# CS264: Beyond Worst-Case Analysis
# Lecture #11: LP Decoding*

Tim Roughgarden†

October 29, 2014

## 1 Preamble

This lecture covers our final subtopic within the "exact and approximate recovery" part of the course. The problem we study concerns *error-correcting codes*, which solve the problem of encoding information to be robust to errors (i.e., bit flips). We consider only binary codes, so the objects of study are $n$-bit vectors. Recall that a *code* is simply a subset of $\{0,1\}^n$; elements of this subset are *codewords*. Recall that the *Hamming distance* $d_H(x,y)$ between two vectors is the number of coordinates in which they differ, and that the *distance* of a code is the minimum Hamming distance between two different codewords.

For example, consider the code $\{x \in \{0,1\}^n : x \text{ has even parity}\}$, where the codewords are vectors that have an even number of 1s. One way to think about this code is as the set of all $(n-1)$-bit vectors, with an extra "parity bit" appended at the end. The distance of the code is exactly 2. If an odd number of bit flips is suffered during transmission, the error can be detected, and the receiver can request a retransmission from the sender. If an even number of bit flips occurs, then the receiver gets a codeword different from the one intended by the sender.

For a code with distance $d$, up to $d-1$ adversarial errors can be detected. If the number of errors is less than $d/2$, then no retransmission is required: there is a unique codeword closest (in Hamming distance) to the transmission, and it is the message originally sent by the receiver. That is, the corrupted transmission can be *decoded* by the receiver.

This lecture studies the computational problem of decoding a corrupted codeword, and conditions under which the problem can be solved efficiently using linear programming. This approach works well for a useful family of codes, described next.



†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

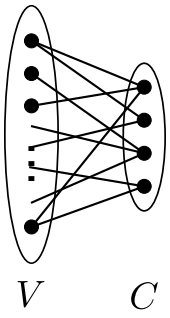


Figure 1: Bipartite graph for expressing parity check codes. The $n$ coordinates of $V$ represent the code word, and each element of $C$ represents a parity check.

# 2 LDPC Codes: From Graphs to Codes

Graphs are very useful for expressing codes and some of their properties. Consider a bipartite graph $G = (V, C, E)$; see Figure 1. The left-hand side nodes $V$ correspond to the $n$ coordinates or variables of a code word. Every right-hand node $j \in C$ corresponds to a "parity check." More formally, a vector $\mathbf{x} \in \{0,1\}^n$ *satisfies* the parity check $j \in C$ if $\mathbf{x}_{N(j)}$ has even parity, where $N(j) \subseteq V$ denotes the neighbors of node $j$ in $G$ and $\mathbf{x}_I$ denotes the vector $\mathbf{x}$ projected onto the coordinates of the set $I$. The code corresponding to $G$ is, by definition, the vectors $\mathbf{x} \in \{0,1\}^n$ that satisfy every parity check of $C$. For instance, the example code of the previous section corresponds to the graph in which $C$ is a single node, connected to all of the variables. For today's lecture, you might want to think of each parity check as containing 20 or so variables, with each variable in 10 or so different parity checks. (With these parameters, $C$ is be roughly half the size of $V$.) These *parity check codes* are also known as binary linear codes (see Homework #6).

Every bipartite graph defines a code, but some graphs/codes are better than others. The goal of this lecture is to identify conditions on the graph $G$ so that it is possible to correct many errors (a constant fraction, even), and moreover in polynomial time (using linear programming).

Following [2], we propose the following conditions.

(C1) Every left-hand side node $i \in V$ has exactly $d$ neighbors, where $d$ is a constant (like 10).

(C2) Every right-hand side node $j \in C$ has at most a constant (like 20) number of neighbors.

(C3) [Expansion condition] There is a constant $\delta > 0$, independent of $n$, such that for all subsets $S \subseteq V$ with size $|S|$ at most $\delta n$,

$$|N(S)| \geq \frac{3}{4} d|S|, \tag{1}$$

where $N(S) \subseteq C$ denotes the nodes (parity checks) of $C$ that have at least one neighbor in $S$. Intuitively, this expansion condition implies that a bunch of errors (represented

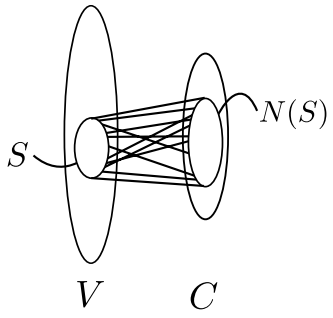


Figure 2: The expansion condition (C3). Every small enough subset $S \subset V$ must have lots of distinct neighbors in $C$.

by $S$) must be reflected in tons of different parity checks (corresponding to $N(S)$). See Figure 2.

