PRINCETON COS 521: ADVANCED ALGORITHM DESIGN

Linear Thinking and Numerical Algorithms

According to conventional wisdom, *linear thinking* describes a thought process that is logical or step-by-step (i.e., each step must be completed before the next one is undertaken). *Nonlinear thinking*, on the other hand, is the opposite of linear: creative, original, capable of leaps of inference, etc.

From a complexity-theoretic viewpoint, conventional wisdom is startlingly right: linear problems are generally computationally easy, and nonlinear problems usually are not.

Example: Solving linear systems of equations is easy. Solving quadratic systems of equations is NP-hard. (Reason: Using the nonlinear constraint $x^2 = x$, we can force variables to be $0/1$ so quadratic programming can encode the integer program e.g.)

Not all nonlinear problems are difficult, but the ones that turn out to be easy are generally those that can leverage linear algebra (eigenvalues, singular value decomposition, etc.).

In mathematics, too, linear algebra is simple and easy to understand. The goal of much higher mathematics seems to be to reduce the study of complicated (nonlinear!) objects to the study of linear algebra.

Simplest example: Solving systems of linear equations

The following is a simple system of equations.

$$\begin{aligned} 2x_1 - 3x_2 &= 5 \\ 3x_1 + 4x_2 &= 6 \end{aligned}$$

More generally, we represent a linear system of m equations in n variables as $Ax = b$ where A is an $m \times n$ coefficient matrix, x is a vector of n variables, and b is a vector of m real numbers. In your linear algebra course, you learnt that this system is feasible iff the rank of $A|b$ (i.e., the matrix where b is tacked on as a new column of A) has rank at most n. The solution is computed via matrix inversion. One subtlety not addressed in most linear algebra courses is whether this procedure is polynomial time.

You may protest that they point out that the system can be solved in $O(n^3)$ operations. Yes, but this misses a crucial point we will address before the end of the lecture.

Systems of linear inequalities and linear programming

If we replace some or all of the $=$ signs with $\geq$ or $\leq$ in a system of linear equations, we obtain a system of linear inequalities.

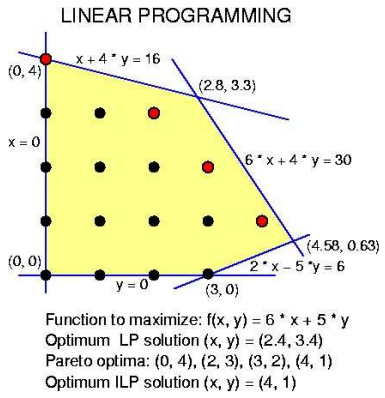

Figure 1: A system of linear inequalities and its feasible region

The feasible region has sharp corners; it is a convex region and is called a *polytope*. However, this method of graphing the inequalities and their feasible region does not scale well with n, m. The number of vertices of this feasible region grows roughly as m^n in general.

In fact a polynomial-time method to determine feasibility of linear inequalities was only discovered in 1979 by Khachiyan, a Soviet mathematician.

We will discuss the core ideas of this method later in the course. For now, we just assume polynomial-time solvability.

In *Linear Programming* one is trying to optimize (i.e., maximize or minimize) a linear function over the set of feasible values. The general form of an LP is

$$\min c^T x \tag{1}$$

$$Ax \geq b \tag{2}$$

Here $\geq$ denotes component-wise "greater than."

This form is very flexible as shown in class. To express *maximization* instead of minimization, just replace c by $-c$. To include an inequality of the form $a \cdot x \leq b_i$ just write it as $-a \cdot x \geq -b_i$. To include an equation as a constraint just replace with two inequalities.

As is clear from the figure above, the optimum of the linear program is attained at some vertex of the feasible region. Thus a trivial but exponential-time algorithm to find the optimum is to enumerate all vertices of the feasible region and take the one with the lowest value of the objective. The famous *simplex* method is a clever method

to enumerate these vertices one by one, ensuring that the objective keeps decreasing at each step.

