Linear-Time Encodable and Decodable Error-Correcting Codes

Daniel A. Spielman*

Abstract

We present a new class of asymptotically good, linear error-correcting codes. These codes can be both encoded and decoded in linear time. They can also be encoded by logarithmic-depth circuits of linear size and decoded by logarithmic depth circuits of size $O(n \log n)$. We present both randomized and explicit constructions of these codes.

Keywords: asymptotically good error-correcting code; linear-time; expander graph, superconcentrators.

1. Introduction

We construct an asymptotically good family of linear error-correcting codes that can be encoded and decoded in linear time. These codes can also be encoded by linear-size circuits of logarithmic depth, and decoded by circuits of logarithmic depth and size $O(n \log n)$. Our construction builds on a construction of linear-time decodable error-correcting codes by Sipser and Spielman [SS96], which also appears in this issue. We will occasionally refer the reader to [SS96] for information on the subtleties of linear-time computation. Except for these few references, the present paper should be self-contained. However, as many of the constructions and proofs in the present paper have simpler analogues in [SS96], the reader might prefer to read that paper before reading this one.

When discussing linear time, one must be careful to specify exactly what is meant. When we say that our algorithms run in linear time, we mean that they can easily be implemented to run in linear time on a Pointer Machine (see [KU58] or [Sch80]) or on a RAM in the uniform cost model (see [AHU74]). They can also be modified to run in linear time on a RAM in the logarithmic cost model by using a technique from Section 7 of [SS96]. For a more detailed description of the models that we consider, we refer the reader to Section 2 of [SS96].

*Dept. of Computer Science, U. C. Berkeley, Berkeley, CA 94720. `spielman@math.mit.edu`. Partially supported by an NSF Postdoc. This work was also supported by the Fannie and John Hertz Foundation, Air Force Contract F49620-92-J-0125, DARPA N00014-92-J-1799, and NSF grant 9212184CCR. Errata to this paper will appear at `http://theory.lcs.mit.edu/~spielman`

Suggestions for how to analyze the complexity issues special to error-correcting codes appeared in the work of Savage [Sav69, Sav71] and Bassalygo, Zyablov, and Pinsker [BZP77]. Bassalygo, Zyablov, and Pinsker point out that the complexity of encoding and decoding should be divided into two parts: The first complexity is that of building the encoders and decoders. This process may involve operations that only need to be performed once, such as computing a check matrix from a generator matrix. The second complexity is the time it takes the encoders and decoders to encode and decode individual transmissions. In our constructions, we devote a polynomial amount of computation to produce encoders and decoders that run in linear time.

It is somewhat tricky to compare the complexity of our algorithms with those for other codes because, historically, the presentation of most encoding and decoding algorithms have not used such fine models of computation or have not dealt with asymptotic complexity. However, we can point to constructions of Justesen [Jus76] and Sarwate [Sar77] which use efficient implementations of Polynomial GCD [AHU74] to show that certain Reed-Solomon and Goppa codes can be encoded in $O(n \log n \log\log n)$ time and decoded in time $O\left(n \log^2 n \log\log n\right)$. While these codes are not necessarily asymptotically good, one can concatenate them with good codes to obtain asymptotically good codes with similar encoding and decoding times. Codes that have had more efficient algorithms for one of these operations seem to have suffered in the other. Gelfand, Dobrushin, and Pinsker [GDP73] presented randomized constructions of asymptotically good codes that could be encoded in linear time. However, they did not suggest algorithms for decoding their codes, and we suspect that a polynomial-time algorithm would be difficult to find. On the other hand, Sipser and Spielman [SS96] presented explicit constructions of error-correcting codes that can be decoded in linear time, but which seem to require quadratic time for encoding. Our constructions and proofs rely heavily on ideas introduced by Sipser and Spielman, and our encoding circuits have some resemblance to those of Gelfand, Dobrushin, and Pinsker.

An important step along the path to our error-correcting codes is the introduction of *error-reduction codes*. An error-reduction code is weaker than an error-correcting code. Loosely speaking, an error-reduction code is a systematic code with a decoder that can remove most of the errors from the message bits of a partially corrupted word. In Section 2, we precisely define error-reduction codes and present a novel recursion that enables us to build asymptotically good linear-time encodable and decodable error-correcting codes from linear-time encodable and "error-reducible" error-reduction codes. We also show that if the error-reduction codes can be encoded and decoded efficiently in parallel, then the derived error-correcting codes can be as well. Our recursive use of error-reduction codes is analogous to Pippenger's use of expander graphs in his construction of linear-size superconcentrators [Pip77]. In fact, the graphs underlying our linear-size encoding circuits bear a strong resemblance to superconcentrators. In Section 6, we explain why this resemblance is necessary. However, the reader should not need to know anything about superconcentrators in order to understand the present paper.

We construct error-reduction codes by modifying the expander codes constructed by Sipser and Spielman [SS96]. Accordingly, we build error-reduction codes from expander graphs. In Section 3, we explain what expander graphs are and give pointers to both randomized and explicit constructions. In Section 4, we show that very good expander graphs can be used to construct error-reduction codes that can be encoded and error-reduced in linear time. We also present parallel algorithms for these operations. Unfortunately, we do not know of explicit constructions of expander graphs of quality sufficient for this construction. However, randomly chosen graphs will work well with high probability.

In Section 5, we present a construction of error-reduction codes that uses known explicit constructions of expander graphs. We show that these error-reduction codes can be encoded and partially error-reduced by linear-size circuits of constant depth (Lemma 18). The main statement of this paper is Theorem 19, in which we incorporate these constructions of error-reduction codes into the recursion of Section 2 to obtain our error-correcting codes.

We conclude the paper with some comments on how these codes might perform in practice.

2. From error reduction to error correction

Our error-correcting codes are constructed from codes that we call error-reduction codes. Loosely speaking, error-reduction codes have the property that, if not too many of their message bits and not too many of their check bits have been corrupted, then it is possible to remove most of the corruption from their message bits. While these codes do not necessarily allow for full error-correction, we can combine them in a recursive construction to obtain codes that enable full error-correction. In this section, we formally define error-reduction codes and demonstrate how they can be used to construct error-correcting codes.

Definition 1. A code $\mathcal{C}$ of length n with rn message bits and $(1-r)n$ check bits is an *error-reduction code* of *rate* r, *error reduction* ϵ, and *reducible distance* δ if there is an algorithm[1] that, when provided with a word $\vec{x}$ that differs from a codeword $\vec{w} \in \mathcal{C}$ in at most v message bits and t check bits, where $v \leq \delta n$ and $t \leq \delta n$, will output a word that differs from $\vec{w}$ in at most ϵt message bits.

