CS168: The Modern Algorithmic Toolbox
Lecture #19: Expander Codes

Tim Roughgarden & Gregory Valiant*

June 1, 2016

In the first lecture of CS168, we talked about modern techniques in data storage (consistent hashing). Since then, the major course themes have been about the representation, analysis, interpretation, and visualization of data. Today we'll close the circle by returning to the "nuts and bolts" of managing data, specifically to data transmission (using error-correcting codes).

1 Expanders

1.1 Definitions and Examples

We begin with a question:

> Can a graph be simultaneously "sparse" and "highly connected?"

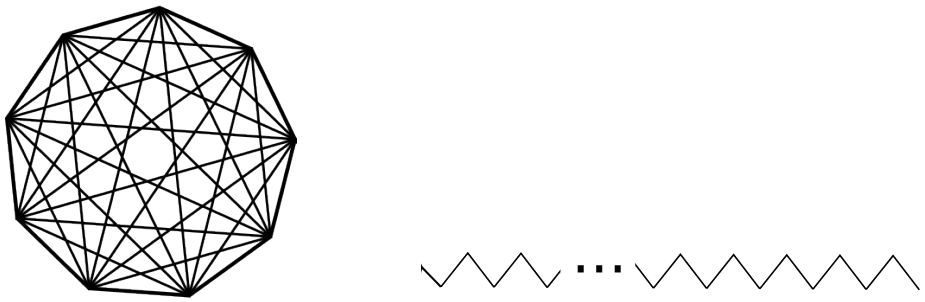
The first half of the lecture explains an affirmative answer to this question. The second half connects the question to the design of error-correcting codes.

What exactly do we mean by "sparse" and "highly connected?" The answers are familiar. One common definition of a sparse graph, as seen in CS161, is that the number m of edges is $O(n)$, where n is the number of vertices. In this lecture we'll impose a stronger condition, that every vertex has $O(1)$ degree. This implies that there are only $O(n)$ edges (why?), while the converse is not true (e.g., consider a star graph).

Intuitively, we want "highly connected" to translate to "acts like a clique," since a clique is clearly the most well connected graph. We'd like a definition that differentiates and interpolates between extreme cases like cliques and paths (Figure 1). Back in Week 6, we saw two essentially equivalent definitions of "highly connected," one combinatorial and one algebraic.

To recap, recall that the *isoperimetric ratio* of a graph $G = (V, E)$ is

$$\min_{S \subseteq V \,:\, |S| \leq |V|/2} \frac{|\delta(S)|}{|S|}.$$



(a) A clique (b) A path

Figure 1: A canonical expander and a canonical non-expander.

The notation $\delta(S)$ refers to the "boundary" of S — the edges with exactly one endpoint in S. Thus the isoperimetric ratio is large if and only if every subset of at most half the vertices has a large "surface area to volume" ratio. For example, the isoperimetric ratio of the clique is $\approx \frac{n}{2}$, while that of the path is $\frac{2}{n}$.

The second definition involves λ_2, the second-smallest eigenvalue of the graph's Laplacian matrix.[1] Computations show that λ_2 is large (like n) for the clique but small (like $\Theta(1/n^2)$) for the path. Thus both definitions agree that the connectivity measure is going to infinity with n for cliques and going to 0 with n for paths. More generally, in Lecture #12 we noted *Cheeger's inequality*, which implies that one of these quantities is large if and only if the other quantity is large.

For d-regular graphs — where every vertex has degree d — the isoperimetric ratio is at most d. (Since $\delta(S)$ can at most contain all of the $d|S|$ edges incident to vertices in S.) In Week 6 we saw that the same is true for λ_2. So when we restrict attention to sparse (constant-degree) graphs, we will not see these quantities tend to infinity with n, as they do for cliques. So a d-regular graph should be considered "highly connected" if these quantities are close to the obvious upper bound of d.

Formally, a family of *expander graphs* is one such that the isoperimetric ratio and λ_2 do not tend to 0 as the number n of vertices tends to infinity — i.e., they stay bounded away from 0 as $n \to \infty$. (Recall that this holds for one of the measures if and only if it holds for the other measure.) Thus cliques are expanders while paths are not, as one would hope.[2]

[1] Recall this is the $V \times V$ matrix $D - A$, where D is the diagonal matrix with entries equal to vertex degrees, and A is the graph's adjacency matrix.

