CONSTRUCTING ERROR-CORRECTING CODES FROM EXPANDER GRAPHS

DANIEL A. SPIELMAN*

Abstract. We survey a derivation of asymptotically good error-correcting codes from expander graphs. These codes, called Expander Codes, can be decoded in linear time. We explain how to modify this construction to produce asymptotically good codes that can be encoded as well as decoded in linear time.

Key words. Asymptotically good error correcting code, expander graph.

1. Introduction. While there are simple randomized constructions of good error-correcting codes and expander graphs, explicit constructions are more difficult to find. In this paper, we survey a derivation of asymptotically good error-correcting codes from explicit constructions of expander graphs that appeared in [7,10]. These error-correcting codes have the advantage that they can be both encoded and decoded in linear time. In the present paper, we provide a simple explanation of how these constructions work, without optimizing the constructions. We will explain the work from a theoretical perspective, obtaining constants but ignoring their impact on engineering realities.

We begin with a short introduction to the theory of expander graphs, in Section 2, and error-correcting codes, in Section 3. In Section 4, we explain how expander graphs can be used to make asymptotically good error-correcting codes. In Section 5 we explain how to decode them in linear time. As known encoding algorithms for these codes use quadratic time, we sketch, in Section 6, how this construction can be modified to produce linear-time encodable and decodable error-correcting codes. In Section 7, we conclude by explaining one way in which we would like to see the construction of Section 4 improved.

Before beginning, I wish to warn the reader that all statements in this paper should be interpreted asymptotically. While I may refer to *an expander graph*, I really mean a *family of expander graphs*. Similarly, the concept of an asymptotically good code only makes sense as n grows large.

2. Expander Graphs. Expander graphs have been the subject of much study in combinatorics and computer science. While everyone agrees that an expander graph should be a graph in which every reasonably small set of vertices has many neighbors, there are many different ways of measuring the quality of an expander. For our purposes, it is best to focus on the size of induced subgraphs.

* Department of Mathematics, M.I.T., Cambridge MA 02139. Supported in part by an NSF mathematical sciences postdoc.

Consider a d-regular graph on n vertices. The expected number of edges in a graph induced by a randomly chosen subset of αn vertices is

$$\alpha^2 \frac{dn}{2}.$$

We will consider a family of d-regular graphs to be a family of good expanders if all subsets of αn vertices have induced subgraphs of roughly this size. A random d-regular graph is a good expander with high probability. Explicit constructions are more difficult to come by.

The property that we require of an expander graph can be witnessed by the eigenvalues of its adjacency matrix. The largest eigenvalue of the adjacency matrix of a d-regular graph is d. Let G be a d regular graph whose next-largest eigenvalue in absolute value is $\lambda < d$. Using straightforward linear algebra, (see [1] for a proof) one can show that, for every set of αn vertices in G, the subgraph induced by these vertices contains at most

$$\left(\alpha^2 + (\alpha - \alpha^2)(\lambda/d)\right) \frac{dn}{2} \tag{2.1}$$

edges.

Margulis and Lubotzky, Phillips, and Sarnak [3,4] have produced, for an infinite number of degrees d, infinite families of d-regular graphs with second-largest eigenvalues bounded by $2\sqrt{d-1}$. These are terrific expanders. Moreover, Alon and Boppona [5] have shown that one cannot produce infinite families of d-regular graphs with smaller second-largest eigenvalues. Thus, these expanders are the best that we can witness using (2.1). However, it is easy to show that random d-regular graphs have better expansion for $\alpha < 1/d$, and it is quite possible that the graphs produced in [3,4] have better expansion as well. The actual statement proved in [3,4] has the following form:

THEOREM 2.1. *For every pair of primes p, q congruent to 1 modulo 4 such that p is a quadratic residue modulo q, there is a simply describable $(p+1)$-regular Cayley graph of $PSL(2, Z/qZ)$ with $q(q^2-1)/2$ vertices such that the second-largest eigenvalue of the graph is at most $2\sqrt{p}$.*

For asymptotic purposes, weaker families of expanders suffice. All that we really need is, for an infinite number of degrees $d_1, d_2, \ldots$, an infinite family of d_i-regular graphs in which all graphs of degree d_i have second-largest eigenvalue at most λ_i and

$$\lim_{i \to \infty} \lambda_i / d_i = 0. \tag{2.2}$$

Of course, graphs with smaller ratios of second-eigenvalue-to-degree will be better for practical purposes.

From any infinite family of d-regular graphs with second-largest eigenvalue bounded by some $\lambda < d$, we can produce a family that satisfies (2.2).