In Section 4, we present a simple randomized construction of a linear-time encodable error-reduction code in which the error-reduction can be performed in linear time. In Section 5, we present a slightly more complicated explicit construction. We now show how these constructions can be used to construct an infinite family of linear-time encodable and decodable error-correcting codes:

Constructing error-correcting codes from error-reduction codes:

Let $\mathcal{C}_0$ be an error-correcting code of block length n_0 and rate $1/4$ from which a $\delta/4$ fraction of error can be corrected. Let $\mathcal{R}_k$ be a family of error-reduction codes with $n_0 2^k$ message bits, $n_0 2^{k-1}$ check bits, reducible distance $\delta > 0$, and error-reduction $1/2$, for $k \geq -1$. We define the codes $\mathcal{C}_k$ for $k > 0$ by describing how they are encoded. $\mathcal{C}_k$ will be a code of block length $n_0 2^k$ and rate $1/4$.

$\mathcal{C}_k$ will have $n_0 2^{k-2}$ message bits. We call this set of message bits M_k. We produce $n_0 2^{k-3}$ of the check bits of $\mathcal{C}_k$ by encoding the $n_0 2^{k-2}$ message bits M_k using the code $\mathcal{R}_{k-2}$. Let A_k denote

[1] This definition is not concerned with the particulars of the algorithm, but rather the structure of the code implied by the algorithm's existence.

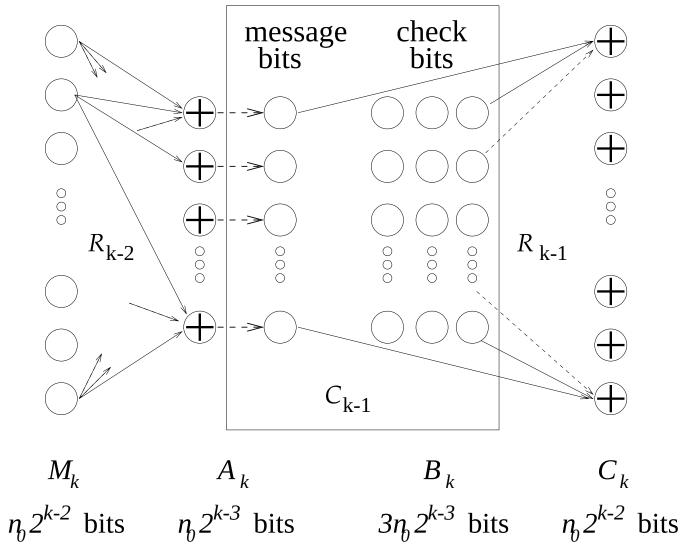

Figure 1: The recursive construction of $\mathcal{C}_k$.

the resulting set of $n_0 2^{k-3}$ check bits. Another $3 \cdot n_0 2^{k-3}$ check bits are produced by using the $n_0 2^{k-3}$ check bits in A_k as the message bits in the code $\mathcal{C}_{k-1}$ and computing the corresponding $3 \cdot n_0 2^{k-3}$ check bits, which we will call B_k. A final $n_0 2^{k-2}$ check bits, C_k, are produced by using the $n_0 2^{k-1}$ check bits in $A_k \cup B_k$ as the message bits in the error-reduction code $\mathcal{R}_{k-1}$. We have produced a total of $3 \cdot n_0 2^{k-2}$ check bits, so the code has total length $n_0 2^k$ and rate $1/4$, as promised (See Figure 1).

Lemma 2. *The codes $\mathcal{C}_k$ are error-correcting codes of lengths $n_0 2^k$ and rate $1/4$ from which a $\delta/4$ fraction of error can be corrected. The codes $\mathcal{C}_k$ can be encoded and decoded in linear time if the codes $\mathcal{R}_k$ have linear-time encoding algorithms and linear-time error-reduction algorithms that will*

a. on input a word that differs from a codeword in v message bits and t check bits, where $v, t \leq \delta n$, output a word that differs from that codeword in at most $\max(v/2, t/2)$ message bits, and

b. on input a word that differs from a codeword in $v \leq \delta n$ message bits and no check bits, output that codeword.

Proof: The proof is by induction. Our base case is the code of length n_0 and rate $1/4$ from which a $\delta/4$ fraction of error can be corrected. Since n_0 is a constant, the errors can be removed from this code in constant time, which is certainly linear; we similarly assert that we can encode this code in constant time.

Let $c_1 n_0 2^k$ be the time required to encode the error-reduction code $\mathcal{R}_k$, and let $c_2 n_0 2^k$ be the time required to perform its error-reduction. Let c_0 be the time required to encode and decode $\mathcal{C}_0$. We will show by induction that $\mathcal{C}_k$ can be encoded in time $3c_1 n_0 2^{k-1} + c_0$ and decoded in time $3c_2 n_0 2^{k-1} + c_0$.

The time required to encode $\mathcal{C}_k$ is the time required to encode $\mathcal{R}_{k-2}$, plus the time to encode $\mathcal{C}_{k-1}$, plus the time to encode $\mathcal{R}_{k-1}$:

$$c_1 n_0 2^{k-2} + \left(3c_1 n_0 2^{k-2} + c_0\right) + c_1 n_0 2^{k-1} = 3c_1 n_0 2^{k-1} + c_0.$$

To correct errors in $\mathcal{C}_k$, we work back to front. That is, we begin by using the linear-time algorithm that satisfies condition (a) to perform error-reduction on the $n_0 2^{k-1}$ bits in $A_k \cup B_k$ using the set of $n_0 2^{k-2}$ check bits in C_k. We then treat the set of $n_0 2^{k-1}$ check bits in $A_k \cup B_k$ as a received word of $\mathcal{C}_{k-1}$, and apply the decoding algorithm for that code. Finally, we assume that the values obtained for the $n_0 2^{k-3}$ check bits in A_k are free of error and use them in the algorithm of condition (b) to perform error-correction on the message bits M_k. The decoding time can now be computed as the encoding time was. It remains to show that this algorithm will correct a $\delta/4$ fraction of error.

