Lecture 12

Expander Codes

12.1. Cycle Codes

Let $G = (V, E)$ be an undirected connected graph. If an edge e connects a vertex v to some other vertex, then we say that e is *incident* to v, and denote it by $e \dashv v$. If $e = (u, v)$, then we call u and v *adjacent*.

The graph *induced* by a subset E' of the edges is the graph (V', E') where V' is the set of vertices incident to at least one edge in E'. A *spanning tree* T on G is a collection of edges such that the graph induced by these edges is a tree, and such that all vertices in V are incident to at least one edge in T.

A *cycle* in G is a collection of edges in G such that in the graph induced by them all vertices have even degree. A cycle is *primitive* if it is a closed path in which all vertices incident to an edge in the set have degree 2. The *girth* of G is the length of the smallest cycle in G.

Definition 12.1. The *cycle code* associated to G, called $C(G)$, is a code of block length $N = |E|$, which is constructed as follows. Index the coordinates of $\mathbb{F}_2^n$ by the edges of G (after fixing an ordering of the edges). Then $C(G)$ equals the set of all $(x_e \mid e \in E)$ such that for all $v \in V$ we have $\sum_{e \dashv v} x_e = 0$.

The cycle code associated to G has very interesting properties, which we will explore in this section.

Theorem 12.2. *$C(G)$ is an $[n, k, d]_q$-code where $k \geq n - |V|$, and d is the girth of G. Moreover, the codewords in $C(G)$ are characteristic functions of cycles in G.*

Proof. Since every vertex imposes a linear constraint, the assertion on the dimension follows.

The codewods in $C(G)$ are rivially the characteristic functions of cycles: if we put a one on the edges participating in a cycle, then all vertices are incident to an even number of edges carrying a one, and hence the sum of the edge labels of edges incident to the vertex is zero, so that the characteristic function of the cycle is in $C(G)$. Conversely, if we have a codeword in $C(G)$ and take all the edges corresponding to the nonzero positions of the codeword, then all vertices incident to at least one edge in the set have even degree.

It follows that the minimum distance of $C(G)$ is the length of the smallest cycle in G, i.e., it is equal to the girth of G. □

The true minimum distance of a cycle code is larger. To see this, note that

$$\sum_{v \in V} \sum_{e \dashv v} x_e = 0,$$

Lecture by Amin Shokrollahi
Scribes by Amin Karbasi

since in this sum every x_e appears twice, one for each endpoint of e. If $w \in V$, then this means that

$$\sum_{w \neq v \in V} \sum_{e \dashv v} x_e = \sum_{e' \dashv w} x_{e'}, \tag{12.1}$$

so that the constraint posed by w is dependent on the other constraints. This means that the dimension of $C(G)$ is at least $n - |V| + 1$. We will now prove that the dimension is equal to this number, thereby also deriving a linear time algorithm for encoding cycle codes.

Theorem 12.3. *Let T be a spanning tree on G, and $E' = E \setminus T$. Then the positions in E' are information positions of $C(G)$ and there is a simple linear time algorithm which calculates the values on the edges in T given the values on the edges in E'.*

Proof. We will develop an algorithm that maintains at every point a set S of edges the values of which have been calculated up to that point in the algorithm. Moreover, at any point in the algorithm, the set $V \setminus S$ will be a tree T'. At the beginning, $S = E'$, since we will put values on the edges of E' according to the information positions, and $T' = T$.

The leaves of T' are incident to exactly one edge in T'. Therefore, the values of these edges can be calculated using the constraints at the leaves: each value is the sum of the values of the edges in S incident to the leaf. We will add these edges into S, thereby increasing the set S. It is clear that the remaining edges (the ones not in S) still form a tree, because we have separated only the leaves from the tree in the previous round.

It remains to be shown that the labelings on the edges produce a valid codeword. Note that all the vertices of V, except for the root w of T, are forced to satisfy the constraints of the cycle code. Using (12.1), we see that the constraint at the root w has to be satisfied as well.

The algorithm provided above is clearly linear time, since for every vertex v (except the root of T) we are performing $\deg(v) - 1$ operations, where $\deg(v)$ is the degree of v. □

It can be shown that if $G_1, G_2, \ldots$ is a sequence of graphs for which the rates of $C(G_i)$ converges to some R, $0 < R < 1$, then the minimum distance of these codes will be at most $c \log(|E_i|)$ for some positive constant c, where E_i is the set of edges of G_i. We will leave this as an exercise to the reader. Instead, we will prove this for d-regular graphs, where $d > 2$.

