Expander Codes

Michael Sipser[*]
MIT
Sandia National Laboratories

Daniel A. Spielman[†]
MIT

Abstract

We present a new class of asymptotically good, linear error-correcting codes based upon expander graphs. These codes have linear time sequential decoding algorithms, logarithmic time parallel decoding algorithms with a linear number of processors, and are simple to understand. We present both randomized and explicit constructions for some of these codes. Experimental results demonstrate the extremely good performance of the randomly chosen codes.

1. Introduction

We present a new class of error correcting codes derived from expander graphs. These codes have the advantage that they can be decoded very efficiently. That makes them particularly suitable for devices which must decode cheaply, such as compact disk players and remote satellite receivers. We hope that the connection we draw between expander graphs and error correcting codes will stimulate research in both fields.

1.1. Error correcting codes

An error correcting code is a mapping from messages to codewords such that the mapping can be inverted even if a few characters of the codeword have been altered. We limit our discussion to the alphabet $\{0,1\}$, although our techniques can be applied to larger alphabets as well.

Formally, we define an error correcting code of block-length n, rate r, and minimum distance δ to be a function $f\colon \{0,1\}^{rn} \rightarrow \{0,1\}^n$ such that for all distinct $x, y \in \{0,1\}^{rn}$ we have that $d(f(x), f(y)) \geq \delta n$ where $d(u,v)$ is the Hamming distance, *i.e.*, the number of bits in which u and v differ.

Encoding is the process of mapping $x \in \{0,1\}^{rn}$ to $f(x)$, and decoding is the process of mapping $u \in \{0,1\}^n$ to the x which minimizes $d(f(x), u)$. The minimum distance δ implies that it is possible to correctly decode a corrupted version of $f(x)$ in which fewer than $\delta n/2$ bits have been changed.

A central problem of coding theory is to find families of codes for which δ and r remain constant as n grows large. Such families are called *asymptotically good.* Asymptotically good families are desirable because they enable one to encode data in fewer blocks of larger size, which decreases the probability that too many errors will occur in any one block. Of course as n grows, it becomes increasingly important to have encoding and decoding procedures that operate efficiently.

An important feature of our codes is that they allow very efficient decoding. They can be decoded in linear time sequentially as well as in logarithmic parallel time with a linear number of processors. As far as we know, this is the first asymptotically good class of codes with linear time sequential decoding. The parallel algorithm can be implemented as a circuit of depth $O(\log n)$ and size $O(n \log n)$, with modest constants. As for encoding, our class consists of linear codes and these always have quadratic time encoding algorithms which may be implemented to run in $O(\log n)$ parallel time with $O(n^2)$ processors. Our codes are especially suitable for applications in which one is more concerned with the cost of decoding than that of encoding.

1.2. Expander graphs

An expander graph is a graph in which every set of vertices has a surprisingly large set of neighbors. It is a perhaps surprising but nonetheless well known fact that expander graphs that expand by a constant fac-

[*]Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139. sipser@math.mit.edu. Partially supported by Air Force Contract F49620-92-J-0125, DARPA N00014-92-J-1799, and NSF grant 9212184CCR.

[†]Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139. spielman@math.mit.edu. Partially supported by the Fannie and John Hertz Foundation, Air Force Contract F49620-92-J-0125, DARPA N00014-92-J-1799, and NSF grant 9212184CCR.

tor, but which have only a linear number of edges, do exist. In fact, a simple randomized process will produce such a graph. Margulis [Mar88] and Lubotzky, Phillips, and Sarnak [LPS88] have a construction of linear-sized expander graphs which are almost as good as the randomly chosen graphs. We shall call a bipartite expander graph *balanced* if it has equal numbers of left and right nodes, otherwise it is *unbalanced.* Balanced bipartite expanders may be easily obtained from ordinary expanders. Good unbalanced expander graphs are easily obtained randomly, but appear more difficult to construct.

Our construction requires unbalanced bipartite expander graphs. We use a construction of unbalanced expander graphs due to Ajtai, Komlós and Szemerédi [AKS87] to give an explicit construction of a family of expander codes.

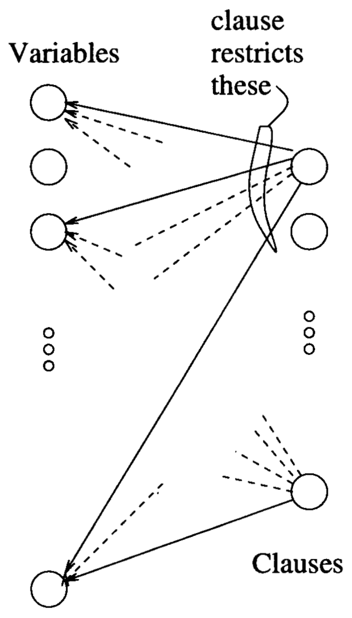
Expander graphs have been used before in a different way to construct error-correcting codes: Alon, Bruck, Naor, Naor and Roth [ABN+92] used expander graphs to construct asymptotically good families of error-correcting codes that lie above the Zyablov bound.

1.3. The construction

Our expander codes fall within the class of low-density parity-check codes introduced by Gallager [Gal63]. In fact, the random codes used by Gallager are almost identical to the codes we discuss in Section 3. Gallager proposed a decoding algorithm similar to that which we use in Section 3.2, but was unable to analyze it. Instead, by taking advantage of the high girth of randomly chosen graphs, he provided a weak analysis of a more complicated algorithm which showed that it would usually correct a linear number of errors, but it was only guaranteed to correct a much smaller number of errors. Zyablov and Pinsker [ZP76] presented a parallel algorithm for decoding Gallager's codes which they proved can, with high probability over the choice of the code, correct a linear number of errors in $O(\log n)$ time with $O(n)$ processors. Their algorithm is somewhat more complicated than the algorithm we present in Section 3.1.

