Analysis of Algorithms	Linear Programming Notes
CS 6820 Fall 2014	Lectures, October 3-20, 2014

1 Linear programming

The linear programming (LP) problem is the following optimization problem. We are given matrix A and vectors b and c and the problem is to find x that

$$\max(cx \text{ such that: } Ax \leq b\}.$$

Assume that A is an n by m matrix, b is an m vector, and c and x are n vectors so the multiplications, and inequalities above make sense. So x is a vector of n variables, and $Ax \leq b$ is a set of m inequalities. An example in two variables would be

$$\begin{aligned} max x_1 + x_2 & \\ 2x_1 + x_2 &\leq 3 \\ x_1 + 2x_2 &\leq 5 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

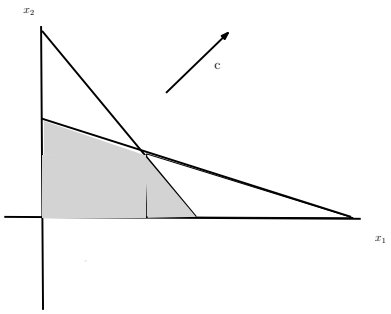

Figure 1: linear program in two dimensions.

An algorithm for linear programming takes A, b and c as input, and returns one of the following three answers:

- "no solutions exist", if there is no solution x such that $Ax \leq b$.
- "the maximum is unbounded", if for any γ there is a solution $Ax \leq b$ with $cx \geq \gamma$.
- return a vector x that satisfies $Ax \leq b$ and achieves the maximum of cx.

2 First example matching and fractional matching

As a first example of linear programming consider the matching problem. We are given a graph $G = (V, E)$. To think of matching this way, we associate a variable x_e with every edge $e \in E$. We would like to think of these variables taking values 0 or 1 with $x_e = 1$ indicating that edge e in the matching, and 0 when its not in the matching. To write the maximum matching problem as a set of inequalities we have

$$\begin{aligned} x_e &\in \{0,1\} \text{ for all } e \in E \\ \sum_{e \text{ adjacent to } v} x_e &\leq 1 \text{ for all } v \in V \\ max \sum_e x_e & \end{aligned}$$

Note that this is an *integer* linear program, as we require x_e to be 0 or 1, and not a fractional value between the two.

Lemma 1 *Integer solutions x to the above inequalities are in one-to-one correspondence to matchings $M = \{e : x_e = 1\}$, with the matching of maximum size corresponding to the optimal solution.*

To define the *fractional matching problem* we replace the constrain $x_e \in \{0, 1\}$ by $0 \leq x_e \leq 1$ for all edges. So the fractional matching problem is

$$\begin{aligned} 0 \leq x_e &\leq 1 \text{ for all } e \in E \\ \sum_{e \text{ adjacent to } v} x_e &\leq 1 \text{ for all } v \in V \\ max \sum_e x_e & \end{aligned}$$

What can we say about the maximum? One way to derive bounds on the sum is to add up all the $n = |V|$ inequalities for the nodes. We get

$$\sum_v \sum_{e \text{ adjacent to } v} x_e \leq n.$$

Each edge $e = (v, u)$ occurs twice on the left hand side, as e is on the list of edges for the sum at the node v and u, so the inequality is

$$2 \sum_e x_e \leq n,$$

i.e., the sum is at most $n/2$. Not only the maximum matching is limited to at most half the number of vertices, but also the maximum fractional matching.

Alternately, we can add up a subset of the inequalities. A subset $A \subset V$ is a *vertex cover* if A contains at least one of the ends at each edge e. Adding the inequalities for $v \in A$ we get

$$\sum_{v \in A} \sum_{e \text{ adjacent to } v} x_e \leq |A|.$$

Since A is a vertex cover, each edge variable x_e occurs at least once on the left hand side, and some occurs twice. However, we also have that $x_e \geq 0$, so we also have

$$\sum_e x_e \leq \sum_{v \in A} \sum_{e \text{ adjacent to } v} x_e \leq |A|.$$

for all vertex covers A. The minimum vertex cover problem is to find a vertex cover of minimum size. The above inequality is true for all vertex covers, and hence also for the minimum vertex cover.

Lemma 2 *The maximum $\sum_e x_e$ for a fractional matching is at most the minimum size $|A|$ of a vertex cover.*

We can further strengthen the inequality by considering *fractional vertex cover*: adding each inequality with a nonnegative multiplier y_v. A vector $y_v \geq 0$ for all $v \in V$ is a fractional vertex cover if for each edge $e = (v, u)$ we have $y_u + y_v \geq 1$. Note that a fractional vertex cover where $y_v \in \{0, 1\}$ is a regular vertex cover.