Given a d-regular graph on n vertices, G, consider the graph whose adjacency matrix is the square of the adjacency matrix of G. This graph will be a d^2-regular graph on n vertices. Each vertex will have d self-loops. After removing these self-loops, we obtain a $d(d-1)$ regular graph that we call G^2. If the second-largest eigenvalue of G is at most λ, then the second-largest eigenvalue of G^2 will be at most $\lambda^2 - d$. As iteration of this process produces graphs whose ratio of second-largest to largest eigenvalue becomes arbitrarily small, we see that we can obtain a family that satisfies (2.2).

3. Error-Correcting Codes. Error-correcting codes were developed to compensate for interference in communication. When a message is transmitted over a channel, what comes out at the other end is usually a slightly distorted version of the original. Thus, instead of sending a raw message, one sends the message along with some additional redundant information so that the receiver can figure out the intended message, even if some distortion occurs. A convenient mathematical abstraction of this situation is to imagine messages as bit strings in $\{0,1\}^m$ which are *encoded* as strings in $\{0,1\}^n$ $(n > m)$ and then transmitted over the channel. If the channel corrupts transmissions by randomly flipping bits that pass through it, then the receiver will receive a string in $\{0,1\}^n$ that differs in some places from the original transmission. It would be natural for the receiver to assume that the intended message is the one whose encoding differs in the least number of places from that which it receives. Thus, one should use a code in which each pair of messages in $\{0,1\}^m$ map to words in $\{0,1\}^n$ that differ in as many places as possible.

The standard formal language for these concepts is to define a *code* to be an injective mapping from $\{0,1\}^m$ into $\{0,1\}^n$. The strings in the image of the mapping are *codewords*. The length of the codewords, n, is the *block length* of the code. The phrase *error-correcting code* has no additional formal meaning, but it has the connotation that each pair of codewords differ in many places. The *Hamming distance* (usually shortened to *distance*) between two words is just the number of places in which they differ. One measure of the quality of a code is its *minimum distance*–the minimum of the distance between all pairs of codewords. As a computer scientist, I like to let n grow large and consider the *minimum relative distance* of a code, which is its minimum distance divided by n. Another important parameter of a code is its *rate*, m/n, which measures how much information is being transmitted with each bit. One objective of the study of error-correcting codes is to find codes of maximal minimum distance for their rate. Remarkably, it is possible to find infinite families of error-correcting codes in which the rate and relative minimum distance remain constant, even as the block-length grows. Such families are called *asymptotically good*.

3.1. Linear Codes. A convenient way to make an error-correcting code is to divide the n bits of a code into m message bits and $n-m$ check

bits. A *linear code* is one in which the check bits are linear combinations of the message bits. In our case, these are linear combinations over $GF(2)$. Alternatively, one could just define a linear code to be one in which the codewords form an m-dimensional subspace of $GF(2)^n$, as one can always divide such a code into message bits and check bits. A convenient property of linear codes is that the minimum distance between two codewords is equal to the minimum weight of a non-zero codeword, where the *weight* of a codeword is its distance from the all-0 word.

If one defines a code by choosing a random subspace of $GF(2)^n$ of the desired dimension, then one probably obtains a good code. In particular, it probably achieves the Gilbert-Varshamov bound:

THEOREM 3.1. *Let $\delta < 1/2$ and let C be a random linear code of length n and rate $r = 1 - H(\delta)$, where $H(x) = -(x \log x + (1-x)\log(1-x))$ is the binary entropy function. With arbitrarily high probability as n grows large, C has relative minimum distance at least δ.*

It is unknown whether there exist codes over $GF(2)$ with a better tradeoff between rate and minimum relative distance.

While it is easy to find a good error-correcting code, explicit constructions are more difficult to come by. In Section 4, we will use explicit constructions of expander graphs to produce explicit constructions of asymptotically good error-correcting codes. We emphasize that these codes are not known to be as good as random codes. However, they will have the advantage that they can be decoded in linear time. In contrast, all known decoding algorithms for random linear codes require exponential time.

4. Construction of Expander Codes. Expander codes fall within a class of codes introduced by Tanner [11] to generalize the low-density parity check codes introduced by Gallager [2].

