CSC 5160 Spectral Algorithms

Week 9: Expander Code

We will see one application of expander graphs in detail, in designing asymptotically good error-correcting codes that are efficiently encodable and decodable. First we will introduce the basics in coding theory, including random linear code. Then we show the expander codes, and finally we present the superconcentrator codes that allow encoding in linear time.

Error Correcting Codes

A transmitter wants to send a message to a receiver through a communication channel. The channel may have errors, and some of the message bits may be flipped. Our goal is to come up with a transmission scheme so that the receiver can always recover the transmitter’s message.

Let’s see one example before the formal setup. Let’s say that only one bit may be flipped. Then an easy way is to transmit every bit three times. So, if the message is of 4 bits, then we need to transmit 12 bits.

Here is a clever way to transmit a 4-bit message using only 7 bits, in such a way that any error can be corrected.

Let $b_3, b_5, b_6, b_7$ be the bits that the transmitter would like to send (message bits).

We send three parity-check bits $b_1, b_2, b_4$ by the following rules:

- $b_1 = b_3 + b_5 + b_7$
- $b_2 = b_3 + b_6 + b_7$
- $b_4 = b_3 + b_5 + b_6$

Where all operations are modulo 2.

We can write down the equations that the codeword satisfies:

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
 b_1 \\
 b_2 \\
 b_3
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
0
\end{pmatrix}.
\]

The transmitter will send the “codeword” $b_1, b_2, \ldots, b_7$, e.g. if $b_3 = 1, b_5 = 0, b_4 = 1, b_7 = 0$, then $b_1 = 1, b_2 = 0, b_4 = 1$.

Let $c_3, c_5, \ldots, c_7$ be the bits that the receiver receives.

Suppose the 6th bit is flipped, from $b_6 = 1$ to $c_6 = 0$.

You can check that:

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
 c_3 \\
 c_5 \\
 c_3
\end{pmatrix}
= \begin{pmatrix}
1 \\
0 \\
0
\end{pmatrix}.
\]

This tells you there is an error, and furthermore the error is in the 6th bit (100 is the binary of 6).

It is also true for any other 4-bit message and any other error location.

This is called the Hamming Code.
Setting

We view an error correcting code as a mapping \( C : \{0,1\}^m \rightarrow \{0,1\}^n \) where \( m < n \).

Every string in the image of \( C \) is called a codeword.

The rate of the code is defined as \( R = \frac{m}{n} \). The higher the rate the more efficient is the code.

The Hamming distance between two words \( c' \) and \( c'' \) is the number of bits that they differ, written as \( \text{dist}(c', c'') \).

The minimum distance of a code is \( d = \min_{c', c'' \in C} \text{dist}(c', c'') \). The larger is the minimum distance the more errors that the code can correct. In particular, if the number of errors is less than \( \frac{d}{2} \), then there will be a unique codeword that is closest to the corrupted codeword.

The minimum relative distance of a code is \( \delta = \frac{d}{n} \).

If there is a constant fraction of errors, then the encoding length has to be increased by a constant.

It turns out that there are codes that the rate and the minimum relative distance are bounded below by absolute constant, even as \( n \) grows.

A sequence of codes \( C_1, C_2, \ldots \) of increasing message length is called asymptotically good if there are absolute constants \( r \) and \( \delta \) such that for all \( i \), \( r(C_i) \geq r \) and \( \delta(C_i) \geq \delta \).

Linear Code

A simple class of codes is the class of linear codes.

There are two different ways to define a linear code. We restrict to arithmetic modulo 2.

One way is to define by a generator matrix. A generator matrix \( G \) is an \( m \times n \) matrix that maps any \( m \)-bit string into an \( n \)-bit string. So it maps a message string into a codeword.

Another way is to define a code by a parity-check matrix. Let \( H \) be an \( (n-m) \times n \) matrix and the set of codewords is defined as \( C = \{ c : Hc = 0 \} \). So each row of \( H \) is a parity check constraint of the codewords.

