Introduction to Coding Theory **CMU: Spring 2010**

Notes 8: Expander Codes and their decoding

March 2010

Lecturer: Venkatesan Guruswami *Scribe: Venkat Guruswami & Ankit Sharma*

In this lecture, we shall look at Expander Codes, Tanner Codes which are a generalization of Expander Codes and Distance amplification codes. These codes visualize the code as a graph where the value of the i^{th} bit of a codeword corresponds to, for example, the value associated with the i^{th} vertex or the i^{th} edge of the graph. The properties of the graph determine the properties such as distance and efficient decodability of the code. Further, the constraints such as parity checks put on the codewords manifest as local constraints on vertices or edges such as constraints on the set of the values the edges incident to a vertex can take.

The motivation to view linear codes as graphs and use a graph-theoretic approachto construct them comes from the parity-check and generator matrix view of the code. For a $[n, n-m]_2$, we can interpret the parity check matrix as representing a $n \times m$ bipartite graph and then describe the properties of the code from the properties of the graph. An interesting class of codes are Low Density Parity Check (LDPC) codes which have a small number of 1's in each row and column of the parity-check matrix and hence manifest, under the graph view, as sparse graphs.

Let us start with a few notations and definitions.

1. $G(V, E)$ denotes a graph G with vertex set V and edge set E. For a bipartite graph G, we shall denote by L the set of vertices in the left partition of G and by R the set of vertices in the right partition of G. A bipartite graph G will usually be denoted as $G(L \cup R, E)$.

2. A graph G is said to be *d-regular* if each vertex of G has degree d. A bipartite graph G is said to be *d-left regular* if all vertices in the left partition of G have degree d. Similarly, we can define a bipartite graph to be *d-right regular*.

3. For any vertex set $S \subseteq V$ in a graph $G(V, E)$, a vertex $q \in V \backslash S$ is said to be a *neighbour* of S if it is adjacent to some vertex in S. We denote by $N(S)$ the set of neighbours of S. For a bipartite graph $G(L \cup R, E)$, if $S \subseteq L$, then $N(S) \subseteq R$ and vice-versa.

4. For any vertex set $S \subseteq V$ in a graph $G(V, E)$, a vertex $q \in V \backslash S$ is said to be a *unique neighbour* of S if it is adjacent to exactly one vertex in S. We denote by $U(S)$ the set of unique neighbours of S. For a bipartite graph, if $S \subseteq L$, then $U(S) \subseteq R$ and vice-versa.

1 Expander Codes

We first give the definition of a bipartite expander graphs. A bipartite expander graph essentially has the property that every 'small' set of vertices in the left partition has a significantly 'large' neighbourhood in the right partition. Formally,

Definition 1 *A $(n, m, D, \gamma, \alpha)$ bipartite expander is a D-left-regular bipartite graph $G(L \cup R, E)$ where $|L| = n$ and $|R| = m$ such that $\forall S \subseteq L$ with $|S| \leq \gamma n$, $N(S) \geq \alpha |S|$.*

In the above definition, γ gives the measure of a 'small' set and α gives the measure of a 'large' neighbourhood. α is called the *expansion factor*. Note that $\alpha \leq D$ trivially. The following theorem shows the existence of expander graphs that approach this optimal bound.

Theorem 2 *$\forall \epsilon > 0, m \leq n$, $\exists \gamma > 0$ and $D \geq 1$ such that a $(n, m, D, \gamma, D(1-\epsilon))$ expander exists. Additionally, $D = \Theta(\frac{\log(n/m)}{\epsilon})$ and $\gamma n = \Theta(\frac{\epsilon m}{D})$.*

Some remarks about the above expanders:

1. Note that $m \leq n$, so we want expansion from the larger side to the smaller side. This is the harder direction as there is less room to expand into.
2. For a given value of m and n, the parameters of the the above expander are optimal up to $O(1)$ factors and the expansion factor can be brought as close to D at the cost of increasing D.
3. $\gamma n D$ is a trivial lower bound on m since sets of size up to γn expand by a factor of almost D. The above result achieves a value of m that is $1/\epsilon$ times larger than this trivial bound.

