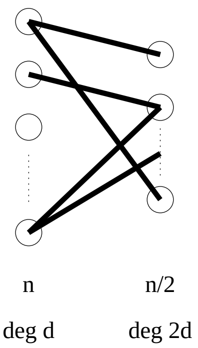
18.419 Applied Extremal Combinatorics	**Fall 1998**
Lecture 15:November 5, 1998	
Lecturer: Daniel Spielman	*Scribe: Arvind Sankar*

In the last class we considered a bipartite expander graph with $(n, n/2)$ nodes. We showed that if the expander is a good expander, we can construct a good error-correcting code from it. The decoding alogorithm was linear time. However, that construction had the disadvantage that encoding is not easy. If we form a generator matrix for the code, then we find that the matrix is dense, and hence the code would take quadratic time to encode.

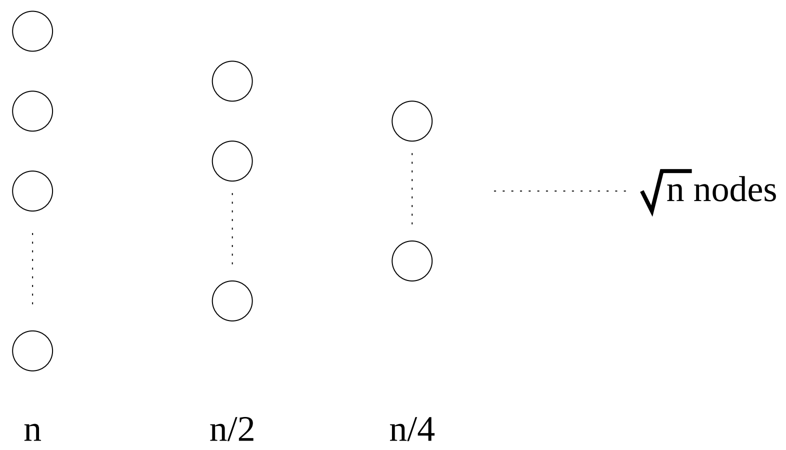
We want a code that can be encoded in linear time and will correct a number of errors proportional to n with decoding also in linear time. Of course, the only linear time algorithms are very simple ones, so we cannot expect the code to be as good.

Consider the same bipartite structure as before, but with the nodes on the right now interpreted as parity gates, that is, they output the sum of their neighbours mod 2. The encoded output consists of the original message together with the outputs of these parity gates. So the encoding algorithm is trivial. We get a code of rate $2/3$ and length $3n/2$. This is not a very good code, however. The minimum distance, for example, is only $d+1$. This kind of code will be used in a recursive construction to get better codes.



As far as decoding goes, if we know the errors in the check bits, then correcting the message bits is the same as with the code considered in the previous lecture. So our idea will be to recursively encode the check bits with the same code, but of half the length, and continue till we come down to short lengths, when we can use a better code with a more complex decoding algorithm.

So the code looks like this:



The final stage consists of $\sqrt{n}$ bits, which can be encoded using any good code. The complexity of encoding/decoding this last code is $O((\sqrt{n})^2) = O(n)$. The big code is decoded by working backward from the last stage.

If we assume that α fraction of the bits are in error, then on average, we expect that about α fraction of the bits in each column will be in error. Then we can show that our decoding algorithm works. Experimentally, we find that the same fraction of errors as in the previous code can be tolerated, but the probability of being able to correct is now smaller. The probability of incorrect decoding now goes down as an exponential in $\sqrt{n}$ rather than in n. We will now show that the decoding algorithm is somewhat robust in the sense that even if some check bits are in error, the number of errors in the message bits which are not corrected will not be too large.

Proposition *Assume that the bipartite graph has the property that for all subsets A of the nodes on the left,*

$$|A| \leq \alpha n \Longrightarrow |N(A)| > \left(\frac{3}{4}d + 2\right)|A|$$

Then if $v \leq \alpha n/2$ message bits are incorrect and $b \leq \alpha n/2$ check bits are incorrect, then after we run the decoding algorithm at most $b/2$ message bits will be in error.

Proof. We call a check bit *satisfied* if it is equal to the parity of its neighbouring message bits. Then the decoding algorithm consists of iteratively finding a message bit such that most of its neighbours are unsatisfied and flipping it, till there are no such message bits left.

So the first thing is to show that we can find such message bits, if there aren't too many errors. Let u denote the number of unsatisfied check bits, and s the number of satisfied check bits that are neighbours of corrupt message bits.