We can construct the parity check matrix from the generator matrix and vice versa.

Let \( G = \begin{bmatrix} I_m \\ P \end{bmatrix} \) be in the standard form. This can be assumed as it generates the same set of codewords and they are differred by a linear transformation.

Let \( H = [ \begin{bmatrix} -P \mid I_{n-m} \end{bmatrix} \) be the parity check matrix.

Then \( HG = -P + P = 0 \). So all the codewords of \( G \) will satisfy all the constraints of \( H \), and these
are the only codewords that satisfy all the constraints because \( H \) is of rank \( n-m \) and its null space is of dimension \( m \).

---

**Random Linear Code**

Let \( M \) be an \( n \times m \) generator matrix where each entry is an independent uniformly random bit.

This is a linear code with rate \( m/n \).

To analyze the minimum distance of the code, let \( C^1 = Mb^1 \) and \( C^2 = Mb^2 \), observe that \( \text{dist}(C^1, C^2) = \text{dist}(0, C^1 - C^2) \), as \( C^1 - C^2 = M(b^1 - b^2) \), and so the minimum distance of \( C \) is equal to \( \min_{\|b\|_1 < m} \|Mb\|_1 \) where \( \|b\|_1 \) denotes the number of 1s in \( b \).

**Lemma** Let \( M \) be a random \( n \times m \) matrix. For any \( d \), the probability that \( C_M \) has minimum distance at least \( d \) is at least \( 1 - \frac{2m H(p)}{n} \binom{n}{d} \).

**Proof** Let \( b \) be an mbit string. We analyze the probability that \( \|Mb\|_1 \leq d \).

Each entry of \( Mb \) is the inner product of a row of \( M \) with \( b \).

Since each row of \( M \) consists of random \( \{0,1\} \) entries, each entry of \( Mb \) is chosen uniformly from \( \{0,1\} \).

Since each row of \( M \) is independent, \( Mb \) is a uniform random vector in \( \{0,1\}^n \).

So, the probability that \( \|Mb\|_1 \leq d \) is equal to \( \frac{\binom{n}{d}}{2^n} \).

Therefore, by the union bound, \( \Pr_b \left[ \exists b \in \{0,1\}^m, \|Mb\|_1 \leq d \right] \leq \frac{2m H(p)}{n} \binom{n}{d} \).

Let's work out the rate that this random linear code achieves.

First we use the fact that \( \binom{n}{d} \approx 2^{-n H(p)} \) where \( H(p) = -p \log p - (1-p) \log (1-p) \) is the binary entropy function.

To see this, note that \( \binom{n}{d} p^d (1-p)^{n-d} \leq \frac{1}{\sqrt{n}} \binom{n}{\frac{n}{2}} p^d (1-p)^{n-d} = (p/2)^d (1-p/2)^{n-d} \leq 1 \) and so

\[ \binom{n}{d} \leq (p/2)^d (1-p/2)^{n-d} = e^{-n H(p)}. \]

On the other hand, argue that \( \binom{n}{d} p^d (1-p)^{n-d} \) is the largest term in the binomial expansion and so \( \binom{n}{d} p^n (1-p)^n \geq \frac{1}{m!} \) and thus \( \binom{n}{d} \geq 2^{-n H(p)/m!} \).

We use the fact that \( 2^{-n H(p)} \approx \frac{1}{\sqrt{n}} \binom{n}{\frac{n}{2}} \).

When we set \( m=n \) and \( d=\delta n \), then the above lemma says that the code \( C_M \) has rate \( 1 \) and minimum relative distance \( \delta \) with positive probability when \( 2^{\delta n} - 2^{-n H(p)} < 1 \), which happens when \( H(p) < 1 \).

This is called the Gilbert-Varshamov bound. It is still not known if binary code exists whose relative
minimum distance satisfies $H(s) > 1 - r$.

So, a random linear code is an asymptotically good code, but the problem is that there is no known efficient decoding algorithm.

---

**Expander Codes**

Gallager proposed to use low-density parity-check codes for error correcting codes.