Example 1. *(Assignment Problem) Suppose n jobs have to be assigned to n factories. Each job has its attendant requirements and raw materials. Suppose a single number captures all of these: c_{ij} is the cost of assigning job i to factory j. Let x_{ij} be a variable that assigns job i to factory j. We hope this variable is either* 0 *or* 1 *but that is not expressible in the LP so we* relax *this to the constraint*

$$x_{ij} \geq 0 \quad and x_{ij} \leq 1 \quad \text{for each } i, j.$$

Each job must be assigned to exactly one factory so we have the constraint $\sum_j x_{ij} = 1$ for each job i. Then we must ensure each factory obtains one job, so we include the constraint $\sum_i x_{ij} = 1$ for each factory j. Finally, we want to minimize overall cost so the objective is

$$\min \sum_{ij} c_{ij} x_{ij}.$$

The assignment problem is equivalent to the maximum weighted bipartite matching problem that you may have seen. There are combinatorial algorithms for the problem (Hint: the max-flow min-cut theorem), but it turns out that the linear program above itself provides a solution.

Fact 1. *The assignment LP admits an integral optimum solution. That is, there is an optimum solution to the LP where each of the x_{ij} variables are set to either* 0 *or* 1*.*

This problem is abstractly studied as the max weight bipartite matching problem: Given a bipartite graph $G = ((A, B), E)$ with edge weights $w : E \to \mathbb{R}_{\geq 0}$ (i.e., the vertices in G can be partitioned into sets A and B and each edge in E is of the form (a, b) for some vertex $a \in A$ and $b \in B$), the max-weight bipartite matching problem is to find a subset of edges $M \subseteq E$ that do not share a vertex while maximizing $\sum_{e \in M} w(e)$. We won't prove it in class but the optimal value of the following linear program returns the max-weight matching:[1]

Observe that the fact does not mean that *every* optimum solution to the LP is integral. This is easy to see – imagine an example where there are two different integral assignments x, x' that are both optimum. Then, notice that $\frac{1}{2}x + \frac{1}{2}x'$ is also an optimum assignment that satisfies all the constraints of the LP and if $x \neq x'$ then, at least some entry of this new assignment must be fractional.

In general one doesn't get so lucky: solutions to LPs end up being nonintegral no matter how hard we pray for the opposite outcome. Next lecture we will discuss what to do if that happens.□

[1] There are (at least) two ways to see this, which we won't prove. But these are some buzzwords if you want to look it up yourself. One way is to use the Birkhoff-Von Neumann Theorem and to treat the fractional matching as a doubly-stochastic matrix, and the integral matching as a permutation matrix. The other way is to consider writing a flow network where the max-flow is equal to the maximum fractional matching, and then using the flow integrality theorem.

In fact linear programming was invented in 1939 by Kontorovich, a Russian mathematician, to enable efficient organization of industrial production and other societal processes (such as the assignment problem).

The premise of communist economic system in the 1940s and 1950s was that centralized planning —using linear programming!— would enable optimum use of a society's resources and help avoid the messy "inefficiencies"of the market system! The early developers of linear programming were awarded the Nobel prize in economics! Alas, linear programming has not proved sufficient to ensure a perfect economic system. Nevertheless it is extremely useful and popular in optimizing flight schedules, trucking operations, traffic control, manufacturing methods, etc. At one point it was estimated that 50% of all computation in the world was devoted to LP solving. Then youtube was invented...

Linear modeling

At the heart of mathematical modeling is the notion of a *system* of variables: some variables are mathematically expressed in terms of others. In general this mathematical expression may not be succinct or even finite (think of the infinite processes captured in the quantum theory of elementary particles). A *linear* model is a simple way to express interrelationships that are linear.

$$y = 0.1x_1 + 9.1x_2 - 3.2x_3 + 7.$$

Example 2. *(Diet) You wish to balance meat, sugar, veggies, and grains in your diet. You have a certain dollar budget and a certain calorie goal. You don't like these foodstuffs equally; you can give them a score between 1 and 10 according to how much you like them. Let l_m, l_s, l_v, l_g denote your score for meat, sugar, veggies and grains respectively. Assuming your overall happiness is given by*

$$m \times l_m + g \times l_g + v \times l_v + s \times l_s,$$

where m, g, v, s denote your consumption of meat, grain, veggies and sugar respectively (note: this is a modeling assumption about you) then the problem of maximizing your happiness subject to a dollar and calorie budget is a linear program. □

Example 3. *(ℓ_1 regression) This example is from Bob Vanderbei's book on linear programming. You are given data containing grades in different courses for various students. You can try to come up with a model for explaining these scores. You hypothesize that a student's grade in a course is determined by the student's innate aptitude, and the difficulty of the course.*