Our construction of expander codes uses Tanner's approach [Tan81] of forming a code out of an unbalanced bipartite graph and smaller codes called subcodes. To obtain a code C having block length n we use a bipartite graph whose n left nodes represent variables corresponding to positions in the codewords. Each right node, which we call a *clause*, has an associated subcode on the variables which are its neighbors. The codewords of C are the assignments of the variables that induce a codeword in each clause. Like Gallager, Tanner's analysis rests on the girth of his graphs. Analysis based on girth seems insufficient to demonstrate that a family of codes is asymptotically good.

We form expander codes using low-degree unbalanced bipartite expander graphs and carefully chosen subcodes. Consider what happens when a few bits of a codeword are corrupted. Expansion guarantees that these bits will be contained in many clauses which will no longer have induced codewords. Most of the corrupted bits will appear in many clauses that "want" to correct them. Expansion also implies that there will be very few uncorrupted bits that many clauses want to flip. Thus if we update all bits as suggested by a vote of the clauses, we will reduce the number of corrupted bits.



1.4. Overview of remaining sections

In Section 2, we define expander codes precisely and prove that they are asymptotically good. In Section 3, we examine the performance of expander codes in which the subcodes consist of all words of even parity. These are the simplest imaginable subcodes, and they require very good expanders to produce good codes. We are not aware of explicit constructions of expanders of the quality needed to implement these codes; however, randomly chosen graphs suffice with high probability. We present both parallel and sequential decoding algorithms for these codes. In fact, we prove that if the sequential decoding algorithm works, then the underlying graph must be an expander. In Section 4, we present an explicit construction of families of expander codes as well as a generalization of our construction due to Noga Alon. In Section 5, we present the results of tests we performed on a randomly chosen code of the type described in Section 3. We conclude by presenting in Appendix A an examination of the expansion properties of randomly chosen graphs.

2. Expander codes

We being by introducing some facts about linear codes. A code of block-length n is called *linear* if the codewords are a subspace of $GF(2)^n$. Some advantages of these codes are:

- The zero vector is always a codeword.
- The codewords in a linear code of rate r form a vector space of dimension rn. The bits in the code can be divided into rn bits which represent the message and $(1-r)n$ redundant bits that are linear combinations of the message bits.
- Any linear code can be encoded in $O\left(n^2\right)$ time: the message is chosen by setting the rn message bits, and there is a $rn \times (1-r)n$ matrix such that the redundant bits can be computed by multiplying the vector of rn message bits by this matrix.
- The minimum distance of a linear code is equal to the minimum weight of a non-zero codeword (the weight of a codeword w is $d(0,w)$; note that $d(v,w) = d(0, w-v)$).

When we build expander codes, we will use linear codes as subcodes. This will insure that the expander codes will themselves be linear. The number of redundant bits in an expander code will be at most the sum of the number of redundant bits in each subcode. Our decoding algorithms are designed so that their correctness can be proved by demonstrating that, when given any input word of sufficiently low weight, they output the zero codeword.

Before explaining how to decode expander codes, we will first demonstrate that expander codes have large minimum distance.

Theorem 1. *Let B be a bipartite graph between n variables and $\frac{d}{c}n$ clauses that is d-regular on the variables and c-regular on the clauses. Let S be a code of block-length c, rate r and minimum distance ϵ. Let $\mathcal{C}(B,S)$ denote the code consisting of the settings of the variables such that, for every clause, the variables that are neighbors of that clause form a codeword of S. If B expands by a factor of more than $\frac{d}{c\epsilon}$ on all sets of size at most αn, then $\mathcal{C}(B,S)$ has rate $dr-(d-1)$ and minimum distance at least α.*

Proof: To obtain the bound on the rate of the code, we will count the number of linear constraints imposed by the clauses. Each clause induces $(1-r)c$ linear constraints. Thus, there are a total of

$$n\frac{d}{c}(1-r)c = dn(1-r)$$

linear constraints, which implies that there are at least $n(dr-(d-1))$ degrees of freedom.

To prove the bound on the minimum distance, we will show that there can be no non-zero codeword of weight less than αn. Let w be a non-zero word of weight at most αn and let V be the set of variables that are 1 in this word. There are $d|V|$ edges leaving the variables in V. The expansion property of the graph implies that these edges will enter more than $\frac{d}{c\epsilon}|V|$ clauses. Thus, the average number of edges per clause will be less than $c\epsilon$, so there must be some clause that is a neighbor of V, but which has a number of neighbors in V that is less than the clause's minimum distance. This implies that w cannot induce a codeword of S in that clause, so w cannot be a codeword in $\mathcal{C}(B,S)$. $\square$

3. A simple example

The simplest example of expander codes is obtained by letting B be a graph with expansion greater than $\frac{d}{2}$ on sets of size at most αn, and letting S be the code consisting of words of even parity. The parity-check matrix of the resulting code, $\mathcal{C}(B,S)$, is just the adjacency matrix of B. The code S has rate $\frac{c-1}{c}$ and minimum distance $\frac{2}{c}$, so $\mathcal{C}(B,S)$ has rate $1-\frac{d}{c}$ and minimum distance at least αn.

To obtain a code that we can efficiently decode, we will need even greater expansion. With greater expansion, small sets of corrupt variables will induce non-codewords in many clauses. By using these "unsatisfied" clauses, we will be able to determine which variables are corrupted. In the next two sections, we will explore how to use this phenomenon to decode the simple codes we just presented.