Assume that this algorithm is provided with a word in which there are at most $n_0 2^k \delta/4$ errors. Then, there are at most $n_0 2^k \delta/4$ errors in C_k and $A_k \cup B_k$. Thus, after the bits in C_k are used to perform error-reduction on the bits in $A_k \cup B_k$, there will be at most $n_0 2^{k-1}\delta/4$ errors in the bits in $A_k \cup B_k$. By the inductive hypothesis, the code $\mathcal{C}_{k-1}$ can correct this many errors! Thus, after we run the decoding algorithm for $\mathcal{C}_{k-1}$ on the bits in $A_k \cup B_k$, there will be no errors left in the bits in A_k. Since there are at most $n_0 2^k \delta/4$ errors in the bits in M_k, when we run the error-correction algorithm (b) on these bits using the bits in A_k, we will actually remove all the errors from M_k. □

Lemma 3. *If the codes $\mathcal{R}_k$ can be encoded by linear-size circuits of constant depth and have error-reducing circuits of linear-size and constant depth that, when given a word with v corrupt message bits and t corrupt check bits for $v, t < \delta n$, produce a word with at most* $\max(v/2, t/2)$ *corrupt message bits, then the codes $\mathcal{C}_k$ can be encoded by circuits of linear size and logarithmic depth, and decoded by circuits of size $O(n \log n)$ and logarithmic depth, where $n = n_0 2^k$.*

Proof: The bounds on the size and depth of the encoding circuit follow immediately from the construction of the codes $\mathcal{C}_k$. The decoding is slightly more complicated. If we naively simulate the recursive algorithm used in the proof of Lemma 2, we would obtain a circuit of depth $O\left(\log^2 n\right)$. To obtain a circuit of depth $O(\log n)$, we will need to perform decoding on many levels of the recursive construction simultaneously.

The beginning of the algorithm is similar to that in the proof of Lemma 2: first the check bits in C_k are used to reduce the number of errors in $A_k \cup B_k$. The bits in $A_k \cup B_k$ are then viewed as belonging to a word received in $\mathcal{C}_{k-1}$, divided into parts M_{k-1}, A_{k-1}, B_{k-1}, and C_{k-1}, and the bits in C_{k-1} are used to reduce the errors in $A_{k-1} \cup B_{k-1}$. This continues all the way down to C_0, which is decoded by brute force in constant time. So far, our circuit has logarithmic depth and linear size. If we began with at most $n_0 2^k \delta/4$ errors, then the same analysis as was used in the proof of Lemma 2 can be used to show that no errors remain in the bits in $\mathcal{C}_0 = A_1 \cup B_1$ and that at most $n_0 2^i \delta/4$ errors remain in the bits in $M_i = A_{i+1}$.

To complete the decoding, we use the bits in $A_i = M_{i-1}$ to reduce the errors in M_i, *simultaneously,* for all levels $1 \le i \le k$. We will show that, after a logarithmic number of iterations of this procedure, no errors will remain in the bits in M_k. In particular, after the j-th iteration, at most $n_0 2^{i-j}\delta/4$ errors remain in the bits in M_i. This is because, at the beginning of the j-th iteration, there are at most $n_0 2^{i-j+1}\delta/4$ errors in the $n_0 2^{i-2}$ bits in M_i and fewer than $n_0 2^{i-j+1}\delta/4$ errors in the bits in A_i, so the j-th iteration of error-reduction will terminate with at most $n_0 2^{i-j}\delta/4$ errors left in M_i. The circuit required for each iteration has linear size, so our final circuit has size $O(n \log n)$ and depth $O(\log n)$. □

3. Expander graphs

Our error-reduction codes are derived from expander graphs, and are closely related to the expander codes constructed by [SS96]. In this section, we will review some facts concerning expander graphs that we need for our constructions.

Expander graphs have been the focus of much study in theoretical computer science and combinatorics. An expander graph is a graph in which every set of vertices has an unusually large number of neighbors. Our constructions require graphs that expand by a constant factor, but which have only a linear number of edges. It is a remarkable fact that such graphs exist. In fact, a simple randomized process will produce one with high probability. Deterministic, polynomial-time constructions also exist.

Let $G = (V, E)$ be a graph on n vertices. To describe the expansion properties of G, we say *every set of at most m vertices expands by a factor of δ* if, for all sets $S \subset V$,

$$|S| \leq m \qquad \Rightarrow \qquad \Big| \{y : \exists x \in S \text{ such that } (x, y) \in E\} \Big| > \delta \, |S| .$$

In our constructions, we will make use of *unbalanced bipartite* expander graphs. That is, the vertices of the graph will be divided into two sets so that there are no edges between vertices in the same set. We call such a graph (c, d)*-regular* if all the nodes in one set have degree c and all the nodes in the other have degree d. By counting edges, we find that the number of c-regular vertices must differ from the number of d-regular vertices by a factor of d/c. We will use graphs in which $d > c$, so they will have more c-regular than d-regular vertices. We will only consider the expansion of sets of vertices contained within the larger side of the graph. We call a graph a (c, d, ϵ, δ) *expander* if it is a (c, d)-regular graph in which every subset of at most an ϵ fraction of the c-regular vertices expands by a factor of δ.

It is well known that a randomly chosen (c, d)-regular graph will probably be a good expander:

Proposition 4. *Let B be a randomly chosen (c, d)-regular bipartite graph between n c-regular vertices and $\frac{c}{d}n$ d-regular vertices. Then, for all $0 < \alpha < 1$, with high probability, all sets of αn c-regular vertices in B have at least*

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

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

Proof: See Appendix B of [SS96]. □

The most common way to prove that a particular graph is a good expander is to examine its second-largest eigenvalue. The largest eigenvalue of a k-regular graph is k. If the second-largest eigenvalue is far from the first, then the graph is a good expander. The greatest possible separation between the first and second-largest eigenvalues in a graph was achieved in explicit constructions by Margulis [Mar88] and Lubotzky, Phillips, and Sarnak [LPS88]:

Theorem 5 (Lubotzky-Phillips-Sarnak, Margulis). *For every pair of primes p, q congruent to 1 modulo 4 such that p is a quadratic residue modulo q, there is a $(p+1)$-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 $2\sqrt{p}$.*

One can show that a graph with the eigenvalue separation displayed by these graphs is a good expander by using results from [Tan84, Alo86, Kah92]. Other constructions that achieve a similar separation have since appeared [Bie89, Mor95, Mor94]. From the fact that these graphs are Cayley graphs, one can show that they have a simple representation:

Proposition 6. *Each graph described in Theorem 5 can be constructed in time polynomial in its number of vertices. Moreover, this polynomial-time computation can be used to construct a description of the graph of size logarithmic in the number of vertices in the graph. There is an algorithm that, given this description, will produce the labels of the neighbors of a node in the graph in time polynomial in the length of the labels (i.e., polylogarithmic in the number of vertices).*

In the remainder of this paper, we can ignore the fact that these graphs are Cayley graphs, and just concentrate on the relation between their degrees and second-largest eigenvalues. Unfortunately, these graphs are not unbalanced bipartite. To obtain unbalanced bipartite expander graphs from these graphs, we use their *edge-vertex incidence graphs*. From a d-regular graph G on n vertices, we derive a $(2, d)$-regular graph with $dn/2$ vertices on one side and n vertices on the other:

Definition 7. Let G be a graph with edge set E and vertex set V. The *edge-vertex incidence graph* of G is the bipartite graph with vertex set $E \cup V$ and edge set

$$\{(e, v) \in E \times V : v \text{ is an endpoint of } e\}.$$

To understand the expansion of these edge-vertex incidence graphs, we use a lemma of Alon and Chung [AC88]:

Lemma 8 (Alon-Chung). *Let G be a d-regular graph on n vertices with second-largest eigenvalue λ. Let X be a subset of the vertices of G of size γn. Then, the number of edges contained in the subgraph induced by X in G is at most*

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

We could use this lemma to characterize the edge-vertex incidence graphs of the graphs constructed in Theorem 5 as $(2, d, \epsilon, \delta)$ expanders for some ϵ and δ. However, we will find it more convenient to work directly with Lemma 8.