One could try various functional forms for how the grade is determined by these factors, but the simplest form to try is linear. Of course, such a simple relationship will not completely explain the data so you must allow for some error. Denoting by $Grade_{ij}$ *the grade of student* i *in course* j *this linear model hypothesizes that*

$$Grade_{ij} = aptitude_i + easiness_j + \epsilon_{ij}, \tag{3}$$

where ϵ_{ij} *is an error term.*

Clearly, the error could be positive or negative. A good model is one that has a low value of $\sum_{ij} |\epsilon_{ij}|$*. Thus the best model is one that minimizes this quantity.*

We can solve this model for the aptitude and easiness scores using an LP. We have the constraints in (3) for each student i *and course* j*. Then for each* i, j *we have the constraints*

$$s_{ij} \geq 0 \quad and - s_{ij} \leq \epsilon_{ij} \leq s_{ij}.$$

Finally, the objective is $\min \sum_{ij} s_{ij}$*.*

This method of minimizing the sum of absolute values is called ℓ_1-regression *because the* ℓ_1 *norm of a vector* x *is* $\sum_i |x_i|$*.* □

Just as LP is the tool of choice to squeeze out inefficiencies of production and planning, linear modeling is the bedrock of data analysis in science and even social science.

Example 4. *(Econometric modeling) Econometrics is the branch of economics dealing with analysis of empirical data and understanding the interrelationships of the underlying economic variables —also useful in sociology, political science etc.. It often relies upon modeling dependencies among variables using linear expressions. Usually the variables have a time dependency. For instance it may a posit a relationship of the form*

$$Growth(T+1) = \alpha \cdot Interest\ rate(T) + \beta \cdot Deficit(T-1) + \epsilon(T),$$

where $Interest\ rate(T)$ *denotes say the interest rate at time* T*, etc. Here* α, β *may not be constant and may be* probabilistic variables *(e.g., a random variable uniformly distributed in* $[0.5, 0.8]$*) since future growth may not be a deterministic function of the current variables.*

Often these models are solved (i.e., for α, β *in this case) by regression methods related to the previous example, or more complicated probabilistic inference methods that we will study later in the course.*□

Example 5. *(Perceptrons and Support Vector Machines in machine learning) Suppose you have a bunch of images labeled by whether or not they contain a car. These are data points of the form* (x, y) *where* x *is n-dimensional (*n*= number of pixels in the image)and* $y_i \in \{0, 1\}$ *where* 1 *denotes that it contains a car. You are trying to train an algorithm to recognize cars in*

other unlabeled images. There is a general method called SVM's that allows you to find some kind of a linear model. (Aside: such simple linear models don't work for finding cars in images; this is an example.) This involves hypothesizing that there is an unknown set of coefficients $\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n$ such that

$$\sum_i \alpha_i x_i \geq \alpha_0 + error_x \quad \text{if } x \text{ is an image containing a car,}$$

$$\sum_i \alpha_i x_i \leq 0.5\alpha_0 + error_x \quad \text{if } x \text{ does not contain a car,}$$

Then finding such α_i's while minimizing the sum of the absolute values of the error terms is a linear program. After finding these α_i's, given a new image the program tries to predict whether it has a car by just checking whether $\sum_i \alpha_i x_i \geq \alpha_0$ or $\leq 0.5\alpha_0$. (There is nothing magical about the 0.5 *gap here; one usually stipulates a gap or* margin *between the yes and no cases.)*

This technique is related to the so-called support vector machines in machine learning (and an older model called perceptrons), though we're dropping a few technical details (ℓ_2-regression, regularization etc.). Also, in practice it could be that the linear *explanation is a good fit only after you first apply a nonlinear transformation on the x's. For instance let z be the vector where the ith coordinate $z_i = \phi(x_i) = \exp(-\frac{x_i^2}{2})$. You then find a linear predictor using the z's. (How to choose such nonlinear transformations is an art.)* □

One reason for the popularity of linear models is that the mathematics is simple, elegant, and most importantly, efficient. Thus if the number of variables is large, a linear model is easiest to solve.

A theoretical justification for linear modeling is *Taylor expansion*, according to which every "well-behaved"function is expressible as an infinite series of terms involving the derivatives. Here is the taylor series for an m-variate function f:

$$f(x_1, x_2, \ldots, x_m) = f(0, 0, .., 0) + \sum_i x_i \frac{\partial f}{\partial x_i}(0) + \sum_{i_1 i_2} x_{i_1} x_{i_2} \frac{\partial f}{\partial x_{i_1} \partial x_{i_2}}(0) + \cdots .$$

If we assume the higher order terms are negligible, we obtain a linear expression.

