CS 521: Linear Programming **Rutgers: Fall 2022**

Lecture 5

October 7, 2022

Instructor: Sepehr Assadi *Scribes: Chengyuan Deng, Parth Mittal, Sharath Punna, Bingyu Xin*

Topics of this Lecture

1 Application: Largest Inner Disk

We start with the following motivating application.

Problem 1. Given a convex polyon, we want to find the largest disk (circle) that can fit inside the polygon.

The input is given as m half-spaces enclosing the polygon, with k half-planes enclosing the polygon from below, and $m-k$ half-planes enclosing the polygon from above:

The convex polygon:

$$\begin{aligned} \forall i \leqslant k \quad & \ell_i : y \geqslant a_i x + b_i \\ \forall i > k \quad & \ell_i : y \leqslant a_i x + b_i \end{aligned}$$

For simplicity, we are going to assume that we do not have any completely vertical or horizontal line. See Figure 1 for an illustration of this problem.

1.1 A Linear Program for Largest Inner Disk

The objective is to maximize of the radius r of the disk such that the distance from the center of the disk $s = (x, y)$ to all bounding lines/edges of the polygon is no smaller than the radius. That is, we want to solve

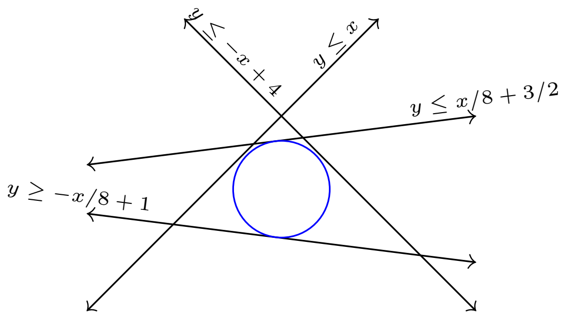

Figure 1: An example input for Problem 1 and its optimal disk.

the following program:

$$\begin{aligned} \max_{s,r} \quad & r \\ \text{subject to} \quad & r \leqslant \min_{i \in [m]} d(\ell_i, s) \\ & s \in P, \end{aligned}$$

where P is the polygon specified by the input. This optimization problem however is not an LP yet.

To convert the program above into an LP, we will use the following geometric fact.

Fact 1. *In the plane, the distance of a point (x_0, y_0) from a line $ax + by + c = 0$ is given by*

$$\frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}.$$

Note that as long as $s \in P$, we can drop the absolute sign in Fact 1, since we know whether the quantity in the numerator is less than or greater than 0. If $s \notin P$, then dropping the absolute value has the added benefit that it will make r to be less than 0 on at least one of the constraints. This enforces $s \in P$ implicitly. Hence we can write the Largest Inner Disk problem as the following LP:

$$\begin{aligned} \max_{x,y,r} \quad & r \\ \text{subject to} \quad & r \leqslant \frac{y - a_i x - b_i}{\sqrt{a_i^2 + 1}} \quad && \forall i \leqslant k \\ & r \leqslant \frac{a_i x + b_i - y}{\sqrt{a_i^2 + 1}} \quad && \forall i > k. \end{aligned}$$

We leave verifying the correctness of this LP as an exercise for the reader.

1.2 Low-Dimensional LPs

In the LP described above for the largest disk problem, there are only three variables (x, y, and r), while the number of constraints is equal to the number of lines used to define the polygon, which can be arbitrarily large. Such LPs where the number of variables (n) is much smaller than the number of constraints (m), namely, $n \ll m$ and even typically $n = O(1)$, are called **low-dimensional LPs**. The Largest Inner Disk problem is a natural example of a Low-dimensional LP.

Though the Simplex algorithm we saw in Lecture 3 solves a low-dimensional LP, it would not be a wise choice to use it here because the simplex algorithm requires the LP to be in the equational form, and turning a low-dimensional LP to the equational form will blow up the dimension of the problem from n to $2n + m$, destroying the nice structure of Low-dimensional LPs.

In this lecture, we are going to see another algorithm, the *Clarkson's Algorithm* [1] that solves low-dimensional LPs much more efficiently. Before that however, we examine yet another (generally inefficient) way of solving LPs, called the *Fourier-Motzkin elimination* method that dates back to way before even the introduction of LPs.