Lemma 3 *Integer solutions y to the above inequalities are in one-to-one correspondence to vertex covers $A = \{v : y_v = 1\}$, with the vertex cover of minimum size corresponding to the integer solution with minimum $\sum_v y_v$.*

Consider a fractional vertex cover y. Multiplying $\sum_{e \text{ adjacent to } v} x_e \leq 1$ by y_v we get

$$y_v \sum_{e \text{ adjacent to } v} x_e \leq y_v$$

and adding the inequalities for all nodes (and turning the sides around to help the write-up), we get

$$\begin{aligned} \sum_{v \in V} y_v &\geq \sum_{v \in V} y_v \sum_{e \text{ adjacent to } v} x_e \\ &= \sum_{e=(u,v) \in E} x_e (y_v + y_u) \\ &\geq \sum_{e=(u,v) \in E} x_e \end{aligned}$$

where the equation in the middle is just reordering the sum, and the inequality follows as y is a fractional vertex cover and x is nonnegative.

Summing up we get the following main theorem

Theorem 4

$$\max_{matching\ M} |M| \leq max_x \sum_e x_e \leq min_y \sum_v y_v \leq min_{A\ \ vertex\ cover} |A|$$

where the maximum in the middle is over fractional matchings x, and the minimum is over fractional vertex covers y.

Remark. Recall from a couple lectures ago we have seen as an application of max-flows and min-cuts that in bipartite graph the size of a maximum matching equals the minimum size of a vertex cover, so there is equation throughout the chain on inequalities above in bipartite graphs. This is not true in general graphs. Consider for example a triangle. The maximum matching is of size 1, the minimum vertex cover needs 2 nodes, and note that $x_e = 0.5$ for all e, and $y_v = 0.5$ for all v define fractional matching and fractional vertex covers with values 1.5. More generally, consider a complete graph on $n = 2k + 1$ nodes. The maximum matching is of size n, we can get a fractional matching of size $n/2$, by say using a triangle with $1/2$ on each edge, and a matching on the rest. Putting $y_v = 1/2$ in each node gives a fractional vertex cover of value $n/2$ also, while the minimum size of an integer vertex cover is $n - 1$.

3 Linear programs and their duals

Using the example of fractional matching, we derived upper bounds on the maximum, by adding up fractional copies of the inequalities (multiplying each by a nonnegative value y_v. Thinking about such bounds more generally leads to the concept of the dual of a linear program. Consider again linear programs in the general form

$$\max(cx \text{ such that: } Ax \leq b\}.$$

Let $a_i x \leq b_i$ denote the inequality in the ith row of this system. For any nonnegative $y_i \geq 0$ we can get the inequality $y_i(a_i x) \leq y_i b_i$, and adding up such inequalities for a vector $y \geq 0$ we get

$$\sum_i y_i(a_i x) \leq \sum_i y_i b_i$$

or using vector notation, we have $y(Ax) \leq yb$. If it happens to be the case, that $yA = c$, than the inequality we just derived is $cx \leq yb$, hence yb is an upper bound of the maximum our linear program seeks to find. The *dual linear program* is the best such upper bound possible. More formally, it is the program

$$\min(yb : y \geq 0 \text{ and } yA = c\}.$$

By the way we derived this program, for each y the value yb is an upper bound of our original linear program, which immediately gives us the following.

Theorem 5 (weak duality) *For any linear program defined by matrix A and vectors b and c we have*

$$\max(cx : Ax \leq b\} \leq \min(yb : y \geq 0 \text{ and } yA = c\}.$$

4 Fractional matchings, flows, and linear programs in nonnegative variables

Going back to fractional matching, the fractional matching problem had inequalities for all vertices, but also had constraints that require each variable $0 \leq x_e \leq 1$. Observe that the constraints $x_e \leq 1$ for an edge $e = (u, v)$ are redundant, as they follow from the inequalities that the variables associated with edges adjacent to, say the vertex v, need to sum to at most 1. However, the constraints $x_e \geq 0$ are important. It is useful to thing about what is the dual of a linear program with $x \geq 0$ constraints. To take the dual of this linear program with the method we have seen so far, we need to introduce nonnegative variables associates with both the $Ax \leq b$ constraints, as well as the $x \geq 0$ constraints (which we may want to write as $-x \leq 0$). Lets call this second set of variables s. Taking the dual we get the following:

$$\min(yb + s0 : y \geq 0, s \leq 0 \text{ and } yA - s = c\}.$$

Since the right hand side of the second set of constraints is 0, the s variables do not contribute to the objective function, so we can simply the dual linear program to be the following

$$\min(yb : y \geq 0 \text{ and } yA \leq c\}.$$

We get the

Theorem 6 (weak duality II) *For any linear program defined by matrix A and vectors b and c where the solution is required to be nonnegative, we have*

$$\max(cx : x \geq 0, Ax \leq b\} \leq \min(yb : y \geq 0 \text{ and } yA \geq c\}.$$

Notice that this applying this to fractional matching we see that fractional vertex cover is the dual linear program for fractional matching. When we write the fractional matching inequalities as a matrix $Ax \leq b$, we have a variable for each edge, and a constraint for each vertex. The matrix A therefore is $m = |E|$ by $n = |V|$. The matrix A has 0/1 entries. A row of A corresponding to vertex v has 1 in positions corresponding to edges e adjacent to v, and hence a column of A corresponding to an edge $e = (u, v)$ has 1 in the two positions associated with the two vertices u and v. So the dual inequality $yA \geq c$ becomes $y_u + y_v \geq 1$ for all edges $e = (u, v)$.