The proof of the above theorem is through the probabilistic method. Explicit constructions were also known for $\alpha \approx D/2$, but for a long time there was no explicit construction known with expansion better than $D/2$. Capalbo, Reingold, Vadhan, and Wigderson [2] in 2002 gave an explicit construction of a constant degree expander with expansion $D(1-\epsilon)$ for any desired $\epsilon > 0$, and any desired imbalance ratio m/n.

We now come to the connection between expanders and codes. For this let us define Factor Graphs. For an $[n, n-m]_2$ linear code, construct the bipartite graph F corresponding to the $(n-m) \times n$ parity-check matrix of C as follows

- there is a node in the left partition L_F corresponding to each bit of a codeword in C ($|L_F)| = n$),
- there is a node in the right partition R_F corresponding to each parity check ($|R_F| = n - m$),
- and there is an edge between a node u in L_F and node v in R_F if and only if the bit corresponding to u participates in the parity check corresponding to v in C.

Such a bipartite graph F is called the *factor graph* of C. We can see that the above construction establishes a correspondence between linear codes and bipartite graphs. The nodes on the left are the bits of the code and the nodes on the right are the parity checks. Each codeword assigns each node on the left the value of the corresponding bit. An assignment of values to the nodes on the left is a valid codeword if and only if it satisfies all parity checks i.e. if we denote by V_q the value of the node q on the left, then V corresponds to a valid codeword if and only if $\forall u \in R_F, \sum_{j \in N(u)} V_j = 0$. The above construction also indicates how we can construct a parity-check matrix corresponding to a bipartite graph that has fewer nodes in the right partition compared to the left partition.

Expander codes are linear codes whose factor graphs are bipartite expander graphs. Let us denote the code corresponding to an expander graph G by $C(G)$.

We now establish a useful property of bipartite expander graphs with expansion close to degree D.

Lemma 3 *Let G be a $(n, m, D, \gamma, D(1-\epsilon))$ expander graph with $\epsilon < 1/2$. For any $S \subseteq L_G$ such that $|S| \leq \gamma n$, $U(S) \geq D(1-2\epsilon)|S|$.*

PROOF: The total number of edges going out of S is $D|S|$ by virtue of G being D-left-regular. By the expansion property, $N(S) \geq D(1-\epsilon)|S|$. Hence, out of $D|S|$ edges emanating out of S, at least $D(1-\epsilon)|S|$ go to unique vertices which leaves at most $\epsilon D|S|$ edges. Therefore, at most $\epsilon D|S|$ vertices out of the at least $D(1-\epsilon)|S|$ vertices in $N(S)$ can have more than one incident edge. Hence, $U(S) \geq D(1-2\epsilon)|S|$. □

The above already implies that the distance $\Delta(C(G))$ of the code satisfies $\Delta(C(G)) \geq \gamma n$. (Why?) The next theorem indicates this bound by roughly a factor of two.

Theorem 4 *Let G be a $(n, m, D, \gamma, D(1-\epsilon))$ expander. Then $\Delta(C(G)) \geq 2\gamma(1-\epsilon)n$.*

PROOF: Since $C(G)$ is a linear code, it is sufficient to establish that the weight of any non-zero codeword is at least $2\gamma(1-\epsilon)n$. For contradiction, let p be a codeword of Hamming weight less than $2\gamma(1-\epsilon)n$. Let S be the set of nodes in G which are set to 1 in p.

Clearly, the parity check of any node in $U(S)$ cannot be satisfied since the parity-check-node shares an edge with exactly one node in set S and this node has value 1 by virtue of being in S and edges to nodes in $L \setminus S$ cannot satisfy the parity since the value of those nodes is 0. Hence, if we show that $U(S)$ is non-empty, we have shown that the purported codeword does not satisfy all parity checks which is a contradiction.

If $|S| \leq \gamma n$, then we are through since by Lemma 3, $|U(S)| \geq D(1-2\epsilon)|S|$. If $|S| \geq \gamma n$, then choose a subset Q of S having exactly γn nodes. By Lemma 3, $|U(Q)| \geq D(1-2\epsilon)|Q| = D(1-2\epsilon)\gamma n$.

