CAS CS 599 B: Mathematical Methods for TCS

Lecturer: Mark Bun Spring 2022

Lecture Notes 22:

LDPC Decoding, Linear Programming

Reading.

- Guruswami-Rudra-Sudan §16
- Useful introduction to linear programming: "Understanding and using linear programming," Matoušek and Gärtner.

1 Expander Codes

Last time, we talked about expander codes (specifically, low-density parity check codes), whose parity check matrices arise as the adjacency matrices of bipartite expanders. Recall the correspondence between a code C with parity check matrix H and a bipartite graph $G = (L = [n], R = [m], E)$ as

$$C = \{c \in \mathbb{F}_2^n \mid Hc = \mathbf{0}\} = \{c \in \mathbb{F}_2^n \mid \forall j \in [m] : \bigoplus_{(i,j) \in E} c_i = 0\}.$$

We saw that if G is an (α, β)-vertex expander, then the corresponding code is an $[n, n-m, \alpha n + 1]$ linear code. The technical claim driving this connection was:

Claim 1. *For $S \subseteq [n]$, define $U(S) = \{j \in R \mid |S \cap N(j)| = 1\}$. If G is an (α, β)-vertex expander, then for every $|S| \leq \alpha n$, we have $|U(S)| \geq (2\beta - 1)D|S|$.*

The magic of LDPC codes is that they have ridiculously simple (linear time and efficiently parallelizable) decoding algorithms. Here's the whole algorithm: If a codeword bit $\hat{c}_i$ looks wrong in the sense that the majority of the constraints it participates in are violated, then flip it. Keep doing this until there are no more candidates to flip.

Theorem 2. *Let $\beta > 3/4$, $c \in C$ be a codeword, and $\hat{c} \in \mathbb{F}_2^n$ such that $\Delta(\hat{c}, c) \leq \alpha n/4$. Then the above decoding algorithm recovers c.*

This works because of the following sequence of claims:

Claim 3. *If $\hat{c} \in \mathbb{F}_2^n$ is not a codeword, there is a candidate bit to flip.*

Proof. Let S be the set of coordinates on which c and $\hat{c}$ disagree. Then by Claim 1, we have $|U(S)| \geq (2\beta - 1)D|S| > D|S|/2$. Therefore, some $i \in S$ has more than $D/2$ neighbors in $U(S)$, so i is a candidate to flip. □

Claim 4. *The number of satisfied parity checks always increases.*

Since the total number of parity checks is m, the above two claims show that the algorithm terminates at some codeword after this many rounds. The final claim ensures that this is the correct codeword.

Claim 5. *The algorithm can never reach a candidate codeword with $\alpha n/2$ errors.*

Proof. The starting codeword $\hat{c}$ had at most $\alpha n/4$ errors, so the starting number of unsatisfied parity checks is at most $D\alpha n/4$. If the algorithm reaches a candidate codeword with $\alpha n/2$ errors, then Claim 1 implies that the number of incorrect parity checks is greater than $D\alpha n/4$, which contradicts Claim 4. □

2 Linear Programming

We're now going to start the last unit of the course, which is on linear and semidefinite programming and applications to non-linear/non-convex optimization and constraint satisfaction problems. As such, the emphasis will (hopefully?) complement what you'll see in a course on optimization algorithms.

A linear program is an optimization problem where the objective and the constraints are all linear. For example:

$$\begin{aligned} \max \quad & 2x + 3y \\ \text{s.t.} \quad & 4x + 8y \leq 12 \\ & 2x + y \leq 3 \\ & 3x + 2y \leq 4 \\ & x, y \geq 0 \end{aligned}$$

This is an example of an LP in "standard form", which in general looks like

$$\begin{aligned} \max \quad & \langle \vec{c}, \vec{x} \rangle \\ \text{s.t.} \quad & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq 0 \end{aligned}$$

where all vector inequalities are interpreted componentwise.