A family of expander codes is specified by a linear code of block length d and an infinite family of d-regular expander graphs. Let $\mathcal{G}$ be an infinite family of d-regular graphs such that λ is an upper bound on the second-largest eigenvalue of each, and let C be a code of rate r, minimum relative distance δ, and block length d. The family of expander codes, $\mathcal{C}(\mathcal{G}, C)$, will contain codes of rate $2r - 1$ and minimum relative distance at least

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

From a graph $G \in \mathcal{G}$ with n vertices, we construct a code of length $dn/2$, called $\mathcal{C}(G, C)$. Each bit of the code is associated with an edge in the graph G. For each vertex of G, we make the restriction that the bits on its edges must form a codeword in C (each edge around a vertex should be associated with a bit in the code C). Since the restriction at each vertex results in $(1-r)d$ linear constraints, and there are n vertices, the $dn/2$ bits of the code suffer a total of $(1-r)dn$ linear constraints.

Each word in $\{0,1\}^{dn/2}$ can be associated with the set of edges in G of which it is the characteristic vector. Each codeword can then be viewed as a set of edges in the graph that induce codewords of C at each vertex. The reason that these codes have no low-weight non-zero codewords is that a low weight codeword would correspond to a small set of edges in the graph that induces codewords at each vertex. But, if the graph is a good expander, then a small set of edges will touch many vertices. In fact, it will touch so many vertices that some vertex will touch fewer than δd of these edges, which implies that these edges cannot induce a codeword at that vertex.

To make this argument formal, we use (2.1) to see that any set of

$$\left(\alpha^2 + (\alpha - \alpha^2)(\lambda/d)\right) \frac{dn}{2}$$

edges much have at least αn vertices as endpoints. Thus, at least one of these vertices will have at most

$$d(\alpha + (1-\alpha)\lambda/d)$$

of these edges as endpoints. Thus, the word in which these edges are set to 1 and all others are set to zero cannot be a codeword if

$$(\alpha + (1-\alpha)\lambda/d) < \delta.$$

This implies the desired bound on the relative minimum distance of the code.

5. Decoding Expander Codes. One reason that expander codes are exciting is that they contain families that can be decoded in linear time. That is, we can find families of expander codes, $\mathcal{C}(\mathcal{G}, C)$, for which there is a linear-time algorithm that will map to a codeword any word of relative distance at most ϵ from that codeword, for some positive constant ϵ. We begin by describing a logarithmic-time parallel algorithm that performs this decoding task. A natural approach to decoding these codes is to examine the status of each vertex individually. Since the edges touching a vertex were originally a codeword in the code C, one might attempt to decode by adjusting the edges around each vertex according to the decoding algorithm for C. To specify such an algorithm, we must decide what to do when the vertices that are the endpoints of an edge conflict in their opinions as to whether or not that edge should be flipped. Our algorithm will flip an edge if either of its neighbors think it should be flipped. The other natural choice, to flip an edge only if both neighbors think it should be flipped, does not necessarily work: for example, if C contains the all-1's word, and if all of the edges neighboring some vertex are corrupted, then they will form a codeword according to that vertex, but will appear corrupt according to their other neighbors (see Figure 5.1).

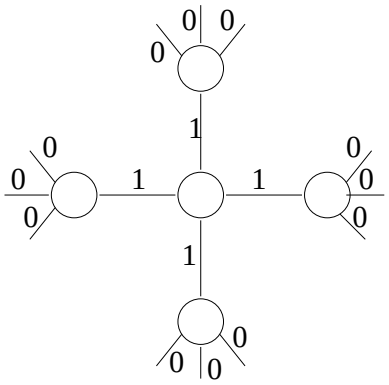

FIG. 5.1. *If the all-1s word is a codeword, then the vertex in the middle thinks it is looking at a codeword, while all the others think that their edge with a 1 is corrupt.*

Since C has minimum distance at least δd, it is possible to correct $\delta d/2$ errors made to codewords of C. However, if we try to use C to correct up to $\delta d/2$ errors, we could wind up creating more errors than we started with. If a vertex has $\delta d/2 + 1$ corrupt neighbors, then these corrupt neighbors might bring it closer to a different codeword. In this case, the vertex will think that it has $\delta d/2 - 1$ corrupt neighbors, which actually correspond to edges that are correct. If the decoding algorithm decides to flip these correct edges that the vertex thinks are corrupt, then the number of corrupt edges will increase. This effect is compounded by the fact that each edge can contribute to the confusion of two vertices. To prevent this problem, our algorithm will flip the edges suggested by a vertex only if the state of the edges around that vertex is within distance $\delta d/4$ of a codeword of C. This means that at least $3\delta d/4$ of the edges around a vertex will need to be corrupt before it can cause $\delta d/4$ additional correct edges to become corrupt.