Now $|S \setminus Q| < \gamma(1-2\epsilon)n$ since $|S| < 2\gamma(1-\epsilon)n$. Hence, there are less than $D\gamma(1-2\epsilon)n$ edges emanating out of $S \setminus Q$ and since $|U(Q)| \geq D(1-2\epsilon)\gamma n$, there cannot be an edge incident to each node in $U(Q)$ from $S \setminus Q$. Hence, there exists a node in $U(Q)$ which has exactly one edge incident to it from S and therefore $U(S) \neq \emptyset$ which completes the proof. □

The rate of expander code is at least $1 - m/n$ and hence if m is smaller by a constant factor compared to n, the expander code has rate bounded away from 0. Thus codes whose factor grpahs are unbalanced expanders with expansion factor $(1-\epsilon)D$ for $\epsilon < 1/2$ are asymptotically good, and therefore explicit constructions of such expanders immediately gives an explicit construction of asymptotically good odes. This was the first constructive method for asymptotically good codes that did not rely on code concatenation. This is in itself a nice feature, but we will now see that these expander codes also admit very efficient (linear time) decoding algorithms.

1.1 Decoding algorithm for Expander Codes

We now present a decoding algorithm for correcting all error patterns of Hamming weight less than $\gamma(1-2\epsilon)n$ assuming $\epsilon < 1/4$ (i.e., when the expansion factor exceeds $3D/4$). Let r be the received word and let V_q be the value assigned to node q in L_G by r. Each parity check in R_G is either satisfied or unsatisfied. The algorithm proceeds by flipping the value of those nodes in L_G which have more unsatisfied checks in their neighbourhood than satisfied checks. This process continues till there no unsatisfied checks left.

Algorithm

While there exists a node y in L_G with more unsatisfied than satisfied checks in $N(y)$

- Flip V_y and update the list of satisfied and unsatisfied checks in R_G.

This completes the description of the algorithm. We now prove the correctness and analyze the time complexity of the algorithm. First we prove that if the number of errors is at most γn, there must exist a node on the left whose neighbourhood of D check nodes has more unsatisfied checks than satisfied checks. This ensures that the algorithm will get started.

Lemma 5 *If the number of errors is at most than γn (and at least 1), then there exists a node in L_G which is adjacent to more then $D/2$ unsatisfied checks. (This assumes that $\epsilon < 1/4$.)*

Proof: Let $T \neq \emptyset$ be the set of error locations. Since $|T| \leq \gamma n$, by Lemma 3, $|U(T)| \geq D(1-2\epsilon)|T| > \frac{D}{2}|T|$ if $\epsilon < 1/4$. All checks in $U(T)$ are clearly unsatisfied. Hence, there exists a node in T that is adjacent to more than $D/2$ unsatisfied checks. Since each node in L_G has D neighbours, therefore the above lemma implies that the node would have more unsatisfied than satisfied checks. □

Lemma 6 *If we start with a received word having less than $\gamma(1-2\epsilon)n$ errors then we can never reach a word with γn errors in any interim step of the algorithm.*

Proof: We flip a node on the left only when the number of unsatisfied checks is greater than the number of satisfied checks in its neighbourhood. Thus with each flip the number of unsatisfied checks decreases by at least 1.

The received word has less than $\gamma(1-2\epsilon)n$ errors and therefore it has less than $D\gamma(1-2\epsilon)n$ unsatisfied checks (by D-left-regularity of the graph) to begin with. If we ever reach an intermediate string with γn errors, then the set of error locations would have at least $D(1-2\epsilon)\gamma n$ unique neighbours (by Lemma 3) and hence there would be at least as many unsatisfied checks. This contradicts the facts that we start with less than $D\gamma(1-2\epsilon)n$ unsatisfied checks and this number cannot increase. □

1.1.1 Correctness of the algorithm

By Lemmas 5 and 6, we see that if we start with an erroneous codeword with less than $\gamma(1-2\epsilon)n$ errors, the algorithm will always find a node (participating in more unsatisfied checks than satusfied checks) to flip. With each node flip, the number of unsatisfied checks goes down by at least 1 and hence the algorithm terminates in at most m iterations. Lemma 6 proves that during the course of the algorithm, there won't be any stage at which the number of errors in the current word (compared to the originally closest codeword) is at least γn. Since the distance of the code is at least $2\gamma(1-\epsilon)n > \gamma n$ ($\because \epsilon < 1/2$), the final codeword to which the algorithm converges must be the original closest codeword.

1.1.2 Running Time of the algorithm

Let d be the maximum degree of a node in R_G.