Corollary 7 *The dual linear program for fractional matching is the linear program for fractional vertex cover.*

Recall that in *bipartite* graphs we have seen that the maximum matching is the same size as the minimum size of a vertex cover. This implies that in bipartite graphs the maximum fractional matching is the same size as the minimum fractional vertex cover also. We also have seen that the integer matching and integer vertex cover is not the same size on a triangle, but we'll show below that the maximum size of a fractional matching is the same as the minimum size of a fractional vertex cover on all graphs. This will follow from the strong linear programming duality.

Next we consider the maximum flow problem. You may recall the formulation of maximum flow with variables on paths. Given a directed graph $G = (V, E)$ with nonnegative capacities $c_e \geq 0$ on the edges, and a source-sink pair $s, t \in V$, the flow problem is defined as a linear program with variables associated with all $s - t$ paths. Let $\mathcal{P}$ denote the set of paths in G from s to t. Now the problem is (using x as a variable name rather than f to make it more similar to our other linear programs):

$$\begin{aligned} x_P &\geq 0 \text{ for all } P \in \mathcal{P} \\ \sum_{P:e\in P} x_P &\leq c_e \text{ for all } e \in E \end{aligned}$$

$$max \sum_P x_P$$

The dual of this linear program has variables associated with the edges (the inequalities of the above system), and has a variable associated with each path $P \in \mathcal{P}$. The dual program then becomes the following.

$$\begin{aligned} y_e &\geq 0 \text{ for all } e \in E \\ \sum_{e\in P} y_e &\geq 1 \text{ for all } P \in \mathcal{P} \end{aligned}$$

$$min \sum_e c_e y_e$$

Notice that since the capacities c_e are nonnegative, an optimal solution will have $y_e \leq 1$ for all edges. Now consider an integer solution when y_e is 0 or 1 on all edges, and let $F = \{e : y_e = 1\}$ be the selected set of edges. The constraint on paths requires that all $s - t$ path must contain an edge in F, so F must contain an (s, t) cut, and by minimality, and optimal solution is then an (s, t)-cut, and its value is exactly the capacity of the cut.

Lemma 8 *Integer optimal solutions to the above dual linear program are in one-to-one correspondence with minimum capacity (s, t)-cuts in the graph.*

We know from linear programming duality that the maximum fractional flow has the same value as the minimum of the dual program. Note that in the case of flows, we have seen that the integer max flow is equal to the min cut value. Our observation that min-cuts are the integer solutions of the dual linear program shows that the dual linear program also has an integer dual solution.

Corollary 9 *The above dual of the max-flow problem is guaranteed to have an optimal solution with variables y integer, and hence the flow linear problem and its dual has the same optimal solution value.*

5 Strong duality of linear programs

We have seen that the primal and dual linear programs have equal values in the max-flow problem. While not all linear programs solve optimally in integer variables, we'll see that the primal and dual linear programs always have equal solution values. This is the main theorem of linear programming, called strong duality, i.e., that in inequality in the weak duality theorem is always equal.

Theorem 10 (strong duality) *For any linear program defined by matrix A and vectors b and c, if there is a solution x such that $Ax \leq b$ then we have*

$$\max(cx : Ax \leq b\} = \min(yb : y \geq 0 \text{ and } yA = c\}.$$

as well as

$$\max(cx : x \geq 0, Ax \leq b\} = \min(yb : y \geq 0 \text{ and } yA \geq c\}.$$

First observe the second statement follows from the first, by simply applying the first to linear programs with $x \geq 0$ as part of the constraint matrix. Second recall that by weak duality, we know that all solutions x and all solutions y have $cx \leq yb$. To prove the equality, all we have to do is to exhibit a pair of solutions x^* and y^* such that $cx^* = y^*b$. Once we do this we have that for al solutions x the value $cx \leq y^*b = cx^*$ so x^* is optimal, and similarly, for all solutions y we have that $yb \geq cx^* = yb^*$, so y^* is optimal.

We will not formally prove this theorem that such an x^* and y^* must exists, rather will present a"proof" based on physics principles, that we hope will give good intuition why the theorem is true without being too complex. We will think of the area $P = \{x : Ax \leq\}$ as a physical area enclosing say a small metal ball. The walls of the area are bounded by the inequalities $a_i x \leq b_i$, and x will denote the location of the ball. We will also imagine that there a strong magnet "at infinity" in direction of c that is puling the ball, however, the ball cannot leave the bounding area P.