The graphs of Theorem 5 will suffice for our constructions in Section 5. However, a more general class of graphs would suffice as well. The following definition captures this class of graphs:

Definition 9. We will say that a family of graphs $\mathcal{G}$ is a *dense family of good expander graphs* if $\mathcal{G}$ contains graphs $G_{n,d}$ of n nodes and degree d so that

> [**density**] for an infinite number of values of d, the family of graphs $G_{n_i,d} \in \mathcal{G}$ is infinite and satisfies $n_{i+1} - n_i = o(n_i)$, and
>
> [**goodness**] for each of these values d, the second-largest eigenvalues of $G_{n_i,d}$ are bounded from above by constants λ_d such that $\lim_{d \to \infty} \lambda_d / d = 0$.

Pippenger [Pip93] points out that we can obtain a family of good expander graphs by exponentiating the expander graphs constructed originally by Margulis [Mar73] or Gabber and Galil [GG81] (see also [Alo86, Proposition 2.6]). To see that the graphs of Theorem 5 are dense, one can apply bounds on the gaps between successive primes in arithmetic progressions.

4. A simple construction

In this section, we exploit very good expander graphs to obtain a simple construction of linear-time error-reduction codes. While we do not know of explicit constructions of graphs with enough expansion for this construction, Proposition 4 can be used to show that a random graph will suffice with high probability.

Let B be a $(d, 2d)$-regular graph between sets of n and $n/2$ vertices. The error-reduction code $\mathcal{R}(B)$ will have n message bits $\{x_1, \ldots, x_n\}$ and $n/2$ check bits $\left\{c_1, \ldots, c_{n/2}\right\}$, corresponding to the d-regular and $2d$-regular vertices, respectively. Let $b(i,j)$ be the function such that, for each check bit c_i, the message bits neighboring c_i are $x_{b(i,1)}, \ldots, x_{b(i,2d)}$. The check bit c_i is defined to be the sum modulo two of its neighbors. Thus, $\mathcal{R}(B)$ can be easily encoded in linear time: one need merely examine $2d$ message bits to compute the value of each check bit (See Figure 2).

We will now show that if B is a sufficiently good expander graph, then $\mathcal{R}(B)$ is a good error-reduction code. To explain the error-reduction algorithm that we will use, we will say that a check bit c_i is *satisfied* by message bits $x_1, \ldots, x_n$ if c_i is the sum modulo two of $x_{b(i,1)}, \ldots, x_{b(i,2d)}$. Otherwise, we say that c_i is *unsatisfied.*

To discuss our error-reduction algorithms, we imagine that a codeword $\vec{w}$ of $\mathcal{R}(B)$ has been transmitted and that a word that is close to $\vec{w}$ has been received. We then call *corrupt* those message and check bits in which the received word differs from $\vec{w}$. The idea behind the error-reduction algorithm is simple: if we find a message bit such that most of its neighbors are unsatisfied and we flip[2] the value of that bit, then we will have reduced the number of unsatisfied

[2]If the bit was 0, make it 1. If it was 1, make it 0.

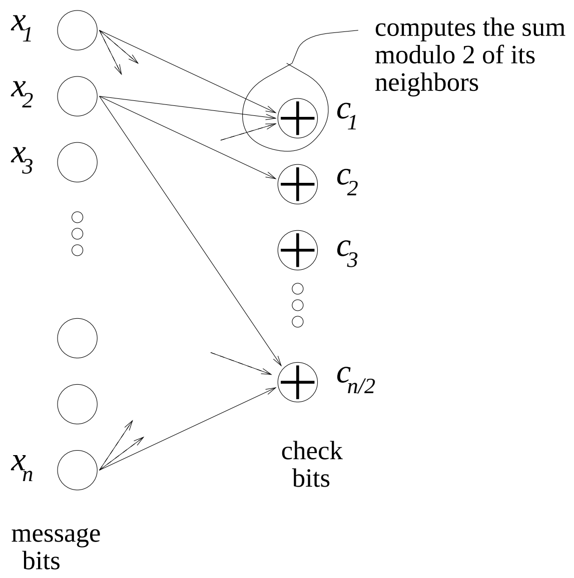

Figure 2: A circuit that encodes $\mathcal{R}(B)$.

check bits. If we iterate this process, we probably also reduce the number of corrupt message bits.

The reader familiar with [SS96] will see a strong resemblance between expander codes and these error-reduction codes. Where expander codes have constraints, these codes have check bits. If none of the check bits is corrupt, one can remove errors from the message bits using the expander code decoding algorithms of [SS96]. We will show that if only a few of the check bits are corrupt, then the decoding algorithms of [SS96] will remove most of the errors from the message bits.

Simple sequential error-reduction algorithm:

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

Lemma 10. *Let B be a $(d, 2d, \alpha, \frac{3}{4}d + 2)$ expander. If the simple sequential error-reduction algorithm for $\mathcal{R}(B)$ is given a word $\vec{x}$ that differs from a codeword $\vec{w}$ of $\mathcal{R}(B)$ in $v \leq \alpha n/2$ message bits and $t \leq \alpha n/2$ check bits, then the algorithm will output a word that differs from $\vec{w}$ in at most $t/2$ of its message bits.*

Proof: We call *corrupt* any bit that differs from its corresponding bit in $\vec{w}$. Let V be the set of corrupt message bits and let T be the set of corrupt check bits. Let $v = |V|$ and $t = |T|$.

Let u be the number of unsatisfied check bits and let s the number of satisfied check bits with neighbors in V. We will view the pair (u, v) as the state of the algorithm. We will first show that if $\alpha n \geq v \geq t/2$, then there is some message bit with more unsatisfied than satisfied neighbors. The expansion of the graph implies that

$$u + s \;\; > \;\; (3d/4 + 2)\, v. \tag{1}$$

Each check bit with an input in V accounts for at least one edge leaving V. A check bit can be unsatisfied either because it has at least one neighbor in V or because it is corrupt. Similarly, a satisfied check bit that is a neighbor of V is either corrupt, or it has at least two neighbors in V. By counting the dv wires leaving V, we obtain

$$dv + t \;\; \geq \;\; u + 2s \tag{2}$$

Combining inequalities (1) and (2), we find $s < (d/4 - 2)\, v + t$, and

$$u \;\; > \;\; (d/2 + 4)\, v - t. \tag{3}$$

When $\alpha n \geq v \geq t/2$, we have $u > dv/2 + t$, so there must be some message bit such that most of its neighbors are unsatisfied. This means that the algorithm will have a message bit to flip. Thus, when the algorithm terminates, either $v < t/2$, or at some point along the way $v = \alpha n$.