1. Preprocessing Stage: The computation of all unsatisfied nodes in R_G as a preprocessing step takes $O(md)$ time. As part of preprocessing step, we also associate with each node in L_G the number of unsatisfied checks it is part of and make a list, call it Q, of nodes in L_G which have more unsatisfied than satisfied checks. This takes an additional $O(Dn) = O(Dmd)$ time.
2. Time complexity of each iteration: In each iteration instead of searching for a node with more unsatisfied than satisfied checks, we remove an element from list Q. Further, after flipping the node, we update the list of unsatisfied checks in R_G in $O(D)$ time. We take further $O(Dd)$ time to update the number of unsatisfied checks associated with each element in L_G and to insert any element which now have more unsatisfied than satisfied check into Q and to remove elements from Q which due to the flip have lesser unsatisfied than satisfied checks. Hence, each iteration can be implemented in $O(Dd)$ time.
3. Number of Iterations: The original number of unsatisfied checks can be at most m. As argued above, in each iteration the number of unsatisfied checks reduces by at least 1. Thus the total number of iterations is at most m.

Hence the algorithm can be implemented to run in $O(Ddm) = O(n)$ time when D, d are constants.

Note: These codes being linear can be encoded in $O(n^2)$ time by multiplying the message vector by the generator matrix.

2 Tanner Codes

Let G be a $n \times m$ bipartite graph which is d-right regular and let $C_0 \subseteq \mathbb{F}_2^q$ be a binary linear code.

Definition 7 (Tanner code) *The Tanner code $X(G, C_0)$ is defined as the set*

$$\{c \in \mathbb{F}_2^n | \forall u \in R_G, c_{|N(u)} \in C_0\}$$

where $c_{|N(u)} \in \mathbb{F}_2^d$ denotes the subsequence of c formed by the bits corresponding to the neighbours of u in L_G.

Remark 8 *We note the following:*

1. *Tanner codes are linear codes since C_0 is a linear code.*
2. *Tanner Codes are a generalization of expander codes since in expander code C_0 was chosen to be the $[d, d-1, 2]_2$ parity check code.*

We claim that the dimension of $X(G, C_0)$ is at least $n - m(d - dim(C_0))$. This is because for each $i \in R_G$, the condition that $c_{|N(i)} \in C_0$ imposes $d - dim(C_0)$ independent linear constraints on the bits of c. Therefore, the condition $\forall i \in R_G, c_{|N(i)} \in C_0$ imposes at most $m(d - dim(C_0))$ independent linear constraints and hence the claim follows.

The usefulness of Tanner codes comes from the fact that they require a much lower expansion factor through the choice of an appropriate C_0. In expander codes which are a special case of Tanner codes, we used parity check codes for C_0. Since parity check codes have distance 2, we required that all sets of size at most γn expand by a factor of more than $D/2$ in order to argue that the code had distance at least γn. However, if we use a local code C_0 of distance d_0, then we only require an expansion factor exceeding D/d_0 to ensure the same code distance. Hence a good choice of C_0 allows us to construct explicit Tanner codes using graphs with weaker expansion properties which are therefore easier to construct.

In order to construct codes with a high rate, we require a highly unbalanced bipartite graph so that there are much fewer parity checks than the number of bits in a code word. One way of achieving this is through the construction of Edge Vertex Incidence Graph which we define next.

Definition 9 *The Edge Vertex Incidence Graph of a graph $G = (V, E)$ is defined as the bipartite graph $H_0 = (L \cup R, E')$ where L has a node corresponding to each edge in E, R has a node corresponding to each node in V and an edge exists in E' between each node e in L and corresponding u_e and v_e in R where u_e and v_e are the nodes in R corresponding to the end-points of edge e in E.*

In the graph-code correspondence which we had so far, the bits of the codeword used to sit on nodes in the left partition and the constraints used to be imposed by nodes in the right partition. The edges of graph G form the left partition of H_0 and the nodes of graph G form the right partition of H_0. Hence, we can equivalently view the codeword bits as "sitting" on the edges of the graph G, with each node of the graph G imposing a local constraint on the values on the d edges incident at that node.