[2] Among friends one might talk about "an expander graph," but strictly speaking the definition of an expander only makes sense for an infinite family of graphs. Note that when we say "cliques" or "paths" we're already talking about an infinite family, since there is one clique and path graph for each value of n.

1.2 Existence of Expanders

We know that expanders exist, since cliques are expanders. But do *sparse* expanders exist?[3] You should not have any a priori intuition about this question. Certainly 1-regular graphs (i.e., perfect matchings) and 2-regular graphs (collections of cycles) can never be expanders. This makes the following fact all the more amazing.

Fact 1.1 *For every $d \geq 3$, there exists a family of d-regular expander graphs.*

In fact, *almost every* d-regular graph with $d \geq 3$ is an expander! This is proved using the "probabilistic method," which is a very simple but very cool idea developed by Paul Erdös (see [1]). The most straightforward way of proving the existence of an object with desired properties is to simply exhibit the object. In the probabilistic method, one instead defines, as a thought experiment, a random experiment that produces an object (like a graph). Then one proves that the produced object has the desired property (like expansion) with positive probability over the random choices in the experiment. Note that this implies existence — if no objects with the desired property exist, then certainly the probability of randomly selecting an object with the desired property is zero!

For Fact 1.1, then, we want to define a random experiment that produces a d-regular graph, and that generates an expander with positive probability. There are several ways this can be done; for example, with an even number n of vertices, one can choose d perfect matchings (i.e., group all vertices into pairs), independently and uniformly at random, and superimpose them to obtain a d-regular graph. One can prove that, for every $d \geq 3$, this process has a positive probability of generating an expander.[4] And it turns out that this probability is not just bigger than 0, but is actually very close to 1.

What about "explicit" constructions of expander graphs, in the form of a deterministic and computationally efficient algorithm that generates "on demand" an expander graph of a given size? These are much harder to obtain. The first explicit constructions are from the 1980s, where the constructions are simple to state (using just the simplest-possible number theory) but require difficult mathematics to analyze. In the 21st century, there have been several iterative constructions for which the expansion requirement can be verified using elementary linear algebra (see [3]).[5]

What good are expander graphs? In situations where you have the luxury of choosing your own graph, expander graphs often come in handy. They were first considered in the mid-20th century in the context of telecommunication network design — in that context, it's clear that both sparsity and rich connectivity (i.e., robustness to failures) are desirable properties. We next study an application in which expander graphs are not physically

[3] Often when one talks about expanders it's a given that you're restricting to graphs of constant degree.

[4] For large enough d (like 20), the proof is not hard, and it shows up as a homework problem in CS264 ("Beyond Worst-Case Analysis").

[5] The difficulty of explicit constructions is one of the paradoxes you just have to get used to in the study of theoretical computer science. While it is easy to believe that finding a needle in a haystack (a euphemism for the the P vs. NP problem) is hard, sometimes it's hard to even find hay in a haystack! (Recall that almost all graphs are expanders...)

realized as networks, but rather a bipartite variant is used logically to define good error-correcting codes.

2 Expander Codes

2.1 Error-Correcting Codes

Error-correcting codes address the problem of encoding information to be robust to errors (i.e., bit flips). We consider only binary codes, so the objects of study are n-bit vectors.

The simplest error-correcting code uses a single parity bit. This involves appending 1 bit to the end of message, a 0/1 as needed to ensure that the resulting string has even parity (i.e., an even number of 1s). So we would encode

$$100 \mapsto 1001$$

and

$$101 \mapsto 1010.$$

The corresponding *code* — that is, the legitimate codewords — is then $C = \{\mathbf{x} \in \{0,1\}^n : \mathbf{x} \text{ has even parity}\}$.

The key point is that the *Hamming distance* — the number of differing coordinates — of two distinct codewords is at least 2: if you flip one bit of a codeword, then you also need to flip a second bit to obtain a string with even parity. This means that the *distance* of the code — the minimum Hamming distance between two distinct codewords — is precisely 2.

A code that has distance 2 can detect one error (i.e., a bit flip): if exactly one bit gets flipped, then the receiver will notice that the received (corrupted) message is not a valid codeword, and can therefore ask the sender to retransmit the message. With a parity bit, an odd number of errors will always be detected, while an even number of errors will never be detected (since it results in a new legitimate codeword).