Unfortunately, we do not know of any explicit constructions of expander graphs with expansion greater than $\frac{d}{2}$. However, a randomly chosen regular graph will have expansion $d - 1 - o(1)$ on small sets with very high probability. In Appendix A, we will explore in more detail the expansion of randomly chosen graphs. The idea of choosing a regular graph at random and using the code induced by using the codes of even weight words as subcodes was first proposed by Gallager [Gal63]. One of his suggestions for how to decode these codes is the same as the decoding algorithm that we analyze in Section 3.2.

3.1. Parallel decoding

Theorem 2. *Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ clauses of degree c such that all sets X of at most $\alpha_0 n$ variables have more than $(\frac{3}{4}+\epsilon)d|X|$ neighbors, for some $\epsilon > 0$. Let $C(B)$ be the code consisting of those settings of the variables that cause every clause to have parity zero. Then, $C(B)$ has rate at least $(1-\frac{d}{c})$ and there exists an algorithm that will correct up to any $\alpha < \frac{\alpha_0(1+4\epsilon)}{2}$ fraction of errors after $\log_{1/(1-2\epsilon)}(\alpha n)$ parallel decoding rounds, where each round requires constant time.*

Proof: We will say that a word satisfies a clause if the word induces even parity in that clause. We decode this code using a parallel decoding algorithm. In each decoding round,

1. Each clause computes the parity of the variables that are its neighbors.
2. Each variable checks if more than half of its neighbors are unsatisfied. If they are, then the variable flips it value.

Assume that the algorithm is presented with a word of weight at most αn, where $\alpha < \frac{\alpha_0(1+4\epsilon)}{2}$. We will refer to the variables that are 1 as the *corrupted variables*, and we will let S denote this set of variables. We will show that after one decoding round, the algorithm will produce a word that has at most $(1-2\epsilon)\alpha n$ corrupted variables.

To this end, we will examine the sizes of F, the set of corrupted variables that fail to flip in one decoding round, and C, the set of variables that were originally uncorrupt, but which become corrupt after one decoding round. After one decoding round, the set of corrupted variables will be $C \cup F$. Define $v = |S|$, and set ϕ, γ, and δ so that $|F| = \phi v$, $|C| = \gamma v$, and $|N(S)| = \delta d v$. By expansion, $\delta > \frac{3}{4} + \epsilon$. To prove the theorem, we will show that

$$\phi + \gamma < \frac{\frac{1}{4}}{\frac{1}{4}+\epsilon} < 1 - 2\epsilon.$$

We will first show that

$$\phi < 4 - 4\delta.$$

We can bound the number of neighbors of S by observing that each variable in F must share at least half of its neighbors with other corrupted variables. Thus, each variable in F can account for at most $\frac{3}{4}d$ neighbors, and, of course, each variable in $S \setminus F$ can account for most d neighbors, which implies

$$\begin{aligned} \delta d v &\leq \frac{3}{4} d\phi v + d(1-\phi)v \quad \Rightarrow \\ \phi &\leq 4 - 4\delta. \end{aligned}$$

We now show that

$$\gamma < \frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}.$$

Assume by way of contradiction that this is false. Let C' be a subset of C of size $\frac{\delta-(\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}v$. Each variable in C' must have at least $\frac{d}{2}$ edges that land in clauses that are neighbors of S. Thus, the total number of neighbors of $C' \cup S$ is at most

$$\frac{d}{2}|C'| + \delta d v.$$

But, because the set $C' \cup S$ has size at most

$$\begin{aligned} &\left(1 + \frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}\right)\left(\frac{1+4\epsilon}{2}\right)\alpha_0 n \\ \leq &\left(\frac{\frac{1}{2}}{\frac{1}{4}+\epsilon}\right)\left(\frac{1+4\epsilon}{2}\right)\alpha_0 n = \alpha_0 n, \end{aligned}$$

it must have expansion greater than $d(\frac{3}{4}+\epsilon)$, which implies that

$$\begin{aligned} \frac{d}{2}|C'| + \delta d v &= \frac{d}{2}\left(\frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}\right)v + \delta d v \\ &> \left(\frac{3}{4}+\epsilon\right) d \left(1 + \frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}\right) v. \end{aligned}$$

After simplifying this inequality, we find that it is a contradiction. Thus, we find

$$\phi + \gamma \quad < \quad 4 - 4\delta + \frac{\delta - (\frac{3}{4}+\epsilon)}{\frac{1}{4}+\epsilon}$$

$$\begin{aligned}
&= \frac{1-(\frac{3}{4}+\epsilon)+4\epsilon-4\epsilon\delta}{\frac{1}{4}+\epsilon} \\
&= \frac{\frac{1}{4}-\epsilon+4\epsilon(1-\delta)}{\frac{1}{4}+\epsilon} \\
&< \frac{\frac{1}{4}-\epsilon+4\epsilon(\frac{1}{4})}{\frac{1}{4}+\epsilon} \\
&= \frac{\frac{1}{4}}{\frac{1}{4}+\epsilon}.
\end{aligned}$$

□

We note that this algorithm can be implemented by a circuit of size $O(n\log n)$ and depth $O(\log n)$.

3.2. Sequential decoding

We will now show how to decoded these codes sequentially in linear time.

We assume that the graph on which the code is based has been presented to the decoding algorithm as two linked lists of clusters of pointers. The first linked list should contain a cluster of pointers for each variable and the second should contain a cluster of pointers for each clause. The pointers associated with a variable should point to the clauses that are its neighbors, and the pointers associated with a clause should point to the variables that it contains.

The sequential decoding algorithm has two phases: a set-up phase which organizes the data, and a decoding phase which is repeated until the word is decoded, or the algorithm gives up. We will continue to say that a clause is satisfied if the variables it contains have parity zero. Otherwise, we will say that it is unsatisfied.

