Applications of Expander Graphs

The three applications of expander graphs which this will cover are:
1. Error Correcting Codes
2. Super Concentrators
3. Hash Functions

The first two applications are covered in the first chapter of Hoory, Linial, and Wigderson. While the last is covered in Cryptographic Hash Functions from Expander Graphs by Charles, Goren and Lauter.

Dylan has already covered most of the background which we will need with the exception to the idea of bipartite expander graphs called magical graphs.

Magical Graphs

For a set $S \subset G$, where G is a graph, let $\Gamma(S)$ denote the neighbours of S in G.

Given a bipartite graph $G = (L, R; E)$ we refer to L as the *left vertex set* and R as the *right vertex set*.

Definition: A bipartite graph $G = (L, R; E)$ is called an $(n, m; d)$-*magical graph* if $|L| = n$, $|R| = m$, every left vertex has d neighbours and for each subset $S \subset L$ the following properties hold:
a) $|\Gamma(S)| > \frac{5d}{8}|S|$ when $|S| \leq \frac{n}{10d}$ and
b) $|\Gamma(S)| > |S|$ when $|S| \leq \frac{n}{10d}$.

Remark 1: As it is a bipartite graph all the edges must lie between a left vertex and a right vertex.

Remark 2: As the left and right vertex sets can have different sizes we no longer require the graph to be d-regular, instead we require the property that every left vertex has d neighbours.

Remark 3: The two properties a) and b) give the 'good expander' properties of the graph.

The following lemma states the existence of magical graphs when the size

of the two sets is large enough. The proof, which we will not give, uses a probabilistic argument which shows that in fact most bipartite graphs are magical.

Lemma 1: There exists a constant n_0 such that for all $d \geq 32, n \geq n_0$ and $m \geq \frac{3n}{4}$ there exists an $(n, m; d)$-magical graph.

We will see shortly that the magical graphs will be useful in the first two applications. The first application is to error correcting codes.

Error Correcting Codes

A common technical issue that arises in data transmission, especially with data sent over long distances through the air where there may be interference, is that sometimes parts of the transmitted data are corrupted during the transmission.

We would like a method of encoding data so that if errors occur then most of the errors can be detected and corrected.

Linear Codes

The idea is to map blocks of n bits to blocks of more bits, say $n + k$ bits, in a way that the redundancies can be used to detect and correct errors.

To start with we will define a metric on the space we are working with.

Definition: Given two strings of $n + k$ bits c_1 and c_2 the *Hamming Distance* between c_1 and c_2, denoted $d_H(c_1, c_2)$, is the number of bits which are different.

For example, $\mathrm{d}_H(0101010, 0101011) = 1$ since they differ only in the last variable.

The method for sending and receiving encoded data using a linear code is as follows:

1. The sender first divides the message into blocks of n bits.
2. These n bits are then mapped to different blocks of $n + k$ bits called codewords using an injective 'encoding' function.

3. These codewords are then transmitted.
4. The receiver divides the message into $n + k$ bit blocks.
5. The receiver compares each block to a list of possible valid codewords.
6. If it the block is not a codeword the receiver replaces it by the closest codeword. (By closest we mean in terms of the hamming distance, so codeword with the fewest number of different digits. This closest entry is assumed to be the initial codeword. This is the step where the errors are detected and corrected.)
7. The receiver then applies the inverse (decoding) function to these codewords to find the initial blocks of n bits. Stringing these blocks together gives the initial message.

A formal definition for a linear code is:

Definition: A *linear code* is a n-dimensional subspace of n of $\mathbb{F}_2^{n+k}$.

Notice that we could use any finite field but we will use $\mathbb{F}_2$ for simplicity as a bit can be thought of as $\mathbb{F}_2$.

Taking the standard basis vectors to a basis for a code gives you a function which you can use to encode your data. The elements in the image of the function are the *codewords*. Thus a code $C \subset \{0,1\}^n$ is the set of possible codewords.

If the distance between any two codewords entries on the list is $2d + 1$ then d errors can always be corrected, while $d + 1$ errors can always be detected. Thus we want there to be a large distance between any two codewords so that we can correct as many errors as possible.

Definition: The *distance* of a code $C \subset \{0,1\}^n$ is

$$\text{distance} = \frac{\min_{c_1 \neq c_2 \in C} \mathrm{d}_H(c_1, c_2)}{n}.$$

However, we also want to send as few bits as possible which leads us to the idea of the rate of a code.