Theorem 12.4. *Let $G_1, G_2, \ldots$ be a sequence of connected d-regular graphs with $d > 2$ for which the number of vertices goes to infinity Then there is some universal constant $c > 0$ such that the minimum distance of $C(G_i)$ is at most $c \log_{d-1}(|E_i|)$, where E_i is the set of edges of G_i.*

Proof. The equivalent assertion is to show that the girth of G_i is at most $c \log_{d-1}(|E_i|)$. Let g denote the girth of G_i, and let v be a vertex of G_i. The *neighborhood* of a subset W of vertices of G_i, denoted $\Gamma(W)$, is the set of neighbors of the vertices in W. The *i-th iterated neighborhood* of W is $\Gamma(\Gamma(\cdots\Gamma(W))) =: \Gamma^{(i)}(W)$, where the iteration goes for i steps. Thus, the first iterated neighborhood of W is just $\Gamma(W)$.

Since g is the girth of the graph, the $(g/2 - 1)$-th iterated neighborhood of $\{v\}$ is a tree: otherwise, if a node w appears twice in the iterated neighborhood, there will be two paths of length $< g/2$ from v to w, which gives rise to a cycle of length $< g$ in G_i.

The number of vertices in the k-th iterated neighborhood, under the assumption that this neighborhood is a tree, is equal to

$$|\Gamma^{(k)}(\{v\})| = \frac{d}{d-2}(d-1)^k - \frac{2}{d-2}.$$

Since the nodes in the $(g/2 - 1)$-th iterated neighborhood of $\{v\}$ are distinct, we have

$$\frac{d}{d-2}(d-1)^{g/2-1} - \frac{2}{d-2} \leq |V_i|,$$

where V_i is the number of vertices of G_i. As a result

$$g = O(\log_{d-1}|V_i|) = O\left(\log_{d-1}\frac{2|E_i|}{d}\right) = O(\log_{d-1}|E_i|),$$

and we are done. □

12.2. Strengthening Cycle Codes

Why is the minimum distance of cycle codes with a constant rate small? The reason is that constraints in a valid codeword can be adjacent to as few as two edges, giving rise to cycles being valid codewords. How can we fix this?

There are vaious ways to do so:

1. Consider more general constraints at the nodes. This gives rise to expander code.
2. Consider hyperedges instead of edges. This gives rise to LDPC codes.
3. Use decoding algorithms that allow for decoding errors at the expense of decoding larger fractions of errors. This gives rise to iterative decoding.

We will follow all these approaches in these notes. More specifically, in this lecture, we will discuss the first and the second approach, and study in detail the last approach in the next lectures.

12.3. Expander codes

To build an expander code, we begin with a bipartite expander graph. Say that G is a (c, d)-regular graph between sets of vertices of size n and $\frac{c}{d}n$, and that $d \geq c$. We will identify each of the n nodes on the large side of the graph with one of the bits in a code of length n. We refer to these n bits as *variables*. Ecah of the $\frac{c}{d}n$ vertices on the small side of the graph will be associated with a constraint on the variables. Each constraint will correspond to a set of linear restriction on the d variables that are its neighbors. In particular, a constraint will require that the variables it restricts form a codewrod in some linear code of length d. Becasue the restrictions we impose on the variables are linear, the resulting expander code will be linear as well.

Definition 12.5. Let G be a (c, d)-regular graph between a set of n nodes $\{v_1, v_2, \cdots, v_n\}$, called *variables*, and a set of $\frac{c}{d}n$ nodes $\{c_1, c_2, \cdots, c_{\frac{c}{d}n}\}$, called *check nodes*. Let $g(i, j)$ be a function designed so that, for each constraint c_i the variables neighboring c_i are $v_{g(i,1)}, \cdots, v_{g(i,d)}$. Let S be an error correcting code of block length d. The *expander code* $C(G, S)$ is the codeword of lenght n whose codewords are the words $(x_1, \cdots, x_n)$ such that, for $1 \leq i \leq \frac{c}{d}n$, $x_{g(i,1)}, \cdots, x_{g(i,d)}$ is a codeword of S.

Note that in this terminology a cycle code is a code of the form $C(G, S)$, where P is the $[d, d-1, 2]_2$ parity code. If G is a sufficiently good expander and if the constraints are identified with suffieciently good codes, then the resulting expander code will be a good code. More formally we have the following theorem:

Theorem 12.6. *Let G be a $(c, d, \alpha, \frac{c}{d\varepsilon})$ exoander and S an error correcting code of block length d, rate $r > (c-1)/c$, and minimum relative distance ε. Then $C(G, S)$ has the following propeties:*

(1) *rate*$\geq cr - (c-1)$.

(2) *minimum relative distance* $\geq \alpha$.