If $v \leq \alpha n$, then we have that

$$u + s > \left(\frac{3}{4}d + 2\right) v$$

and

$$u + 2s \leq dv + b$$

Hence we have

$$u > \frac{dv}{2} + 4v - b$$

If we have additionally $v \geq b/2$, then we get

$$u > \frac{dv}{2} + b$$

Since only b check bits are in error, hence more than $dv/2$ check bits are unsatisfied and are neighbours of corrupt message bits. Hence we must have at least one corrupt message bit most of whose neighbouring checks are unsatisfied.

We have thus shown that the algorithm can run as long as

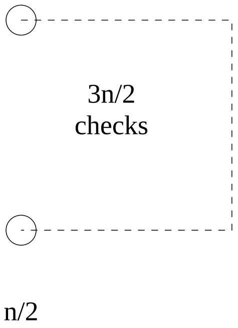
$$\frac{b}{2} \leq v \leq \alpha n$$

In addition, we need to prove that the decoding stops because v decreases below $b/2$ rather than increasing above αn. Note that at each time step, the number of unsatisfied checks decreases. Initially, we had at most $d\alpha n/2 + b$ unsatisfied checks. If we ever have $v = \alpha n$, then we would have

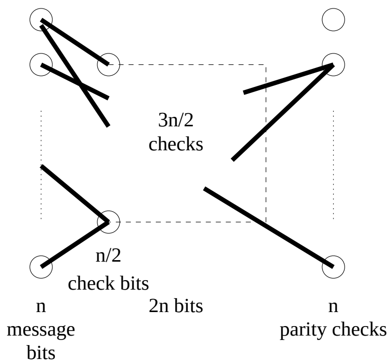
$$u > \frac{dv}{2} + 4v - b > \frac{d\alpha n}{2} + b$$

which is a contradiction.

Using this proposition, we can give a recursive construction of long codes with fast encoding/decoding algorithms. We start with a code like this:



Now put n message bits on the left, and make the $n/2$ nodes their check bits. Take the entire block of $2n$ bits in the middle and construct n parity checks for them and put them on the right:

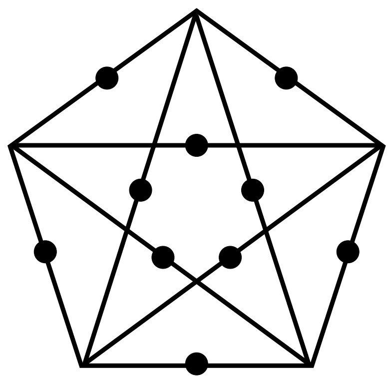


So we have a code for n message bits which is again of rate $1/4$. We run the decoding algorithm first on the $2n$ bits in the box and their check bits. Then invoke the algorithm recursively on the box, which is the same code but half the length. Then use the algorithm on the n message bits and their (now decoded) check bits.

If there were at most $\alpha n/2$ errors in the transmission, then there are at most $\alpha n/2$ errors in the parity checks on the right, and at most $\alpha n/2$ errors in the box, hence by the proposition, we will be left with at most $\alpha n/4$ errors in the box after decoding. Hence we can assume by induction that our recursive decoding algorithm will work on the box, thus correcting all the errors in the original message's check bits. So the final decoding step will work and correct all the errors in the message (since again there are at most $\alpha n/2$ errors there, and now all the check bits are known correctly).

Thus we have obtained long codes of rate $1/4$ which can correct a constant fraction of errors, with linear encoding and decoding algorithms. The trouble is that the constant fraction is very small for explicit expanders. For random graphs, we can get a better constant, but still we cannot prove anything better than approximately $\alpha = 0.01$.

We now generalize our parity check codes as follows. We still have n nodes on the left, but on the right, instead of parity check nodes, we put more general constraint nodes. Each constraint node says that my neighbours form a codeword in such-and-such an error-correcting code. We assume each node on the left has degree two, and each node on the right has degree d. Then we can think of a message bit as an edge between two constraint nodes, and draw the graph as:



In this graph, every vertex is a constraint, and every edge is a message bit. The graph is d-regular, and will be chosen to be an expander. As an example, consider the K_5 shown above, with each constraint being simply a parity check. There are 5 constraints and the code length is 10. Every cycle in this graph represents a codeword, and thus the minimum distance is 3. It turns out that one of the constraints is actually redundant, so the code has dimension 6.

We will show in the next class that if we have degree d nodes, each imposing a constraint corressponding to a code of rate r and relative minimum distance δ, and the second adjacency matrix eigenvalue of the graph satisfies $\lambda_2 \leq \lambda$, then we get a code with rate at least $2r - 1$, and relative minimum distance at least[1]

$$\left(\frac{\delta - \lambda/d}{1 - \lambda/d}\right)^2$$

[1] I think we actually ended up proving something like $\delta(\delta - \lambda/d)/(1 - \lambda/d)$.