Definition: The *rate* of a code $C \subset \{0,1\}^n$ is

$$\text{Rate} = \frac{\log |C|}{n}.$$

Thus we want to maximize the rate as well as the distance because this will mean that the codewords are spread out throughout the string and you are using as much of the string as possible. For example, you don't want all the codewords to be clumped in the first few bits because this would be inefficient; you would be transmitting more bits than you would need.

This leads us to the natural problem.

Problem: Is it possible to design a linear code with $|C| = 2^k$ with rate $> R_0$ and distance $> d_0$?

Using magical graphs we can construct a linear code the following properties.

Theorem: There exists a linear code with rate $\geq \frac{1}{4}$ and distance $\geq \frac{1}{10d}$.

We will start with the following lemma.

Lemma 2: Given a $(n, \frac{3n}{4}; d)$-magical graph G and a subset $S \subset L$ where $|S| < \frac{n}{10d}$ there exists a vertex $u \in R$ with exactly one neighbour in S.

Proof of Lemma 2: By the definition of a magical graph

$$\Gamma(S) \geq \frac{5d}{8}.$$

Rewriting this equation gives us

$$|S| \leq |\Gamma(S)|\frac{8}{5d}.$$

As $d \geq 1$ we see that

$$|S| < 2|\Gamma(S)|.$$

This means that although each vertex in S has d neighbours, the average number of neighbours for a vertex in $\Gamma(S)$ is less than 2. Obviously a vertex in $\Gamma(S)$ has at least one neighbour. Thus there is a vertex $u \in \Gamma(S)$ with only one neighbour, i.e., $|\Gamma(u) \cap S| = 1$.

Proof of the Theorem: Let G be an $(n, \frac{3n}{4}; d)$-magical graph. Let $S \subset L$ where $|S| < \frac{n}{10d}$.

We will index all the vertices in R by v_i and in L by w_j. Start by creating a $\frac{3n}{4}$ by n matrix A where $a_{i,j}$ is 1 if v_i is adjacent to w_j, i.e.,

$$a_{i,j} = \begin{cases} 1 & \text{if } v_i \in R \text{ adjacent to } w_j \in L \\ 0 & \text{otherwise.} \end{cases}$$

Normally the adjacency matrix is a square matrix but for a bipartite graph the diagonal blocks are zero and the remaining blocks are the transpose of each other (as the adjacency matrix is a symmetric). Thus we are only concerned with one of these remaining blocks. The following gives the form for a adjacency matrix of a bipartite graph in terms of our matrix A

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}.$$

Recall the formal definition of a code we gave was a subspace of $\mathbb{F}_2^n$. Let $C = \ker A$. We want to show that C is a code.

As there are only $\frac{3n}{4}$ rows we see

$$\text{Rank } A \leq \frac{3n}{4}.$$

This implies

$$\dim C \geq \frac{n}{4}.$$

Thus

$$|C| \geq 2^{\frac{n}{4}}.$$

As the rate was $\log_2$ of the code's size this proves Rate $\geq \frac{n}{4}$.

As the code is a linear subspace, given two non-zero codewords c_1 and c_2 you could shift them preserving the distance so that one is zero and the other is $c_2 - c_1$. Thus the distance of the code equals the least number if ones in a non-zero codeword.

Suppose there was $x \in C \subset \mathbb{F}_2^n$ where x had less than $\frac{n}{10d}$ ones. Let $S \subset L$ be the vertices which correspond to non-zero digits of x. Then $|S| < \frac{n}{10d}$. From Lemma 2 there exists v_i such that $|\Gamma(v_i) \cup S| = 1$.

$$Ax = \begin{pmatrix} \vdots \\ 1 \ (i^{\text{th}} \text{ entry}) \\ \vdots \end{pmatrix}.$$

Thus $x \notin C$ as x is not in the kernel which is a contradiction. This shows there is no codeword with less than $\frac{n}{10d}$ ones. Therefore $d_H \geq \frac{1}{10d}$ which concludes the proof.

Next we will use magical graphs to construct super concentrators.

Super Concentrators

Definition: A graph G is called a *super concentrator* if there are subsets I and O, called the input and output sets respectfully, such that for every k and for all $S \subset I$ and $T \subset O$ with $|S| = |T| = k$, there exits k vertex disjoint paths in G from S to T.

Super concentrators are graphs that have a large number of totally independent paths between 'input nodes' and 'output nodes'.

If you model a data network in a way where there are a lot of independent paths between sender and receiver then the network is resilient to line breaks.

This can be useful when you design a circuit. Although it also makes sense that you would want to be using as few edges as possible to reduce the cost of the wire, (when circuits were still made of wire), which leads us to the following problem.