In the inequality (1), we use $N(S)$ to denote the vertices of $C$ that have at least one neighbor in $S$. Observe that by (C1), the number of edges sticking out of $S$ is only $d|S|$, so $|N(S)|$ is certainly at most $d|S|$. The condition (1) asserts that the number of distinct neighbors of $S$ is almost as large as the trivial upper bound, and moreover this holds for *every* one of the exponentially many choices for $S$. This sounds like a strong property, so you would be right to question if there really exist any graphs that satisfy (C3). It turns out that *almost all* graphs satisfy the condition — with high probability (approaching 1 as $n \to \infty$), a random graph that satisfies (C1) and (C2) also satisfies (C3). (See Homework #6 for details.) Conditions (C1)–(C3) more or less define the codes known as *low-density parity check (LDPC) codes*.

Our interest is computationally efficient decoding algorithms; as a prerequisite, we establish next the fact that LDPC codes have large distance (linear in $n$). The proof also develops intuition for why the expansion condition (C3) leads to good codes.

**Proposition 2.1** *A code satisfying conditions (C1)–(C3) has distance at least $\delta n$, where $\delta > 0$ is the same constant as in condition (C3).*

*Proof:* Let $\mathbf{x}$ be a code word, and let $\mathbf{z} \in \{0,1\}^n$ be a vector with $d_H(\mathbf{x}, \mathbf{z}) < \delta n$. We need to show that $\mathbf{z}$ is not a code word.

Let $S$ denote the $d_H(\mathbf{x}, \mathbf{z})$ coordinates in which $\mathbf{x}$ and $\mathbf{z}$ differ. We claim that there exists a parity check $j \in C$ that involves exactly one coordinate of $S$. Observe that this claim implies the proposition: since $\mathbf{x}$ is a code word, it satisfies $j$, and since exactly one of the coordinates of $j$ is flipped in $\mathbf{z}$, relative to $\mathbf{x}$, $\mathbf{z}$ does not satisfy the parity check $j$. Hence $\mathbf{z}$ is not a code word.

To prove the claim, note that condition (C1) implies that there are precisely $d|S|$ edges sticking out of the vertices of $G$ that correspond to $S$. Since $|S| < \delta n$, condition (C3) implies that $|N(S)| \geq \frac{3}{4}d|S|$. That is, at least 75% of the edges sticking out of $S$ go to distinct

vertices of $C$. This leaves at most 25% of the edges with the capability of donating a second neighbor of $S$ to a parity check in $N(S)$. It follows that at least

$$\left(\frac{3}{4} - \frac{1}{4}\right) d|S|$$

vertices of $N(S)$ have a unique neighbor in $S$, which completes the claim and the proof. ■

# 3 LP Decoding

We now know the it is possible in principle to decode LDPC codes up to a constant fraction of errors. But can we do it efficiently?

## 3.1 Integer Programming Formulation

We begin with an integer programming formulation of the decoding problem. Given a corrupted code word $\mathbf{z}$, we want to solve the optimization problem

$$\min d_H(\mathbf{x}, \mathbf{z}) \tag{2}$$

subject to

$$\mathbf{x} \text{ is a code word.} \tag{3}$$

It is straightforward to express (2) as a linear objective function. Let $I, J \subseteq \{1, 2, \ldots, n\}$ denote the coordinates for which $z_i = 0$ and $z_i = 1$, respectively. Introduce a binary variable $x_i \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. Then (2) is equivalent to

$$\min \sum_{i \in I} x_i + \sum_{i \in J} (1 - x_i).$$

The "1"s in the second sum contribute the constant value $|J|$ to every solution, so the optimization problem is unchanged if we simplify the objective to

$$\min \sum_{i \in I} x_i - \sum_{i \in J} x_i. \tag{4}$$

Formulating the constraints (3) is trickier. Intuitively, we introduce variables and constraints that ensure that each parity check $j \in C$ is "locally satisfied," and also constraints ensuring that all of the "local solutions" to the parity checks are "globally consistent," in the sense that they are all induced by a single vector $\mathbf{x}$. More precisely, define