1. if the ball keeps accelerating in c direction forever, the value of cx will go to infinity as the ball moves, so $\max cx = \infty$.

2. If the ball cannot accelerate forever, it must come to a stop. Let x denote the place it stops.

3. At this point the ball has a force c on it from the magnet, the walls of the bounding area must exert force that compensates the magnet's force. The wall $a_i x \leq b_i$ can exert force in a_i direction, so this force can be of the form $y_i a_i$ for a nonnegative number $y_i \geq 0$. The ball stopped, so the forces acting on it sum to 0, so we get $c - \sum_i y_i a_i = 0$.

4. Finally observe that only walls that touch the ball can exert any force on it, so if $a_i x <, b_i$ we must have $y_i = 0$.

We claim that the place x^* where the ball stops and the vector $y^* = (y_1^*, \ldots, y_m^*)$ form optimal primal and dual solutions.

Lemma 11 *The properties listed above that are derived from physical principles, imply that the place x^* where the ball stops and the vector $y^* = (y_1^*, \ldots, y_m^*)$ form optimal primal and dual solutions.*

Proof. First we note that x^* and y^* are feasible solutions. The ball is inside the region P, so $Ax^* \leq b$ by definition. We also have that $y^* \geq 0$ as the wall holds the ball inside by exerting force, but is not pulling the ball towards the wall, so $y_i^* \geq 0$ for all i, and we have seen that $c = \sum_i y_i^* a_i$ as the forces add up to 0.

Next we want to show that x^* is of maximum value cx and y^* has minimum value yb. Recall that all we have to do to show this is to argue that $cx^* = y * b$. To see this consider the chain of inequalities

$$cx^* = (y^*A)x^* \leq y^*(Ax^*) = \sum_i y_i^*(a_i x^*) \leq \sum_i y_i^* b_i,$$

that is true for all feasible solutions x^* and y^* (the first equality is by $c = y^*A$, the second by rearranging terms, and the inequality follows as its true term by term: $y_i^*(a_i x^*) \leq y_i^* b_i$ for all i, as $y_i \geq 0$ and $a_i x^* \leq b_i$. Now recall the last property, that force can be only exerted by walls that touch the ball x^*. This property implies that $y_i > 0$ only possible if $a_i x^* = b_i$, or in other words either $y_i^* = 0$ or $a_i x^* = b_i$ for all i. In either case $y_i^*(a_i x^*) = y_i^* b_i$, and so the last inequality above is actually equal. ■

Notice that the chain on inequalities is true for all feasible vectors x and feasible dual solutions y. The argument we just went through can be used to recognize a pair of optimal solutions.

Corollary 12 *For a solution x of $Ax \leq b$ and a vector $y \geq 0$ that satisfies $c = yA$, x and y are optimal solutions of the linear program and its dual if and only if for each i either $y_i = 0$ or $a_i x = b_i$. Or put it differently, x and y are optimal solutions of the linear program and its dual if and only if for all i we have $y_i(b_i - a_i x) = 0$.*

6 The ellipsoid method

Next we will roughly sketch one of the methods for solving linear programs, the ellipsoid method. This was the first polynomial time algorithm discovered for the problem. Its not the most efficient practically: practical algorithms either use the simplex method (which may be exponential in the worst case) or interior point methods. However, the ellipsoid method is based on a simple geometric idea, and it is the most powerful in the sense of being able to solve extensions of linear programs also in polynomial time.

The basis idea of the ellipsoid method is the following. Suppose we know that our feasible region $P = \{x : Ax \leq b\}$ is contained in a ball, say the ball $B = \{x : x^2 = \sum_i x_i^2 \leq 1\}$. If we also knew that say $x_1 \geq 0$ for all points in P, then P is contained in a half ball $B \cap \{x : x_1 \geq 0\}$, which is only half as big as B. Unfortunately, a half-ball is a much more complex object. The idea is to enclose the half ball in an ellipsoid E. The ellipsoid will still be smaller than B (tough by much less smaller than the half-ball), it will still contain the region of our interest P, and be again a simple shape, like a ball, so we will be able to recurse.

To develop the idea above, we need to find an ellipse E that encloses the half-ball as shows in the figure below. Our ellipse will be centered at $c = (\frac{1}{n+1}, 0, \ldots, 0)$. So the ball B

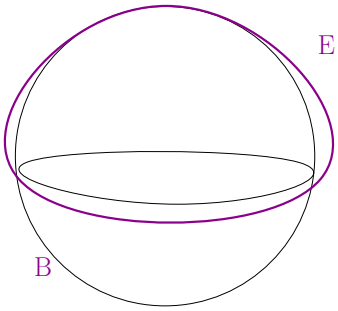

Figure 2: The ellipse E enclosing the top half of the ball B.

translated be centered at a point c would have definition $B(c) = \{x : \sum_i (x_i - c_i)^2 \leq 1\}$. The ellipse we are interested in is a bit squashed version of this ball defined as $E = \{x : \frac{n^2-1}{n^2} \sum_{i \leq 2} x_i^2 + (\frac{n+1}{n}(x_1 - \frac{1}{n+1}))^2 \leq 1\}$, where n is the dimension of the space $x \in R^n$.