2 Fourier-Motzkin Elimination Method

The Fourier-Motzkin elimination method is a reminiscient of Gaussian elimination for solving linear systems, applied to LPs instead. The algorithm is named after Joseph Fourier who proposed the method in 1826 and Theodore Motzkin who re-discovered it in 1936.

Consider an LP in the following generic form:[1]

$$\max_{x \in \mathbb{R}^n} \quad c^T x \quad \text{subject to } Ax \leqslant b.$$

We will begin by adding one additional variable z and observing the following LP has the same optimal value as the above LP:

$$\begin{aligned} \max_{z \in \mathbb{R}, x \in \mathbb{R}^n} \quad & z \\ \text{subject to} \quad & Ax \leqslant b \\ & c^T x \geqslant z. \end{aligned} \tag{1}$$

The Fourier-Motzkin elimination algorithm works by converting this new LP with $n+1$ variables into an LP with a *single* variable z. At that point, solving the problem becomes trivial as such an LP would simply be a list of lower bounds and upper bounds on the value of z, namely,

$$\begin{aligned} \max_{z \in \mathbb{R}} \quad & z \\ \text{subject to} \quad & z \geqslant \ell_1, \quad z \geqslant \ell_2, \quad \ldots, \quad z \geqslant \ell_{m'} \\ & z \leqslant h_1, \quad z \leqslant h_2, \quad \ldots, \quad z \leqslant h_{m''}, \end{aligned}$$

where $\ell_1, \ldots, \ell_{m'}$ and $h_1, \ldots, h_{m''}$ are all fixed numbers. Define $\ell := \max_j \ell_j$ and $h := \min_j h_j$. These constraints can be compactly represented as $z \in [\ell, h]$. Now, if $\ell > h$, we have that the LP is *infeasible*; otherwise, $z = h$ is the optimal value of the LP. We will also be able to find the optimal assignment as discussed later.

The main step in this approach, as the name suggests, is to *eliminate* the variables one by one, and then adding a series of constraints to *nullify* the effect of removing the variable. We discuss this step next.

2.1 Eliminating a Variable

Without loss of generality, let us say we want to remove the variable x_1. Observe that x_1 appears only in the constraints. Each constraint is of the following form, where x_{-1} is the vector of all variables (including z) except for x_1 and $a_{i,-1}$ are their coefficients in the i-th constraint:

$$a_{i,1} x_1 + \langle a_{i,-1}\,,\, x_{-1} \rangle \leqslant b_i.$$

(Here, we also consider representing the constraint $c^T x \geqslant z$ in the form $z - c^T x \leqslant 0$ and thus can treat it exactly the same as all other constraints).

We will begin by partitioning the constraints into 3 sets: P, N and Z, depending on whether the coefficient of x_1 in a constraint is positive, negative, or zero, respectively. Observe that this partition puts

[1]Note that this differs from the standard form, in that there are no non-negativity constraint on x. However, it can be seen that this form is more general, since one can always include the non-negativity constraints in the matrix A.

all the constraints which put an upper bound on x_1 in P, all lower bounds on x_1 in N, and all constraints that do not contain x_1 in Z, because:

$$\forall j \in P \qquad x_1 \leqslant \frac{b_j - \langle a_{j,-1}\,,\, x_{-1}\rangle}{a_{j,1}},$$

and

$$\forall k \in N \qquad x_1 \geqslant \frac{b_k - \langle a_{k,-1}\,,\, x_{-1}\rangle}{a_{k,1}}.$$

(the inequality flips in the second case because $a_{k,1} < 0$).

The key idea is that now we can remove x_1 from the LP by chaining together the inequalities in P and N to nullify the effect of removing x_1. For each constraint $j \in P$ and $k \in N$, we add the constraint:

$$\frac{b_k - \langle a_{k,-1}\,,\, x_{-1}\rangle}{a_{k,1}} \leqslant \frac{b_j - \langle a_{j,-1}\,,\, x_{-1}\rangle}{a_{j,1}}.$$

Thus, we replace $|P| + |N|$ constraints with $|P| \cdot |N|$ constraints this way.

