CS 229r Essential Coding Theory, Lecture 13 Mar 7, 2017

Lecture 13

Instructor: Madhu Sudan *Scribe: Alexander Wei*

1 Overview

Lecture today will cover graph-theoretic codes, as well as linear time encoding and decoding. So far, we have seen codes with relatively efficient runtimes: For example, the best known algorithms for the Reed-Solomon code can encode and decode erasures in time $\mathrm{O}(n \log n)$ and list-decode in time $O(n \operatorname{polylog}(n))$. Thus, a natural question to ask is whether we can improve on this bound and obtain codes with linear time encoding and decoding. It turns out that we can, but at a price. We can correct $\Omega(1)$-fraction errors with rate $R > 0$ using the **low-density parity-check** (LDPC) codes constructed by Sipser-Spielman and Spielman. The state-of-the-art, due a result of Guruswami and Indyk, is codes that correct $\frac{\delta}{2}$ errors uniquely with $R = 1 - \delta$ over large alphabets.

There are two classes of graph-theoretic codes—based on whether the graph represents the parity-check matrix or the generator matrix of the code. (In particular, all graph-theoretic codes are linear codes.) Today, we'll look at the former class of graph-theoretic codes—in particular, expander codes.

2 Expander Codes

Consider a bipartite graph B with N vertices on the left and M vertices on the right. We let $[N]$ and $[M]$ denote the vertices on the left and right, respectively. We now describe how B gives rise to a linear code C_B over $\mathbb{F}_2$. We define the codewords of C_B to be the binary strings $x_1 x_2 \cdots x_N$ such that

$$\bigoplus_{i \leftrightarrow j} x_i = 0$$

for all $j \in [M]$. (Here, the $\leftrightarrow$ symbol denotes adjacency.) In other words, the parity-check matrix is the adjacency matrix of the bipartite graph, where the rows corresponds to vertices in $[N]$ and the columns correspond to vertices in $[M]$. By definition, the rate of this code is $1 - M/N$.

To analyze the distance of this code, we now define some notation. For $S \subseteq [N]$, let $\Gamma(S)$ denote the neighborhood of S. In addition, define

$$\Gamma^{\text{odd}}(S) := \{j \mid \#\{i \mid i \leftrightarrow j, i \in S\} \text{ is odd}\}$$

and

$$\Gamma^{\text{unique}}(S) := \{j \mid \exists!\ i \in S \text{ such that } i \leftrightarrow j\}.$$

By definition, C_B has distance D if and only if $\Gamma^{\text{odd}}(S) \neq \emptyset$ for all subsets $S \subseteq [N]$ of size less than D. However, Γ^{odd} is rather difficult to understand. But we do know that $\Gamma^{\text{odd}}(S) \supseteq \Gamma^{\text{unique}}(S)$. Because Γ^{unique} is more feasible to study, we bound the distance of C_B by showing that $\Gamma^{\text{unique}}(S) \neq \emptyset$ for all $S \subseteq [N]$ such that $|S| < D$.

We now define some graph-theoretic notions that will yield codes with good distance:

Definition 1. *A bipartite graph B with N vertices on the left-hand side and M vertices on the right-hand side is (c, d)-**regular** if every $i \in [N]$ has degree c and ever $j \in [M]$ has degree d.*

We restrict our attention to regular bipartite graphs, since they have the following nice "expansion" property: For $S \subseteq [N]$, notice that $|\Gamma(S)| \leq c|S|$, and if equality holds, then $\Gamma^{\text{unique}}(S) = c|S|$. We'll also want our bipartite graph to have a similar property under slightly weaker constraints. This motivates the following two definitions involving the "expansion" property:

Definition 2. *A (c, d)-regular graph is an (α, δ)-**expander** if for all $S \subseteq [N]$ such that $|S| \leq \delta N$, we have*

$$|\Gamma(S)| \geq \alpha \cdot c|S|.$$

Definition 3. *A (c, d)-regular graph is an (α, δ)-**unique expander** if for all $S \subseteq [N]$ such that $|S| \leq \delta N$, we have*

$$\left|\Gamma^{\text{unique}}(S)\right| \geq \alpha \cdot c|S|.$$

Note that for expander graphs, we'll want α to be as close to 1 as possible.

Lemma 4. *Suppose B is a (c, d)-regular bipartite graph. If B is an (α, δ)-expander, then B is also a $(2\alpha - 1, \delta)$-unique expander.*

Proof. Fix $S \subseteq [N]$ of size at most δN. Let $U = \Gamma^{\text{unique}}(S)$ and $T = \Gamma(S) \setminus U$. Notice that $c|S| \geq |U| = 2|T|$ and that $2|U| + 2|T| \geq 2\alpha \cdot c|S|$. Subtracting the first expression from the second gives us $|U| \geq (2\alpha - 1) \cdot c|S|$ as desired. □

Theorem 5. *Suppose B is a (c, d)-regular bipartite graph that is also an (α, δ)-expander with $\alpha > 1/2$. Then C_B is a code with relative distance at least δ.*

Such codes, built using bipartite expander graphs, are known as **expander codes**.