Sipser and Spielman [2] and Spielman [3] show how to analyze these codes using expansion, and also give very efficient encoding and decoding algorithms.

Let's say we are constructing codes of rate $1/4$.

Let $G$ be a bipartite graph with $n$ vertices on the left and $3n/4$ vertices on the right.

Every vertex on the left has exactly $d$ neighbors, and $|N(S)| \geq \frac{5d}{8}|S|$ for every $S$ on the left with $|S| \leq \frac{n}{8}$, where $d$ is a constant.

Such a graph is known to exist for $d > 32$ by a probabilistic argument.

We show how to construct a good code from such a graph.

Let $H$ be an $(3n/4 \times n)$ parity-check matrix where each row is indexed by a vertex on the right and each column is indexed by a vertex on the left. The $i$-th entry of $H$ is one if there is an edge in $G$ from the $i$-th node on the right to the $j$-th node on the left. You can think of the nodes on the right are the parity check constraints and the nodes on left are the bits of the codewords.

The nullspace of $H$ is of dimension $\geq n/4$, and so the code has rate $\geq 1/4$.

We argue that the minimum distance is at least $n/od$. Suppose by way of contradiction, that there is a codeword $c$ with less than $n/od$ bits. Let this set of bits be $S$. Since $|S| < n/od$, its neighbor set $N(S)$ in $G$ satisfies $|N(S)| \geq \frac{5d}{8}|S|$. This implies that there must exist a vertex $v \in N(S)$ with only one neighbor in $S$, but then that constraint cannot be satisfied, a contradiction.

This shows that the rate and the minimum relative distance are constant.
Constructions

We state the constructions in graph theoretic terminology.

Let $G$ be a $(c,d)$-regular graph with $n$ vertices on the left, each of degree $c$, $(c/d)n$ vertices on the right, each of degree $d$.

Each vertex on the left corresponds to one bit of the codeword, called the variables.

Each vertex on the right corresponds to a constraint on the $d$ variables that it is connected to. In particular, a constraint will require that the variables it restricts form a codeword in some linear code of length $d$ (e.g. each constraint could be a random linear code on $d$ variables).

An $n$-bit string is a codeword if it satisfies all the constraints.

Theorem. Let $G$ be a $(c,d,a,\frac{c}{d})$-graph, i.e. $(c,d)$-regular and for each subset $S$ with $1 \leq |S| \leq n$
then $|\text{IN}(S)| \geq \frac{c}{d}|S|$. Let $P$ be a code of $d$ variables, rate $r > \frac{(c-1)c}{c}$, and minimum relative distance $\delta$. Then the resulting code has rate at least $cr-(c-1)$ and minimum relative distance at least $\frac{\delta}{2}$.

Proof. Each constraint imposes $(1-r)d$ linear constraints. So there are at most $\frac{cr}{d} - (1-r)d = o(n(1-r))$ linear constraints. So the subspace satisfying all the constraints is of dimension at least $n(\sqrt{r}-c+1)$, and thus the rate is at least $cr-c+1$.

To see the minimum distance property, let $w$ be a codeword with $\frac{c}{d}n$ non-zero bits $S$. There are $\frac{c}{d}|S|$ edges going out of $S$. By the expansion property, $|\text{IN}(S)| > \frac{c}{d}|S|$. Therefore, there must be a constraint $v \in \text{IN}(S)$ connected to $< d$ nodes in $S$, contradicting this satisfies the constraint at $v$, since it should have at least $d$ non-zeros for a codeword in $P$.

Decoding

The simplest example of expander codes is when the constraints are just parity check constraints.

The parity check code $P$ has rate $\frac{d-1}{d}$ and minimum relative distance $\frac{2}{d}$, and so the resulting expander code has rate $1 - \frac{2}{d}$.