Our parallel decoding algorithm will consist of a logarithmic number of rounds. In each round, each vertex whose edges are within $\delta d/4$ of a codeword will send a *flip* message to each edge that differs from that codeword. Then, each edge that receives at least one *flip* message will flip its bit. To formally analyze the effect of a decoding round, we call a vertex *confused* if it sends a *flip* message to a correct edge, and *unhelpful* if its edges are not within distance $\delta d/4$ of a codeword.

Assume that at most $\alpha dn/2$ edges have been corrupted. As each edge touches two vertices, there can be at most

$$\frac{2 \cdot \alpha dn/2}{3\delta d/4} = \frac{4\alpha n}{3\delta}$$

confused vertices and at most

$$\frac{\alpha dn}{\delta d/4} = \frac{4\alpha n}{\delta}$$

unhelpful vertices. The edges that are corrupt after the decoding round are those that receive inappropriate *flip* messages from confused vertices and those that are already corrupt and have both endpoints in unhelpful vertices. The number of inappropriate *flip* messages sent by the confused vertices is at most

$$\frac{4\alpha n}{3\delta}(\delta d/4) = (\alpha/3)\frac{dn}{2}$$

By (2.1), the number of edges with both endpoints in unhelpful vertices is at most

$$\left(\left(\frac{4\alpha}{\delta}\right)^2 + (\lambda/d)\left(\frac{4\alpha}{\delta} - \left(\frac{4\alpha}{\delta}\right)^2\right)\right)\frac{dn}{2}$$

Thus, the total number of corrupt edges at the end of the decoding round is at most

$$\left((\alpha/3) + \left(\frac{4\alpha}{\delta}\right)^2 + (\lambda/d)\left(\frac{4\alpha}{\delta} - \left(\frac{4\alpha}{\delta}\right)^2\right)\right)\frac{dn}{2}$$

To ensure that the decoding algorithm will succeed in a logarithmic number of rounds, we need this term to be less than $\alpha dn/2$ by a constant multiplicative factor. We observe that $(4\alpha/\delta)^2$ can be made arbitrarily small by choosing α to be small. We can then make $(\lambda/d)(4\alpha/\delta)$ arbitrarily small by choosing a family of graphs that satisfy (2.2). By Theorem 3.1, we know that we can vary d without changing r or δ. Once we fix d, we can use a constant-size brute-force search to find a code guaranteed by Theorem 3.1, or just use a code from another explicit asymptotically good family.

To transform this algorithm into a linear-time algorithm, observe that the number of vertices whose edges do not form a codeword of C decreases by a constant factor after each round. As the algorithm only needs to account for those vertices whose edges do not form a codeword of C, the total amount of work performed by the algorithm is linear.

6. Linear encoding complexity. Since expander codes are linear codes, they can be encoded in quadratic time. However, we would like to construct codes that can be both encoded and decoded in linear time. To do this, we modify the construction of expander codes and then combine the modified codes using a recursion introduced in [10].

We begin with a modification of expander codes that are linear-time encodable: for a d-regular graph G, associate to each vertex a code C with d message bits and k check bits. The code $\mathcal{R}(G, C)$ has $dn/2$ message

bits, which are identified with the edges of G. To produce the nk check bits of the code, each vertex produces the check bits of the code C that correspond to the encoding of the message bits on its edges. Clearly, this encoding operation can be performed in linear time. Moreover, if one knows all the values of all the check bits, then it is possible to correct errors in the message bits using the algorithm described in Section 5. Unfortunately, this code is asymptotically very poor: if a message bit and the check bits corresponding to the vertices it touches are corrupted, then there is no way to recover that message bit.

It is possible to prove that, if G is a good expander, then the code $\mathcal{R}(G, C)$ has a useful property: given a limited amount of corruption in its message and check bits, the decoding algorithm of Section 5 can be used to remove most of the corruption from the message bits. For this reason, we have called these codes *error-reduction codes*. If the errors are randomly distributed, then this algorithm will probably remove most of the errors from the message bits. For all reasonably small error patterns, this algorithm can reduce the number of errors in the message bits to half the number of errors in the check bits. Intuitively, this is reasonable because one error in a message bit affects two vertices, while an error in a check bit only affects one vertex. Thus, an error in a message bit has twice as much impact on the arguments of Section 5 as an error in a check bit. This intuition can be made rigorous, provided one makes slight modifications to the decoding algorithm of Section 5.