Set-up:

1. Walk down the list of clauses. For each clause, compute the parity of the variables it contains.
2. Initialize lists $L_0, \ldots, L_d$.
3. Walk down the list of variables. For each variable, count the number of unsatisfied clauses in which it appears. If this number is i, put the variable in list L_i.

Decoding:

1. Choose the greatest i such that L_i is not empty. If $i = 0$, then exit and report "decoded". If $0 < i \leq \frac{d}{2}$, then exit and report "could not decode".
2. Choose the variable, v, that is at the head of list L_i.
3. Flip the value of v. For each clause that contains v, flip the parity of that clause. For each variable w that is in a clause that contains v, adjust the count of the number of unsatisfied clauses that contain w, remove w from its current list, and put it at the tail of the appropriate list.
4. Return to step 1.

Proposition 3. *The above algorithm runs in linear time for all constant-degree graphs and all input words.*

Proof: Because the out-degree of every variable and clause is constant, the set-up phase requires linear time. Observe that every time a variable is flipped in a decoding phase, the number of clauses that have parity 1 decreases. Thus, the decoding loop can be executed at most once for each clause in the graph. □

Theorem 4. *Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ clauses of degree c such that all sets X of at most αn variables have at least $(\frac{3}{4}+\epsilon)d|X|$ neighbors, for some $\epsilon > 0$. Let $C(B)$ be the code consisting of those settings of the variables that cause every clause to have parity zero. Then the sequential decoding algorithm will correct up to an $\alpha/2$ fraction of errors while executing the decoding loop at most $d\alpha n/2$ times.*

Proof: We will say that the decoding algorithm is in state (v, u) if v variables are corrupted and u clauses are unsatisfied. We view u as a potential associated with v. Our goal is to demonstrate that the potential will eventually reach zero. To do this, we will show that if the decoding algorithm begins with a word of weight at most $\alpha n/2$, then, at every step, there will be some variable with more unsatisfied neighbors than satisfied neighbors.

First, we consider what happens when the algorithm is in a state (v, u) with $v < \alpha n$. Let s be the number of satisfied neighbors of the corrupted variables. By the expansion of the graph, we know that

$$u + s \geq \left(\frac{3}{4}+\epsilon\right) dv.$$

Because each satisfied neighbor of the corrupted variables must share at least two edges with the corrupted variables, and each unsatisfied neighbor must have at least one, we know that

$$dv \geq u + 2s.$$

By combining these two inequalities, we obtain

$$s \leq \left(\frac{1}{4} - \epsilon\right) dv \qquad \text{and} \qquad u \geq \left(\frac{1}{2} + 2\epsilon\right) dv.$$

Since each unsatisfied clause must share at least one edge with a corrupted variable, and since there are only dv edges leaving the corrupted variables, we see that at least a $(\frac{1}{2} + 2\epsilon)$ fraction of the edges leaving the corrupted variables must enter unsatisfied clauses. This implies that there must be some corrupted variable such that a $(\frac{1}{2} + 2\epsilon)$ fraction of its neighbors are unsatisfied. Of course, this does not mean that the decoding algorithm will decide to flip a corrupted variable.

However, it does mean that the only way that the algorithm could fail to decode is if it flips so many uncorrupt variables that v becomes greater than αn. Assume by way of contradiction that this happens. Then there must be some time at which v equals αn. Because v was initially at most $\alpha n/2$, we know that at least $\alpha n/2$ flips must have occurred, and that each flip can only decrease the potential u. This would imply that the algorithm was in a state $(\alpha n, u)$, where $u < \frac{d}{2}\alpha n$, because u was initially at most $\frac{d}{2}\alpha n$. But, this would contradict the analysis above. □

We like to represent this analysis by the following figure. The argument first shows that there is a "hill" in the state-space that the state can never occupy. At every step the state must descend, whether or not it moves left or right. The reason that the decoding always works is that if the algorithm starts out far enough to the left, then it can never get over the hill.

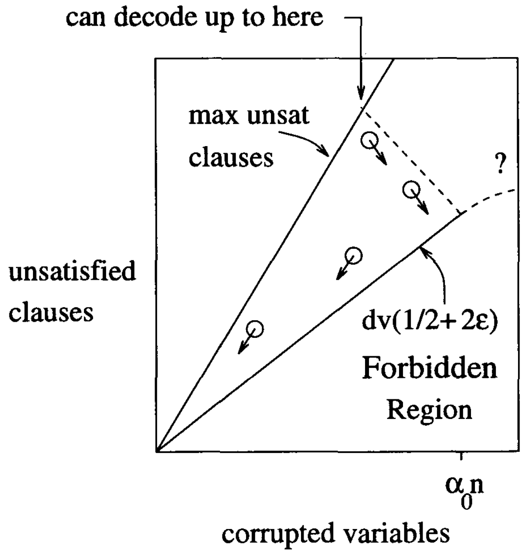
We note that it is possible to improve the constants in this analysis by taking into account the fact that after each decoding step, the number of corrupted variables actually decreases by at least $2\frac{u}{v} - d$.

We will now show that, not only does the expansion of B imply that the decoding algorithm works, but if the sequential decoding algorithm works, then the graph B must be an expander.

Theorem 5. *Let B be a bipartite graph between n variables of degree d and $\frac{d}{c}n$ clauses of degree c such that the sequential decoding algorithm presented at the beginning of this section successfully decodes all sets of at most αn errors in the code $C(B)$. Then, all sets of αn variables must have at least*

$$\alpha n \left(1 + \frac{2\frac{d-1}{2c}}{3 + \frac{d-1}{2c}}\right)$$

neighbors in B.