Lemma 13 *The ellipse E contain the half ball $B \cap \{x : x_1 \geq 0\}$.*

Proof. First test two points $x = (1, 0 \ldots, 0)$. This point satisfies the ellipse inequality as

$$\frac{n^2-1}{n^2} \sum_{i \leq 2} x_i^2 + (\frac{n+1}{n}(x_1 - \frac{1}{n+1}))^2 = (\frac{n+1}{n}\frac{n}{n+1})^2 = 1.$$

Second, consider a point $x = (0, x_2, \ldots, x_n)$ on the "equator", i.e. where $\sum_{i \geq 0} x_i^2 = 1$. For such a point we get

$$\frac{n^2-1}{n^2} \sum_{i \leq 2} x_i^2 + (\frac{n+1}{n}(x_1 - \frac{1}{n+1}))^2 = \frac{n^2-1}{n^2} + (\frac{n+1}{n}\frac{1}{n+1})^2 = \frac{n^2-1}{n^2} + \frac{1}{n^2} = 1.$$

Finally, consider a general point x in the half ball, and let $\sum_{i \leq 2} x_i^2 = s^2$ for some value s. If $x_1 \leq \frac{1}{n+1}$, i.e., the point is below the level of the center, the argument we just used for

the equator works again. For the point is above the center, $\frac{1}{n+1} \leq x_1$ we use the fact that $x_1 \leq \sqrt{1-s^2}$. So we get

$$(x_1 - \frac{1}{n+1})^2 \leq (\sqrt{1-s^2} - \frac{1}{n+1})^2$$

In this case we get a rather complicated expression in s for our bound

$$\frac{n^2-1}{n^2}\sum_{i\leq 2} x_i^2 + (\frac{n+1}{n}(x_1 - \frac{1}{n+1}))^2 \leq \frac{n^2-1}{n^2}s^2 + (\frac{n+1}{n})^2(\sqrt{1-s^2} - \frac{1}{n+1})^2$$

Maybe the simplest way to show that this is at most 1 is to use calculus to show that the maximum of this expression on the interval $s \in [0,1]$ occurs at the two ends, and we have just seen that at $s = 0$ or 1 the value is 1. ■

Next we need to show that our ellipse E indeed has significantly smaller volume that the ball B, as we hope to claim to make progress by shrinking the volume. To do this, its useful to remember the expression for the volume of a ball with radius r in n dimension is $\gamma_n r^n$ for a constant γ that depends on the dimension. For example, in 2 dimension $\gamma_2 = \pi$, while in 3 dimension $\gamma_3 = \frac{4}{3}\pi$. We are thinking about ellipses of the form $E_0 = \{x : (\alpha_i x_i)^2 \leq 1\}$. A ball of radius r is expressed this way with $\alpha_i = 1/r$ for all i. More generally, the volume of the ellipse just defined is $V(E_0) = \gamma_n / \prod_i \alpha_i$. Using this expression we can get the ratio of the ellipse E containing the half-ball and the ball.

Lemma 14 *The ratio of the volumes of the ellipse E and the ball B is bounded by $V(E)/V(B) \leq e^{-1/4(n+1)}$*

Proof. The ball B has radius 1, so its volume is γ_n. In computing the ratio, γ_n cancels. In defining the ellipse E we used $\alpha_1 = \frac{n+1}{n}$, while α_i for $i \geq 2$ we have $\sqrt{\frac{n^2-1}{n^2}}$. So we get

$$V(E)/V(B) = \frac{n}{n+1}(\frac{n^2}{n^2-1})^{(n-1)/2}$$

To estimate this expression we can use that $1+x \approx e^x$ for small x, and use that $\frac{n}{n+1} = (1 - \frac{1}{n+1})$ and similarly $\frac{n^2}{n^2-1} = (1 + \frac{1}{n^2-1})$ so we get

$$V(E)/V(B) \approx e^{-1/(n+1)}(e^{1/(n^2-1)})^{(n-1)/2} = e^{-1/(n+1)+1/2(n+1)} = e^{-1/2(n+1)}$$

Being a bit more careful with the small error in the $1 + x \approx e^x$ approximation, decreases the bound a bit further, but we can get the bound claimed by the lemma. Unfortunately, the decrease is quite minimal. ■

Now we are ready to use the above geometry in an algorithm. To help the presentation, we will make a number of simplifying assumptions. As we make each assumption, we comment on how one may be able to solve problems without the assumption, but we will not elaborate in these further.