To obtain a code that is efficiently decodable, we require a stronger expansion property, so that not only there exists a vertex $v \in \text{IN}(S)$ with only one neighbor in $S$, as in the proof of minimum distance, but to ensure that the average degree is small so that there are many such neighbors.
For this purpose, we require the expander graph to satisfy the property that every subset $S$ with at most $6n$ vertices has at least $\frac{3}{4}|S|$ neighbors.

There was no known explicit constructions of expander graphs with such a strong vertex expansion property. Now it is known how to construct such a "lossless" expanded using zig-zag product. Anyway, a random $(c,d)$ graph would have this property w.h.p., so if one is not worried about randomized construction there is nothing to worry about.

**Decoding Algorithm**

The algorithm is very simple. Recall that each constraint is just a parity check constraint.

- If there is a variable that is in more unsatisfied than satisfied constraints, then flip the value of that variable.
- Repeat until no such variable remains.

**Running Time**

After a linear time initialization to count the number of satisfied neighbors at each variable, each iteration can be implemented in constant time (since $c$ and $d$ are constants). There will be at most a linear number of rounds, since the number of unsatisfied constraints is strictly decreasing.

**Theorem** Let $B$ be a $(c,d,n,3c/4)$-expander. Then the simple algorithm will correct an $9/2$ fraction of errors.

**Proof** We say a variable is corrupted if its bit is differer from the bit in the codeword.

Let $u$ be the number of corrupted variables, denoted this set by $V$.

Let $u$ be the number of unsatisfied constraints.

Let $s$ be the number of satisfied constraints with a neighbor in $V$.

By the expansion property, we have $u + s > \frac{3}{4}cu$.

Each satisfied constraint with a neighbor in $V$ must have at least two edges to $V$, while each unsatisfied constraint must have at least one edge in $V$, and hence $cu > u + 2s$.

Combining these two inequalities, we obtain $u + s > \frac{3}{4}(u + 2s) \Rightarrow u + 2s \Rightarrow u > \frac{cs}{2}$. ($\star$)

So, out of the $c$ edges going out of $V$, more than $\frac{cs}{2}$ edges are connected to unsatisfied constraints. This implies that at least one variable has more unsatisfied neighbors than satisfied neighbors.

Therefore, as long as $u \leq an$, the algorithm will flip some variable (not necessarily correct).

In the case that the algorithm fails, it is a uniformly random vector (by Assumption 1).
there is only case that the algorithm fails is when \( v \) becomes greater than \( w \). Assume by contradiction that it happens. Consider the time when \( v = w \). Then (4) implies that \( w > C \alpha n^2 \). But this is a contradiction because initially \( w = C \alpha n^2 / 2 \) and \( w \) is strictly decreasing throughout the algorithm.

---

**Parallel Algorithm**

There is a parallel decoding algorithm that works in \( O(\log n) \) iterations.

- In parallel, flip each variable with more unsatisfied neighbors than satisfied neighbors.
- Repeat until no such variables exist.

**Theorem** Let \( B \) be a \((c,d,a,\frac{3+\delta}{4})\)-expander. The parallel algorithm can correct up to \( \alpha(1+\epsilon) / 2 \) fraction of errors in \( O(\log n) \) iterations.

**Proof Idea** Let \( F \) be the set of corrupted variables that the algorithm fail to flip in one decoding round. Let \( C \) be the set of uncorrupted variables that the algorithm (incorrectly) flip in one round. Then the set of corrupted variables in next round will be \( F \cup C \).

If we can show that \( |F \cup C| < (1-\delta)|V| \), then we are done.

To bound \( |F \cup C| \), just observe that each vertex in \( C \) has many common neighbors with the vertices in \( V \), and similarly each vertex in \( F \) shares at least \( c / 2 \) neighbors with other vertices in \( V \).

So, the vertices in \( F \cup C \) are not contributing to the expansion (at most \( 2p / 4 \)) and so there cannot be many of them.

---

**Explicit Construction**

The known deterministic constructions have weaker expansion properties. To deal with that we can use a stronger code as the constraints, e.g. the random linear code achieving the GV-bound.

The decoding algorithm is also slightly modified.