Geometrically, each linear constraint corresponds to halfspace. The feasible region of vectors satisfying all constraints is an intersection of halfspaces, which if bounded, is a convex polytope. The maximum of an LP is attained at one of the vertices ("basic feasible solutions") of this polytope.

A cornerstone of algorithms is that linear programs can be solved efficiently both in theory and practice. The first practical method, the simplex method, for solving LPs was introduced by Dantzig in the 40s and gave rise to the principle that if you can cast a problem as an LP, you can hope to solve it efficiently. Around 1979, Leonid Khachiyan analyzed the ellipsoid algorithm for solving LPs to prove that they can be solved in polynomial time, in the bit complexity of the input, even in the worst case. Starting with the work of Karmarker on interior-point methods around 1984, there has been a lot of success in designing LP algorithms that both run in provable polynomial time and are efficient in practice.

2.1 LP Variants

There are lots of variations on LPs which are useful both for modeling different problems and for setting up different classes of algorithms to solve them.

Equational form If you don't like inequality constraints, you could also study systems with equality constraint like $\langle \vec{a}, \vec{x} \rangle = b$. An equality constraint can be converted into the two inequality constraints $\langle \vec{a}, \vec{x} \rangle \leq b, -\langle \vec{a}, \vec{x} \rangle \leq -b$. Conversely, an inequality constraint $\langle \vec{a}, \vec{x} \rangle \leq b$ can be turned into an equality constraint by adding a slack variable: $\langle \vec{a}, \vec{x} \rangle + y = b, y \geq 0$.

In general, an LP in equational form looks like $\max \langle c, \vec{x} \rangle$ s.t. $A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}$. The feasible region is the intersection of an affine subspace with the positive orthant in $\mathbb{R}^n$. By removing linearly dependent rows from A, we can without loss of generality assume that $A \in \mathbb{R}^{m \times n}$ for $m \leq n$ and that A has full rank.

Non-negativity If you want to set up an LP where a variable x is not constrained to be nonnegative, you can replace each occurrence of x with $x^+ - x^-$ for $x^+, x^- \geq 0$.

Optimization vs. Search vs. Decision Sometimes you don't want to solve an optimization problem, and are happy just to find *any* feasible point $\vec{x} \in K$, where K is a region specified by linear constraints. This is called the LP search problem. You can relax this further to the LP decision problem: Given a region K, is $K = \emptyset$ or does there exist a $\vec{x} \in K$? Optimization is certainly at least as hard as search, which is at least as hard as decision.

An extremely useful fact is that there are polynomial-time reductions going the other way. That is, given a subroutine that solves the decision version of LP feasibility in polynomial time, one can design poly-time algorithms for search and for optimization. Here are the main ideas behind these reductions:

Search-to-decision reduction. Assume the feasible region is $K = \{\vec{x} \in \mathbb{R}^n \mid A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}\}$ where A has full rank. Then K is non-empty if and only if it intersects one of the coordinate hyperplanes, i.e., there exists an i such that $K_i := K \cap \{x_i = 0\}$ is non-empty. Now by trying at most n possibilities for x_i, one can use a subroutine for LP feasibility to find a non-empty K_i, then add the constraint $x_i = 0$ and recurse. The base case of the recursion is when A has exactly n linearly independent constraints, in which case one just solves the linear system $A\vec{x} = \vec{b}$.

Optimization-to-search reduction. Consider an optimization problem of the form $\max \langle \vec{c}, \vec{x} \rangle$ s.t. $\vec{x} \in K$. One can compute a crude upper bound M such that the objective is finite iff it is at most M, and then solve the feasibility problem $\langle \vec{c}, \vec{x} \rangle \geq M, \vec{x} \in K$ to determine which is the case. Now using $\log(M/\varepsilon)$ rounds of binary search, one can find an ε-approximation to the maximum value of s such that the LP $\langle \vec{c}, \vec{x} \rangle \geq s, \vec{x} \in K$ is feasible, and then use a subroutine for LP search to find such an $\vec{x}$.

