Linear-Time Encodable and Decodable Error-Correcting Codes

Daniel A. Spielman*
MIT

Abstract

We present a class of asymptotically good error-correcting codes that can be both encoded and decoded in linear sequential time and logarithmic parallel time with a linear number of processors. We present both randomized and explicit constructions of these codes. An important step in our construction is the introduction of *error-reducing codes*: codes that have a decoder that can very quickly remove a constant fraction of the errors from a corrupted codeword.

1. Introduction

Error-correcting codes were developed to enable communication over noisy channels. The situation is described by Figure 1, which appears in the beginning of almost every book on error-correcting codes.

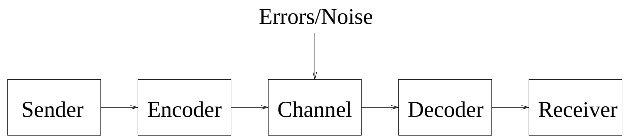

Figure 1: *Communication over a noisy channel.*

The sender passes a message to the encoder, which then adds redundant information to the message to form a codeword. Distinct codewords should differ in many places so that, if the channel corrupts a codeword in a only few places, the decoder will still be able to determine which codeword was sent.

It is standard to assume that the channel creates errors in a transmitted message by flipping some of the bits in the message. A widely studied channel is the *binary symmetric channel*, in which the channel flips each bit independently with a fixed probability $p < 1/2$. We will say that a bit has been "corrupted" if it has been flipped by the channel.

Formally, we define an error correcting code of *block-length* n, *rate* $r < 1$, and *relative minimum distance* δ to be the image of a function $f\colon \{0,1\}^{rn} \to \{0,1\}^n$ 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 (the number of bits in which u and v differ). The codewords of the code are the individual images under f of $\{0,1\}^{rn}$.

Encoding is the process of mapping $x \in \{0,1\}^{rn}$ to $f(x)$, and decoding is the process of mapping some $u \in \{0,1\}^n$ to the x which minimizes $d(f(x), u)$. If a code has relative minimum distance δ, then there is a decoding algorithm that will produce x from a corrupted version of $f(x)$ in which fewer than $\delta n/2$ bits have been flipped. Of course, some decoding algorithms will be able to correct more errors than others. We will say that a decoding algorithm can correct ϵn errors if, for all $u \in \{0,1\}^n$ and an $x \in \{0,1\}^{rn}$ such that $d(f(x), u) < \epsilon n$, the decoding algorithm outputs x on input u.

Coding schemes that use this type of code are called *block codes*: when a long message is to be sent, it is first divided into pieces of length rn. Each piece is encoded to form a block of length n. These blocks are then sent over the channel.

A central problem of coding theory is to find families of codes for which δ and r remain constant as the block-length grows. Such families are called *asymptotically good.* Asymptotically good families are desirable because they enable one to encode data in fewer blocks of larger size, thereby decreasing the probability that

*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.

an uncorrectable number of errors will occur in any one block. No additional communication costs are incurred because the rate of the codes remains constant even as the block-length grows. However, the amount of computation needed to encode and decode a block code usually grows disproportionately with the length of the code. In this paper, we present the first asymptotically good family of error-correcting codes in which the work needed to encode and decode the codes is strictly proportional to the length of the codes.

1.1. Previous work

Early formalizations of the problem of finding computationally efficient error-correcting codes can be found in [Ziv67, BZP77, Sav71].

Certain Reed-Solomon and Goppa codes can be encoded in roughly $O(n \log n)$ time and decoded in roughly $O(n \log^2 n)$ time [Jus76, Sar77]. While these codes are not necessarily asymptotically good, they can be concatenated with Justesen codes to obtain asymptotically good codes with similar encoding and decoding times. Other good coding schemes that have been more efficient in one operation have suffered in the other.

Gelfand, Dobrushin and Pinsker [GDP73] presented a randomized construction of asymptotically good error-correcting codes that could be encoded in linear time. They did not suggest an algorithm for decoding their codes.

Zyablov and Pinsker [ZP76] and Kuznetsov [Kuz73] showed that it is possible to decode Gallager's randomly chosen low-density parity-check codes [Gal63] in logarithmic time with a linear number of processors. Since these codes are linear, they can be encoded by performing a binary matrix-vector multiplication. We are not aware of a more efficient means of encoding these codes.

Sipser and Spielman [SS94] used expander graphs to construct error-correcting codes that can be decoded in linear sequential time and logarithmic parallel time with a linear number of processors. These "expander codes" are linear codes so they can be encoded using quadratic work. They presented both randomized and explicit constructions of their codes. We build on the techniques developed in their paper.