Problem: Given n what is the smallest K such that there is a super concentrator with n input and output vertices and Kn edges?

Trivially it is always possible to have a super concentrator with n^2 edges. We would like to construct one with fewer edges. Using magical graphs we can prove the following theorem.

Theorem: There exists a constant K which depends only on the degree d and n_0, where the n_0 is the constant mentioned in Lemma 1.

To do this we will need a result from graph theory.

Definition: A *perfect matching* for a bipartite set is a set of edges in a graph such that each vertex in the left set is incident to precisely one vertex in the right set.

Hall's Marriage Theorem: Given a bipartite graph $G = (L, R, E)$ there exists a perfect matching if and only if $|X| \leq |\Gamma(X)|$ for any $X \subset L$.

Proof: Given a $\left(n, \frac{3n}{4}; d\right)$-magical graph and $S \subset L$.

We will use Hall's theorem to show that there is a perfect matching from S to $\Gamma(S)$.

By the definition of a magical graph

$$|\Gamma(S)| = \begin{cases} \frac{5d}{8}|S| & \text{if } |S| \leq \frac{n}{10d} \\ |S| & \text{if } \frac{n}{10d} < |S| \leq \frac{n}{2} \end{cases}$$

In the first case if $d = 1$ then $|\Gamma(S)| \geq |S|$ since every vertex in S has a unique vertex in $\Gamma(S)$ while if $d > 1$ then $|\Gamma(S)| \geq \frac{10}{8}|S| < |S|$.

In the second case we also have $|\Gamma(S)| \geq |S|$.

Thus in both cases $|\Gamma(S)| > |S|$.

This implies by Hall's Theorem there exists a perfect matching from S to $\Gamma(S)$.

We will recursively construct a super concentrator. We proceed by induction on n. Recall there was a constant n_0 in Lemma 1 which proves the existence of magical graphs. If $n < n_0$ choose a complete bipartite graph. It has n^2 edges and is a super concentrator.

If $n \geq n_0$ choose two copies $G_1 = (L_1, R_1, E_1)$ and $G_2 = (L_2, R_2, E_2)$ of a $(n, m; d)$-magical graph given by Lemma 1 (where $m = \left\lceil \frac{3n}{4} \right\rceil$ and $d = 32$).

Let C be a super concentrator between R_1 and R_2. We choose C to be the super concentrator found by induction. Thus we glue G_1 and G_2 together using C. We also add an edge from each vertex in L_1 to the corresponding vertex in L_2.

Thus the vertices of the graph are

$$V = L_1 \cup R_1 \cup R_2 \cup R_1.$$

Thus the edges of the graph are

$$E = E_1 \cup E_2 \cup E(C) \cup \{\text{edges from } v_i \in L_1 \text{ to } w_i \in L_2\}.$$

We want to show that this graph is a super concentrator with input set L_1 and output set L_2.

Choose $S \subset L_1$ and $T \subset L_2$ such that $|S| = |T| = k$.

If $k \leq \frac{n}{2}$ then $|\Gamma_{G_1}(S)| \geq |S|$ and $|\Gamma_{G_2}(T)| \geq |T|$ since G_1, G_2 are magical.

By Hall's Theorem there exists a perfect matching from S to $S' \subset \Gamma_{G_1}(S)$ and T to $T' \subset \Gamma_{G_2}(T)$.

As C is a super concentrator this implies there are disjoint paths from S' to T'. This is disjoint paths from S to S' and S' to T' and T' to T. Thus linking them together you get k disjoint paths from S to T.

If $k > \frac{n}{2}$ identifying the vertices in L_1 and L_2 there exists an overlap. Thus match $S \cap T$ and $S \cap T$ directly using the edges between L_1 and L_2.

Then you are left with sets $S \setminus T$ and $T \setminus S$ which are smaller than $\frac{n}{2}$. Then you can use the previous method to construct disjoint paths from $S \setminus T$ and $T \setminus S$.

Thus C' is a super concentrator.

The number of edges are:

$$e(n) = \begin{cases} 2nd + n + e(\frac{3n}{4}) & \text{for } n > n_0 \\ n^2 & \text{for } n \leq \ n_0. \end{cases}$$

Solving this recurrence relation gives a constant K which depends only on n_0 and d. More precisely we want K such that $2nd + n + K(\frac{3n}{4}) \leq Kn$ and $K \geq n_0$. Thus $K = \max(4(2d+1), n_0)$ satisfies these conditions and hence this concludes the proof.

Finally we will use expander graphs to generate hash functions.

