COMPSCI 650 Applied Information Theory April 12, 2016

Lecture 20

Instructor: Arya Mazumdar *Scribe:* [redacted]

1 Recap

In bipartite graph-based linear codes, the bipartite graph (Tanner graph) needs to be sparse to be solved easily. Such codes are also known as "Low Density Parity Check Codes" (LDPC codes), which are now part of the CDMA standard. If the graph has no cycles, then we can assume local independence. If there are no short cycles in the graph, then local independence is approximate.

Suppose we have a regular tree graph with n nodes, each with degree d. Then the number of nodes at a given level l is d^l. Since $d^l \leq n \Rightarrow l \leq \frac{\log\ n}{\log\ d}$. This implies that in a non-tree graph, there will always exist a cycle of length $\mathcal{O}(\log\ n)$.

Now we want to show that there exist bipartite graph codes that can correct $\mathcal{O}(n)$ errors (or αn errors, $\alpha \in (0,1)$). Below is a timeline of related methods being proposed:

- Belief propagation to find marginals, 1986, Pearl
- In the context of codes, BP appeared in 1998, MacKay & Neal
- Expander graph, 1971, Pinsker
- Expander codes, 1996, Sipser & Spielman

2 Expander Graph

Consider the graph shown in Figure 1. There are n vertices on the left and $n-k$ vertices on the right. The vertices on the left, V_1, have degree d_v each, and the vertices on the right, V_2, have degree d_c each.

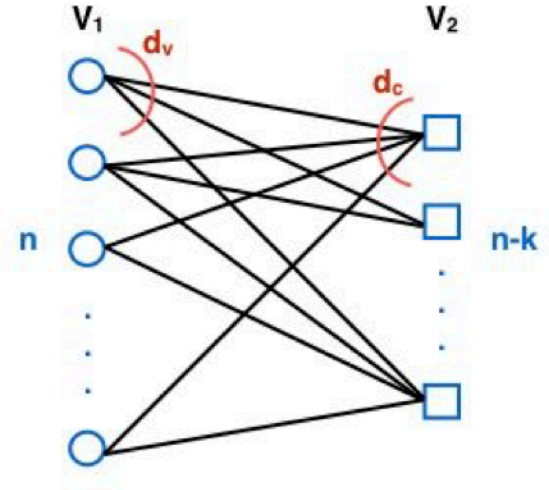

Figure 1: Bipartite graph.

Definition: A bipartite graph (*e.g.* as in Figure 1) is an (α, γ)-expander graph if

$$N(S) > \gamma d_v |S| \ \forall\ S:\ |S| \leq \alpha n$$

Note that the maximum number of vertices that the subset S can touch is $d_v S$, while the minimum number of vertices that S can touch is $\frac{d_v}{d_c}|S|$. Hence, we have the following:

$$d_v|S| \geq N(S) \geq \frac{d_v}{d_c}|S|$$

Theorem: (Sipser & Spielman) For a bipartite $(\alpha, \frac{3}{4})$-expander with n vertices on the left (with degrees d_v), $n\frac{d_v}{d_c}$ vertices on the right (with degrees d_c), there exists a simple, iterative, linear-time decoding that corrects $\frac{\alpha n}{2}$ errors. Note that the number of errors that can be corrected is linear in the size of the codeword.

Proof: Suppose we transmit a codeword and some of the bits get flipped. Hence, some of the parity check constraints will be unsatisfied. Let S = set of bits that are flipped. We want to find a vertex $v \in V_1$ which has more unsatisfied checks than satisfied checks (we can always find such a vertex through iterative decoding). Once we find that vertex v, we simply flip it, causing the number of unsatisfied constraints to reduce. See Table 1 for the notation that will be used in the rest of the proof.

Name	Description
r	Number of corrupted vertices on the left
t	Number of unsatisfied checks in the neighborhood
s	Number of satisfied checks in the neighborhood
$S \in V_1$	The set of corrupted vertices

Table 1: Notation

Note that an $(r, t)-$tuple can be viewed as the 'state' of the decoding algorithm at each iteration.

Suppose that $r \leq \alpha n$. Then

$$\begin{aligned} |N(S)| &> \frac{3}{4}d_v|S| = \frac{3}{4}d_v r \\ |N(S)| &= t + s > \frac{3}{4}d_v r \end{aligned} \tag{1}$$

Note that because of how parity check constraints work, s must contribute at least two edges per vertex in s, while each vertex in t can contribute just one edge. Hence, we also have the following constraint:

$$2s + t \leq d_v r \tag{2}$$

Combining (1) and (2), we have:

$$\begin{aligned} d_v r &> t + 2\left(\frac{3}{4}d_v r - t\right) \\ \Rightarrow d_v r &> \frac{3}{2}d_v r - t \\ \Rightarrow t &> \frac{d_v r}{2} \qquad (3) \\ \Rightarrow \frac{t}{r} &> \frac{d_v}{2} \qquad (4) \end{aligned}$$

Therefore, there must exist a vertex v that has more unsatisfied checks than satisfied checks.