To show that this reduction is complete and sound, we have the following pair of claims:

Claim 2. *If (x, z) is a feasible solution to the original LP, then (x_{-1}, z) is feasible for the modified LP.*

Proof. Since (x, z) is feasible, x_1 is sandwiched between the inequalities of P and N, and hence the inequalities agree with each other on (x_{-1}, z). □

Claim 3. *If (x_{-1}, z) is a feasible solution to the modified LP, then there exists a choice of x_1 such that (x, z) is feasible for the original LP.*

Proof. If (x_{-1}, z) is a feasible solution of the modified LP, then in particular the constraints from $P \times N$ are satisfied, and hence there exists a choice of x_1 that satisfies all upper (lower) bounds in P (N). □

We also note crucially that the value of the objective function is the same as we transfer feasible solutions between the original and modified LP, and hence the optimal values are equal too. Hence, using the above procedure, we can iteratively remove the variables $x_1 \ldots x_n$ leaving behind a one-variable LP in z, which we saw is trivially solved at the beginning of this section.

It remains to analyse the running time of this algorithm.

Claim 4. *The Fourier-Motzkin Algorithm runs in $O(m^{2^n})$ time.*

Proof. Observe that in each iteration $|P| + |N|$ constraints are being replaced by $|P| \cdot |N|$ constraints. In the above construction, if m is the number of constraints before removing x_1, then in the worst-case, these are replaced by $O(m^2)$ constraints. Since we need to perform n iterations to remove all the n variables, the number of constraints can potentially blow up to $O\left(m^{2^n}\right)$ in the one-variable LP, which can then be solved trivially by iterating through the constraints. □

So the Fourier-Motzkin Method is extremely slow — when n and m are both large, its running time can be double exponential in the input size! But observe that in the case of low-dimensional LPs with constant number of variables, the run-time is polynomial in m (albeit still quite slow).

3 Clarkson's Algorithm

We want to solve the following n-variable m-constraints LP:

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & A \cdot x \leqslant b \end{aligned} \tag{2}$$

To present the algorithm, we first need to review some more important preliminaries in linear programming, as well as stating some simplifying assumptions that makes the analysis much easier. We will then present the algorithm.

3.1 Preliminaries and Assumptions

To describe the algorithm, we need a (seemingly) different notion of basic feasible solutions than the one we introduced in Lecture 2, which we define next.

Definition 5. A **basic feasible solution** is a feasible solution $x \in \mathbb{R}^n$ such that there is a subset B of the *rows* of size n such that A_B is full rank and $A_B \cdot x = b_B$[a]. In other words, n linearly independent constraints are tight in the solution x.

[a]Here A_B is the sub-matrix of A induced by rows in B, and similarly for vector b_B.

As in the case of the equational form, we would like to argue that an optimal solution can be found among the basic feasible solutions. However, this is not true, as can be seen with the following example (which is in the same form as Eq (2)):

Example 6. *Consider the following linear program:*

$$\begin{aligned} \max \quad & x + y \\ \text{subject to} \quad & x + y \leqslant 1. \end{aligned}$$

All the points on the line $x + y = 1$ are optimal solutions to this LP, but there are no basic feasible solutions.

But setting aside this trivial situation, we have the following result.

Proposition 7. *If a linear program in the form of Eq (2) has a basic feasible solution, then there is an optimal basic feasible solution.*

We postpone the proof of this result to a later lecture. We need one more definition before we can talk about the algorithm.

Definition 8. A basic feasible solution x is **degenerate** if *more than* n constraints are tight at x.

Recall that degeneracy previously came up in the Simplex algorithm which forced us to make non-improving pivot steps (which in turned led to the issue of cycling). Degeneracy is also become problematic in the analysis of the Clarkson's algorithm. Thus, in the following, we are going to make some related assumptions to simplify the analysis.

Assumptions for Clarkson's Algorithm:

(*i*) The LP has a basic feasible solution.

(*ii*) The LP does *not* have any degenerate basic feasible solutions.

(*iii*) Optimum of any sub-constraints are either unbounded or unique.

Remark. We shall note that *all* of these assumptions can be lifted in Clarkson's algorithm, but it makes the presentation harder, so we will keep them (see the original paper [1]).