1. We will assume that we are looking for is known to be contained in a large ball $B_0 = \{x : x^2 \leq R^2\}$ for a parameter R. For example, if we know that the variables are $0 \leq x_i \leq 1$ than we can use $R = \sqrt{n}$. Similarly, if there are upper bounds on the variables, this implies an upper bound on R. Without any such bound, one would have to argue that if the maximum of the linear program is not infinite, it occurs in a bounded region.

2. To simplify the presentation, we will focus on just finding a feasible solution x satisfying $Ax \leq b$ without a maximization. One can incorporate an objective function of $\max cx$, for example, by adding a constraint that $cx \geq \gamma$ and binary searching on the maximum value of γ for which the system remains feasible.

3. Instead of looking for an exact solution of $Ax \leq b$, we will be satisfied with an approximate solution. Assume that we are given an error parameter $\epsilon > 0$. By dividing the inequalities in $Ax \leq b$ by appropriate constants, we can assume that each entry in the matrix is at most 1, i.e.,that $|a_{ij}| \leq 1$ for all i and j. With this assumption in place we will accept an x as a solution if for each constraint i it satisfies $a_i x \leq b_i + \epsilon$. However, the algorithm can conclude that no solution exists, assuming the original system $Ax \leq b$ has no solution.

   Notice that there are cases that it is not clear upfront what this approximate solution algorithm should output. If there is no solution to $Ax \leq b$, but there is a solution to the related system of $a_i x \leq b_i + \epsilon$ for all i the algorithm can find either answer. In most applications such an approximate solution is OK. To make the algorithm precise we would have to do two things. First find a small enough value $\epsilon > 0$ that guarantees that of $Ax \leq b$ has no solution, than $a_i x \leq b_i + \epsilon$ for all i also has no solution. Second, we need to show that for a small enough $\epsilon > 0$ a solution x of the approximate system $a_i x \leq b_i + \epsilon$ for all i can be rounded to become a solution of the original system.

The main idea of the algorithm is to start with the ball in assumption 1 that is known to contain all solutions. While we have not found a solution, we will have an ellipsoid E_i that contains all solutions. In each iteration, we test the center c^i of our ellipsoid E_i. If c^i is an (approximate) solution to our system, we return $x = c^i$ and we are done. If c^i is not a solution, than it must violate one of the constraints of the system $a_i c^i > b_i + \epsilon$. In this case, all solutions, even all approximate solutions, are in the half of the ellipsoid defined by the cut through the center of our ellipsoid $a_i x \leq a_i c^i$. We then define the next ellipsoid E_{i+1} to contain this half-ellipsoid $E_i \cap \{x : a_i x \leq a_i c^i\}$.

We defined an enclosing ellipsoid for a half-ball, with the ball centered at 0, and the half defined by one of the coordinate directions. However, now we need this for an ellipsoid with a different center, and a direction that may not be one of the coordinate directions. While working out the algebraic expression for this new ellipsoid is a bit complex, the geometric idea is simple via a geometric steps. To see how this translates to an algebraic expression, you may want to look at the notes by Santosh Vempala posted in the course web page.

- By translating the space we can assume that any given point c is the origin.

- For an ellipsoid E defined by the inequality $\sum_i(\alpha_i x_i) \leq 1$ we can stretch the coordinates, by using a new coordinate system with $y_i = \alpha_i x_i$, and the ellipsoid E becomes the unit ball in the new coordinate system.
- Finally, we want to take a half-space defined by an arbitrary vector $ax \geq 0$, rather than a coordinate direction. To do this, we can again change coordinate systems, by letting the unit vector in direction a become our first coordinate, and extending this to an orthogonal system of coordinates for the space.

Using these reductions allows us to take the ellipsoid defined in the beginning of this section for the unit ball as part of an iterative algorithm. From Lemma 14 we know that the volume in each iteration decreases by at least a bit. The last question we need to answer is how many iterations we need before we can conclude. In fact, we need to wonder about how the algorithm can conclude. It may find that the centers of one of the ellipsoids is a solution, and hence can terminate. But how does it terminate when there is no solution? The idea is to note that if $Ax \leq b$ has a solution, than the set of approximate solutions must have a volume δ. This is useful, as if we ever find that our ellipsoid E_i at some iteration has volume smatter than δ than we can conclude that $Ax \leq b$ cannot have a solution, and hence can terminate.