Whenever you see an article in the newspaper describing certain quantitative relationships —eg, the effect of more policing on crime, or the effect of certain economic policy on interest rates—chances are it has probably been obtained via a linear model and ℓ_1 regression (or the related ℓ_2 regression). So don't put blind faith in those numbers; they are necessarily rough approximations to the complex behavior of a complex world.

Meaning of polynomial-time

Of course, the goal in this course is designing polynomial-time algorithms. When a problem definition involves numbers (as opposed to combinatorial/discrete inputs we are more familiar with), the correct definition of polynomial-time is "polynomial in the number of bits needed to represent the input. "

Thus the input size of an $m \times n$ system $Ax = b$ is not mn but the number of bits used to represent A, b, which is at most mnL where L denotes the number of bits used to represent each entry of A, b. (We assume that the numbers in A, b are rational, and in fact by clearing denominators we may assume wlog they are integer.)

Let's return to the question we raised earlier: *is Gaussian elimination a polynomial-time procedure?* The answer is yes. The reason this is nontrivial is that conceivably during Gaussian elimination we may produce a number that is too large to represent. We have to show it runs in $\mathsf{poly}(m, n, L)$ time.

First, note that standard arithmetic operations $+, -, \times$ run in polynomial time.

The reason this works out is that size of the numbers produced during the algorithm are related to the determinant of $n \times n$ submatrices of A, and this determinant has value at most $n!2^{Ln}$. To see this, just recall that the formula for determinant of an $n \times n$ matrix is

$$\det(A) = \sum_{\sigma} \operatorname{sgn}(\sigma) \prod_{i} A_{i\sigma(i)},$$

where σ ranges over all permutations of n elements.

The number of bits used to represent determinant is the log of this, which is $n \log n + Ln$, which is indeed polynomial. Thus doing arithmetic operations on these numbers is also polynomial-time.

This has some consequence for linear programming as well. Recall that the optimum of a linear program is attained at a vertex of the polytope. The vertex is defined as the solution of all the equations obtained from the inequalities that are tight there. We conclude that each vertex of the polytope can be represented by $n \log n + Ln$ bits. This at least shows that the solution can be *written down* in polynomial time (a necessary precondition for being able to compute it in polynomial time!).

A cautionary tale It is easy to lull yourself into a false sense of "everything will be alright" with numerical algorithms. After all, it did work out for Gaussian elimination and linear programming, even if we were a little sloppy at first. But we ought to tread carefully as the next example shows:

Example 6 (The sum-of-square-roots problem)**.** *Consider the following innocuous-looking problem: given two n-vertex polygons in the plane with each vertex described by a pair of rational numbers, each with* $\mathsf{poly}(n)$ *bits, decide if the first has a strictly larger perimeter than the second. Notice that the input to this problem is of size* $\mathsf{poly}(n)$*, accounting for the number of bits required to describe each vertex of the polygons.*

There is a simple algorithm to try: for any polygon, we can estimate the perimeter by going over each "edge" of the polygon, estimating its Euclidean length, and adding all the n numbers obtained up. We can then compare the two. The lengths however involve square roots of some rational numbers that, for the natural strategy above, would require approximating the square root by a rational number. Several methods give a strong approximation guarantee so we seem to be in luck – you can obtain a rational estimate of the square root within any ϵ *error in time polynomial in the bit size of the input numbers and* $\mathsf{poly}\log(1/\epsilon)$*. But what* ϵ *should we choose?*

The desired accuracy dictates the choice of ϵ *in estimating the perimeter. Thus, to solve the decision problem we are interested in, we must understand the following question:* given two polygons described by $\mathsf{poly}(n)$-size rational vertices of unequal perimeter, what is the minimum possible difference between their perimeters? *If this quantity can be made arbitrarily small, then the natural strategy to solve the problem above clearly fails since we cannot hope for any* $\epsilon > 0$ *to work. Despite quite a bit of effort so far, it turns out we do not know the answer to the above question. Consequently, we do not know if the polygon comparison problem (or the sum-of-square-roots comparison problem) has a polynomial time algorithm.*

For a very related reason, the Euclidean traveling salesperson problem (TSP but where the vertices are described as points in $\mathbb{R}^d$ *and distances are Euclidean) is known to be NP-hard but not known to be in NP – given a rational c and a TSP tour, we do not know how to verify if the cost of the TSP tour is at most c.*