Using these assumptions, we can prove the following statements, which form the core idea of the algorithm. Throughout the proof, we let x^* be the optimal basic feasible solution of the original LP with the basis B.

Claim 9. *The following linear programs are equivalent, in the sense that they have the same value of the optimal solution:*

$$\begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & Ax \leqslant b, \end{aligned} \qquad\qquad \begin{aligned} \max \quad & c^T x \\ \text{subject to} \quad & A_B x \leqslant b_B. \end{aligned}$$

In words, even if we throw out all constraints other than B, the objective value does not change.

Proof. Note that any solution that is feasible for the first LP is automatically feasible for the second one, since it contains a subset of the constraints of the original. So the optimal of the second LP is at least as much as the optimal of the original. The proof strategy to show the converse is to exhibit an optimal solution to the dual of the original LP which is also feasible for the dual of the second LP.

Let y^* be the optimal dual solution corresponding to x^*, then since x^* is not degenerate (by assumption (*ii*)) $A_i x_i^* = b_i$ only for $i \in B$. But now by complementary slackness conditions (from Lecture 4) y^* is 0 everywhere outside B. Hence if we just project y^* to the coordinates B, we get a feasible solution for the dual of the second linear program as well with the same value. By the weak duality, this implies that optimal value of the second LP is also at most that of the first one, concluding the proof. □

Claim 10. *Suppose R is any subset of the rows (constraints) and x' is the optimal solution of this LP:*

$$\begin{aligned} x' = \max \quad & c^T x \\ \text{subject to} \quad & A_R \cdot x \leqslant b_R. \end{aligned}$$

If $A_B x' \leqslant b_B$ then x' is also an optimal solution for the original LP.

In words, if we get "lucky" and the optimal solution x' of the LP on a subset of constraints does not violate the basis constraints B of x^*, then in fact $x' = x^*$.

Proof. For this proof, we will consider the following three LPs:

$$\begin{aligned} \max \quad & x \\ \text{s.t} \quad & A_R \cdot x \leqslant b_R. \end{aligned} \qquad (3) \qquad \begin{aligned} \max \quad & x \\ \text{s.t} \quad & A_R \cdot x \leqslant b_R \\ & A_B \cdot x \leqslant b_B. \end{aligned} \qquad (4) \qquad \begin{aligned} \max \quad & x \\ \text{s.t} \quad & A_B \cdot x \leqslant b_B. \end{aligned} \qquad (5)$$

First, note that x' is feasible for (4) by the hypothesis of the claim. But since it is optimal for (3), whose constraints are a subset of (4), it must also be optimal for (4).

On the other hand, x^* (the optimal basic feasible solution with basis B for the original LP) is also feasible for (4), while being optimal for (5) by Claim 9. So by the same argument, it is also optimal for (4).

But now by assumption *(iii)* on the uniqueness of optimal solution, we have $x' = x^*$ and are done. $\square$

At this point, we pause and interpret what the claims above are telling us. First, Claim 9 tells us that if we (magically) had access to the (unique) basis corresponding to the (unique) optimal solution to our linear program, we could forget about all the other constraints, and solve only an n-constraint linear program to recover the optimal solution.

Of course, we do not actually have the basis in hand; but Claim 10 tells us that we do not need to! If we can find a subset R of the constraints such that a feasible solution to these constraints is also feasible for the (unknown) basis, then the claim reduces our problem to solving an $|R|$-constraint LP. The trivial choice of R is to take the whole set, but this does not make any progress — the ingenuity of Clarkson's algorithm is in (randomly and iteratively) constructing a small set R that works.

3.2 The Algorithm

At a high level, the algorithm is as follows. We start by picking a random subset R of the constraints of size roughly $O(n^2)$ and solve the problem optimally only on these constraints using any algorithm available to us[2]. We then check this solution against all constraints of the original LP and collect the violated ones. We re-insert these violated constraints to the system, to increase their "priority", and continue like this. The main part of the proof is the so-called *constraint sampling lemma* that ensures that in each iteration we are not going to insert too many constraints back in the system. On the other hand, Claim 10 implies that we are also always reinserting at least one constraint from the basis back to the system. The outcome of this is that eventually we will "flood" the system with only basis constraints and thus get a random sample that contains all of the basis constraints inside it; by Claim 9 solving this LP is as good as the original one, which concludes the proof.