Lemma 15 *If the system of inequalities $\{x : Ax \leq b\}$ has a solution, and $|a_{ij}| \leq 1$ for all entries than the volume of the set $\{x : a_i x \leq b_i + \epsilon$ for all $i\}$ must be at least $\delta = (2\epsilon/n)^n$.*

Proof. Consider a solution x^* such that $Ax^* \leq b$. We define a small box around x^* as

$$B(x^*) = \{x : |x_i - x_i^*| \leq \epsilon/n \text{ for all } i\}$$

Observe that all points $x \in B(x^*)$ must satisfy the approximate inequalities, and the volume $V(B(x^*))$ is exactly δ proving the lemma. ■

Theorem 16 *Under the simplifying assumptions 1-3 made above, the ellipsoid algorithm solve the problem of finding a feasible solution to a system of inequalities in n dimension in $O(n^2 \log(Rn/\epsilon)$ iterations.*

Proof. By the 1st assumption we start out with a ball of volume $R^{n/2}$. By Lemma 14 each iteration decreases the volume of our ellipsoid by a factor of $e^{1/2(n+1)}$, so $2(n+1)$ iterations decrease the volume by a factor of e, i.e., by a constant factor. what is the range of volume that we need to decrease? we need to go from $R^{n/2}$ to $(2\epsilon/n)^n$, a range less than $(Rn/\epsilon)^n$. This happens after the volume decreases $\log((Rn/\epsilon)^n) = n\log(Rn/epsilon)$ times by a constant factor, so overall will take $O(n^2 \log(Rn/\epsilon)$ iterations as claimed. ■

7 Linear Programming and Randomized Rounding

As an application of linear programming, we will consider the following disjoint path problem. Given a directed graph $G = (V, E)$, capacities on the edges $c_e \geq 0$ for $e \in E$, and pairs of

nodes $s_i, t_i \in V$ for $i = 1, \ldots, k$, the problem is to find paths P_i from s_i to t_i that don't use any edge e more than c_e times. For example, when $c_e = 1$ for all e we are looking for disjoint paths. There may not be k such paths in G, so we will try to find as many as possible.

The first natural idea is to reduce the problem to max-flow, but unfortunately, this won't work out so well. To see why, note that the natural reduction would add a supersource s with edges to each s_i with capacity 1, and a supersink t with edges from each t_i to t with capacity 1, and than find a max-flow from s to t. We have seen that a flow can be decomposed into paths from s to t, with integer capacities, each flow will carry an integer amount of flow, and with the source and sink edges having capacity 1, each paths will have one unit of flow. What goes wrong is the pairing of the sources and sinks. There is nothing guaranteeing that the path starting at s_i ends at its pair t_i, rather than at some other terminal t_j. In fact, this approach cannot as finding disjoint paths (when $c_e = 1$ for all e) is an NP-complete problem. Here we will find a close to optimal solution when the capacities are high enough.

The high level idea is to take advantage that linear programs can be solved in polynomial time. The above paths problem can be formulated as an integer problem. We solve a fractional version, and then will want to use the fractional solution to get an integer solution. The most natural way to formulate the path problem is to have a variable x_P associated with every paths. Let $\mathcal{P}_i$ denote the set of paths in G from s_i to t_i. Then we write

$$\begin{aligned} x_P &\geq 0 \text{ for all } P \in \cup_i \mathcal{P}_i \\ \sum_{P:e\in P} x_P &\leq c_e \text{ for all } e \in E \\ \sum_{P\in\mathcal{P}_i} x_P &\leq 1 \text{ for all } i \\ max \sum_P x_P & \end{aligned}$$

The inequality for the edges enforces the capacities, the other set of inequalities is asking to select at most one total of the paths between any source and sink. While we didn't include the $x_P \leq 1$ constraint, this is implied by the inequality that is asking to select at most one total of the paths between any source and sink. So integer solutions will have x_P either 0 or 1, and we can think of the paths P with $x_P = 1$ as being selected.

Lemma 17 *Integer solutions to the above inequalities are in one-to-one correspondence to paths that satisfy the capacity constraints.*

However, unlike maximum flow, this linear program does **not** solve in integers, the optimal solution will result in fractional values. Before we start thinking about how to use the solution to this linear program, we need to worry if we can solve it at all. The problem is that the linear program as stated can have exponentially many variables, one for each paths. We will give a compact, polynomial size version by using the traditional flow formulation with flows on edge, but doing this separately for each edge e. We'll use variables $f_i(e)$ to denote the amount of flow from s_i to t_i on edges e, that is $f_i(e) = \sum_{P\in\mathcal{P}_i:e\in P} x_P$.

$$
\begin{aligned}
f_i(e) &\geq 0 \text{ for all } e \in E \text{ and all } i \\
\sum_i f_i(e) &\leq c_e \text{ for all } e \in E \\
\sum_{e \text{ enters } v} f_i(e) - \sum_{e \text{ leaves } v} f_i(e) &= 0 \text{ for all } i \text{ and all } v \in V,\ v \neq s_i, t_i \\
\sum_{e \text{enters } t_i} f_i(e) - \sum_{e \text{ leaves } t_i} f_i(e) &\leq 1 \text{ for all } i \\
\max \sum_i \sum_{e \text{ enters } t_i} f_i(e) - \sum_{e \text{ leaves } t_i} f_i(e)
\end{aligned}
$$

We have a separate set of flow conservation constraints for each flow f_i, a bound of 1 on flow between any given source-sink paths, a joint capacity constraint founding the total flow $\sum_i f_i(e)$, and the goal is to maximize the total flow. The advantage of this linear program is its compact size. For a graph with n nodes and m edges, we have mk nonnegative variables, and $m + nk$ constraints. Integer solutions to this linear program also are solutions to the original path problem. Its not hard to see that in an integer solution the set of edges with $f_i(e) = 1$ for a given index i can form cycles and possibly a single path from s_i to t_i. Cycles do not contribute to the objective value, and path contribute 1. So ignoring the possible cycles in a solution, we get the following.