Let G be a d-regular graph with N vertices and $Nd/2$ edges. Under the modified view of code-bits sitting on the edges of the graph, the Tanner code $X(H_0, C_0)$ can alternately been defined directly in terms of G as

$$T(G, C_0) = \{c \in \mathbb{F}_2^{\frac{Nd}{2}} | \forall v \in V(H), c_{|\Gamma(v)} \in C_0\} .$$

(We use the notation $T(\cdot, \cdot)$ instead of $X(\cdot, \cdot)$ to highlight this distinction.) Again, $T(G, C_0)$ is a linear code of dimension at least $Nd/2 - N(d - dim(C_0))$ and a sufficient condition for positive dimension is that $dim(C_0) > \frac{d}{2}$.

We will now use for G a good "spectral" expander.

Definition 10 *A graph $G = (V, E)$ is said to be a (n, d, λ)-expander if G is a d-regular graph on n vertices and $\lambda = min\{\lambda_2, |\lambda_n|\}$ where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ are the eigenvalues of the adjacency matrix of H.*

What makes (n, d, λ) expander useful? This is because of Expander Mixing Lemma, which we state next and crucially use in our analysis. We do not prove this lemma here, but it is not hard to prove and a proof can be found in several places (eg. the book by Alon and Spencer).

Lemma 11 (Expander mixing lemma) *Let $G = (V, E)$ be a (n, d, λ) expander graph. Then $\forall S, T \subseteq V$ we have*

$$\left| \; |E(S,T)| - \frac{d|S||T|}{n} \; \right| \leq \lambda \sqrt{|S||T|} \tag{1}$$

where $|E(S, T)|$ is the number of edges between sets S and T with the edges in $(S \cap T) \times (S \cap T)$ counted twice.

Remark 12 *Some comments:*

1. *Since G is a d-regular bipartite graph, therefore the highest eigen value λ_1 would be d.*
2. *Since G is a d-regular graph, we know that there are $d|S|$ edges coming out of S. Now, if it were a purely random graph, we would expect that $|T|/n$ fraction of these edges to end up in T. Thus the expected value of $|E(S, T)|$ would be $\frac{d|S||T|}{n}$. The expander mixing lemma upper bounds the deviation of $|E(S, T)|$ from the random graph expected value in terms of how small the second largest eigenvalue (in absolute value) is in comparison to d. Note that the lemma bounds this deviation for all pairs of sets S, T.*

For technical reasons, we will find it convenient to work with a "bipartite version" of G called its double cover instead of G itself.

Definition 13 (Double cover) *The Double Cover of a graph $G = (V_G, E_G)$ is defined as the bipartite graph $H = (L_H \cup R_H, E_H)$ with both left and right partitions of graph H being equal to V_G i.e. $L_H = R_H = V_G$ and $\forall (u, v) \in E_G$, there is (u, v) and (v, u) in E_H. Hence, the double cover H of graph G has two copies of each node of G, one in the left partition and the other in the right partition, and there are two copies of each edge (u, v) of G.*

Our final code will be $T(H, C_0)$ for suitable C_0 where H is the double cover of an (n, d, λ)-expander G. Working with the bipartite graph H (instead of G) makes the description and analysis of the decoding algorithm simpler and cleaner.

Hence, if G is (n, d, λ) expander, we now take H, the double cover of G. The code-bits still sit on the edges of H as they were in G and the constraints on the values which the edges can take are also essentially the same as they were in G. However, as we will see the analysis becomes a lot more simplified.

Clearly H is a $n \times n$ d-regular bipartite graph i.e. there are n nodes in both left and right partition and each node has d neighbours. What is the analogue of expander mixing lemma for H? It is

essentially the same. Consider the $n \times n$ matrix corresponding to H that has 1 in $(i,j)^{th}$ position if there is an edge in H between the i^{th} node in left partition and j^{th} node in right partition else it is 0. This is exactly the adjacency matrix of G. Hence $d = \lambda_1 \geq \lambda_2 \geq \lambda_3 \cdots \geq \lambda_n$ would be the eigenvalues of the matrix (same as that of adjacency matrix of G) and define $\lambda = \min\{\lambda_2, |\lambda_n|\}$.

Lemma 14 (Expander mixing lemma: bipartite version) *Let H be a $n \times n$ d-regular bipartite graph with λ as defined above. Then $\forall S \subseteq L, T \subseteq R$,*

$$\left| \; |E(S,T)| - \frac{d|S||T|}{n} \; \right| \leq \lambda \sqrt{|S||T|}$$

where $E(S,T)$ is the set of edges between sets S and T. The double counting which we had stated explicitly in Lemma 11 is implicit here.

