18.409 Applied Extremal Combinatorics	Fall term, 1998-99
Lecture 14: November 5, 1998	
Lecturer: Daniel Spielman	Scribe: Rados Radoicic

1 Construction of The Expander Codes

Expander codes fall within a class of codes introduced by Tanner to generalize the low-density parity check codes introduced by Gallager.

A family of expander codes is specified by a linear code of block length d and an infinite family of d-regular expander graphs. Let $\mathcal{G}$ be an infinite family of d-regular graphs such that λ is an upper bound on the second-largest eigenvalue of each, and let C be a linear code of rate r, minimum relative distance δ and block length d. The family of expander codes, $\mathcal{C}(\mathcal{G}, C)$ will be constructed as follows:

> From a graph $G \in \mathcal{G}$ with n vertices, we construct a code of length $dn/2$, called $\mathcal{C}(G, C)$. Each bit of the code is associated with an edge in the graph G. For each vertex of G, we make the restriction that the bits on its edges must form a codeword in C (each edge around a vertex should be associated with a bit in the code C).

We'll show that for reasonable values of $d, \lambda, r, \mathcal{G}$, the code $\mathcal{C}(G, C)$ is a good code.

2 Rate of The Expander Code

Since C has rate r, for each vertex in G, C imposes $(1-r)d$ linear constraints on the edges that are its neighbors. Therefore, the total number of constraints is $\leq nd(1-r)$.

So, we have $\leq nd(1-r)$ linear constraints on $nd/2$ variables. Thus, the space of solutions has rank $> \frac{nd}{2} - nd(1-r) = \frac{nd}{2}(2r-1)$. Therefore, the rate of the expander code $\mathcal{C}(G, C)$ is $2r-1$ (as long as $r > 1/2$, there are solutions with the solution space).

Note: The constraints can never be contradictory since C is linear (and therefore $\bar{0}$ word always satisfies the constraints).

So, we proved the following theorem:

Theorem 1 *The rate of the expander code $\mathcal{C}(G, C)$ is $2r-1$.*

3 The relative minimum distance of the expander code

We'll use the following lemma which was proved earlier in the semester.

Lemma 1 *Let A be a set of αn vertices in G(here, $G \in \mathcal{G}$). There are at most $\frac{nd}{2}\left(\alpha^2 + \frac{\lambda}{d}(\alpha - \alpha^2)\right)$ edges of G between the vertices of A.*

Corollary 1 *Let E be a set of at least $\frac{nd}{2}\left(\alpha^2 + \frac{\lambda}{d}(\alpha - \alpha^2)\right)$ edges in G. Then E touches at least dn vertices.*

proof: It follows from lemma 1

Each word in $\{0,1\}^{dn/2}$ can be associated with the set of edges in G of which it is the characteristic vector. Each codeword can then be viewed as a set of edges in the graph that induce codewords of C at each vertex. The reason that these codes have no low-weight non-zero codewords is that a low-weight codeword would correspond to a small set of edges in the graph that induces codewords at each vertex. But, if the graph is a good expander, then a small set of edges will touch many vertices. In fact, it will touch so many vertices that some vertex will touch fewer than δd of these edges, which implies that these edges cannot induce a codeword at that vertex. To make this argument formal, we use corollary 1 to see that any set of

$\left(\alpha^2 + \frac{\lambda}{d}(\alpha - \alpha^2)\right) \frac{dn}{2}$ edges must have at least αn vertices as endpoints. Thus, at least one of these vertices will have at most $d(\alpha + (1-d)\frac{\lambda}{d})$ of these edges as endpoints. Thus, the word in which these edges are set to 1 and all other are set to zero cannot be a codeword if $\alpha + (1-\alpha)\frac{\lambda}{d} < \mathcal{G}$. This implies the following theorem:

Theorem 2 *The minimum relative distance of th expander code $\mathcal{C}(G, C)$ is at least $\left(\frac{\mathcal{G}-\lambda/\lceil}{1-\lambda/d}\right)^2$.*

4 Linear Codes

A convenient way to make an error-correcting code is to divide the n bits of a code into m message bits and $n - m$ check bits. A *linear code* is one in which the check bits are linear combinations of the message bits. In our case, these are linear combinations over $GF(2)$. Alternatively, one could just define a linear code to be one in which the codewords form an m-dimensional subspace of $GF(2)^n$, as one can always divide such a code into message bits and check bits. A convenient property of linear codes is that the minimum distance between two codewords is equal to the minimum weight of a non-zero codeword, where the *weight* of a codeword is its distance from the all-0 word.