- For each constraint, if the variables in that constraint differ from a (local) codeword in at most \( d/4 \) places, then send a "flip" message to each variable that differs.
- In parallel, flip the value of each variable that receives at least one "flip" messages.

The analysis is very similar. using expander mixing lemma to establish that there are no dense subgraphs, while the errors create dense subgraphs.
Superconcentrator Codes

The above expander codes can be encoded in quadratic time, one just needs to construct a generator matrix from the parity check matrix, and then do a matrix-vector multiplication.

Spielman [3] showed how to modify the code construction to achieve linear time encoding, by introducing a concept called “error-reduction code” and a new recursive construction similar to that of superconcentrators.

**Error Reduction Code**: A code \( C \) of length \( n \) with \( r \) message bits and \( (1-r)n \) check bits is an error reduction code of rate \( r \), error reduction \( \varepsilon \), and reducible distance \( s \). If there is an algorithm that differs from a codeword \( w \in C \) in at most \( v \) message bits and \( t \) check bits with \( u \leq n \) and \( v \leq n \), will output a word that differs from \( w \) in at most \( s \) message bits.

**Recursive Construction**

Let \( C_0 \) be an error correcting code of \( n_0 \) variables, with rate \( 1/4 \), correcting a \( 1/4 \) fraction of errors.

Let \( R_k \) be a family of error correcting codes with \( n_0 \cdot 2^k \) message bits, \( n_0 \cdot 2^{k-1} \) check bits, reducible distance \( b^2 \), and error reduction \( 1/2 \), for \( k \geq 1 \).

We define the code \( C_k \) for \( k \geq 0 \) of \( n_0 \cdot 2^k \) variables and rate \( 1/4 \):

- \( C_k \) has \( n_0 \cdot 2^k \) message bits, call \( M_k \).
- We produce \( n_0 \cdot 2^k \) check bits using \( M_k \) and the code \( R_{k-1} \). Call these \( n_0 \cdot 2^k \) check bits \( A_k \).
- Produce \( 3n_0 \cdot 2^k \) check bits using \( A_k \) as message bits using the code \( C_{k-1} \). Call the new check bits \( B_k \).
- Produce \( n_0 \cdot 2^k \) check bits using \( A_k \) and \( B_k \) as message bits and using the code \( R_{k-1} \). Call the new check bits \( C_k \).

So, \( M_k \) are the \( n_0 \cdot 2^k \) message bits, \( A_k \), \( B_k \), \( C_k \) are the \( 3n_0 \cdot 2^k \) check bits. Thus the code is of rate \( 1/4 \).

**Encoding Time** Let \( C_1 n_0 \cdot 2^k \) be the time required to encode \( R_k \). Let \( C_2 n_0 \cdot 2^k \) be the time to do error reduction for \( R_k \). Let \( C_0 \) be the time to encode and decode \( C_0 \).

We prove by induction that \( C_k \) can be encoded in time \( 3C_1 n_0 \cdot 2^{k-1} + C_0 \), and decoded in time \( 3C_1 n_0 \cdot 2^{k-1} + C_0 \).
To encode \( C_k \), the time required is \( c_1 n_0 2^{k/2} + (3c_1 n_0 2^{k/2} + c_0) + c_1 n_0 2^{k/2} = 3c_1 n_0 2^{k/2} + c_0 \).

**Decoding**

**Theorem.** The codes \( C_k \) are error correcting codes of \( n_0 2^k \) variables with rate \( 1/4 \) from which a \( 3/4 \) fraction of error can be corrected, assuming \( R_k \) satisfies the following two properties:

1. On input a word that differs from a codeword in \( v \) message bits and \( t \) check bits, where \( v \leq 6n \) and \( t \leq 6n \), output a word that differs from that codeword in at most \( \max \{v, \frac{1}{2} t \} \) message bits.
2. On input a word that differs from a codeword in \( u \leq 6n \) message bits and no check bits, output that codeword.

To correct errors in \( C_k \), we work back to front.

Assume the algorithm is provided with a word which has at most \( 6n_0 2^{k/4} \) errors.