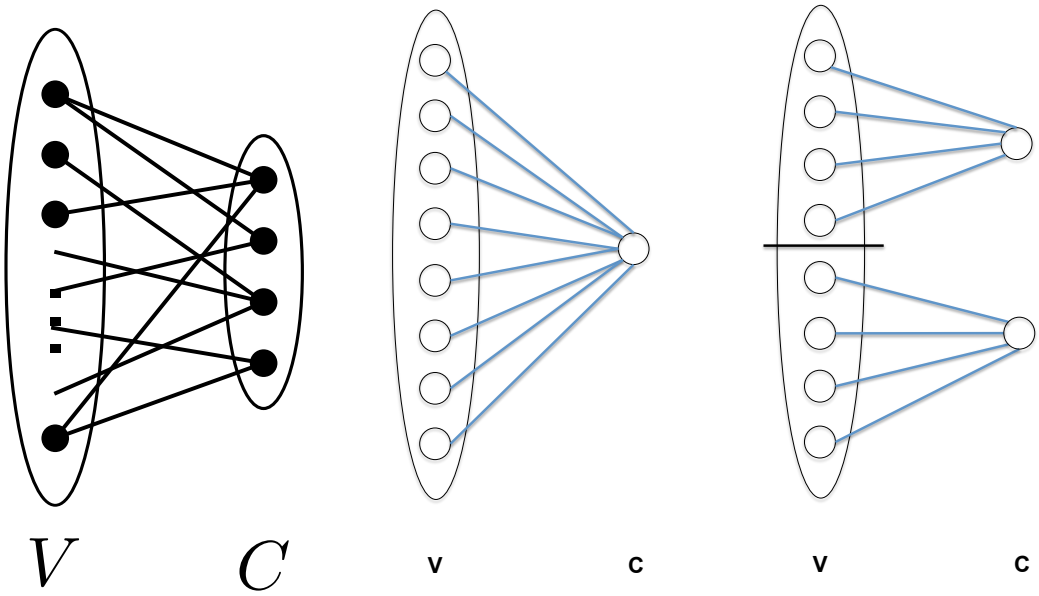
Generalizing, if a code has distance d, up to $d-1$ adversarial errors can be detected. If the number of errors is less than $d/2$, then no retransmission is required: there is a unique codeword closest (in Hamming distance) to the transmission, and it is the message originally sent by the receiver.[6] That is, the corrupted transmission can be *decoded* by the receiver.

How can we generalize the parity bit idea to obtain codes with larger distance? To be useful, it is also necessary to have computationally efficient algorithms for encoding messages and decoding corrupted codewords.

2.2 From Graphs to Codes

Every bipartite graph $G = (V, C, E)$ can be used to define an error-correcting code (Figure 2(a)). The left-hand side vertices V correspond to the n coordinates or variables of a

[6] The Hamming distance satisfies the triangle inequality (why?), so if there are two different codewords with Hamming distance less than $d/2$ to a common string, then they have Hamming distance less than d from each other.



(a) Codes from graphs (b) 1 parity bit (c) 2 parity bits

Figure 2: Bipartite graphs define error-correcting codes. The n coordinates of V represent the code word, and each element of C represents a parity check.

code word. Every right-hand vertex $j \in C$ corresponds to a "parity check." More formally, a vector $\mathbf{x} \in \{0,1\}^n$ *satisfies* the parity check $j \in C$ if $\mathbf{x}_{N(j)}$ has even parity, where $N(j) \subseteq V$ denotes the neighbors of vertex j in G and $\mathbf{x}_I$ denotes the vector $\mathbf{x}$ projected onto the coordinates of the set I. The code corresponding to G is, by definition, the vectors $\mathbf{x} \in \{0,1\}^n$ that satisfy every parity check of C. For instance, the code generated by using a single parity bit corresponds to the graph in which C is a single vertex, connected to all of the variables (both the original bits and the parity bit) — see Figure 2(b). For a simple extension, we could imagine having one parity check for the first half of the coordinates and a second for the second half (Figure 2(c)). With this code, two errors might or might not be detected (depending on whether they occur in different halves or the same half, respectively).

Every bipartite graph defines a code, but some graphs/codes are better than others. The goal of this lecture is to identify conditions on the graph G so that it is possible to correct many errors (a constant fraction, even), and moreover in linear time.

Following [4], we propose the following conditions.

(C1) Every left-hand side vertex $i \in V$ has exactly d neighbors, where d is a constant (like 10).

(C2) Every right-hand side vertex $j \in C$ has at most a constant (like 20) number of neighbors.