To show that the algorithm must terminate with $v < t/2$, we show that v must always be less than αn. We assume that when the algorithm begins $v \leq \alpha n/2$ and therefore $u \leq dv + t \leq d\alpha n/2 + \alpha n/2$. As the algorithm proceeds, u must steadily decrease. However, if the algorithm is ever in a state (u, v) in which $v = \alpha n$, then inequality (3) would imply that $u > d\alpha n/2 + 7\alpha n/2$, which would be a contradiction.

Thus, the algorithm must always maintain the condition that $v < \alpha n$, which implies that the algorithm will not terminate until it is in a state in which $v < t/2$. $\square$

Proposition 11. *The simple sequential error-reduction algorithm can be implemented to run in linear time.*

Proof: The basic idea behind the proof is that, because the degrees of the graph are constant, the algorithm only needs to perform a constant amount of work for each message bit that is flipped. Because the number of unsatisfied check bits decreases each time a message bit is flipped, there will only be a linear number of flips. For a more complete proof of this proposition, see the proof of Lemma 9 in [SS96]. $\square$

It is also possible to perform error-reduction on these codes in parallel. The algorithm proceeds in rounds, each of which is the natural generalization of the sequential algorithm:

Simple error-reduction round:

- For each message bit, count the number of unsatisfied check bits among its neighbors.
- Flip each message bit such that most of its neighbors are unsatisfied.

Proposition 12. *A simple error-reduction round can be performed by a linear-size circuit of constant depth.*

Lemma 13. *Let B be a $(d, 2d, \alpha, \frac{3}{4}d+4)$ expander. Assume that a simple error-reduction round for $\mathcal{R}(B)$ is given an input $\vec{x}$ that differs from a codeword $\vec{w}$ of $\mathcal{R}(B)$ in $v \leq \alpha n/2$ message bits and $t \leq \alpha n/2$ check bits. Then, the round will output a word with the same check bits that differs from $\vec{w}$ in at most*

$$v(d-4)/d \tag{4}$$

message bits, if $v \geq t/2$. If $v \leq t/2$, then the round will output a word in which at most $t/2$ message bits differ from those in $\vec{w}$.

Proof: Let V denote the set of corrupt message bits, T the set of corrupt check bits, F the set of corrupt message bits that fail to flip during the round, and C the set of message bits that were originally clean, but which are flipped so as to become corrupt during the round. The message bits that are corrupt in the output of the round are those in $C \cup F$. For a set of message bits, S, let $N(S)$ denote the set of check bits that have neighbors in S. Set δ so that $\delta d|V| = |N(V)|$.

We first show by contradiction that $|V \cup C| < \alpha n$. If $|V \cup C| \geq \alpha n$, then pick a subset C' of C such that $|V \cup C'| = \alpha n$. At least $d/2$ of the edges leaving each message bit in C must either connect to check bits that have inputs from V or that are corrupt. So, $V \cup C'$ has at most $\delta d|V| + (d/2)\,|C'| + |T|$ neighbors. On the other hand, $V \cup C'$ must expand by a factor of more than $(3d/4+4)$, so we obtain the inequality

$$(3d/4+4)\alpha n \;<\; |N(V \cup C')| \;\leq\; \delta d|V| + (d/2)(\alpha n - |V|) + |T|,$$

which implies $(d/4+4)\alpha n \;<\; (\delta - 1/2)d|V| + |T|$. Because we assume $|T| \leq \alpha n/2$, and δ must be less than 1, this contradicts our assumption that $|V| \leq \alpha n/2$. Thus, $|V \cup C| < \alpha n$, and we can assert

$$(3d/4+4)\,|V \cup C| < |N(V \cup C)| \leq \delta d|V| + (d/2)\,|C| + |T|. \tag{5}$$

We next bound $\delta d|V|$ in terms of $|F|$. Every message bit in F is a neighbor of at least as many satisfied as unsatisfied check bits. At most $|T|$ of these satisfied check bits are satisfied because they have been corrupted. So, at least $(d/2)|F| - |T|$ of the edges leaving F end in a check bit that contains an input from another element of V. Thus, the set V can have at most

$$\delta d|V| \leq d|V| - (d/4)|F| + |T|/2$$

neighbors. Plugging this bound into inequality (5), we find

$$\begin{array}{rcll} (3d/4+4)(|V|+|C|) & < & d|V| - (d/4)\,|F| + (d/2)\,|C| + (3/2)|T| & \Rightarrow \\ (d/4+4)\,|C| + (d/4)\,|F| & < & (d/4-4)|V| + (3/2)|T| & \Rightarrow \\ |C \cup F| & < & \frac{(d-16)|V|+6|T|}{d}|V|. & \end{array}$$

Note that d must be greater than 16 because the graph has expansion at least $(3/4)d+4$, and the expansion of a graph cannot be greater than its degree. We can now verify that if $|V| \geq |T|/2$, then $|C \cup F| < \frac{d-4}{d}|V|$, and if $|V| < |T|/2$, then $|C \cup F| < \frac{d/2-2}{d}|T| < |T|/2$. $\square$

By iterating simple parallel error-reduction rounds, we obtain the simple parallel error-reduction algorithm:

Simple parallel error-reduction algorithm:

- Iterate $\log_{d/(d-4)} 2$ simple parallel error-reduction rounds.

Corollary 14. *Let B be a $(d, 2d, \alpha, \frac{3}{4}d+4)$ expander. Assume the simple parallel error-reduction algorithm for $\mathcal{R}(B)$ is given a word $\vec{x}$ that differs from a codeword $\vec{w}$ of $\mathcal{R}(B)$ in $v \leq \alpha n/2$ message bits and $t \leq \alpha n/2$ check bits. Then, the simple parallel error-reduction algorithm will output a word that differs from $\vec{w}$ in at most* $\max(v/2, t/2)$ *of its message bits. Moreover, this algorithm can be implemented as a circuit of constant depth and linear size.*

By combining Corollary 14 with Lemma 3 and Lemma 10 with Lemma 2, we can show that, from a family of very good expander graphs, one can obtain linear-time encodable and decodable error-correcting codes. However, we repeat that the only way we know to obtain such graphs in polynomial time is through a randomized construction.