2.2 Modeling with LPs

There are plenty of classic examples of combinatorial optimization problems that can naturally be formulated as linear programs, e.g., network and multicommodity flow problems, fitting or separating data points by lines or planes, scheduling jobs, etc. Here's an example of how to use linear programming to solve a problem that isn't obviously captured by an LP.

In the *bipartite maximum cardinality matching* problem, there is a bipartite graph $G = (L, R, E)$. You can think of the left vertices as people and the right vertices as jobs. The goal is to assign as many people to jobs as possible so as to maximize the number of jobs completed. This problem can be cast exactly as

$$
\begin{aligned}
P := \max \quad & \sum_{\ell,r} x_{\ell,r} & \\
\text{s.t.} \quad & \sum_{(\ell,r)\in E} x_{\ell,r} \leq 1 & \forall \ell \in L \\
& \sum_{(\ell,r)\in E} x_{\ell,r} \leq 1 & \forall r \in R \\
& x_{\ell,r} \in \{0,1\} & \forall (\ell,r) \in E
\end{aligned}
$$

This is an "integer linear program" in that it is a linear program with the additional integrality constraints of the form $x_e \in \{0,1\}$. A natural way to try to solve this using an actual linear program is to relax the integrality constraint:

$$
\begin{aligned}
R := \max \quad & \sum_{\ell,r} x_{\ell,r} & \\
\text{s.t.} \quad & \sum_{(\ell,r)\in E} x_{\ell,r} \leq 1 & \forall \ell \in L \\
& \sum_{(\ell,r)\in E} x_{\ell,r} \leq 1 & \forall r \in R \\
& 0 \leq x_{\ell,r} \leq 1 & \forall (\ell,r) \in E
\end{aligned}
$$

Note that a feasible solution to P is also a feasible solution to R, so the maximum value of R is always at least that of P. But a priori, it might be higher, and indeed without additional structural knowledge of the problem, R might be feasible while P is infeasible. It might be the case that the optimal solution to R, say, sets $x_{\ell,r} = x_{\ell,r'} = 1/2$.

It turns out that for bipartite matching, this situation can't happen: Every vertex of the feasible region for the relaxed LP R is integral. There's a nice combinatorial proof of this fact by contraposition. Suppose x is a non-integral feasible point. Then there are two cases:

Case 1: There exists an edge e such that $0 < x_e < 1$ and for every edge e' that shares a vertex with e, we have $x_{e'} = 0$. If this is the case, then we can perturb x_e slightly and stay inside the feasible region, so x can't be a vertex. Specifically, for small enough ε, we have that both $x + \varepsilon \mathbf{1}_e$ and $x - \varepsilon \mathbf{1}_e$ are both feasible, and since x is a convex combination of these points, it isn't a vertex.

Case 2: Every edge e with positive fractional weight shares a vertex with another edge with positive fractional weight.

In this case, let $e_1 = (\ell_1, r_2)$ be an edge for which $0 < x_{e_1} < 1$. Then there must be a different edge $e_2 = (r_2, \ell_3)$ such that $0 < x_{e_2} < 1$. Continue finding such distinct edges until we eventually repeat a vertex. This gives us a simple cycle of even length which, WLOG, takes the form

$$e_1 = (\ell_1, r_2), e_2 = (r_2, \ell_3), e_3 = (\ell_3, r_4), \ldots, e_k = (r_k, \ell_1).$$

Now define the vector Δ by $\Delta_e = 1$ if $e = e_j$ for some odd j and $\Delta_e = -1$ if $e = e_j$ for some even j. Then for sufficiently small ε, we have that both $x + \varepsilon\Delta$ and $x - \varepsilon\Delta$ are feasible, but x is a convex combination of these two points, so it is not a vertex of the feasible region.

It turns out that there's an algebraic way to see a generalization of this fact. Any linear program with feasible region $\{\vec{x} \mid A\vec{x} \leq \vec{b}, \vec{x} \geq 0\}$ where A is *totally unimodular* and $\vec{b}$ is integral has the property that every vertex is integral. Here, a matrix A is totally unimodular if every square submatrix has determinant 0 or ± 1.