$$A_j = \{\mathbf{y} \in \{0, 1\}^{N(j)} \, : \, \mathbf{y} \text{ has even parity}\}$$

as the set of assignments to the variables in the parity check $j$ that satisfy the check. For each check $j \in C$ and satisfying assignment $a \in A_j$, we introduce a new binary decision variable $y_{j,a}$ with the intended semantics that

$$y_{j,a} = \begin{cases} 1 & \text{if } \mathbf{x}_{N(j)} = a \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Observe that the number of such variables scales with $|A_j|$, which in turn is exponential in the number of variables $|N(j)|$ in the parity check. Property (C2) of LDPC codes implies that $|N(j)| = O(1)$, and hence $|A_j| = O(1)$ for every $j \in C$.

Our first set of constraints asserts that every parity check is satisfied:

$$\sum_{a \in A_j} y_{j,a} = 1 \tag{6}$$

for every $j \in C$. These constraints are not enough: we need additional constraints that relate $\mathbf{x}$ to $\mathbf{y}$, as otherwise the optimal solution simply picks independently an assignment $a \in A_j$ for each $j \in C$ and then sets $\mathbf{x} = \mathbf{z}$ to minimize the objective (4).

Our second set of constraints asserts that

$$x_i = \sum_{a \in A_j \,:\, a(i)=1} y_{j,a} \tag{7}$$

for every variable-check pair $(i, j) \in E$, where $a(i)$ denotes the value assigned to $x_i$ by $a$. As intended, $x_i$ will be either 0 or 1, depending on the value chosen for the variable by the local assignment $a \in A_j$ for the parity check $j \in C$. For every $i \in V$, this holds for every local assignment $a \in A_j$ in which $i$ participates (i.e., with $j \in N(i)$). The only way for the constraints (7) to hold for a 0-1 solution is if $\mathbf{y}$ is induced by $\mathbf{x}$ according to the intended semantics in (5).

Summarizing, the feasible 0-1 solutions to the constraints (6) and (7) are precisely the code words, and the feasible solution that optimizes the objective function (4) is the nearest code word to the received message $\mathbf{z}$.

## 3.2 Linear Programming Relaxation

Computing the nearest code word is, in general, an $NP$-hard problem. To obtain a computationally efficient heuristic, we consider the natural relaxation of the above integer program: maximize the objective (4) subject to $\mathbf{x}, \mathbf{y} \geq 0$ and the constraints (6) and (7). We denote this linear program by (LP). It has polynomial size (using (C2)) and hence can be solved in polynomial time.

## 3.3 The Main Result

If the optimal solution (LP) happens to be integral, then it must be the code word nearest to $\mathbf{z}$ — every code word is a feasible and integral solution and the objective function is the Hamming distance. The worry is that the optimal solution to (LP) is fractional, and thus does not yield a code word. Since computing the nearest code word is an $NP$-hard problem in general, we expect this to occur in some instances. Our goal is to identify conditions — on the code-defining graph $G$, and the distance between $\mathbf{z}$ and the nearest code word — under which the optimal solution to (LP) is integral. We'll achieve this goal in the form of the following theorem.

**Theorem 3.1 ([1])** *If the graph $G$ satisfies conditions (C1) and (C3) and the corrupted code word $\mathbf{z}$ has Hamming distance at most $\delta n/2$ from its nearest code word $\mathbf{x}$, then the unique optimal solution to (LP) is integral (and equal to $\mathbf{x}$).*

The constant $\delta$ in Theorem 3.1 is the same one as in the expansion condition (C3).

# 4 Proof of Theorem 3.1

We prove Theorem 3.1 in two parts, which mirror several of our other exact recovery proofs. Section 4.1 identifies a somewhat technical sufficient condition for the exactness of (LP). Section 4.2 proves that the hypotheses of Theorem 3.1 imply that this sufficient condition is met. Before explaining either of these steps, a preliminary observation: by a simple shifting argument (see Homework #6), Theorem 3.1 reduces to the special case in which the nearest code word to the input $\mathbf{z}$ is the all-zero vector (which is always a code word of a parity check code). Thus, for the rest of this section, we assume that the nearest code word to $\mathbf{z}$ is $\mathbf{0}$, and hence $\mathbf{z}$ has at most $\delta n/2$ non-zero variables.