Theorem 15. *From a family of $(d, 2d, \alpha, 3d/4+4)$ expander graphs between sets of $2^k n_0$ and $2^{k-1} n_0$ vertices, for all $k \geq -1$, one can construct an infinite family of error-correcting codes that have linear-time encoding algorithms and linear-time decoding algorithms that will correct an $\alpha/8$ fraction of error. Moreover, their encoding can be performed by linear-size circuits of logarithmic depth and their decoding can be performed by logarithmic depth circuits of size $O(n \log n)$.*

Proof: Apply Lemma 2 to Proposition 11 and Lemma 10, and Lemma 3 to Corollary 14. $\square$

5. Explicit Constructions

To produce explicit constructions of error-reduction codes, we will use explicit constructions of expander graphs. Our construction is a generalization of that in Section 4.

Definition 16. Let B be a (c, d)-regular bipartite graph between sets of n and $(c/d)n$ vertices and let $\mathcal{S}$ be an error-correcting code with d message bits and k check bits. $\mathcal{R}(B, \mathcal{S})$ is a code with n message bits and $(c/d)nk$ check bits. Each of the nodes in the set of n vertices is identified with one of the message bits $\{x_1, \ldots, x_n\}$, and each of the nodes in the other set is identified with a set of k check bits called a *cluster*. Let $b(i,j)$ be the function such that, for each cluster C_i, the message bits neighboring C_i are $x_{b(i,1)}, \ldots, x_{b(i,d)}$. The check bits in cluster C_i are set to be the check bits of the codeword of $\mathcal{S}$ that has message bits $(x_{b(i,1)}, \ldots, x_{b(i,d)})$.

In this section, we will show that if B is the edge-vertex incidence graph of a good expander graph, such as one of those produced in Theorem 5, and if $\mathcal{S}$ is a good error-correcting code, then $\mathcal{R}(B, \mathcal{S})$ is a good error-reduction code. We first note that such codes can be encoded in linear time:

Proposition 17. *For a fixed code $\mathcal{S}$, the family of codes $\mathcal{R}(B, \mathcal{S})$ can be encoded in linear time on a sequential machine or by a linear-size circuit of constant depth.*

Our parallel algorithms for error-reducing these codes will proceed in rounds. Our sequential algorithms will be simulations of the parallel algorithms.

Parallel error-reduction round for $\mathcal{R}(B, \mathcal{S})$:
(where ϵ is the minimum relative distance of $\mathcal{S}$)

- In parallel, for each cluster, if the the check bits in that cluster and the associated message bits are within relative distance $\epsilon/6$ of a codeword, then send a "flip" signal to every message bit that differs from the corresponding bit in the codeword.
- In parallel, every message bit that receives at least one "flip" signal flips its value.

The reader familiar with [SS96] will observe a strong resemblance between its explicit construction of expander codes and this construction of error-reduction codes. Where the expander codes have constraints, these codes have clusters of check bits. If there are no errors in these check bits, then the expander code decoding algorithms can be used to correct errors in the message bits. The lower threshold for sending a "flip" signal in the parallel error-reduction rounds seems necessary to deal with corrupt check bits.

Lemma 18. *Let $\mathcal{S}$ be a linear code of rate r, block length d, and minimum relative distance ϵ, and let B be the edge-vertex incidence graph of a d-regular graph on n vertices with second-largest eigenvalue λ. Then, $\mathcal{R}(B, \mathcal{S})$ is a code with $dn/2$ message bits and $dn(1-r)/r$ check bits. If a parallel error-reduction round for $\mathcal{R}(B, \mathcal{S})$ is given an input that differs from a codeword $\vec{w}$ of $\mathcal{R}(B, \mathcal{S})$ in at most $\alpha dn/2$ message bits and at most $\beta dn/2$ check bits, Then, the error-reduction round will output a word that differs from $\vec{w}$ in at most*

$$(2\alpha + \beta) \left(\frac{1}{5} + \frac{9(2\alpha + \beta)}{\epsilon^2} + \frac{3\lambda}{\epsilon d} \right) \frac{dn}{2}$$

message bits.

Proof: Let G be the d-regular graph from which B is derived. Since $\mathcal{S}$ has rate r, each of the n clusters of $\mathcal{R}(B, \mathcal{S})$ will contain $d(1-r)/r$ check bits; so, $\mathcal{R}(B, \mathcal{S})$ will have a total of $dn(1-r)/r$ check bits.

Let V be the set of message bits in which the input differs from $\vec{w}$. A message bit will be corrupt at the end of the decoding round if it receives a "flip" signal, but is not in V, or if it is in V, but it does not receive a "flip" signal. Thus, we will say that a cluster is *confused* if it sends a "flip" signal to a message bit that is not corrupt, and we will call a cluster *unhelpful* if it contains a message bit of V, but it fails to send a "flip" signal to that message bit.

In order for a cluster to be confused, it must have at least $5\epsilon d/6$ corrupt message and check bits. As each message bit is an input to two clusters, there can be at most

$$\frac{(2\alpha+\beta)\frac{dn}{2}}{5\epsilon d/6}$$

confused clusters. Each of these can send at most $\epsilon d/6$ "flip" signals, so at most

$$\frac{(2\alpha+\beta)\frac{dn}{2}}{5\epsilon d/6}\cdot\frac{\epsilon d}{6}=\frac{(2\alpha+\beta)\frac{dn}{2}}{5}$$

message bits not in V can receive "flip" signals.

Similarly, there can be at most

$$\frac{(2\alpha+\beta)\frac{dn}{2}}{\epsilon d/6}$$

unhelpful clusters. Applying Lemma 8, we see that there are at most

$$\frac{dn}{2}\left(\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\frac{\lambda}{d}\left(\frac{6\alpha+3\beta}{\epsilon}\right)\right)$$

message bits both of whose neighbors are unhelpful clusters.

Thus, at most

$$\begin{aligned}&\frac{dn}{2}\left(\frac{2\alpha+\beta}{5}+\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda}{d}\right)\\&=\frac{dn}{2}(2\alpha+\beta)\left(\frac{1}{5}+\frac{9(2\alpha+\beta)}{\epsilon^2}+\frac{3\lambda}{\epsilon d}\right)\end{aligned}$$

message bits will be corrupt at the end of the error-reduction round. □

We will show that for each sufficiently small ϵ, there is a value of λ/d such that the error-reduction round decreases the number of corrupt message bits for some α and β.

To construct our asymptotically good family of error-correcting codes, we will construct error-reduction codes from the edge-vertex incidence graphs of a family of good expander graphs and a good code known to exist by the Gilbert-Varshamov bound (see [MS77]). We then show that parallel error-reduction rounds can be used to perform error reduction on these codes.