If one defines a code by choosing a random subspace of $GF(2)^n$ of the desired dimension, then one probably obtains a good code. In particular, it probably achieves the Gilbert-Varshamov bound:

Theorem 3 *Let $\delta < 1/2$ and let C be a random linear code of length n and rate $r = 1 - H(\delta)$, where $H(x) = -(x \log x + (1-x)\log(1-x))$ is the binary entropy function. With arbitrarily high probability as n grows large, C has relative minimum distance at least δ.*

It is unknown whether there exist codes over $GF(2)$ with a better tradeoff between rate and minimum relative distance.

5 Special Family of Expanders

For asymptotic purposes, weaker families of expanders suffice. All that we really need is, for an infinite number of degrees $d_1, d_2, \ldots$, an infinite family of d_i-regular graphs in which all graphs of degree d_i have second-largest eigenvalue at most λ_i and

$$\lim_{i \to \infty} \lambda_i / d_i = 0. \tag{1}$$

Of course, graphs with smaller ratios of second-eigenvalue-to-degree will be better for practical purposes.

¿From any infinite family of d-regular graphs with second-largest eigenvalue bounded by some $\lambda < d$, we can produce a family that satisfies (1). Given a d-regular graph on n vertices, G, consider the graph whose adjacency matrix of G. This graph will be a d^2-regular graph on n vertices. Each vertex will have d self-loops. After removing these self-loops, we obtain a $d(d-1)$ regular graph that we call G^2. If the second-largest eigenvalue of G is at most λ, then the second-largest eigenvalue of G^2 will be at most $\lambda^2 - d$. As iteration of this process produces graphs whose ratio of second-largest to largest eigenvalue becomes arbitrarily small, we see that we can obtain a family that satisfies (1).

6 Decoding Expander Codes

One reason that expander codes are exciting is that they contain families that can be decoded in linear time. That is, we can find families of expander codes, $\mathcal{C}(\mathcal{G}, C)$, for which there is a linear-time algorithm that will map to a codeword any word of relative distance at most ϵ from that codeword, for some positive constant ϵ. We begin by describing a logarithmic-time parallel algorithm that performs this decoding task. A natural approach to decoding these codes is to examine the status of each vertex individually. Since the edges touching a vertex were originally a codeword in the coded C, one might attempt to decode by adjusting the edges around each vertex according to the decoding algorithm for C. To specify such an algorithm, we must decide what to do when the vertices that are the endpoints of an edge conflict in their opinions as to

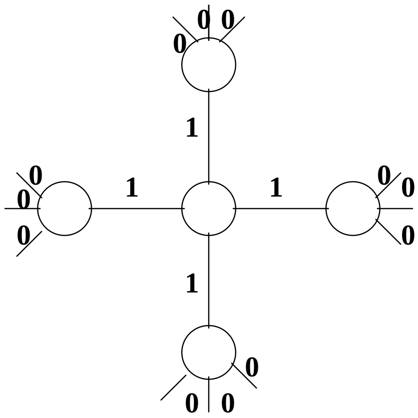

Figure 1: *If the all-1s word is a codeword, then the vertex in the middle thinks it is looking at a codeword, while all the others think that their edge with a 1 is corrupt.*

whether or not that edge should be flipped. Our algorithm will flip an edge if either of its neighbors think it should be flipped. The other natural choice, to flip an edge only if both neighbors think it should be flipped, does not necessarily work: for example, if C contains the all-1's word, and if all of the edges neighboring some vertex are corrupted, then they will form a codeword according to that vertex, but will appear corrupt according to their other neighbors (see Figure 1).