Hash Functions

A hash function is a function which takes a large, unbounded amount of data and maps it to a string of n-bits. As well it is a publicly known function. However, while everyone should know how to compute the hash, it should be hard to find the inverse to the function.

One main reason for hash functions is to give a digital signature for documents. For example, Microsoft will sign their files by calculating the hash of its files. After the computer downloads one of their files the computer calculates the hash of the file and compares this hash to the digital signature. This way it checks to make sure the file is the authentic file and not a virus pretending to be Microsoft's file.

In order for the hash to be useful for digital signatures it should be infeasible for someone to create another document with the same hash.

More formally, four properties which we would want in any *cryptographic hash function* are:
1. easy to compute the hash value for any given message,
2. infeasible to find a message that has a given hash,
3. infeasible to modify a message without changing its hash,
4. infeasible to find two different messages with the same hash.

The problem is that many hash functions which are used today are in the same family. Some of the hash functions in this family have already been proven to be vulnerable to attacks. The fear is that the rest will eventually

also prove to be vulnerable to attacks. Thus it would be desirable to an alternative which is provably secure while still easy to compute.

The idea of having a provably secure hash is to have a hash where solving the problems given in 2, 3 and 4 requires you to solve a mathematical difficult problem like factoring numbers.

We want a method which is mathematically secure but still easy to compute.

Right now the methods which are provably secure take more time to compute. However, perhaps with some work they can be streamlined and there can be circumstances where it would still be worthwhile.

We will discuss methods for constructing provably secure hash function using Ramanujan expander graphs.

Expander Graphs

Suppose we are given an 3-regular expander graph G.

We are choosing 3-regular for simplicity since we are working base 2. However, if we had a k-regular expander graph we could write the data base $k-1$ and the same idea would work.

The steps for getting hash (using a random walk on a 3-regular graph) from unbounded string on bits is given as follows:

1. We choose a vertex v_0. (Choosing a different vertex would give us a different hash function.)
2. Order the edges for each vertex from 1 to 3. The orderings of the edges will not be consistent for different vertices.
3. Starting at v_0 if the first digit is 0 then take the least edge, if it is 1 take the next edge.
4. Call the the other endpoint of the edge v_1.
5. If the second digit is 0 take the least remaining edge (don't include the edge you just used), if it is 1 use the other edge.
6. Call the other endpoint of this edge v_2.
7. Repeating in this fashion gives a random walk which has no backtracking.
8. The endpoint of the random walk is the output.

Thus the output is a number between 1 and n where n is the number of vertices in the graph, or $\log_2 n$ bits.

A random walk on an expander graph mixes quickly. By this we mean the endpoint approximates uniform distribution after $O(\log(n))$ steps.

Next we will give a method for constructing expander graphs using elliptic curves.

Pizer's Ramanujan Graphs

We will assume a bit of background on elliptic curves. See Silverman's Elliptic Curves for more details.

Let p be a large prime say 256 bits and let l be a small prime as in our previous discussion we can assume it is 2.

An *elliptic curve* can be thought of an a non-singular curve of degree 3.

Adding a point at infinity turns the points on this curve into an abelian group where the point at infinity was the identity.

We will be interested in a certain type of elliptic curve called a supersingular elliptic curves. The isomorphism classes of supersingular elliptic curves can be classified by an invariant called the *j-invariant*. We will let the values for the j-invariant be the vertices for the graph we will be constructing.

The j-invariant must lie in the field. As we will be dealing with finite fields this means there are only finitely many vertices.

Recall an *l-isogeny* is a surjective homomorphism (when you work over the algebraic closure) where the kernel is a cyclic group of order l. We will draw an edge between two vertices if there is an l-isogeny between them.

Again when work over the algebraic closure the l-torsion points form a group $\frac{\mathbb{Z}}{l\mathbb{Z}} \times \frac{\mathbb{Z}}{l\mathbb{Z}}$ which means there are $l+1$ of these l-isogenies. This means we have an $l+1$-regular graph.

Although we will not prove it here, it follows from the fact that it is a Ramanujan graph that this graph has good expander properties.

As we can put an ordering on the $(l + 1)$ isogenies, because we can order their generators, we can order the edges in a natural fashion.

Next we make a random walk as was mentioned earlier.

The problem of finding an input which has a hash output of a particular value reduces to the problem of factoring a map into l-isogenies, which is a hard problem.

The graph can also be shown to have a large girth (in other words, there are no small cycles).

The other method for constructing good expander graphs mentioned in the paper by Charles, Goren and Lauter has been shown to be vulnerable to attacks.