Theorem 19. *There exists a polynomial-time constructible family of error correcting codes of rate $1/4$ that have linear-time encoding algorithms and linear-time decoding algorithms that can* *correct any $\gamma < \epsilon_{4/5}^2/2160$ fraction of error, where $\epsilon_{4/5}$ is the lesser solution to $4/5 = 1 - H(\epsilon_{4/5})$ and $H(\cdot)$ is the binary entropy function. The encoding can be performed by linear-size circuits of logarithmic depth and decoding can be performed by circuits of size $O(n \log n)$ and logarithmic depth.*

Proof: We will build the error-correcting codes by constructing a family of error-reduction codes of error reduction $1/2$ and reducible distance 4γ to which we can apply Lemmas 2 and 3. These error-reduction codes will be of the form $\mathcal{R}(B, \mathcal{S})$ where $\mathcal{S}$ is a particular good code known to exist by the Gilbert-Varshamov bound (see [MS77]) and the graphs B are the edge-vertex incidence graphs of a dense family of good expander graphs.

Choose an $\epsilon < \epsilon_{4/5}$ such that $\gamma < \epsilon^2/2160$. From the Gilbert-Varshamov bound, we know that for all sufficiently large block lengths, d, there exist codes of minimum relative distance ϵ and rate $r = 1 - H(\epsilon)$. We will use one such code in our construction. Since $\epsilon < \epsilon_{4/5}$, we have $r > 4/5$. We now find a suitable block length, d.

Let $\mathcal{G} = \{G_{n_i,d}\}$ be a polynomial-time constructible dense family of good expander graphs (we know such a family exists by Theorem 5 and Proposition 6) and let λ_d be the upper bound on the second-largest eigenvalues of its graphs of degree d. Because we wish to apply Lemma 18 with $\alpha, \beta \leq 4\gamma < \epsilon^2/540$, we choose a d so that

$$\left(\frac{1}{5} + \frac{9(2\alpha + \beta)}{\epsilon^2} + \frac{3\lambda_d}{\epsilon d}\right) < 1/4$$

and there exists a code $\mathcal{S}$ of block length d, rate r, and minimum relative distance ϵ. Such a d exists because

$$1/5 + 9(2\alpha + \beta)/\epsilon^2 < 1/4,$$

for $\alpha, \beta < \epsilon^2/540$. We fix this d and the code $\mathcal{S}$ for the rest of the proof.

Let $B_{n_i,d}$ be the edge-vertex incidence graph of $G_{n_i,d}$. Our family of error-reduction codes consists of the codes $\mathcal{R}(B_{n_i,d}, \mathcal{S})$. The i-th code in this family has $dn_i/2$ message bits and $dn_i(1-r)/r$ check bits. However, to apply Lemmas 2 and 3, we need codes with $2^k n_0$ message bits and $2^{k-1} n_0$ check bits for all $k \geq -1$ and some sufficiently large n_0. Because the family of graphs $\mathcal{G}$ is dense, and $(1-r)/r < 1/4$, there exists an n_0 such that for all $k \geq -1$ there is a graph $G_{n_i,d}$ in the family such that

$$dn_i/2 > 2^k n_0 \qquad \text{and} \qquad 2^{k-1} n_0 > dn_i(1-r)/r.$$

Thus, we can use the error-reduction code $\mathcal{R}(B_{n_i,d}, \mathcal{S})$ to obtain a suitable error-reduction code with $2^k n_0$ message bits and $2^{k-1} n_0$ check bits by eliminating some of its message bits by fixing them to zero and adding some dummy check bits. Using the Gilbert-Varshamov bound, we can find a code $\mathcal{C}_0$ of block length n_0, rate $1/4$ and minimum relative distance ϵ, because $1/4 < 1 - H(\epsilon)$ (this minimum relative distance ϵ for $\mathcal{C}_0$ is much more than we actually need for our construction).

It remains to show that the error-reduction codes can remove the required fraction of error. Let

$$\mu = \left(\frac{1}{5} + \frac{9 \cdot 12\gamma}{\epsilon^2} + \frac{3\lambda_d}{\epsilon d}\right) < 1/4.$$

By Lemma 18, if a parallel error-reduction round is given as input a word that differs from a codeword in $v \leq 4\gamma(dn/2)$ message bits and $t \leq 4\gamma(dn/2)$ check bits, then the output of the round will differ from that codeword in at most $(2v+t)\mu$ message bits. If $v \geq t/2$, then $(2v+t)\mu \leq 4\mu v$. So, the iteration of $\log_{(1/4\mu)} 2$ parallel error-reduction rounds will output a word that differs from the codeword in at most $\max(v/2, t/2)$ message bits. Since the iteration of $\log_{(1/4\mu)} 2$ parallel error-reduction rounds can be computed by a linear-size circuit of constant depth, we can now apply Lemma 3 to prove the parallel portion of the theorem.

To obtain our sequential algorithms, we first note that the $(\log_{1/(4a)} 2)$-fold iteration of the parallel error-reduction rounds can be performed sequentially in linear time. Thus, we can satisfy condition (a) of Lemma 2. To satisfy condition (b), we need to be slightly trickier. If there are no corrupt check bits, then it is clear that the application of a logarithmic number of parallel error-reduction rounds will correct all the errors in the message bits. However, a direct simulation would take $O(n \log n)$ time. To obtain a linear-time simulation, we observe that the number of corrupt message bits decreases by a constant factor with each iteration. Thus, the number of unsatisfied clusters will also decrease by a constant factor with each iteration, so only a linear amount of work is required to simulate the iteration of a logarithmic number of parallel error-reduction rounds. For a more detailed analysis of such a simulation, we direct the reader to the end of the proof of Theorem 19 of [SS96].

$\square$

Remark 20. *Some may consider the use of the Gilbert-Varshamov bound in the preceding argument to be "non-constructive". To us, a constant amount of non-constructivity is negligible. However, we point out that one could replace this argument by using any known asymptotically good code, or just fixing d and picking an appropriate error-correcting code (say, from the back of [MS77]).*

Remark 21. *One can combine Lemma 18 and Theorem 19 to construct an asymptotically good family of linear-time encodable and decodable error-correcting codes of any fixed rate. To construct codes of rate r, first build a family of error-reduction codes of the form $\mathcal{R}(B, \mathcal{S})$ where $\mathcal{S}$ is a code of rate $8r/(1+7r)$. The codes in this family with m message bits will have $m(1-r)/4r$ check bits. These check bits can be used as the message bits in one of the codes constructed in Theorem 19. After encoding these check bits with such a code, we obtain $m(1-r)/r$ check bits for our m message bits, yielding a code of rate r. We can decode such a code by first correcting errors in the part corresponding to the codes of Theorem 19. Having removed all errors from the check bits of the error-reduction code used to encode the m message bits, we can use Lemma 18 to see that the iteration of a logarithmic number of parallel error-reduction rounds will correct the errors in the m message bits. As explained in the proof of Theorem 19, the action of the parallel error-reduction rounds can be simulated in linear time on a sequential machine.*

6. The superconcentrator connection

It is not an accident that our linear-size encoding circuits look like superconcentrators. They have to. This fact was the motivation behind our construction. In the following explanation, we assume familiarity with superconcentrators.