Theorem 15 *Let $C_0 \subset \mathbb{F}_2^d$ have distance $\geq \delta_0 d$. Then the relative distance of $T(H, C_0)$ is $\geq \delta_0(\delta_0 - \frac{\lambda}{d})$*

PROOF: Since $T(H, C_0)$ is a linear code, it will be sufficient to prove that no non-zero codeword has weight less than $\delta_0(\delta_0 - \frac{\lambda}{d})nd$.

Let c be a codeword in $T(H, C_0)$ and let F be the set of edges in H which have their corresponding bits non-zero in c. Let S be the set of nodes in L which have at least one edge belonging to F incident on them. Similarly, let T be the set of nodes in R which have at least one edge from F incident on them.

Since the distance of C_0 is $\delta_0 d$ any node in H which is incident to a non-zero edge (i.e., an edge of F) should have at least $\delta_0 d$ edges from F incident to it. (This is because we know that any codeword $c \in T(H, C_0)$ satisfies the condition that for each node in H, the values assigned to the edges incident to the node forms a codeword in C_0.)

This implies that each vertex in S and T must have at least $\delta_0 d$ non-zero edges incident to it.

Hence, $|F| \geq \delta_0 d|S|$ and $|F| \geq \delta_0 d|T|$ and therefore, $|F| \geq \delta_0 d\sqrt{|S||T|}$.

Now, $|F| \leq |E(S,T)|$ and by Expander Mixing Lemma, $|E(S,T)| \leq \frac{d|S||T|}{n} + \lambda\sqrt{|S||T|}$. From above we get,

$$\delta_0 d\sqrt{|S||T|} \leq \frac{d|S||T|}{n} + \lambda\sqrt{|S||T|}$$

which implies that $\sqrt{|S||T|} \geq (\delta_0 - \frac{\lambda}{d})n$. Recalling that $|F| \geq \delta_0 d\sqrt{|S||T|}$, we conclude $|F| \geq \delta_0(\delta_0 - \frac{\lambda}{d})nd$ which completes the proof. □

Remark 16 *Note that when H is the $n \times n$ complete bipartite graph, the code $T(H, C_0)$ is simply the tensor product of C_0 with itself and thus has relative distance exactly δ_0^2. The above theorem states that for a good expander with $\lambda = o(d)$, in the limit of large degree d, the relative distance becomes $\approx \delta_0^2$. Thus we can obtain distance as good as the product construction, but we can get much longer codes (compared to the product construction, which only gives a code of block length d^2 starting with a code of block length d).*

2.1 Rate-Distance Tradeoff

Denote the rate of code C_0 by R. Then rate of $T(H, C_0)$ is at least

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

Let $\delta = \delta_0^2$ be approximately the relative distance of $T(H, C_0)$ in the limit of large d. By picking C_0 to satisfy $R \geq 1 - h(\delta_0)$ (meeting the Gilbert-Varshamov bound), we obtain the following rate vs relative distance for $T(H, C_0)$:

$$R(T(H, C_0) \geq 2(1 - h(\delta_0)) - 1 \approx 1 - 2h(\sqrt{\delta}) \ .$$

The rate is positive $\delta < 0.01$. This implies that we get a positive rate for $T(H, C_0)$ only when the relative distance of $T(H, C_0)$ is rather small.

2.2 Algorithm for Decoding

Sipser and Spielman in 1995 [3] introduced the above construction Tanner codes and gave an algorithm to correct a fraction $\delta_0^2/48$ of errors. Zémor in 2001 [4] gave an improved algorithm to correct a fraction $\delta_0^2/4$ of errors for $T(H, C_0)$ by exploiting the bipartiteness of H. We describe this algorithm and its analysis now.

Specifically, we will give an algorithm to decode a received word to the correct codeword assuming that the number of errors is at most $(1-\epsilon)\frac{\delta_0}{2}(\frac{\delta_0}{2} - \frac{\lambda}{d})nd$ which is roughly a quarter of the distance of the code.

Let $y \in \mathbb{F}_2^{|E|}$ be the received word which we wish to decode. Denote by $y_{|\Gamma(u)} \in \mathbb{F}_2^d$ the subsequence of y formed by the bits corresponding to the edges incident to u.