We now formalize the algorithm.

Clarkson's Algorithm:

Input: An LP in Eq (2) with n variables and m constrains.

1. Let H be *multi-set* of constraints (initially the constraints of the LP).
2. Set $r = 6n^2$ and pick r constraints R uniformly at random from H.
3. Solve $x' = \max \quad c^T x \quad \text{subject to} \quad A_R \cdot x \leqslant b_R$.
4. Let V_R be the *multi-set* of constraints from H violated by x'.
   - If $V_R = \emptyset$, return x' and terminate the algorithm.
   - If $|V_R| \leqslant \frac{2n|H|}{r}$, add V_R to H, and go to step (2) (this is called a *good* iteration).
   - Otherwise (ignore this iteration and) go to step (2) (this is called a *bad* iteration).

An immediate observation about the algorithm is that it only invokes an LP solver on programs of size $O(n^2)$, so the algorithm achieves the dramatic reduction in the size of the LP that we were looking for. Note that the algorithm is trivially correct, since it only returns x' if it satisfies all the constraints of the original LP, and since x' was optimal on a *subset* of the constraints, its value can only be larger than or equal to the

[2]Clarkson's algorithm is basically a *reduction* from solving n-variable m-constraint LPs to solving multiple n-variable $\Theta(n^2)$-constraint LPs instead – given that $n \ll m$, this should lead to a much more efficient algorithm for the problem instead of trying to handle all the m constraints at the same time.

value of x^*; but since it is a feasible solution of the LP we are done. The main part is in proving that the algorithm terminates after not too many iterations (in expectation and with high probability).

Note that whenever V_R is non-empty, it certainly contains at least one constraint from the basis B. The aim behind separating the algorithm into good and bad iterations is to boost the relative weight of the basis constraints. Since we only add V_R into H when it is "small" (i.e. the basis constraints have large relative weight in V_R), we should intuitively expect them to eventually dominate H.

The proof will proceed in two steps. First, we will bound the number of *good* iterations, i.e., the iterations where the violated constraints are added to H.

Lemma 11. *The number of good iterations is at most* $3n \ln m$.

To prove the lemma, we need the following claims:

Claim 12. *After* k *good iterations,* $|H| \geqslant 2^{k/n}$.

Proof. If x' does not violate any basis constraint, then by Claim 10, $x' = x^*$ and thus we will have $V_R = \emptyset$. Hence in each good iteration, at least one of the basis constraints has its number of copies in H doubled. But this means that after k good iterations, by the pigeonhole principle, at least one basis constraint is doubled k/n times, giving us $|H| \geqslant 2^{k/n}$. □

Claim 13. *After* k *good iterations,* $|H| \leqslant \left(1 + \frac{2n}{r}\right)^k m$.

Proof. In a good iteration, $|V_R| \leqslant 2n|H|/r$, which means that H grows by a factor of at most $1 + 2n/r$. Hence after k such steps, we have $|H| \leqslant (1 + 2n/r)^k m$. □

These claims now immediately give us the proof of Lemma 11.

Proof of Lemma 11. The lower bound $2^{k/n}$ on the size of H grows much faster (as a function of k) than the upper bound $(1 + 2n/r)^k \cdot m = (1 + 1/3n)^k \cdot m$. Hence this gives us the following upper bound on the number of good iterations:

$$\text{minimum } k \in \mathbb{N} \text{ such that the following holds:} \quad \left(2^{k/n} > \left(1 + \frac{1}{3n}\right)^k \cdot m\right).$$

We are going to show that $k = 3n \ln m$ already satisfies the equation above, which implies that the number of good iterations is bounded by this quantity, proving the lemma.

Since $1 + x \leqslant e^x$ for all $x \geqslant 0$, we have that for $k = 3n \ln m$,

$$\left(1 + \frac{1}{3n}\right)^k \cdot m \leqslant \exp\left(\frac{k}{3n} + \ln m\right) = \exp\left(2 \ln m\right),$$

while

$$2^{k/n} = \exp\left(\frac{k}{n} \cdot \ln 2\right) = \exp\left((3 \cdot \ln 2) \cdot \ln m\right) > \exp\left(2 \ln m\right),$$

as $\ln 2 > 2/3$. This concludes the proof. □

Lemma 11 bounds the number of good iterations by $3n \ln m$. We are now going to prove that each iteration with probability at least 1/2 is a good iteration, which means that after expected $6n \ln m$ iterations we will be done. Our main tool is the following lemma.