Proof: We will first deal with the case in which d is even. In this case, every time a variable is flipped, the number of unsatisfied clauses decreases by at least 2. Consider the performance of the decoding algorithm on a word of weight αn. Because the algorithm stops when the number of unsatisfied clauses reaches zero, the algorithm must decrease the number of unsatisfied clauses by at least $2\alpha n$ as it corrects the αn corrupted variables. Thus, every word of weight αn must cause at least $2\alpha n$ clauses to be unsatisfied, so every set of αn variables must have at least $2\alpha n$ neighbors. Because we assume that $c > d$,

$$2 > 1 + \frac{2\frac{d-1}{2c}}{3 + \frac{d-1}{2c}},$$

and we are done with the case in which d is even.

When d is odd, we can only guarantee that the number of unsatisfied clauses will decrease by 1 at each iteration. This means that every set of αn variables must induce at least αn unsatisfied clauses. Alone, this is insufficient to demonstrate expansion by a factor greater than 1. However, let us consider what must happen for the algorithm to be in a state in which αn variables are corrupted, but there is no variable that the decoding algorithm can flip that will cause the number of unsatisfied clauses to decrease by more than 1. This means that each corrupted variable has at least $\frac{d-1}{2}$ of its edges in satisfied clauses. Because each satisfied clause can have at most c incoming edges, this implies that there must be at least $\alpha n\frac{d-1}{2c}$ satisfied neighbors of the αn variables. Thus, the set of αn variables must have at least $\alpha n(1+\frac{d-1}{2c})$ neighbors.

On the other hand, if the algorithm decreases the number of unsatisfied clauses by more than 1, then it must decrease the number by at least 3. For some word of weight αn, assume that the algorithm flips $\beta \alpha n$ variables before it flips a variable that decreases the number of unsatisfied clauses by only 1. The original set of αn variables must have had at least

$$3\beta\alpha n + (1-\beta)\alpha n$$

neighbors. On the other hand, once the algorithm flips a variable that causes the number of unsatisfied clauses to decrease by 1, we can apply the bound of the previous paragraph to see that the variables must have at least

$$(1-\beta)\alpha n\left(1+\frac{d-1}{2c}\right)$$

neighbors. We note that this bound is strictly decreasing in β, while the previous bound is strictly increasing in β, so the lower bound that we can obtain on the expansion occurs when β is chosen so that

$$\begin{aligned}
3\beta\alpha n + (1-\beta)\alpha n &= \left(1+\frac{d-1}{2c}\right)(1-\beta)\alpha n \Rightarrow \\
1+2\beta &= (1-\beta)\left(1+\frac{d-1}{2c}\right) \Rightarrow \\
\beta\left(3+\frac{d-1}{2c}\right) &= \frac{d-1}{2c} \Rightarrow \\
\beta &= \frac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}.
\end{aligned}$$

When we plug β back in, we find that the set of αn variables must have at least

$$\begin{aligned}
&\alpha n\left(3\left(\frac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right)+\left(1-\frac{\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right)\right) \\
&= \alpha n\left(\frac{3+3\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right) = \alpha n\left(1+\frac{2\frac{d-1}{2c}}{3+\frac{d-1}{2c}}\right)
\end{aligned}$$

neighbors. □

4. Explicit constructions

In order to construct expander codes, we will need some explicit construction of unbalanced expander graphs. Our first construction is to take some d-regular expander graph on n vertices, $G_{n,d}$, and derive from it an unbalanced bipartite graph $B_{n,d}$ between a set of $\frac{d}{2}n$ vertices that are identified with the edges of $G_{n,d}$ and a set of n vertices that are identified with the vertices of $G_{n,d}$. The edges in $B_{n,d}$ will go from edges of $G_{n,d}$ to the vertices that are the edge's endpoints. That is, a vertex of $B_{n,d}$ that is associated with the edge (u,v) of $G_{n,d}$ will have edges to the vertices of $B_{n,d}$ that are associated with the vertices u and v of $G_{n,d}$. To use this graph in the formation of an expander code, the set of $\frac{d}{2}n$ vertices will become the variables and the set of n vertices will become the clauses.

The careful reader might observe that the resulting graph is 2-regular at the variables, and thus cannot have expansion by a factor greater than 1 from the variables to the clauses! The point is that the graph is very unbalanced yet doesn't shrink too much, which is in the spirit of ordinary expansion. This property is sufficient to enable us to construct expander codes.

For our initial family of expander graphs, $G_{n,d}$, we will use the expanders constructed by Lubotzky, Phillips and Sarnak [LPS88] and, independently, by Margulis [Mar88]:

Theorem 6 ([LPS88, Mar88]). *For every pair of primes p, q congruent to 1 modulo 4 such that p is a quadratic residue modulo q, there is an easily computed $(p+1)$-regular graph with $q(q^2-1)/2$ vertices such that the second-largest eigenvalue of the graph is at most $\frac{2\sqrt{p}}{p+1}$.*

This theorem will supply us with an infinite number of good expander graphs. To examine the expansion properties of the $B_{n,d}$'s that we derive from these graphs, we will make use of a lemma due to Alon and Chung:

Lemma 7 ([AC88]). *Let $G_{n,d}$ be a d-regular graph on n nodes with second-largest eigenvalue bounded by λ. Let S be a subset of the vertices of $G_{n,d}$ of size γn. Then, the number of edges contained in the subgraph induced by S in $G_{n,d}$ is at most*

$$\frac{n}{2}d\left(\gamma^2+\lambda\gamma(1-\gamma)\right).$$

In particular, we will want the following corollary.

Corollary 8. *Let $G_{n,p+1}$ be one of the graphs known to exist by Theorem 6 and let $B_{n,p+1}$ be the derived bipartite graph. Let V be a subset of the variables of $B_{n,p+1}$ of size*

$$\frac{n(p+1)}{2}\left(\gamma^2+\frac{2\sqrt{p}}{p+1}(\gamma-\gamma^2)\right).$$

Then, the number of clauses that are neighbors of V is at least γn.