Consider a circuit C that takes as input some message bits $x_1, \ldots, x_n$, and produces as output check bits $c_1, \ldots, c_m$ such that the words $x_1, \ldots, x_n, c_1, \ldots, c_m$ form a good error-correcting code. This means that there is a constant δ such that even if we erase any δn of the message bits and any δm of the check bits, we can still recover the erased δn message bits. We will show that this means that there must be gate-disjoint paths from the erased inputs to some subset of the un-erased outputs.

Assume that we cannot find δn vertex-disjoint paths from the erased inputs to the un-erased outputs. Then, Menger's Theorem implies that there is some set of $\delta n - 1$ gates in the circuit such that all paths in the circuit from the erased inputs to the un-erased outputs must go through these gates. This contradicts our assumption that it is possible to recover the values of the erased inputs because there are δn bits of information in the erased input gates, but only $\delta n - 1$ bits of information can get through to the un-erased output gates.

Thus, we see that vertex disjoint paths can be drawn in the underlying graph from any δn input gates into any $(1 - \delta)m$ output gates. While this property is not quite as strong as the property required of superconcentrators, it is close enough that we decided that the easiest way to create linear-size encoding circuits would be to base them on Pippenger's [Pip77] construction of linear-size superconcentrators.

7. Some implementation issues

As our error-correcting codes are closely related to the codes of [SS96], we will only discuss issues that are not addressed in that paper.

In order to make our codes effective against worst-case errors as well as linear-time encodable, we sacrifice on rate. For comparison, one should consider the expander codes of Sipser and Spielman [SS96]. Our codes of rate 1/4 will correct about as many errors on average as their codes of rate 1/2. While there are many ways that our construction can be modified (for example, changing the ratios of message bits to check bits in the error-reduction codes), these codes will always correct fewer errors on average than expander codes of similar rates. We believe that it should be possible to improve our constructions to overcome this difficulty.

In the meantime, we observe that if one is willing to sacrifice worst-case error-correction, one can obtain average-case error-correction as good as that obtained by expander codes. The idea is to recursively apply the error-reduction codes in the natural way: first, the message bits are protected with an error-reduction code, the check bits of this code are then used as the message bits of another error-reduction code, and so on. One can iterate this a few times and then use a known error-correcting code to protect the last set of check bits. Such a code is vulnerable to worst-case errors that only strike the last and smallest code in the recursion. However, if the probability that each bit is corrupt is independent, then it is unlikely that too many errors will occur in any one level of the recursion, so decoding from the smallest code up will probably succeed. Moreover, one can interleave the bits from different levels to prevent bursts from corrupting too many bits on any one level.

We do not recommend using the codes constructed in Theorem 19. These codes are designed to be asymptotically good in theory, but we would be surprised if they were useful in practice. However, we expect that the error-reduction codes described in Section 4 and constructions such as Definition 16 will prove useful.

8. Conclusion

Having presented asymptotically good error-correcting codes that can be encoded and decoded in linear time, many questions remain. The foremost is: how good can such codes be? In particular, can one construct a linear-time encodable code for which one can correct in linear time a number of errors matching the Gilbert-Varshamov bound? For that matter, can one correct in linear time such a number of errors in any code, regardless of its encoding complexity?

Less daunting problems remain open as well: While the codes that we construct are asymptotically good, the rate versus error-correction tradeoffs that we prove are very poor. We hope that someone will construct linear-time codes with better constants. The sequential decoding algorithm that we present for the explicit construction of error-reduction codes is a simulation of a parallel algorithm and therefore unnatural. We would like to see a proof of correctness of a more natural sequential algorithm.

9. Acknowledgement

I would like to thank Mike Sipser for fruitful discussions of this material and Leonard Schulman for simplifying some of the proofs in this paper.

References

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

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley, Reading, Massachusetts, 1974.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[Bie89] F. Bien. Constructions of telephone networks by group representations. *Notices of the American Mathematical Society*, 36(1):5–22, 1989.

[BZP77] L. A. Bassalygo, V. V. Zyablov, and M. S. Pinsker. Problems of complexity in the theory of correcting codes. *Problems of Information Transmission*, 13(3):166–175, 1977.

[GDP73] S. I. Gelfand, R. L. Dobrushin, and M. S. Pinsker. On the complexity of coding. In *Second International Symposium on Information Theory*, pages 177–184, Akademiai Kiado, Budapest, 1973.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22:407–420, 1981.

[Jus76] J. Justesen. On the complexity of decoding Reed-Solomon codes. *IEEE Transactions on Information Theory*, 22(2):237–238, March 1976.

[Kah92] N. Kahale. On the second eigenvalue and linear expansion of regular graphs. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 296–303, 1992.

[KU58] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *Uspehi Math. Nauk*, 13(4):3–28, 1958. English translation in [KU63].

[KU63] A. Kolmogorov and V. Uspenskiĭ. On the definition of an algorithm. *American Mathematical Society Translations*, 29:217–245, 1963.

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

[Mar73] G. A. Margulis. Explicit constructions of concentrators. *Problemy Peredachi Informatsii*, 9(4):71–80, October-December 1973. English translation in [Mar75].

[Mar75] G. A. Margulis. Explicit constructions of concentrators. *Problems of Information Transmission*, 9:325–332, 1975.

[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.

[Mor94] M. Morgenstern. Existance and explicit constructions of $q + 1$ regular Ramanujan graphs for every prime power q. *Journal of Combinatorial Theory, Series B*, 62:44–62, 1994.

[Mor95] M. Morgenstern. Natural bounded concentrators. *Combinatorica*, 15(1):111–122, 1995.

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

[Pip77] N. Pippenger. Superconcentrators. *SIAM Journal on Computing*, 6:298–304, 1977.

[Pip93] N. Pippenger. Self-routing superconcentrators. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 355–361, 1993.

[Sar77] D. V. Sarwate. On the complexity of decoding Goppa codes. *IEEE Transactions on Information Theory*, 23(4):515–516, July 1977.

[Sav69] J. E. Savage. The complexity of decoders–part I: Decoder classes. *IEEE Transactions on Information Theory*, 15:689–695, November 1969.

[Sav71] J. E. Savage. The complexity of decoders–part II: Computational work and decoding time. *IEEE Transactions on Information Theory*, 17(1):77–85, January 1971.

[Sch80] A. Schönhage. Storage modification machines. *SIAM Journal on Computing*, 9(3):490–508, 1980.

[SS96] M. Sipser and D. A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 1996. this issue.

[Tan84] R. M. Tanner. Explicit concentrators from generalized n-gons. *SIAM Journal Alg. Disc. Meth.*, 5(3):287–293, September 1984.