3, since our cost vectors can be made all nonnegative. Finally, plugging in our oracle gives a final runtime of

$$O\left(\frac{mn \log^2 n}{\epsilon^3}\right),$$

where m is the number of edges in our graph.

Note: We can now scale the "average" matrix $\widetilde{X}$ by n to get a matrix $\widehat{X}$ satisfying:

$$\begin{aligned}\tfrac{1}{4}L \bullet \widehat{X} &\geq b(1-\epsilon) \\ \widehat{X}_{ii} &\geq 1-\epsilon \qquad \forall i \\ \mathrm{tr}(\widehat{X}) &= n \\ \widehat{X} &\succeq 0\end{aligned}$$

The attentive reader will observe that this is not as nice as we'd like. We'd really want each $\widehat{X}_{ii} \in [1-\epsilon, 1+\epsilon]$—then we could transform this solution into one where $X_{ii} = 1$ and $\frac{1}{4}L \bullet \widehat{X} \geq b(1 - \epsilon^{O(1)})$.

What we have only guarantees that $X_{ii} \in [1-\epsilon, 1+n\epsilon]$, and so we'd need to set $\epsilon \leq 1/n$ for any non-trivial guarantees. This would still give us a run-time of $O(\epsilon^{-3} mn^4 \mathrm{poly}\log n)$—still polynomial (and useful to examplify the technique), but it could be better. One can avoid this loss by defining K differently—in fact, in a way that is similar to Section 17.2.1—the details can be found in [KL96]. One can do even better using the *matrix* multiplicative weights algorithms: see, e.g., [AK07, Ste10].

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1

Bibliography

[AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, Princeton University, 2005. 16, 17.1

[AK98] Noga Alon and Nabil Kahale. Approximating the independence number via the ϑ-function. *Mathematical Programming*, 80:253–264, 1998. 11.12

[AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 17.3.1

[AW00] T. Asano and D.P. Williamson. Improved approximation algorithms for max sat. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 96–105. Society for Industrial and Applied Mathematics, 2000. 14.1

[CCL+05] Maria Chudnovsky, Gerard Cornuejols, Xinming Liu, Paul Seymour, and Kristina Vuskovic. Recognizing berge graphs. *Combinatorica*, 25:143–186, 2005. 11.7

[CMM07] M. Charikar, K. Makarychev, and Y. Makarychev. Near-optimal algorithms for maximum constraint satisfaction problems. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 62–68. Society for Industrial and Applied Mathematics, 2007. 14.1

[CRST06] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006. 11.5

[DP93] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557–574, 1993. 12.2.5, 12.2.5

[Fei97] Uriel Feige. Randomized graph products, chromatic numbers, and the lovász ϑ-function. *Combinatorica*, 17:79–90, 1997. 11.13

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55:119–139, August 1997. 16.3

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652 (electronic), 2007. 3

[GLS88] Martin Grötschel, Lásló Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988. 9, 9.5, 9.7

[Gup11] Anupam Gupta. Lecture #17: Multiplicative weights, discussion. http://lpsdp.wordpress.com/2011/11/09/lecture-17-multiplicative-weights-discussion/, 2011. 16.2

[Hås99] Johan Håstad. Clique is hard to approximate within a factor of $n^{1-\epsilon}$. *Acta. Math.*, 182:105–142, 1999. 11.4

[HZ99] E. Halperin and U. Zwick. Approximation algorithms for max 4-sat and rounding procedures for semidefinite programs. *Integer Programming and Combinatorial Optimization*, pages 202–217, 1999. 14.1

[KG98] Jon Kleinberg and Michel X. Goemans. The lovász theta function and a semidefinite programming relaxation of vertex cover. *SIAM J. Discret. Math.*, 11:196–204, May 1998. 11.14

[KL96] Philip Klein and Hsueh-I Lu. Efficient approximation algorithms for semidefinite programs arising from MAX CUT and COLORING. In *Proceedings of the Twenty-eighth Annual ACM Symposium on the Theory of Computing (Philadelphia, PA, 1996)*, pages 338–347, New York, 1996. ACM. 17.3.1

[Kon81] S. V. Konyagin. Systems of vectors in euclidean space and an extremal problem for polynomials. *Mathematical Notes*, 29:33–40, 1981. 11.4

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *focs*, page 406. Published by the IEEE Computer Society, 1997. 14.1

[LLZ02] D. Livnat, M. Lewin, and U. Zwick. Improved rounding techniques for the max 2-sat and max di-cut problems. In *Proc. of 9th IPCO*, pages 67–82, 2002. 14.1

[Lov72] László Lovász. Normal hypergraphs and the perfect graph conjecture. *Discrete Math.*, 2:253–267, 1972. 11.4

[Lov79] László Lovász. On the shannnon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979. 11.3

[LW89] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989. 16.1.1

[MP90] Bojan Mohar and Svatopluk Poljak. Eigenvalues and max-cut problem. *Czechoslovak Math. J.*, 40(115)(2):343–352, 1990. 12.2.5

[Ste10] David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *SODA*, pages 684–697, 2010. 17.3.1

[Tar86] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986. 9.1

[YN76] David Yudin and Arkadi Nemirovski. Informational complexity and effective methods of solution of convex extremal problems. *Economics and mathematical methods*, 12:357–369, 1976. 9.7

[Zwi02] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. Society for Industrial and Applied Mathematics, 2002. 14.1