We will use these graphs to derive an infinite family of good codes from a particular good code.

Theorem 9. *Let S be a linear code with rate r, minimum distance ϵ and block-length $p+1$ where p is a prime congruent to 1 modulo 4. Then, there is an easily constructible infinite family of linear codes of rate $2r-1$ and minimum distance at least*

$$\delta = \gamma^2 + (\gamma - \gamma^2)\frac{2\sqrt{p}}{p+1}, \text{ where}$$

$$\gamma(p+1) + (1-\gamma)2\sqrt{p} \leq \epsilon(p+1).$$

Moreover, if

$$\frac{\epsilon^2}{768} > \frac{\epsilon}{16} \cdot \frac{2\sqrt{p}}{(p+1)},$$

then there exists a parallel decoding algorithm that will decode up to a $\frac{\epsilon^2}{64}$ fraction of errors in $O(\log n)$ rounds, each of which takes constant time. This algorithm can be simulated on a sequential machine in linear time.

Proof: Our family of codes will consist of the codes $\mathcal{C}(B_{n,p+1}, S)$ where $B_{n,p+1}$ is a graph derived from one of the graphs $G_{n,p+1}$ known to exist by Theorem 6.

Corollary 8 implies that $B_{n,p+1}$ expands by a factor of at least

$$\frac{\gamma n}{\frac{n(p+1)}{2}\left(\gamma^2 + \frac{2\sqrt{p}}{p+1}(\gamma - \gamma^2)\right)}$$

on sets of variables of size

$$\frac{n(p+1)}{2}\left(\gamma^2 + \frac{2\sqrt{p}}{p+1}(\gamma - \gamma^2)\right).$$

The bounds on the rate and minimum distance of $\mathcal{C}(B_{n,p+1}, S)$ then follow immediately from Theorem 1.

Our decoding algorithm will repeat the following two operations:

1. In parallel, for each clause, if the setting of the variables in that clause is within $\frac{\epsilon}{4}$ of a codeword, then send a "flip" message to every variable that needs to be flipped to obtain that codeword.

2. In parallel, every variable that receives the message "flip", flips its value.

Let X be a set of at most $\alpha n \frac{p+1}{2}$ variables. We will show that if one phase of the decoding algorithm is given a word that is 1 for all $v \in X$, and 0 elsewhere, then it will output a word whose weight is less by a constant factor.

We will say that a clause is *confused* if it sends a "flip" message to a variable that is not in X. In order for this to happen, the clause must have at least $\frac{3}{4}\epsilon(p+1)$ variables of X as neighbors. Each variable of X can be a neighbor of two clauses, so there can be at most

$$\frac{\alpha \frac{p+1}{2} n \cdot 2}{\frac{3}{4}\epsilon(p+1)} = \frac{4\alpha n}{3\epsilon}$$

confused clauses. Each of these can send at most $\frac{\epsilon}{4}(p+1)$ "flip" signals, so at most

$$\frac{4\alpha n}{3\epsilon} \cdot \frac{\epsilon}{4}(p+1) = n\frac{p+1}{2} \cdot \frac{2\alpha}{3}$$

variables not in X will receive "flip" signals.

We will call a clause *unhelpful* if it has more than $\frac{\epsilon}{4}$ neighbors in X. A variable in X will flip unless both of its neighbors are in unhelpful clauses. There are at most

$$\frac{\alpha \frac{p+1}{2} n \cdot 2}{\frac{1}{4}\epsilon(p+1)} = \frac{4\alpha n}{\epsilon}$$

unhelpful clauses, which by Lemma 7 can have at most

$$\frac{n(p+1)}{2}\left(\left(\frac{4\alpha}{\epsilon}\right)^2 + \left(\frac{4\alpha}{\epsilon}\right)\frac{2\sqrt{p}}{p+1}\right)$$

edges among them. So, the weight of the output codeword must decrease by at least

$$\frac{n(p+1)}{2}\left(\alpha - \frac{2\alpha}{3} - \left(\frac{4\alpha}{\epsilon}\right)^2 - \frac{4\alpha}{\epsilon} \cdot \frac{2\sqrt{p}}{p+1}\right)$$

The worst case is when α is maximized. If we set $\alpha = \frac{\epsilon^2}{64}$, then we obtain

$$\frac{n(p+1)}{2}\left(\frac{\epsilon^2}{768} - \frac{\epsilon}{16} \cdot \frac{2\sqrt{p}}{p+1}\right).$$

Thus the weight of the output codeword must decrease by a constant factor.

It is not very difficult to simulate this algorithm in linear time on a sequential machine. The key is to realize that in each round, the number of "interesting" clauses and variables is decreasing by a constant factor, and the algorithm only needs to look at the "interesting" clauses and variables in each round. □

To obtain an infinite family of good codes, we will use for the subcode a good code known to exist by the Gilbert-Varshamov bound:

Corollary 10. *For all ϵ such that $1 - 2H(\epsilon) > 0$, where $H(x) = -x\log_2 x - (1-x)\log_2(1-x)$ is the binary entropy function, there exists an easily constructible family of linear codes with rate $1 - 2H(\epsilon)$*

and minimum distance arbitrarily close to ϵ^2 such that the parallel decoding algorithm will decode up to a $\frac{\epsilon^2}{64}$ fraction of errors in $O(\log n)$ parallel decoding rounds. This decoding algorithm can be simulated in linear time on a sequential machine.