## 4.1 A Sufficient Condition for the Exactness of (LP)

Our sufficient condition for the integrality of the optimal solution to (LP) is derived from weak linear programming duality. No worries if this means nothing to you; we'll include a self-contained argument. We alluded to the idea of a "dual certificate" last lecture: we claimed without proof that the SDP relaxation of the minimum bisection problem has an integral optimal solution, and that the proof involves "guessing and checking" a suitable dual solution. This is exactly the approach we execute here, albeit in the simpler setting of linear (not semidefinite) programs.

The key technical definition is the following.

**Definition 4.1** Let $G = (V, C, E)$ be a bipartite graph and $\mathbf{z} \in \{0,1\}^V$. Let $I \subseteq V$ denote the coordinates with $z_i = 0$, and $J$ the coordinates with $z_i = 1$. The real-valued edge weights $w : E \to \mathcal{R}$ are *feasible* if:

(P1) For every $i \in I$, $\sum_{j \in N(i)} w_{ij} < 1$.

(P2) For every $i \in J$, $\sum_{j \in N(i)} w_{ij} < -1$.

(P3) For every $j \in C$ and distinct $i, i' \in N(j)$, $w_{ij} + w_{i'j} \geq 0$.

Why should you care about this fairly inscrutable definition? Well first of all, existence of feasible weights is a sufficient condition for the integrality of the linear programming relaxation.

**Lemma 4.2** *If $G$ and $\mathbf{z}$ admit a set of feasible weights, then $\mathbf{x} = \mathbf{0}$ is the unique optimal solution to (LP).*

We prove Lemma 4.2 below. The point of the lemma is that it reduces a difficult-to-understand property (integrality of a linear program) to a slightly-less-difficult-to-understand property (existence of feasible weights). For example, suppose $J = \emptyset$ — i.e., $\mathbf{z} = \mathbf{0}$. In this case, it is obvious that $\mathbf{x} = \mathbf{0}$ is the optimal solution to (LP). It is equally obvious that feasible weights exist — just set $w_{ij} = 0$ for all $(i, j) \in E$. Suppose, on the other hand, that $J$ is non-empty but very small. Now it is easier to argue about the existence of feasible weights than to argue directly that the optimal solution to (LP) is integral. For example, one idea is to set $w_{ij} \approx -\frac{1}{d}$ whenever $i \in J$ and $w_{ij} \approx \frac{1}{d}$ whenever $i \in I$. This certainly satisfies properties (P1) and (P2). It satisfies property (P3) if and only if there is no parity check $j \in C$ that contains two variables of $J$ (why?). This is already slightly non-trivial, and in Section 4.2 below we give a more general and powerful version of this argument.

*Proof of Lemma 4.2:* First, observe that $\mathbf{x} = \mathbf{0}$ has objective function value 0. Thus, the goal is to prove that every other feasible solution to (LP) has a strictly positive objective function value. The following derivation may look involved, but really, the Definition 4.1 was chosen to make the derivation work (this is the essence of linear programming duality). All we have to do is follow our nose: at each step, we apply a feasibility condition (of (LP) or of Definition 4.1) or, if all else fails, reverse a double summation.

Let $\mathbf{w}$ denote a set of feasible weights and $(\mathbf{x}, \mathbf{y})$ a feasible solution to (LP). First, by properties (P1) and (P2) of feasible weights and the fact that $\mathbf{x} \geq 0$, we have

$$\sum_{i \in I} x_i - \sum_{i \in J} x_i \geq \sum_{i \in I} \left( \sum_{j \in N(i)} w_{ij} \right) x_i + \sum_{i \in J} \left( \sum_{j \in N(i)} w_{ij} \right) x_i, \tag{8}$$

with equality holding if and only if $\mathbf{x} = 0$. Thus, the proof reduces to showing that the right-hand side of (8) is nonnegative. Continuing the derivation completes the proof:

$$\begin{aligned}
\sum_{i \in I} \left( \sum_{j \in N(i)} w_{ij} \right) x_i + \sum_{i \in J} \left( \sum_{j \in N(i)} w_{ij} \right) x_i &= \sum_{(i,j) \in E} w_{ij} x_i && (9) \\
&= \sum_{(i,j) \in E} w_{ij} \left( \sum_{a \in A_j \,:\, a(i) = 1} y_{j,a} \right) && (10) \\
&= \sum_{j \in C} \sum_{a \in A_j} y_{j,a} \underbrace{\sum_{i \in V \,:\, a(i) = 1} w_{ij}}_{\geq 0 \text{ by (P3)}} && (11) \\
&\geq 0. && (12)
\end{aligned}$$