Proof. To obtain the bound on the rate of the code, we will count the number of linear restrictions imposed by the constraints. As each constraint imposes $(1-r)d$ linear restrictions, the variables suffer at most

$$n\frac{c}{d}(1-r)d = cn(1-r)$$

linear restrictions, which implies that they have at least $n(cr-(c-1))$ degrees of freedom. To prove the bound on the minimum distance, we will show that there cannot be nonzero word of weight αn or less. let w be a nonzero word of weight at most αn and V be the set of variables that are 1 in this word. There are $c|V|$ egdes leaving the variables in V. The expansion property of the graph implies that these edges will enter more than $(\frac{c}{d\varepsilon})|V|$ constraints. Thus the average number of edges per constraint will be less than $d\varepsilon$, so there must be some constraint that is a neighbor of V, but which has a number of neighbors in V that is less than the minimum distance of S. this implies that w cannot be a codeword in $C(G, S)$. □

Let G be a graph with edge set E and vertex set V. The *edge-vertex incidence graph* of G is the bipartite graph with vertex set $E \bigcup V$ and edge set

$$\{(e, v) \in E \times V : v \text{ is an endpoint of } e\}.$$

To understand the expansion of these edge-vertex incodence graphs, we use a lemma of Alon and Chung

Lemma 12.7. *Let G be a d-regular graph on n vertices with second largest eigenvalue λ. Let X be a subset of vertices of G of size γn. Then, the number of edges contained in the subgraph induced by X in G is at most*

$$\frac{dn}{2}\left(\gamma^2 + \frac{\lambda}{d}\gamma(1-\gamma)\right). \tag{12.2}$$

Now we present an explicit construction of asymptotically good expander codes. We will construct codes of the form $C(G, S)$, where S is itself a good code of constant block length and G is the edge-vertex incidence graph.

Theorem 12.8. *If S is a linear code of rate r, block length d, and minimum relative distance ε, and G is the edge-vertex incidence graph of a d-regular graph with second largest eigenvalue λ, then the code $C(G, S)$ has the following properties:*

(1) *block length* $= \frac{nd}{2}$.

(2) *dimension* $\geq \frac{nd}{2}(2r-1)$.

(3) *minimum distance* $\geq \frac{nd}{2}\left(\frac{\varepsilon - \frac{\lambda}{d}}{1-\frac{\lambda}{d}}\right)^2$.

Proof. Let n be the number of vertices of G, so the number of variables of $C(G, S)$ is $dn/2$ which means the block length is $nd/2$.

Again to obtain the bound on the rate of the code, we will count the number of linear restrictions imposed by the constraints. As each constraint imposes $(1-r)d$ linear restrictions and the number of constraints is n, the variables suffer at most $nd(1-r)$ linear restrictions, which implies that they have at least

$$\frac{nd}{2} - nd(1-r) = \frac{nd}{2}(2r-1)$$

degrees of freedom.

Lemma 12.7 implies that any set of

$$\frac{dn}{2}\left(\gamma^2 + \frac{\lambda}{d}(\gamma - \gamma^2)\right)$$

variables will have at least γn constraints as neighbors. Since each variable has two neighbors, the average number of edges per constraint will be less than

$$\frac{2\frac{dn}{2}\left(\gamma^2 + \frac{\lambda}{d}(\gamma - \gamma^2)\right)}{\lambda n} = d\left(\gamma + \frac{\lambda}{d}(1-\gamma)\right).$$

Therefor if

$$d\left(\gamma + \frac{\lambda}{d}(1-\gamma)\right) < \varepsilon d, \tag{12.3}$$

or equivalently

$$\lambda < \frac{\varepsilon - \frac{\lambda}{d}}{1-\frac{\lambda}{d}},$$

then a word of relative weight $(\gamma^2 + \frac{\lambda}{d}(\gamma - \gamma^2)$ cannot be a codeword of $C(G, S)$. So in particular, $C(G, B)$ cannot have a nonzero codeword of weight

$$\mathbf{d}_{min} \geq \frac{nd}{2}\left(\frac{\varepsilon - \frac{\lambda}{d}}{1-\frac{\lambda}{d}}\right)^2.$$

□

12.4. Decoding

The *flipping algorithm* is the algorithm of choice if we consider codes built from expanders. It comes in many favours. We will limit our discussion to its generic version, so called *sequential* flipping algorithm.

Definition 12.9. Associate to every variable node a current *estimate* of the associated bit. Initially this estimate is equal to recieved value. In each iteration exactly one of these estimates is flipped until either a valid code word is reached or the decoding procedure stops. To decide which estimate to flip, one proceeds as follows. Given the current estimates, call a check node *satisfied* if the modulo two sum of the estimates of its connected variable nodes is zero and *unsatisfied* otherwise. Choose one of those variable nodes which is connected to more unsatisfied constraints than satisfied constraints and flip its estimate. if no such variable node exists stop.

We will now see that for graphs with sufficient expansion the flipping algorithm is guaranteed to correct a linear fraction of errors.