Since C has minimum distance at least δd, it is possible to correct $\delta d/2$ errors made to codewords of C. However, if we try to use C to correct up to $\delta d/2$ errors, we could wind up creating more errors than we started with. If a vertex has $\delta d/2 + 1$ corrupt neighbors, then these corrupt neighbors might bring it closer to a different codeword. In this case, the vertex will think that it has $\delta d/2 - 1$ corrupt neighbors, which actually correspond to edges that are correct. If the decoding algorithm decides to flip these correct edges that the vertex thinks are corrupt, then the number of corrupt edges will increase. This effect is compounded by the fact that each edge can contribute to the confusion of two vertices. To prevent this problem, our algorithm will flip the edges suggested by a vertex only if the state of the edges around that vertex is within distance $\delta d/4$ of a codeword of C. This means that at least $3\delta d/4$ of the edges around a vertex will need to be corrupt before it can cause $\delta d/4$ additional correct edges to become corrupt.

Our parallel decoding algorithm will consist of a logarithmic number of rounds. In each round, each vertex whose edges are within $\delta d/4$ of a codeword will send a *flip* message to each edge that differs from that codeword. Then, each edge that receives at least on *flip* message will flip its bit. To formally analyze the effect of a decoding round, we call a vertex *confused* if it sends a *flip* message to a correct edge, and *unhelpful* if its edges are not within distance $\delta d/4$ of a codeword.

Assume that at most $\alpha\delta d/2$ edges have been corrupted. As each edge touches two vertices, there can be at most

$$\frac{2 \cdot \alpha dn/2}{3\delta d/4} = \frac{4\alpha n}{3\delta}$$

confused vertices and at most

$$\frac{\alpha dn}{\delta d/4} = \frac{4\alpha n}{\delta}$$

unhelpful vertices. The edges that are corrupt after the decoding round are those that receive inappropriate *flip* messages from confused vertices and those that are already corrupt and have both endpoints in unhelpful vertices. The number of inappropriate *flip* messages sent by the confused vertices is at most

$$\frac{4\alpha n}{3\delta}(\delta d/4) = (\alpha/3)\frac{dn}{2}$$

By Corollary 1, the number of edges with both endpoints in unhelpful vertices is at most

$$\left(\left(\frac{4\alpha}{\delta}\right)^2 + (\lambda/d)\left(\frac{4\alpha}{\delta} - \left(\frac{4\alpha}{\delta}\right)^2\right)\right)\frac{dn}{2}$$

Thus, the total number of corrupt edges at the end of the decoding round is at most

$$\left((\alpha/3) + \left(\frac{4\alpha}{\delta}\right)^2 + (\lambda/d)\left(\frac{4\alpha}{\delta} - \left(\frac{4\alpha}{\delta}\right)^2\right)\right)\frac{dn}{2}$$

To ensure that the decoding algorithm will succeed in a logarithmic umber of rounds, we need this term to be less than $\alpha dn/2$ by a constant multiplicative factor. We observe that $(4\alpha/\delta)^2$ can be made arbitrarily small by choosing α to be small. We can then make $(\lambda/d)(4\alpha/\delta)$ arbitrarily small by choosing a family of graphs that satisfy (1). By Theorem 3, we know that we can vary d without changing r or δ. Once we fix d, we can use a constant-size brute-force search to find a code guaranteed by Theorem 3, or just use a code from another explicit asymptotically good family.

To transform this algorithm into a linear-time algorithm, observe that the number of vertices whose edges do not form a codeword of C decreases by a constant factor after each round. As the algorithm only needs to account for those vertices whose edges do not form a codeword of C, the total amount of work performed by the algorithm is linear.

7 Next Time

Next time, we will show how to use the expander codes to improve the algorithm for decoding the low density parity check code.