1.2. Superconcentrator codes

The codes that we present in this paper have encoding circuits that strongly resemble superconcentrators.[1] Accordingly, we will refer to them as "superconcentrator codes". The intuition behind why we can decode these codes efficiently is that each of the expander graphs used in the construction of the "superconcentrators" can be decoded using the expander code decoding algorithms of [SS94].

All the codes that we discuss in this paper will be systematic linear codes. This means that the codewords can be divided into rn "message bits" and $(1-r)n$ "parity bits". The message bits contain the message that the sender wants to encode, so there are no restrictions on their values. The parity bits contain the redundant information that the encoder adds to the message bits, and are uniquely determined by the message bits. They are called parity bits because each is determined by some linear combination over $GF(2)$ of the message bits. These codes are *linear* because the set of codewords form a vector space over $GF(2)$.

1.3. Outline of paper

We begin in Section 2 by explaining why linear-size encoding circuits must look like superconcentrators. In Section 3, we introduce the concept of an error-reduction code and present randomized constructions of these codes that can be encoded and decoded in linear-time. These codes are used in Section 4 to construct superconcentrator codes. We conclude in Section 5 by presenting explicit constructions of linear-time encodable and decodable error-reduction codes. We then show how to use these to construct linear-time encodable and decodable error-correcting codes.

2. Motivating the construction

It is not an accident that our linear-time encoding circuits look like superconcentrators. They have to. In this section, we will explain why. While this fact was the motivation behind our construction, one should be able to understand our construction without reading this section.

Consider a circuit C that takes as input some bits $x_1, \ldots, x_m$, and produces as output bits $p_1, \ldots, p_n$ such that the words $x_1, \ldots, x_m, p_1, \ldots, p_n$ form a good error-correcting code. This means that there is a constant δ such that even if we erase[2] any δm of the input bits and any δn of the output bits, we can still recover the erased δm input bits. We will show that this means that there

[1] No knowledge of superconcentrators is assumed in this paper. If the reader would like to learn more about superconcentrators, we recommend reading [Pip77].

[2] Usually, we discuss errors in which a bit's value is flipped. It is also possible to consider the case in which no value is received for some bit. In this case, the decoder knows that the value of the bit has been lost. These errors are called *erasure errors*. Any error-correcting code that can tolerate the first type of error can also tolerate erasure errors. If one is uncomfortable with erasure errors, then just flip a coin and assign its value to each erased bit. One will probably be left with half as many errors of the first type.

must be gate-disjoint paths from the erased inputs to some subset of the un-erased outputs.

Assume that we cannot find δm vertex-disjoint paths from the erased inputs to the un-erased outputs. Then, Menger's Theorem implies that there is some set of $\delta m - 1$ 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 δm bits of information in the erased input gates, but only $\delta m - 1$ 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 δm input gates into any $(1-\delta)n$ output gates. While this property is not quite as strong as the property required of superconcentrators, it is sufficiently close that we decided that the easiest way to create linear-size encoding circuits would be to base them on Valiant's [Val76] construction of linear-size superconcentrators.

3. Error-reducing codes

In this section, we introduce the concept of an error-reducing code. While we are not sure whether this idea will have practical applications, it will be useful for understanding our main construction.

An error-reducing code has the property that if a decoder receives a partially corrupted codeword, then it can correctly compute a large fraction of its message bits. The fraction of the message bits that the decoder cannot compute should be smaller than the fraction of the bits of the codeword that were corrupted in transmission. One could imagine using such a code if one is encoding data that is already encoded by an error-correcting code. In such a situation, it might only be necessary to remove most of the errors from a message.

We will define an *error-reducing code* of rate $r < 1$, error-reduction $\epsilon < 1$, and reducible distance $\delta < 1$ to be the image of a function

$$f : \{0,1\}^{rn} \rightarrow \{0,1\}^{n}$$

such that there exists a function

$$g : \{0,1\}^{n} \rightarrow \{0,1\}^{rn}$$

such that for all $x \in \{0,1\}^{rn}$ and $z \in \{0,1\}^{n}$ such that $d(f(x), z) \leq \delta$, we have $d(x, g(z)) < \epsilon d(f(x), z)$. f is the encoding function, and g is the decoding (error-reduction) function.

We will now modify an idea from [SS94] to construct an error-reducing code. Let B be an unbalanced bipartite expander graph with n nodes of degree d on one side and $n/2$ nodes of degree $2d$ on the other (we will hereafter call such a graph a $(d, 2d)$-regular graph). We will turn this graph into a circuit by directing all the edges from the large side of the graph to the small side, letting the nodes on the large side be the input gates, and letting the nodes on the small side be parity gates (see Figure 2). These parity gates are the outputs of the circuit.