- First, use the \( n_0 2^{k/4} \) check bits in \( C_k \) to perform error reduction on the \( n_0 2^{k/4} \) bits in \( A_k \cup B_k \).

  Since there are at most \( 6n_0 2^{k/4} \) errors in \( C_k \) and \( A_k \cup B_k \), after the error reduction there are at most \( n_0 2^{k/4} \) errors in the bits \( A_k \cup B_k \).

- Apply the error correcting codes of \( C_{k-1} \) to correct all the errors in \( A_k \cup B_k \). Since there are at most \( n_0 2^{k/4} \) errors, \( C_{k-1} \) can correct all of them.

- Finally, use the error reduction code \( R_{k-2} \) with \( A_k \) as the check bits to correct the errors in \( N_k \).

  Since there are no errors in \( A_k \), we can correct all the errors in \( N_k \) by \( R_{k-2} \).

The decoding time is linear, using the same analysis as encoding time.

**Error Reduction Codes**

What is this mysterious error reduction code?

Surprisingly or unsurprisingly, it is just about the same as the expander codes.

Let \( G \) be a \((d, 2d)\)-regular graph, with \( n \) vertices on the left and \( n/2 \) vertices on the right.

Each check bit is just a sum of its neighboring message bits.

If there are no errors in the check bits, then this is just the same as decoding expander codes and all the errors can be removed. So we have the second property of the error reduction codes.

If there are some errors in the check bits, the idea is that the decoding algorithm can still be...
If there are some errors in the check bits, the idea is that the decoding algorithm can still be used to remove some errors in the message bits.

**Error Reduction Algorithm**

- If there is a message bit that has more unsatisfied than satisfied neighbors, then flip the value of that message bit.
- Repeat until no such message bits remains.

**Lemma.** Let \( G \) be a \( (d, 2d, \epsilon, \frac{3d}{4} + \epsilon) \)-expander graph. When given a word \( x \) that differs from a codeword \( \mathbf{w} \) in \( v \leq \alpha n/2 \) message bits and \( t \leq \alpha n/2 \) check bits, then the algorithm will output a word that differs from \( \mathbf{w} \) in at most \( t/2 \) of its message bits.

**Proof.** Let \( U \) be the set of corrupt message bits and \( T \) be the set of corrupt check bits.

Let \( u \) be the number of unsatisfied check bits and \( s \) be the number of satisfied check bits with neighbors in \( V \).

By the expansion property, \( u + s > (\frac{3d}{4} + \epsilon) v \).

Each unsatisfied check bit either has \( \epsilon \) one edge to \( V \) or it is corrupted.

Each satisfied check bit that has a neighbor in \( V \) either has \( \epsilon \) two edges to \( V \) or it is corrupted.

So, \( dv + t \geq u + s \).

Combining, we have \( s < (\frac{d}{2} - \epsilon) u + t \), implying that \( u > (\frac{d}{2} + \epsilon) v - t \). (**)

When \( \alpha n \geq \frac{3}{4} t/2 \), we have \( u > dv/2 + t \), and thus there must be a vertex with more unsatisfied neighbors, and so the algorithm will flip a variable when \( \alpha n \geq \frac{3}{4} t/2 \).

We show that the algorithm must terminate with \( u < t/2 \), as \( u \) cannot be larger than \( \alpha n \).

Initially, \( u \leq \alpha n/2 \) and therefore \( u + dv/2 \leq \alpha d n/2 + \alpha n/2 \).

If \( u > \alpha n \), then (***) implies that \( u > (\frac{d}{2} + \epsilon) \alpha n - t \geq \alpha d n/2 + 7\alpha n/3 \), a contradiction because \( u \) is strictly decreasing throughout the algorithm.

So the algorithm must terminate with \( u < t/2 \) errors in the message bits.

Again the algorithm can be made parallel, and the constraints can use more powerful code.

The ideas are the same, and we won't worry about the details, which can be reconstructed in a matter of time.
References