By taking advantage of the error-reducing properties of these codes, we can construct linear-time encodable and decodable error-correcting codes. We begin with a code R that has m message bits and $m/2$ check bits, that has a linear-time encoding algorithm, and that has a linear-time decoding algorithm that will correct ϵm errors in the message bits, given the correct values for all the check bits. We then encode the check bits produced by this code with a linear-time error-correcting code, A, that takes $m/2$ message bits, produces $3m/2$ check bits, and can correct $\epsilon m/2$ errors in any of its bits. Finally, the $2m$ bits of the code A are encoded by a code R_2 that has $2m$ message bits, m check bits, can be encoded in linear time, and has a linear-time error-reduction algorithm that will terminate with fewer than $\epsilon m/2$ errors in its message bits, if it began with at most ϵm errors in its message and check bits (See Figure 6.1). The check bits of the code we have built will be all the check bits produced by R, A, and R_2. The code A will be supplied by the recursion, and the codes R and R_2 will be error-reduction codes such as those described earlier in this section.

It is clear that the code we have just described can be encoded in linear time. To see that it can also correct ϵm errors, we begin by applying the error-reduction algorithm for R_2. We assumed that, if at most ϵm errors have occurred, then the error-reduction algorithm for R_2 will leave at most $\epsilon m/2$ errors among the $2m$ bits of the code A. As A is a code that can correct $\epsilon m/2$ errors in its $2m$ bits, and the message bits of A are the check

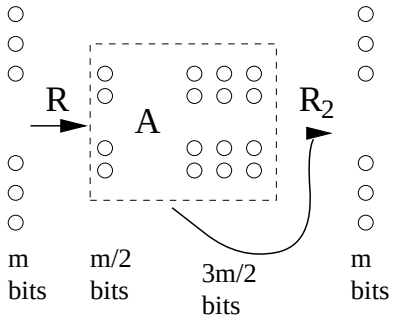

FIG. 6.1. *Error-reduction codes, R and R_2, are combined with a code, A, to produce a larger code, which serves in the place of A for the next recursive construction.*

bits of R, the decoding algorithm for A can be used to eliminate all the errors from these check bits. Once no errors remain among the check bits of R, and there are at most ϵm errors among its message bits, the error-correction algorithm described in Section 5 can be used to correct these errors. Thus, we have described a code with m message bits, $3m$ check bits, that is linear time encodable, and that has a linear-time algorithm that can correct ϵm errors. Such a code can now serve in the role of A in the construction of a new code that is twice as large.

While this construction results in codes of rate 1/4, it is easy to modify the construction to produce codes of any rate–one need only add or remove a few of the error-reduction codes from the front or back of the construction.

7. The quest for a canonical construction. It should be possible to improve the construction of expander codes presented in Section 4. We have not specified which code should be associated with the vertices. Since this code will be of constant size, we do not consider this terribly important–one can search for one by brute force, or use a well-known small code. However, we do not know how to take advantage of any particularly good choice for this code.

There should be some way to choose a code for the vertices that somehow respects the structure of the graph. For example, the edges in the graphs produced in Theorem 2.1 have a very special structure; perhaps there is a code that naturally lives on this structure. It is not clear what one should hope to achieve by finding a canonical construction, so we will make one suggestion: when determining the rate of the code produced in Section 4, we made the worst-case assumption that all of the constraints imposed by the vertices were independent. This need not be true. While there cannot be small redundancies among the constraints imposed by the vertices, it is possible for there to be redundancies among large sets of these constraints. Such redundancies would increase the rate of the code. They might even result in codes in which the number of corrupt variables

is roughly proportional to the number of unsatisfied constraints. Such proportionality is not possible with redundancies among constraints, because without these redundancies there will be words that are far from codewords, but which satisfy all but one constraint. For more information about possible applications of such a construction, we refer the reader to Chapter 5 of [8].

REFERENCES

[1] N. Alon and F. R. K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72:15–19, 1988.

[2] R. G. Gallager. *Low Density Parity-Check Codes*. MIT Press, Cambridge, MA, 1963.

[3] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[4] G. A. Margulis. Explicit group theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24(1):39–46, July 1988.

[5] A. Nilli. On the second eigenvalue of a graph. *Discrete Math*, 91:207–210, 1991.

[6] M. Sipser and D. A. Spielman. Expander codes. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 566–576, 1994.

[7] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. preliminary version appeared as [6].

[8] D. A. Spielman. *Computationally efficient error-correcting codes and holographic proofs*. PhD thesis, M.I.T., May 1995. Available at `http://www-math.mit.edu/~spielman`.

[9] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 388–397, 1995.

[10] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. preliminary version appeared as [9].

[11] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, September 1981.