Equation (9) is just writing the sum of $w_{ij} x_i$ over the edges $E$ in two different ways. Equation (10) follows from the constraints (7), and equation (11) from a reversal of the summations. To see why inequality (12) follows from the the final property (P3) of Definition 4.1 (and the fact that $\mathbf{y} \geq 0$), fix $j \in C$ and $a \in A_j$. By the definition of $A_j$, $a$ assigns a "1"

to an even number of the variables $V$. If this number is 0, then the sum $\sum_{i \in V\,:\,a(i)=1} w_{ij}$ is vacuously 0. If this number is 2, then it is nonnegative by (P3). If it is a bigger even number, the variables $i$ with $a(i) = 1$ can be partitioned into pairs (arbitrarily) and (P3) can be applied to each pair, implying that the sum is nonnegative. ■

## 4.2 Verification of Sufficient Condition

We now prove that, under the hypotheses of Theorem 3.1, there exist feasible weights and hence (by Lemma 4.2) the unique optimal solution of the linear programming relaxation (LP) is the nearest code word to the received transmission $\mathbf{z}$ (as opposed to a fractional solution). Recall that we are assuming (without loss of generality) that the nearest code word is the all-zero word. This means that the indices $I$ (where $z_i = 0$) are precisely the uncorrupted indices, whereas the indices $J$ (where $z_i = 1$) are precisely the corrupted indices. Thus one hypothesis of Theorem 3.1 translates to $|J| < \delta n/2$, where $\delta$ is the constant in the expansion condition in (C3).

Before proving Lemma 4.2, we developed intuition by proving the existence of feasible weights in the case where no parity check contains more than one corrupted variable. This argument is much too weak to prove Theorem 3.1: since we want to tolerate a constant fraction of errors, and each parity check contains only a constant number of variables, there might well be checks containing only corrupted variables.

We now exhibit the feasible weights, following an argument in [3]. To define them, we need to consider three types of variables separately. First, the corrupted variables $J$. Second, the "compromised" variables $K$, defined as uncorrupted variables with lots of corrupted "friends":

$$K = \{i \in I \,:\, \text{at least } 50\% \text{ of the parity checks } N(i) \text{ contain at least one corrupted variable}\}. \tag{13}$$

The third variable type is the uncorrupted and uncompromised variables $V \setminus (J \cup K)$.

We first claim that there are fewer than $\delta n$ corrupted and compromised variables: $|J \cup K| \leq \delta n$. This is the first but not the last point in the proof where we use the expansion assumption (C3). For if not, by supplementing $J$ appropriately, we can choose a set $S$ such that $J \subseteq S \subseteq J \cup K$ and $|S| = \delta n$. Note that the expansion condition (C3) applies to $S$. On the other hand, we can write

$$|N(S)| = \underbrace{|N(J)|}_{\leq d|J| \text{ by (C1)}} + \underbrace{|N(S \cap K) \setminus N(J)|}_{\leq \frac{d}{2}|S \cap K| \text{ by (13)}}.$$

Since $|J| < \delta n/2$ and $|J| + |S \cap K| = |J| + |S \setminus J| = |S|$, we have

$$|N(S)| < \frac{\delta n}{2} \cdot d + \frac{\delta n}{2} \cdot \frac{d}{2} = \frac{3}{4} d|S|.$$

This contradicts the hypothesis that the set $S$ satisfies the expansion condition (C3) and completes the proof of the claim.

We now proceed to defining the weights. Eyeballing the requisite properties (P1)–(P3), the third one seems the hardest to understand. For this reason, we define the weights so that (P3) obviously holds; we'll have to do some work to check that the other two properties hold as well.