Exercise 6. *Verify that Theorem 1 follows from the above lemma.*

3 A Bit of History

The first graph-theoretic codes were proposed in 1963 by Gallagher, but due to the limited computational resources at the time, his codes weren't able to be implemented, and so the nice properties of his codes went unnoticed. In 1984, Tanner rediscovered and generalized graph-theoretic codes, replacing the parity-check constraint with more general linear codes, and also brought up the idea of expander graphs. This idea of graph-theoretic codes was again rediscovered by Sipser and Spielman, who put together error-correcting codes and expander graphs. But at that time, explicit constructions of expander graphs with $\alpha > 1/2$ were not yet known. It was only later, in a result due to Capalbo, Reingold, Vadhan, and Wigderson, that the construction of good expanders (with $\alpha > 1/2$) were discovered.

4 Decoding Expander Codes

To decode an encoded message, suppose we have a message $x_1 x_2 \cdots x_N$ with at most a fixed number of errors. We decode this message using the FLIP algorithm, which is specified as follows:

Call a vertex $j \in [M]$ *satisfied* if the parity condition involving j is satisfied by $x_1 x_2 \cdots x_N$. Otherwise, j is *unsatisfied.* For each $i \in [N]$, we count the number of satisfied and unsatisfied neighbors of i. If i has more unsatisfied neighbors than satisfied neighbors, we flip the parity of x_i. We repeat this process until all constraints are satisfied; that is, until $x_1 x_2 \cdots x_N$ becomes a codeword. If not all constraints are satisfied, but no flip will decrease the number of unsatisfied vertices, then we also terminate.

Theorem 7. *Suppose $\varepsilon < \frac{\delta}{1+c}$ and $\alpha > \frac{3}{4}$. If there are at most εN errors, then the FLIP algorithm will run for $O(N)$ iterations and terminate with $x_1 x_2 \cdots x_n$ being the nearest codeword.*

Proof. It is clear that this algorithm will run for at most $N > M$ iterations, because the number of unsatisfied vertices decreases with each iteration. To prove the rest of the theorem, we proving a stronger bound on the number of flipping iterations and then use the fact that the total distance from the initial codeword is at most

$$\varepsilon n + (\text{\# of iterations}) < \delta n.$$

Let S be the set of flipped bits. For a vertex $j \in [M]$ to be unsatisfied, it must be in the expansion of one of the flipped bits. In other words, we must have $j \in \Gamma(S)$. Because $|\Gamma(S)| < c|S|$, there are at most εnc unsatisfied vertices in $[M]$. This observation implies the algorithm runs for at most εnc iterations for ε such that $\varepsilon n(1+c) < \delta n$.

It remains to show we always end at a codeword—that is, the algorithm never gets stuck with unsatisfied constraints. Making this property hold is where the constraint on α comes in. Again, let S be the set of flipped bits, with $\delta n \geq |S|$. Observe that the lemma gives us

$$\Gamma^{\text{unique}}(S) \geq (2\alpha - 1)c|S|,$$

which implies there exists an $i \in S$ with at least $(2\alpha - 1)c$ unique neighbors. Now, if $\alpha > \frac{3}{4}$ holds, then $(2\alpha - 1)c > c/2$. Hence i has a majority of its neighbors unsatisfied, which means should be flipped. $\square$

It is not difficult to show that with the right data structures, this algorithm decodes expander codes in linear time. With parallelization, it is even possible to achieve decoding that uses only $O(\log n)$ rounds of flipping. Recently, Viderman also showed that it is possible to decode in linear time when $\alpha > \frac{2}{3}$. Finally, note that with families of expander codes, only n goes off to infinity; all of the other parameters else remain constant.

Exercise 8. *Show that the decoding algorithm we described indeed takes linear time.*

5 Spielman Codes

We just showed that we can decode in linear time with expander codes, but it turns out that encoding is hard to do efficiently, since only the parity-check matrix is defined. However, Spielman came up with the **superconcentrator codes**, which have both linear time encoding and linear time decoding. We give an overview of such a code for a fixed rate of $R = 1/4$.

Spielman's code for message length k are defined recursively in terms of the code for shorter message lengths. Given a message $x_1 x_2 \cdots x_k \in \{0,1\}^k$, the first k bits of the encoded message is the message itself. We then compute $k/2$ parity-check bits. We use Spielman's code for message length $k/2$ and rate $1/4$ to encode the parity-check bits into $2k$ bits total. This encoded string forms the next $2k$ bits of our encoded message. The final k bits of the encoded message are k parity-check bits computed from this length $2k$ encoding of the parity-check bits. Observe that the runtime of the encoding algorithm is a recurrence that solves to $O(k)$.

We obtain the parity bits in the above construction from a linear code where the adjacency matrix of a sparse bipartite expander acts as the generator. We do the same for the parity check bits of the length $2k$ string. The codes used here are known as **error reduction codes**. If all the parity-check symbols are correct, and the number of message symbols incorrect is at most εk, then our previous flipping algorithm will correct all errors. We can even relax this a little—if the number of parity check-symbols is less than τk and the number of message symbols incorrect is at most εk, then the flip algorithm corrects all but $\frac{1}{2}\tau k$ errors. This follows from executing the previous analysis a little more carefully with $\alpha > \frac{7}{8}$. Applying this fact lets us achieve a more robust decoding. By solving the recurrence, note that the runtime of the decoding algorithm is also linear in k.