Lemma 18 *Integer solutions to the new inequalities correspondence to paths that satisfy the capacity constraints with the number of paths equal to the objective value.*

Further, we have seen that an edge-flow can be converted to be a path flow of equal value, so any (fractional) solution to the second linear program can be converted to a solution to the first linear program in polynomial time. Recall, the way we get around the exponential number of variables is that our solution will have most of them set to 0, and will define the solution by listing the non-zero values only.

Lemma 19 *The two linear programs have the same optimal value, and a solution to one can be converted to a solution to the other in polynomial time. The resulting solution of the path flow linear program will have only polynomially many paths with nonzero x_P values. Further, the linear programming optimum value is at least as high as the maximum number of paths that can be selected.*

Now suppose we have a solution to our path flow linear program. The next question is, how is this useful to find real paths. The idea is to use the variables x_P for all $P \in \mathcal{P}_i$ as probabilities. For each i independently, we want to select at most one path from s_i to t_i, selecting a given (s_i, t_i) path with probability x_P. This is possible as

$$\sum_{P \in \mathcal{P}_i} x_P \leq 1$$

Note that if we have a strict inequality here, with some probability we will not select any (s_i, t_i) path. There are some basic facts we can get about this selection method using linearity of expectation.

Lemma 20 *The expected number of paths selected by our random selection is $\sum_P x_P$, and the expected number of paths using any edge e is at most c_e for each e.*

Proof. A path P contributes to the expected number of path by its probability of being selected, so the expected number of path overall, by linearity of expectation is $\sum_P x_P$ as claimed. The expected number of paths using an edge e is $\sum_{P:e\in P} x_P$, which is bounded by c_e by our capacity constraint. ■

We would like to show that the number of paths on any edge is below its capacity high probability. This simply won't be true if the expectation is an high as c_e. To help with this, we need to modify the algorithm to use $(1-\epsilon)x_P$ as the probability of selecting path P for all P. This decreases the expected number of paths selected by a factor of $(1-\epsilon)$, but allows us to bound the probability that the number of paths exceeds c_e on any edge.

Lemma 21 *If the expected number of paths on an edge e is at most $(1-\epsilon)c_e$, and $(1-\epsilon)c_e \geq 2\epsilon^{-2}\log m$ than the probability that there are more paths than c_e using edge e is at most $1/m^2$.*

Proof. We use Chernoff bound. Focusing on edge e let $X_i = 1$ of the s_i to t_i path selected uses edge e. By construction, the X_i variables are 0/1 and independent, and the number of paths using e is $X = \sum_i X_i$. Also note that $E(X) \leq (1-\epsilon)c_e$, which we can call $\mu = (1-\epsilon)c_e$, and $(1+\epsilon)\mu = (1-\epsilon^2)c_e \leq c_e$, so we get

$$Pr(X > c_e) \leq (e^{-0.5\epsilon^2})^{(1-\epsilon)c_e} \leq \frac{1}{m^2}$$

as claimed. ■

Theorem 22 *Assuming that all capacities $c_e \geq \epsilon^{-2}\log m$, then the above randomized rounding method, solving the linear program, using $(1-\epsilon')x_P$ for some $\epsilon' > 0$ as probabilities independently for each i, finds a solution to the path problem with high at high probability (say probability at east 3/4) using a number of path no less than a factor $1 - O(\epsilon)$ less than the optimal.*

Proof. We have seen by Lemma 19 that the expected number of paths using this method is at least $(1-\epsilon')$ times the optimal. For each edge, Lemma 21 shows that, we can get the probability of a single violated very low: $Pr(X > c_e) \leq \frac{1}{m^2}$. Let $X(e)$ denote the number of paths on edge e. Using Lemma 21 and the union bound for all edges, we get

$$Prob(\exists e : X_e > c_e) \leq \sum_e Prob(X_e > c_e) \leq m\frac{1}{m^2} = \frac{1}{m}.$$

Similarly, the probability that the total number of paths is below $(1-\epsilon')$ times its expectation can also be bounded by $\frac{1}{m^2}$ due to the lower bound version of the Chernoff bound. Using the union bound on these two events, the probability that the method has $(1-\epsilon')^2 \approx (1-2\epsilon')$ times the maximum possible number of path, and all edges have at most c_e paths is at least $1 - \frac{1}{m} - \frac{1}{m^2}$, which is high enough assuming the graph is not too small. ■