(C3) [Expansion condition] There is a constant $\delta > 0$ (like 0.1), independent of n, such that

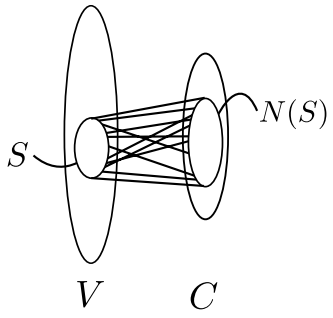


Figure 3: The expansion condition (C3). Every small enough subset $S \subset V$ must have lots of distinct neighbors in C.

for all subsets $S \subseteq V$ with size $|S|$ at most δn,

$$|N(S)| \geq \frac{3}{4} d|S|, \tag{1}$$

where $N(S) \subseteq C$ denotes the vertices (i.e., parity checks) of C that have at least one neighbor in S. Intuitively, this expansion condition implies that a bunch of errors (represented by S) must be reflected in tons of different parity checks (corresponding to $N(S)$). See Figure 3.

In the inequality (1), we use $N(S)$ to denote the vertices of C that have at least one neighbor in S. Observe that by (C1), the number of edges sticking out of S is only $d|S|$, so $|N(S)|$ is certainly at most $d|S|$. The condition (1) asserts that the number of distinct neighbors of S is almost as large as the trivial upper bound, and moreover this holds for *every* one of the exponentially many choices for S. This sounds like a strong property, so you would be right to question if there really exist any graphs that satisfy (C3). Again using the probabilistic method, it is not too hard to prove that a random graph that satisfies (C1) and (C2) also satisfies (C3) with high probability.

Before addressing encoding and decoding, we establish next the fact that expander codes have large distance (linear in n) and hence can detect/correct codewords where a constant fraction of the bits have been corrupted. The point of going over this proof is to develop intuition for why the expansion condition (C3) leads to good codes.

Theorem 2.1 ([4]) *A code satisfying conditions (C1)–(C3) has distance at least δn, where $\delta > 0$ is the same constant as in condition (C3).*

Proof: Let $\mathbf{w}$ be a code word, and let $\mathbf{x} \in \{0,1\}^n$ be a vector with $d_H(\mathbf{w}, \mathbf{x}) < \delta n$. We need to show that $\mathbf{x}$ is not a code word.

Let S denote the $d_H(\mathbf{w}, \mathbf{x})$ coordinates in which $\mathbf{w}$ and $\mathbf{x}$ differ. We claim that there exists a parity check $j \in C$ that involves exactly one coordinate of S. Observe that this claim implies the proposition: since $\mathbf{w}$ is a code word, it satisfies j, and since exactly one

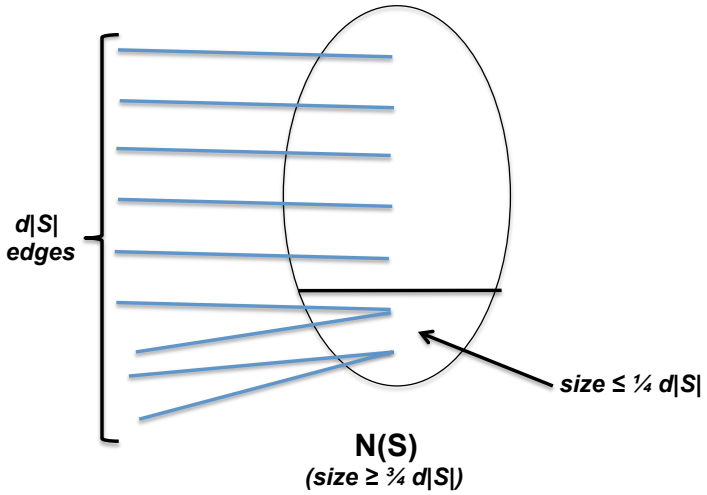


Figure 4: Proof of Theorem 2.1. Because $d|S|$ edges go to at least $\frac{3}{4}d|S|$ different vertices, at least $\frac{1}{2}d|S|$ of the vertices in $N(S)$ are connected to exactly one vertex of S.

of the coordinates of j is flipped in $\mathbf{x}$, relative to $\mathbf{w}$, $\mathbf{x}$ does not satisfy the parity check j. Hence $\mathbf{x}$ is not a code word.