Theorem 12.10. *Let G be a (c, d, α, γ) expander where $\gamma > \frac{3}{4}$. Then the flipping algorithm correctly decodes all error patterns of weight $\frac{\alpha}{2}n$ or less.*

Proof. We call a variable node *good* if its current estimate is correct and *bad* otherwise. Let b_l denote the number of bad variable nodes at the beginning of l-th iteration. Let s_l and u_l denote the number of satisfied and unsatisfied check nodes which are connected to bad variable nodes in the l-th iteration. Note that by definition of the algorithm the sequence u_l is strickly decreasing until the algorithm terminates. This is true since an unsatisfied check node must be connected to bad vraible node and since by definition of the algorithm the total number of unsatisfied check nodes is a strictly decreasing function in the iteration. By assumption

$$b_1 \leq \frac{\alpha}{2}n,$$

and we will show that if $0 < b_l < \alpha n$ then the algorithm flips a variable node in the l-th decoding step and the $b_{l+1} < \alpha n$. Since u_l is strictly decreasing, it will follow that the algorithm does not terminate until u_l has reached zero. To see this claim, assume that $0 < b_l < \alpha n$. From the expansion property we know that

$$s_l + u_l > \frac{3}{4}b_l.$$

Further, a check node which is connected to a bad varaible node but which is satisfied must be connected to at least two bad variables nodes. Since there are exactly $c \cdot b_l$ edges emanating from the set of bad variable nodes, it follows that

$$2s_l + u_l \leq cb_l.$$

Combinig the last two inequalities we conclude that

$$u_l > \frac{c}{2}b_l.$$

In other words the average number of unsatisfied check nodes which is connected to a bad variable node is greater than $\frac{c}{2}$, so that there must exist at least one (bad) variable node which is connected to more unsatisfied check nodes than satisfied ones. It follows that in the l-th step, the algorithm flips a variable node. It remaines to show that $b_{l+1} < \alpha n$. Since $0 < b_l < \alpha n$, and the algorithm flips ter value of only one variable node in each step, the $b_{l+1} \leq \alpha n$. We proceed by contraposition and assume that $b_{l+1} = \alpha n$. By the same reasoning as above we conclude that in this case

$$u_{l+1} > \frac{c}{2} = \frac{c\alpha}{2}n.$$

But this is an obvious contradiction, since as pointed out above, u_l is strictly decreasing and $u_1 \leq l.b_1 \leq \frac{c\alpha}{2}n$. □

12.5. Low Density Parity Check (LDPC) codes

Let's satart this section with the definition of *Tanner graph*.

Definition 12.11. Let C be a binary linear code and let H be a parity-check matrix of C. Assume that H has dimension $m \times n$. Associated to H the following respresentation in terms of a bipartite graph, called the *Tanner* graph. The bipartite graph has n variable nodes, corresponding to the components of the codeword, and m check nodes, corresponding to the set of parity check costraints. Check node i is connected to variable node j if $H_{i,j} = 1$.

Since there are many parity check matrices representing the same code, there are many Tanner graphs corresponding to a given C. Although all of these Tanner graphs describe the same code, they are not equivalent from the point of view of iterative decoder.

Now we are ready to introduce a class of codes which have at least one sparse Tanner graph and called LDPC codes. The main motivation for looking at such codes is that these are the codes for which the iterative algorithm works well.

Definition 12.12. A (c, d)-regular LDPC code is a binary linear code such that every variable node has degree l and every check node has degree d.

The performance of LDPC codes can be dramatically improved by allowing nodes of different degrees. We define an *irregular* LDPC code as a LDPC for which the degrees of nodes are chosen according to some distribution.

Another way of looking at LDPC codes is by using the notion of *hypergraph* which is defined below:

Definition 12.13. A *hypergraph* is a generalization of a graph, where edges can connect any number of vertices. Formally, a hypergraph $H$is a pair (V, E) where V is a set of elements, called nodes or vertices, and E is a set of subsets of V, called *hyperedges*.

While graph edges are pairs of nodes, hyperedges are arbitrary sets of nodes, and can therefore contain an arbitrary number of nodes. Let $V = \{v_1, v_2, \cdots, v_n\}$ and $E = \{e_1, e_2, \cdots, e_m\}$. Every hypergraph has an *incident matrix* $A_{n\times m}(a_{i,j})$ where

$$a_{i,j} = \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{otherwise} \end{cases}$$

The bipartite graph of a hypergraph is directly derived form its indident matrix ie. a check node e_j is connected to a variable node v_i if and only if $v_i \in e_j$.

The LDPC code associated to H is a code of block length $|E|$ such that for each node the modulo two sum of connected hyperedges is zero. Note that in this terminology a cycle code is a LDPC code where all left variable nodes have degree two in the corresponding bipartite graph.