At the 0th iteration of the algorithm, we have $r^{(0)} \leq \frac{\alpha n}{2}$. We need to show that $r^{(i)} \leq \alpha n$ (this is the condition under which $t \to 0$). Suppose $r^{(i)} = \alpha n$. Then $t^{(i)} > \frac{d_v}{2}\alpha n$ by Eq. 3. But

$$t^{(0)} \leq d_v r^{(0)} \leq \frac{d_v}{2}\alpha n$$
$$\Rightarrow t^{(i)} > \frac{d_v}{2}\alpha n \geq t^{(0)}$$

This is a contradiction.

3 Network Coding

In this section, we show two examples of network coding:

Example 1: the Butterfly Network. The network is shown in Figure 2. Assume the following for this network:

- it is corruption free
- there is no delay
- each edge has unit capacity
- the network sends one packet every second

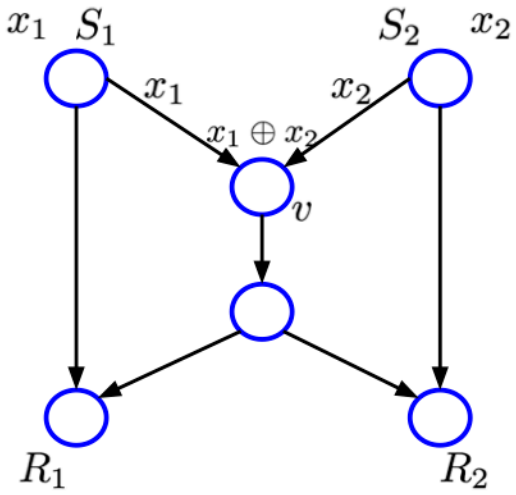

Figure 2: Butterfly Network

As shown in the figure, there are two sources (S_1 and S_2) and two receivers (R_1 and R_2). The sources would like to send the messages x_1 and x_2 to both of the receivers. But in order to send x_1 to R_2 and x_2 to R_1, the packets must form a queue at node v. Instead, we can achieve a better utilization of the network through network coding using the following steps:

1. S_1 sends x_1 to R_2
2. S_2 sends x_2 to R_1

3. S_1, S_2 send x_1, x_2 to v, which XORs the two messages and broadcasts them to R_1 and R_2. Each receiver can then recover the message they do not yet have.

Example 2: Consider the network shown in Figure 3. Two nodes are connected to a broadcasting node, B. The first node (on the left in the figure) has a message x_1, which it would like to send to the second node (on the right in the figure). The second node has a message x_2, which it would like to send to the first node. One method of exchanging the information between the two nodes would involve B first

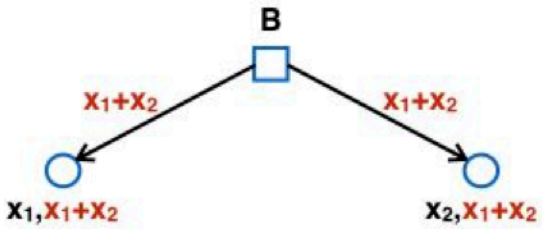

Figure 3: Network for exchanging files.

forwarding x_1 to the node on the right, and then forwarding x_2 to the node on the left. But a more efficient method would utilize network coding: the left and right node can forward x_1 and x_2 to B, respectively, and B can broadcast $x_1 \oplus x_2$ to both nodes. This method achieves the information swap in less time than the first method, and it also has an inherent property of security.

4 Max flow-Min cut

Consider the scenario depicted in Figure 4, showing a set S of sources connecting to a single receiver R through a network G.

Definition: A *cut* of a graph G is defined to be the set of edges that if removed from G, would disconnect S and R.

Definition: A *min cut* is a cut of G that has minimum size of all possible cuts. A min cut can be thought of us (a)the bottleneck in the network.

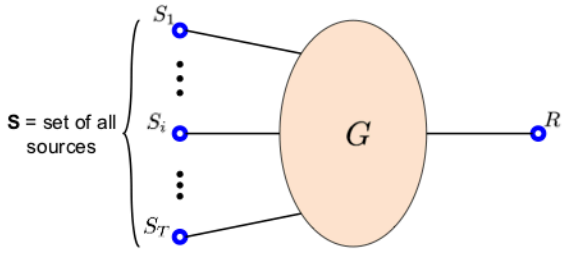

Figure 4: Example of a unicast network.

Suppose that each edge has unit capacity. Let |min cut| $= h$. This means that we cannot send more than h packets simultaneously through the network. In addition, if the minimum cut has size h, then

there exist h edge-disjoint paths from the sources to the destination (if not, then one could find a cut smaller than h, which would be a contradiction).

Example: Multicast. Consider the multicast network shown in Figure 5.

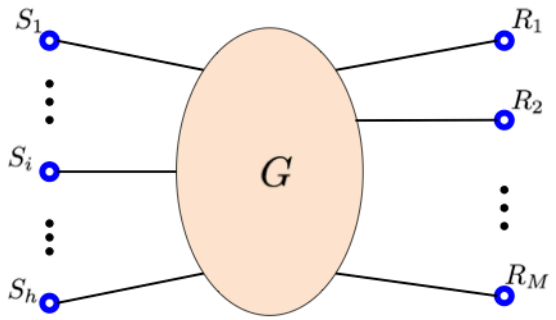

Figure 5: Example of a multicast network.

Suppose $\text{MinCut}(\{S_1, \ldots, S_n\}, R_i) = h, \ \forall \ i \in \{1, M\}$. Then within one time slot, all h sources can send all h packets (1 per source) to all M destinations using network coding.