To prove the claim, note that condition (C1) implies that there are precisely $d|S|$ edges sticking out of the vertices of G that correspond to S (Figure 4). Since $|S| < \delta n$, condition (C3) implies that $|N(S)| \geq \frac{3}{4}d|S|$. That is, at least 75% of the edges sticking out of S go to distinct vertices of C. This leaves at most 25% of the edges with the capability of donating a second neighbor of S to a parity check in $N(S)$. It follows that at least

$$\left(\frac{3}{4} - \frac{1}{4}\right) d|S|$$

vertices of $N(S)$ have a unique neighbor in S, which completes the claim and the proof. ■

What role does expansion play in the proof of Theorem 2.1? For a graph-based code to have small distance, it must be the case that starting from a codeword and flipping a small number of bits causes, for each parity check, an even number of flips among the variables in that check. The expansion condition asserts that there is minimal overlap between the parity checks in which two different variables participate, and this rules out such a "conspiracy of cancellations."

2.3 Encoding

How does one actually use an expander code to do encoding and decoding? Encoding a message $\mathbf{y}$ is done via matrix-vector multiplication — i.e., the codeword is $\mathbf{Ay}$, where $\mathbf{A}$ is a suitable matrix with more rows than columns. This is the generic encoding procedure for

any linear code — i.e., where the sum modulo 2 of two codewords is again a codeword, as is the case for our codes (why?) — and it is not specific to expander codes. You can think of the columns of $\mathbf{A}$ as a basis (in the linear algebra sense) of the vector space (over $\mathbb{F}_2$) of codewords; then encoding just means interpreting the message $\mathbf{y}$ as the coefficients for a linear combination of these basis codewords.

2.4 Decoding

What's interesting about expander codes is their decoding properties. Decoding boils down to solving the *nearest codeword* problem. The input is a (corrupted) message $\mathbf{x}$, with the promise that it has Hamming distance less than $\delta n/2$ from a legitimate codeword $\mathbf{w}$.[7] The goal is to output $\mathbf{w}$.

We consider the following amazingly simple decoding algorithm.

Decoding Algorithm

while there is a variable i such that more than half of the parity checks involving i are unsatisfied **do**
  flip i `// if several candidates, pick one arbitrarily`

With our running example of $d = 10$: if at least 6 of the 10 parity checks that include a variable are unsatisfied, then this variable is a candidate to be flipped.

This algorithm is guaranteed to terminate, no matter what the graph G is. Why? When we flip a variable, the parity checks that contain it toggle between being satisfied and unsatisfied. So if pre-flip there were more unsatisfied than satisfied parity checks, post-flip the reverse is true. So every flip strictly increases the number of satisfied parity checks, and the algorithm must terminate within $|C|$ iterations. Additionally, the algorithm can be implemented in linear time.

The bigger issue is: how do we know that the algorithm terminates with the nearest codeword $\mathbf{w}$? The concern is that the algorithm terminates with a non-codeword that nevertheless satisfies at least 50% of the parity checks for each variable. And indeed, with an arbitrary choice of the graph G, this failure case can occur.

This brings us to our second result.

Theorem 2.2 ([4]) *If the graph G satisfies conditions (C1) and (C3) and the corrupted code word $\mathbf{x}$ has Hamming distance at most $\delta n/2$ from its nearest code word $\mathbf{w}$, then the decoding algorithm above is guaranteed to terminate at $\mathbf{w}$.*

The proof is a slightly more elaborate version of the counting arguments we used to prove Theorem 2.1.[8] The expansion condition again rules out the conspiring cancellations of

[7] Recall that $\mathbf{w}$ is unique — if there were two different codewords both with Hamming distance less than $\delta n/2$ from $\mathbf{x}$, then the distance of the code would be less than δn.

[8] It's not too hard, and shows up as a homework problem in CS264.

a bunch of flipped variables that would be required for the algorithm to get stuck somewhere other than the nearest codeword $\mathbf{w}$.

When expander codes were first proposed in [4], there was interest from several media companies. For example, for encoding CDs and DVDs, the property of linear-time decodability is very attractive. The companies were concerned about how to choose the bipartite graph G. The probabilistic method shows that a random graph would almost always work, but how can you be sure? Explicit constructions of sufficiently good expanders came along only in 2002 [2], and the inability to determinisitically find "hay in a haystack" was an initial impediment to the adoption of expander codes. These days, codes inspired by expander codes are indeed used in practice.

References

[1] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley-Interscience, 2008. Third edition.

[2] Michael R. Capalbo, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 659–668, 2002.

[3] S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43:439–561, 2006.

[4] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 2007.