The first step is to have each parity check $j \in C$ pick a *favorite* variable $i \in N(j)$. We'll see below that favorite variables are given negative weights, so in light of property (P2) we want corrupted variables to be chosen often as favorites. We'll also see that compromised variables are vulnerable to accumulating positive weights to compensate for the corrupted variables in the same parity checks, and to combat this we also want compromised variables to be chosen often as favorites. You will show on Homework #6 that, because of the expansion condition (C3) and the fact that $|J \cup K| < \delta n$, it is possible to do this in a way that:

(*) each variable $i \in J \cup K$ is chosen as the favorite by at least 75% of the parity checks in which it participates — at least $\frac{3}{4}d$ times.

(Since a parity check can only choose one favorite and might have multiple corrupted variables, it's impossible for every corrupted variable to always be chosen as the favorite.) The argument is simply an iterated application of Hall's theorem.

After favorite variables have been chosen, the weights are defined independently for each group of edges corresponding to a parity check $j$. If $j$'s favorite variable $i$ is corrupted (i.e., is in $J$), then we define $w_{ij} = -\frac{2}{d-\epsilon}$ and $w_{i'j} = \frac{2}{d-\epsilon}$ for every $i' \in N(j)$ other than $i$. Here $\epsilon > 0$ is a sufficiently small positive number.[1] Otherwise, if $j$'s favorite variable $i$ is merely compromised, or is neither corrupted nor compromised, then we set $w_{i'j} = 0$ for all $i' \in N(j)$ (including for $i$).

As promised, the weights are defined so that condition (P3) clearly holds. To verify conditions (P1) and (P2), we treat the variable sets $J$, $K$, and $V \setminus (J \cup K)$ separately. For a corrupted variable $i \in J$, property (*) implies that $w_{ij} = -\frac{2}{d-\epsilon}$ for at least 75% of the parity checks $j \in N(i)$, and $w_{ij} \leq \frac{2}{d-\epsilon}$ for the rest. Thus,

$$\sum_{j \in N(i)} w_{ij} \leq \left(\frac{3}{4}d\right)\left(-\frac{2}{d-\epsilon}\right) + \left(\frac{1}{4}d\right)\left(\frac{2}{d-\epsilon}\right) = \left(-\frac{d}{2}\right)\left(\frac{2}{d-\epsilon}\right) < -1,$$

which verifies (P2). For a compromised variable $i \in K$, property (*) implies that $w_{ij} = 0$ for at least 75% of the parity checks $j \in N(i)$, and $w_{ij} \leq \frac{2}{d-\epsilon}$ for the rest. Thus,

$$\sum_{j \in N(i)} w_{ij} \leq \left(\frac{3}{4}d\right) \cdot 0 + \left(\frac{1}{4}d\right)\left(\frac{2}{d-\epsilon}\right) = \frac{1}{2}\frac{d}{d-\epsilon} < 1,$$

provided $\epsilon$ is sufficiently small. This verifies (P1) for variables $i \in K$. Finally, for a variable $i \notin J \cup K$, strictly more than 50% of the parity checks of $N(i)$ contain no corrupted variables

[1] Recall that in the earlier simplistic argument that worked whenever no parity check had more than one corrupted variable, we defined $w_{ij} \approx -\frac{1}{d}$ for the corrupted variable and $w_{i'j} \approx \frac{1}{d}$ for the rest. In the present variant, we use $\approx -\frac{2}{d}$ to compensate for the fact that not all parity checks containing a corrupted variable will be in position to contribute negative weight to that variable.

(since $i \notin K$). Since $w_{ij} = 0$ for all such parity checks $j$ and $w_{ij} \leq \frac{2}{d-\epsilon}$ for the rest, we have

$$\sum_{j \in N(i)} w_{ij} \leq \left(\frac{1}{2}(d+1)\right) \cdot 0 + \left(\frac{1}{2}(d-1)\right)\left(\frac{2}{d-\epsilon}\right) = \frac{d-1}{d-\epsilon} < 1,$$

provided $\epsilon$ is sufficiently small. This verifies (P1) for the remaining variables, and completes the proof of Theorem 3.1.

# References


[1] J. Feldman, T. Malkin, R. A. Servedio, C. Stein, and M. J. Wainwright. LP decoding corrects a constant fraction of errors. *IEEE Transactions on Information Theory*, 53(1):82–89, 2007.

[2] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[3] M. Viderman. LP decoding of expander codes: a simpler proof. arXiv:1206.2568, 2012.