Proof: As we let $p+1$ grow large in Theorem 9, the $\frac{2\sqrt{p}}{p+1}$ terms tend to zero. Moreover, the Gilbert-Varshamov bound [MS77] tells us that, for sufficiently large block-length $p+1$, there are linear codes of rate r and minimum-distance ϵ if $r \leq 1-H(\epsilon)$. We use such a code as the subcode in the construction of Theorem 9. □

It is interesting to note that the above construction doesn't actually require the expanders of Theorem 6. Any infinite family of expanders in which the second-largest eigenvalues tend to zero as the degree grows large will suffice.

4.1. Alon's generalization

Noga Alon has pointed out that the "edge to vertex" construction that we use to construct expander codes is a special case of a construction due to Ajtai, Komlós and Szemerédi [AKS87]. They construct unbalanced expander graphs from regular expander graphs by identifying the large side of their graph with all paths of length k in the original graph and the small side of their graph with the vertices of the original graph. A node identified with a path is connected to the nodes identified with the vertices along that path. The construction used in Theorem 9 is the special case in which $k=1$.

Alon suggested that the codes produced by applying the technique of Theorem 9 to this more general class of graphs can be analyzed by applying the following theorem of Alon, Feige, Wigderson and Zuckerman [AFWZ]

Theorem 11 ([AFWZ]). *Let $G_{n,d}$ be a d-regular graph on n nodes with second-largest eigenvalue bounded by λ. Let S be a subset of the vertices of $G_{n,d}$ of size γn. Then, the number of paths of length k contained in the subgraph induced by S in $G_{n,d}$ is at most*

$$\gamma n d^k \left(\gamma + \lambda(1-\gamma)\right)^k .$$

We should note that the difference of a factor of two between Theorem 11 and Lemma 7 is due to the fact that in Theorem 11 each path is being counted twice: once for each end-point. This difference does not substantially effect the analysis.

Theorem 12. *For all integers $k \geq 2$ and all ϵ such that $1-kH(\epsilon) > 0$, there exists an easily constructible family of linear codes with rate $1-kH(\epsilon)$ and minimum distance arbitrarily close to $\epsilon^{1+1/(k-1)}$ such that the parallel decoding algorithm will decode up to some constant fraction of errors in $O(\log n)$ parallel decoding rounds. Moreover, this algorithm can be implemented to run in linear time on a sequential machine.*

Proof: [Sketch] As in Theorem 9, we will use the graphs known to exist by Theorem 6. We will form a code by using the graph in which all paths of length k in $G_{n,p+1}$ are mapped to the vertices that they contain and by using a Gilbert-Varshamov good code as the subcode. The Gilbert-Varshamov bound implies that for sufficiently large block-length $p+1$, there exist linear codes of rate r and minimum distance ϵ provided that $r \leq 1-H(\epsilon)$. Moreover, as $p+1$ grows large, the terms involving λ in Theorem 11 goes to zero. If this term were zero, then we would know that every set containing an $\epsilon^{\frac{k+1}{k}}$ fraction of the variables has at least an $\epsilon^{\frac{1}{k}}$ fraction of the clauses as neighbors. Because there are nd^k variables of degree k and n clauses of degree kd^k, Theorem 1 would imply that no word of weight up to $\epsilon^{\frac{k+1}{k}}$ can be a codeword. However, because λ never actually reaches zero, we can only come arbitrarily close to this bound.

These codes can be decoded by techniques similar to those used in Theorem 9. □

5. Experimental results

We implemented the parallel and sequential decoding algorithms for the parity-based code described in Section 3. We randomly generated a bipartite graph between 20,000 variables of degree 5 and 10,000 clauses of degree 10. We implemented the parallel decoding algorithm so that it began decoding with threshold 4 and, when it stopped making progress at threshold 4, it switched to threshold 3. We then tested the performance of the decoding algorithms on randomly generated error words of various weights. The results of the tests are summarized in these tables:

To put these results in perspective, one should realize that a simple sphere-packing argument shows that no $\frac{1}{2}$-rate code of block length 20,000 should be able to correct on average more than 2,205 corrupted variables.

Parallel Decoding			
number of variables corrupted	number of tests performed	number of times succeeded	average number of rounds
600	40,000	40,000	13.6
700	135,000	134,999	13.5
730	30,000	29,974	14.3
750	5,000	4,908	15.2

Sequential Decoding			
number of variables corrupted	number of tests performed	number of times succeeded	average number of steps
800	100,000	100,000	942
850	25,000	24,906	1088
900	25,000	17,376	1265
950	25,000	1,031	1385

6. Acknowledgements

We would like to thank the many people who made helpful comments after reading an early draft of this paper. Among them, we would especially like to thank Noga Alon, Oded Goldreich and Shanghua Teng. We would also like to thank Noga Alon for supplying us with the material that appears in Section 4.1 and for Theorem 14.

References

[ABN+92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ron M. Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38(2):509–516, March 1992.

[AC88] Noga Alon and F.R.K. Chung. Explicit construction of linear sized tolerant networks. *Discrete Mathematics*, 72:15–19, 1988.

[AFWZ] Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Computational Complexity*. "To appear".

[AKS87] Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in logspace. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 132–139, 1987.

[AS92] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* John Wiley & Sons, New York, 1992.

[BW74] W. E. Bleick and Peter C. C. Wang. Asymptotics of Stirling numbers of the second kind. *Proceedings of the American Mathematical Society*, 42(2):575–580, Feb 1974.

[Gal63] R. G. Gallager. *Low Density Parity-Check Codes.* MIT Press, Cambridge, MA, 1963.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[Mar88] 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.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North Holland, Amsterdam, 1977.

[Sta86] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Wadsworth & Brooks/Cole, Monterey, CA, 1986.

[Tan81] R. Michael Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, September 1981.

