6.1 Recap From Last Class

6.1.1 Hamming Code
- Binary \((2^r - 1 - 1, 2^r - 1 - r, 3)\) codes.
- \((n, n - \log n, 3)\)

6.1.2 Hadamard Code
- Binary \((2^r - 1, r, 2^r - 1)\) codes.
- \((n, \log n, n/2)\)

6.1.3 Reed Solomon Code
- Binary \((n, k, n - k + 1)\) codes.
- Optimal, but large alphabet.

6.1.4 Concatenation of Codes
- Combining two codes, to gain the benefit of both.
- \((N, K, D)_q^k\) and \((n, k, d)_q\) codes \(\Rightarrow (Nn, Kk, Dd)_q\) code.

6.2 Overview

This lecture introduces another set of codes, optimized for fast (de)coding, based on graphical constructions.
6.3 Graph Theory

6.3.1 \((\alpha, \beta)\) Expander Graphs

- Expansion: every small subset of vertices \((k \leq \alpha n)\) has many \((\geq \beta k)\) neighbors.
- Low Degree.

6.3.2 Bipartite \((\alpha, \beta)\) Expander Graphs

- Bipartite: a graph \(G\) is bipartite if the vertices of \(G\) can be separated into two disjoint sets such that there is edge connecting two vertices in the same set.
- It’s an undirected graph.
- Expansion: every small subset of vertices \((k \leq \alpha n)\) on the left has many \((\geq \beta k)\) neighbors on the right.
- Typically assumed to be low degree, meaning that each vertex has a small number of neighbors.
- The purpose of such graph is to have some sparse (low degree) graph with the \((\alpha, \beta)\) expander property.
6.3.3 \textit{d}-regular graphs

\begin{itemize}
\item An undirected graph is \textit{d}-regular if every vertex has \textit{d} neighbors.
\item A bipartite graph is \textit{d-left-regular} if every vertex on the left has \textit{d} neighbors on the right.
\end{itemize}

6.3.4 Expander Graph Construction

\textbf{Theorem 6.1.} For every constant $0 < c < 1$, can construct bipartite graphs with

\begin{itemize}
\item $n$ nodes on left,
\item $cn$ nodes on right,
\item \textit{d}-regular (left),
\end{itemize}

that are $(\alpha, 3d/4)$ expanders, for constants $\alpha$ and $d$ that are functions of $c$ alone.

In other words, any set containing at most $\alpha$ fraction of the left has $(3d/4)$ times as many neighbors on the right.

The proof of this theorem is not covered in class because of its complexity.

6.4 Low Density Parity Check (LDPC) Codes

In order to construct a $(n, k)$ LPDC code with binary alphabet, we first construct an undirected bipartite $(\alpha, 3d/4)$ expander graph with $n$ nodes on the left, $(n - k)$ nodes on the right, and all nodes on the left are \textit{d}-regular. According to \textbf{Theorem 6.1}, we can always construct such graph.

Next, we construct the parity check matrix $H$ from the graph in the following way:

\begin{itemize}
\item Each row is a vertex on the right.
\end{itemize}
• Each column is a vertex on the left.
• A codeword on the left is valid if each right “parity check” vertex has parity 0.
• The graph has $O(n)$ edges (low density).

Thus, we can obtain a valid parity check matrix $H$ for the code, which means the code is a linear code, and we can construct the corresponding generator matrix $G$ from $H$. A great advantage of such LDPC code is its low-density, meaning that the number of code bits associated with a single parity check bit is small, which allow for fast encoding and decoding.

### 6.4.1 Distance of LDPC codes

**Theorem 6.2.** Distance of LDPC code (codeword of length $n$ and constructed from bipartite $(\alpha, 3d/4)$ expander graph with $n$ vertices on the left and these $n$ vertices are $d$-regular) is greater than $\alpha n$.

**Proof:** Since LDPC is linear code, the distance equals the minimum weight of non-zero codeword.

Assume for the sake of contradiction, there exists a codeword with weight $\leq \alpha n$. The codeword should have all its parity check bits on the right equal to zero.

Let $W$ be the set of 1 bits in the codeword.

Thus, the number of edges from $W$ on the left equals $|W|d$, because the vertices on the left are $d$-regular.
By the \((\alpha, 3d/4)\) expander property, the number of neighbors on the right is at least 
\(\frac{3}{4}|W|d\), so at least one of these neighbors sees a single 1-bit on the left, because otherwise 
there are at least \(2 \cdot \frac{3}{4}|W|d = \frac{3}{2}|W|d\) edges between \(W\) and its neighbors, but the number of 
these edges is \(|W|d\).

Thus, since at least one of these neighbors sees a single 1-bit on the left, it contradicts the 
fact that a code word should have all its parity bits on the right equal to zero. Contradiction! 
\(\square\)

### 6.4.2 Correcting Errors in LPDC codes

We want a fast (linear time) error-correcting algorithm for LPDC codes. The convergence 
to the closest codeword property will be discussed in the next lecture.

**Algorithm**

We say a vertex (on the right) is unsatisfied if parity \(\neq 0\) (i.e. the xor of its neighbors on 
the left equals 1).

While there are unsatisfied check bits :

- Find a bit on the left for which more than \(d/2\) of its neighbors are unsatisfied.
- Flip that bit.

This algorithm converges because the number of unsatisfied check bits reduces by at least 
1 every step. This algorithm runs in linear time because the number of step is bounded by 
the length of codeword due to convergence, and each step the running time is constant due 
to constant maximum degree on the right.

**Theorem 6.3.** There always exists a node (on the left) with more than \(d/2\) unsatisfied 
neighbors if we’re not at a codeword (i.e. there exists unsatisfied check bit on the right).

**Proof:** Let \(S\) be the set of corrupted bits on the left. Assume for the sake of contradiction, 
each node in \(S\) has majority of satisfied neighbors (i.e. more that \(d/2\) of its neighbors are 
satisfied). Let \(N(S)\) denote the set of the neighbors of \(S\).

Consider that each corrupted bit sends 1 dollar to each of its unsatisfied neighbors, and 
0.5 dollar to each of its satisfied neighbors. Assuming \(d\) is odd, the total money sent by \(S\) is 
less than \(|S| \cdot (\frac{d}{2} + 0.5 \cdot \frac{d}{2}) = \frac{3}{4}d|S|\), because the majority of its neighbors are satisfied.

Note that each of \(N(S)\) receives at least 1 dollar, because if it’s unsatisfied, then it 
receives at least 1 dollar by our construction. If it’s satisfied, then it has to be connected to 
at least two corrupted bits, and so receives at least \(2 \cdot 0.5 = 1\) dollar.

Thus, the total money collected on the right is at least \(|N(S)|\).

Since the total money sent by the left is less than \(\frac{3}{4}d|S|\), we have \(|N(S)| < \frac{3}{4}d|S|\), which 
contradicts the property of our \((\alpha, 3d/4)\) Expander Graph, where \(|N(S)| \geq \frac{3}{4}d|S|\) for every 
\(|S| \leq \alpha n\). 
\(\square\)