Define a *Left-decoding iteration* to be the following step performed in parallel for all left nodes:

> $\forall v \in L$ replace $y_{|\Gamma v}$ by $c \in C_0$ such that $\Delta(c, y_{|\Gamma(v)})$ is minimized.

We can efficiently find the closest c to $y_{|\Gamma v}$ by brute force since code C_0 is a small sized code. Note that since this is a bipartite graph, therefore, no two v's in L share an edge and hence there are no conflicts when changing the value of an edge. Similarly, we can define *Right-decoding iteration.*

The algorithm for decoding is as follows:

> alternately perform left-decoding and right-decoding iterations for $A \log n$ iterations

for a large enough constant $A < \infty$.

2.2.1 Correctness of the decoding algorithm

Lemma 17 *Assume that $\lambda/d < \delta_0/3$. When the number of errors is at most $(1-\epsilon)\frac{\delta_0}{2}(\frac{\delta_0}{2} - \frac{\lambda}{d})nd$, the above algorithm converges to the correct codeword when run for $A(\epsilon) \log n$ iterations.*

PROOF: We use the terminology that an error is incident to a vertex if the value associated with at least one edge incident to the node is incorrect in comparison to the correct codeword. Let

$$S_1 = \{v \in L_H | \exists \text{ an error incident to } v \text{ after the first left side decoding step}\} .$$

After the first left-decoding step a node v in L_H has an error incident onto itself only if it had more $\delta_0 d/2$ error edges incident onto itself before the decoding step. This is because any v which had less than $\delta_0 d/2$ incident error edges would, after the decoding step, have corrected all the incident errors. Let E_0 denote the number of errors in y before the decoding step.

Since each node in S_1 had more than $\delta_0 d/2$ errors incident on itself, therefore $E_0 \geq |S_1|\frac{\delta_0 d}{2}$. However, from assumption $E_0 \leq (1-\epsilon)\frac{\delta_0}{2}(\frac{\delta_0}{2} - \frac{\lambda}{d})nd$ and therefore, $|S_1| \leq n(\frac{\delta_0}{2} - \frac{\lambda}{d})(1-\epsilon)$.

Now we run the right-side decoding over R_H.

Let $T_1 = \{v \in R_H | \exists \text{ an error incident to } v \text{ after the first right side decoding step}\}$.

We want to show that $|T_1| \leq \alpha |S_1|$ for some $\alpha < 1$.

Using the same reasoning as above, after the right-decoding step, a node in R_H would have error incident onto itself only if it had more than $\delta_0 d/2$ errors incident onto itself before the right-decoding step. Now, $|E(S_1, T_1)|$ is clearly an upper bound on the number of error edges incident onto the nodes in T_1 before the right-decoding step. Since each node in T_1 had at least $\delta_0 d/2$ errors incident onto itself before the right decoding step, therefore, $\frac{\delta_0 d}{2}|T_1| \leq |E(S_1, T_1)|$.

However, by Expander Mixing Lemma, we can upper bound $|E(S_1, T_1|$ by $\frac{d|S_1||T_1|}{n} + \lambda\sqrt{|S_1||T_1|}$. On using the fact that $|S_1| \leq n(\frac{\delta_0}{2} - \frac{\lambda}{d})(1-\epsilon)$ and AM-GM inequality, $\frac{|S_1|+|T_1|}{2} \geq \sqrt{|S_1||T_1|}$, we can change the upper bound to

$$|E(S_1, T_1)| \leq |T_1|(\frac{d\delta_0}{2} - \lambda)(1-\epsilon) + \lambda\frac{|S_1| + |T_1|}{2} .$$

Together with $\frac{\delta_0 d}{2}|T_1| \leq |E(S_1, T_1)|$, we get

$$\frac{\delta_0 d}{2}|T_1| \leq |T_1|(\frac{d\delta_0}{2} - \lambda)(1-\epsilon) + \lambda\frac{|S_1| + |T_1|}{2}$$

which upon rearranging yields

$$|T_1| \leq \frac{\lambda}{\epsilon\delta_0 d + (1-2\epsilon)\lambda}|S_1| \leq \frac{|S_1|}{1+\epsilon}$$

provided $\delta_0 d > 3\lambda$.