[ZP76] V. V. Zyablov and M. S. Pinsker. Estimation of the error-correction complexity of Gallager low-density codes. *Problems of Information Transmission*, 11(1):18–28, May 1976.

A. The expansion of random graphs

In order to understand how good expander codes can be, we need to know what quality expander graphs we can produce.

We first upper bound the quality of an expander by its expected expansion on a randomly chosen set.

Theorem 13. *Let B be a bipartite graph between n variables and $\frac{d}{c}n$ clauses that is d-regular at the variables and c-regular at the clauses. For all $0 < \alpha < 1$, there exists a set of αn variables with at most*

$$n\frac{d}{c}(1-(1-\alpha)^c) + O(1) \text{ neighbors.}$$

Proof: Choose a set of αn variables uniformly at random. Now, consider the probability that a given clause is not a neighbor of the set of variables. A clause has c neighbors, each of which is in the chosen set with probability α. Thus, the probability that the clause is not a neighbor is

$$\prod_{i=0}^{c-1} \frac{n-\alpha n-i}{n},$$

which tends to $(1-\alpha)^c$ as n grows large. This implies that the expected number of non-neighbors tends to $n\frac{d}{c}(1-\alpha)^c$. $\square$

Noga Alon has pointed out to us that this simple upper bound becomes tight as d grows large.

To choose a random regular bipartite graph, we first choose a random matching between dn "left" nodes and dn "right" nodes. We collapse sets of d left nodes to form the n variables and we collapse sets of c right nodes to form the $\frac{d}{c}n$ clauses. It is possible that this graph will have multiedges that should be thrown away, but this does not hurt the lower bound on the expansion of this graph.

Noga Alon provided us with the following simple proof that such a graph is a good expander on large sets with high probability.

Theorem 14. *Let B be a bipartite graph chosen at random according to the distribution described above. Then, for all $0 < \alpha < 1$, with high probability all sets of αn variables in B have at least*

$$n\left(\frac{d}{c}(1-(1-\alpha)^c) - \sqrt{2dH(\alpha)/\log_2 e}\right)$$

neighbors, where $H(\cdot)$ is the binary entropy function.

Proof: First, we fix a set of αn variables, V, and estimate the probability that V's set of neighbors has size less than

$$n\frac{d}{c}(1-(1-\alpha)^c) - \gamma.$$

The probability that a given clause is a neighbor of V is at least $1-(1-\alpha)^c$. Thus, the expected number of neighbors of V is at least $n\frac{d}{c}(1-(1-\alpha)^c)$. To bound the probability that the size of the set of neighbors deviates from this, we will form a Martingale (See [AS92]).

Each node in V will have d outgoing edges. We will consider the process in which the destinations of these edges are revealed one at a time. We will let X_i be the random variable equal to the expected size of the set of neighbors of V given that the first i edges leaving V have been revealed. $X_1, \ldots, X_{dn}$ form a Martingale such that

$$|X_{i+1} - X_i| \leq 1,$$

for all $0 \leq i < nd$. Thus, by Azuma's Inequality (See [AS92]),

$$\text{Prob}[E[X_{nd}] - X_{nd} > \lambda\sqrt{nd}] < e^{-\lambda^2/2}.$$

Moreover, X_{nd} is the expected size of the set of neighbors of V given that *all* edges leaving V have been revealed, which is exactly the size of the set of neighbors of V.

Since there are only $\binom{n}{\alpha n}$ choices for the set V, it suffices to choose λ so that

$$\binom{n}{\alpha n}e^{-\lambda^2/2} << 1.$$

Let $H(\cdot)$ denote the binary entropy function ($H(x) = -x\log_2 x - (1-x)\log_2(1-x)$). It suffices to choose λ to satisfy

$$\begin{aligned} nH(\alpha)/\log_2 e &< \lambda^2/2 \quad \Rightarrow \\ \sqrt{2nH(\alpha)/\log_2 e} &< \lambda. \end{aligned}$$

$\square$

We note that it is possible to prove a slightly better lower bound by replacing the Martingale argument with a tighter argument. We begin by considering a slightly different distribution on graphs in which the d neighbors of each variable are chosen uniformly at random among the $\frac{d}{c}n$ clauses. We will then derive a very precise estimate of the probability that a set of αn variables has at most $\beta\frac{d}{c}n$ neighbors. It is not difficult to show that the probability that a set of αn variables has at most $\beta\frac{d}{c}n$ neighbors is strictly greater in this distribution than in the original distribution, so an upper bound here will serve as an upper bound for the original distribution. It is also interesting to note that the proof of Theorem 14 only distinguishes between these two distributions in the computation of the expectation, and this difference is very slight.

We now return to the problem of computing the probability that the set of αn variables has at most $\beta\frac{d}{c}n$ neighbors. We note that it is possible to compute the probability that some set C is exactly the set of neighbors of the set of αn variables. The number of ways of choosing the destinations of the $d\alpha n$ edges leaving the set of variables is $\left(\frac{d}{c}n\right)^{d\alpha n}$. The number of ways of choosing these edges so that C is exactly their set of endpoints is equal to the number of surjective functions from a set of size $d\alpha n$ to a set of size $|C|$, which is the product of a factorial and a Stirling number of the second kind: $|C|! \cdot S(d\alpha n, |C|)$ (See [Sta86]). An approximation of the Stirling numbers of the second kind that will give the correct exponential term for the ranges of values we are interested in may be found in [BW74]. Unfortunately, this approximation requires solving an equation, so we cannot present a closed form for the estimate. Having found this estimate, it suffices to sum over all sets of clauses C of size $\beta\frac{d}{c}n$, and plug the result into the proof of Theorem 13 in place of the bound obtained by the Martingale argument.