Lemma 14 (*"Constraint Sampling Lemma"*)**.** *In each iteration,*

$$\mathbb{E}\,|V_R| \leqslant \frac{n \cdot |H|}{r}.$$

Proof. First, by definition of the expectation:

$$\mathbb{E}\,|V_R| = \sum_{\substack{R\subseteq H\\ \text{s.t. } |R|=r}} \binom{|H|}{r}^{-1} \cdot |V_R|.$$

Now define the indicator random variable $Y_{h,R}$ that is 1 iff $h \in V_R$ (i.e. the constraint h is violated by the optimal solution to the constraints R – in other words, if we sample R and compute x', we include h in V_R). Then we can rewrite the expectation as:

$$\mathbb{E}\,|V_R| = \binom{|H|}{r}^{-1} \cdot \sum_{\substack{R\subseteq H\\ \text{s.t. } |R|=r}} \sum_{h\in H\setminus R} Y_{h,R}.$$

We use a different way to enumerate over r-multisets (of constraints) and another constraint outside them: we pick an $(r+1)$-multiset of constraints, and choose r of them into R.

$$\mathbb{E}\,[|V_R|] = \binom{|H|}{r}^{-1} \cdot \sum_{\substack{Q\subseteq H\\ \text{s.t. } |Q|=r+1}} \sum_{g\in Q} Y_{g,Q-g} \qquad (Q-g \text{ and } R \text{ are the same})$$

To complete the proof, we make two quick observations. First, if some constraint g is present twice in Q, $Y_{g,Q-g}$ is never 1, because the other copy ensures that g remains satisifed. Second, if B_Q is a basis corresponding to an optimal basic feasible solution for the constraints Q, and $g \notin B_Q$, $Y_{g,Q-g} = 0$ (by Claim 9). As the size of B_Q is at most n (the dimension of the LP), the innermost sum is at most n. Thus,

$$\mathbb{E}\,|V_R| \leqslant \binom{|H|}{r}^{-1} \cdot \sum_{\substack{Q\subseteq H\\ |Q|=r+1}} n = \binom{|H|}{r}^{-1} \cdot \binom{|H|}{r+1} \cdot n = \frac{(|H|-r)\cdot n}{r+1} \leqslant \frac{|H|\cdot n}{r},$$

concluding the proof. □

With Lemma 14 at hand, we can use Markov's bound to get that each iteration is bad with probability at most $1/2$. This implies that the expected number of iterations before having $3n \ln m$ good iterations is $6n \ln m$ iterations. Thus, by Lemma 11, the algorithm takes $6n \ln m$ iterations in expectation (a slightly more careful analysis can also bound the number of iterations with high probability, but we omit that here).

Finally, let $T(n', m')$ denote the runtime of our subroutine in Clarkson's algorithm for solving an n-variable m'-constraint LP. We have that the expected runtime of the Clarkson's algorithm is:

$$T(n, 6n^2) \cdot O(n \ln m) + O(mn \cdot n \ln m);$$

the first term account for solving an LP on constraints R in each iteration, and the second term is for checking which constraints are violated in each iteration which can be done in $O(mn)$ time each. The important fact here is that any LP we solve in this algorithm has at most $6n^2 \ll m$ constraints and thus the bound above can be much better than $T(n, m)$, namely, by just running our subroutine over the entire input. In fact, even if we use the trivial algorithm of Lecture 2 (by enumerating each basic feasible solution) as our subroutine which gives $T(n', m') = (m' + n')^{O(n')}$ time, the runtime of Clarkson's algorithm would be

$$O(n^{O(n)} \cdot \ln m) + O(mn^2 \cdot \ln m)$$

For small values of n, say a constant, or $O(\log \log m)$, which is the premise of a low-dimensional LP, this leads to a near-linear time algorithm for solving such LPs.

References

[1] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. Assoc. Comput. Mach.*, 42(2):488–499, 1995. 3, 6