Hence, in each iteration set of nodes on which the error-edges are incident decreases geometrically by a factor of $(1+\epsilon)$. Therefore in $O_\epsilon(\log n)$ rounds, no error edges remain. □

2.2.2 Running Time

The above decoding algorithm can clearly be implemented in $O(n \log n)$ time since each iteration takes $O(n)$ time and the number of iteration by the Lemma 17 above is $O(log(n))$. We now show a

linear time implementation of the decoding algorithm.The key observation is that in each iteration, the only vertices (in the relevant side for that iteration) that need to be locally decoded are those that are adjacent to some vertex of the opposite side which had some incident edges flipped in the local decoding of the previous iteration. The latter set shrinks geometrically in size by Lemma 17. Let us be somewhat more specific. After the first (left) iteration, for each $v \in L$, the local subvector $y_{|\Gamma(v)}$ of the current vector $y \in \{0,1\}^E$ belongs to the code C_0. Let $T(y) \subset R$ be the set of right hand side vertices u for which $y_{|\Gamma(u)}$ does not belong to C_0. Let $z \in \{0,1\}^E$ be the vector after the running the right side decoding on y. Note that for each $w \in L$ that is not a neighbor of any vertex in $T(y)$, its neighborhood is untouched by the decoding and hence the incident edges on such nodes still form a valid codeword in C_0. This means that in the next iteration (left side decoding), all these vertices w need not be examined at all.

The algorithmic trick therefore is to keep track of the vertices whose local neighborhoods do not belong to C_0 in each round of decoding. In each iteration, we only perform local decoding at a subset of nodes D_i that was computed in the previous iteration. (This subset of left nodes gets initialized after the first two rounds of decoding as discussed above.) After performing local decoding at the nodes in D_i, we prepare the stage for the next round by computing D_{i+1} for the opposite side as the set of neighbors of all nodes in D_i some of whose incident edges were flipped during the current iteration. Using an argument similar to Lemma 17 one can show that the D_i's are geometrically decreasing in size. This implies that the total number of nodes whose local neighborhoods have to be examined over all iterations is $O_\epsilon(n)$. As each local decoding step can be performed in $O_d(1)$ time, overall we get a linear time implementation of the decoding algorithm.

3 Distance Amplification of Codes

The expander codes constructed so far have rather small relative distance. We now see a way to boost the distance of a code by combining it with certain expander-like graphs. The construction is originally due to Alon, Bruck, Naor, Naor, and Roth [1].

Let $G = (L, R, E)$ be a bipartite graph with $L = \{1, 2, \ldots, n\}$ and $R = \{1, 2, \ldots, m\}$ which is D-left-regular and d-right-regular. Let C be a binary linear code of block length $n = |L|$, so that bits of codewords of C can be "placed" on the left vertices of G.

We define the distance amplified code $G(C) \subset \Sigma^m$ where $\Sigma = \{0,1\}^d$ as follows. Note that the alphabet size of the code $G(C)$ is 2^d.

Definition 18 *For $c \in \{0,1\}^n$, define $G(c) \in (\{0,1\}^d)^m$ by*

$$G(c)_j = (c_{\Gamma_1(j)}, c_{\Gamma_2(j)}, \cdots, c_{\Gamma_d(j)})$$

for $j = 1, 2, \ldots, m$ where $\Gamma_i(j) \in L$ denotes the i'th neighbor of $j \in R$. Now define the code $G(C)$ as

$$G(C) = \{G(c) \mid c \in C\} .$$

Since each bit of a codeword $c \in C$ is repeated D times in the associated codeword $G(c) \in G(C)$, we get the following.

Lemma 19 *Rate of Code $G(C)$ is $1/D$ times the rate of C.*

Lemma 19 follows from the definition of distance amplification code $G(C)$.

Definition 20 *A bipartite graph $G = (L, R, E)$ is said to be (γ, β) disperser if $\forall S \subseteq L_G$ with $|S| \geq \gamma n$, $|N(S)| \geq \beta m$.*

The lemma below follows from the definition of a (γ, β) disperser.

Lemma 21 *If G is a (γ, β) disperser and $\Delta(C) \geq \gamma n$, then $\Delta(G(C)) \geq \beta m$.*

References

[1] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992. 11

[2] M. R. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 659–668, 2002. 2

[3] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. 9

[4] G. Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001. 9