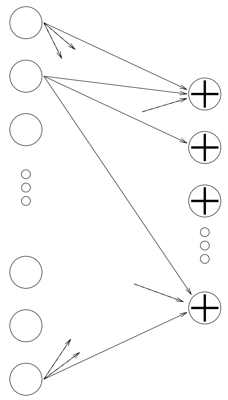

Figure 2: *A circuit that encodes an error-reducing code.*

We now use this circuit to define a code $\mathcal{R}_{n,d}$ of rate $2/3$ by placing the message bits at the inputs to the circuit and using the outputs of the parity gates as the parity bits. The code we thereby obtain is a horrible error-correcting code: it has a word of weight $d+1$ (see Figure 3). However, if the graph is a good expander

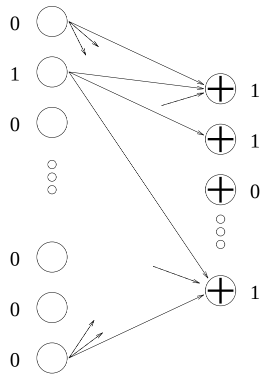

Figure 3: *A low-weight codeword: only one input bit is 1, and only those parity gates that read this bit are 1; all others are 0.*

graph, then this code is a good error-reducing code. It is obvious that we can encode this code in linear time (we need merely compute the values of the parity gates, each of which has a constant number of inputs). We now will show that it is possible to perform error-reduction on this code in linear time.

The decoder naturally associates each bit of the word that it receives with one of the gates in the encoding circuit. We will say that a parity gate in the circuit is *satisfied* by a word if the bit associated with the gate is the parity of the bits associated with its inputs. Otherwise, we will say that it is *unsatisfied* (see Figure 4). The decoder will successively flip the bits associated with the input gates in an effort to decrease the number of unsatisfied parity gates.

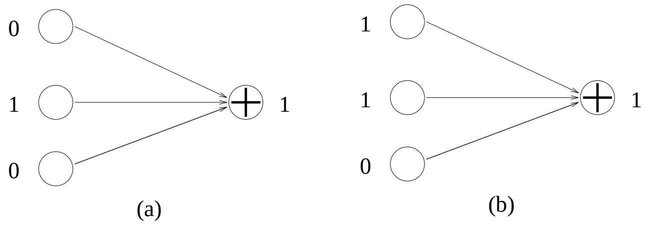

Figure 4: *(a) is a satisfied parity gate. (b) is unsatisfied.*

Sequential error-reduction algorithm:

- If there is an input that is an input to more unsatisfied than satisfied parity gates, then flip the value of that input.
- Repeat until no such inputs remain.

We note that it is easy to implement this algorithm so that it takes constant time for each iteration. At each iteration, it decreases the total number of unsatisfied parity gates, so it can run for at most a linear number of iterations.

Lemma 1. *Let $\mathcal{R}_{n,d}$ be derived from a degree $(d, 2d)$ bipartite graph between a set of n inputs and $n/2$ parity gates such that all sets of at most αn inputs expand by a factor of at least $\left(\frac{3}{4}d+1\right)$. Assume that the sequential error-reduction algorithm is given a word that resembles a codeword of $\mathcal{R}_{n,d}$ except that at most $\alpha n/2$ of the inputs have been corrupted and at most $\alpha n/2$ of the gates have been corrupted. Then, after the termination of the sequential error-reduction algorithm, at most $\alpha n/4$ of the inputs will be corrupted.*

Proof: We will let V denote the corrupted inputs, v the size of V, u the number of unsatisfied parity gates with inputs in V, and s the number of satisfied parity gates with inputs in V. We will view the pair (u, v) as the state of the algorithm. We will first show that if $\alpha n/4 \leq v \leq \alpha n$, then there is some input in more unsatisfied than satisfied parity gates. The expansion of the graph implies that

$$u + s \geq \left(\frac{3}{4}d + 1\right) v. \tag{1}$$

Each gate with an input in V accounts for at least one wire leaving V. It is possible that as many as $\alpha n/2$ of the satisfied parity gates with inputs in V are satisfied because they have only one wire from V, but they have been corrupted. The rest must have two wires from V. By counting the dv wires leaving V, we obtain

$$dv \geq u + s + (s - \alpha n/2) \Rightarrow dv + \alpha n/2 \geq u + 2s \tag{2}$$

Combining equations (1) and (2), we find

$$\begin{aligned} s &\leq \left(\frac{1}{4}d - 1\right) v + \alpha n/2, \qquad \text{and} \\ u &\geq \left(\frac{1}{2}d + 2\right) v - \alpha n/2. \end{aligned} \tag{3}$$

When $\alpha n \geq v \geq \alpha n/4$, we have $u > dv/2$, so there must be some input in more unsatisfied than satisfied parity gates.

To show that the algorithm must terminate with $v \leq \alpha n/4$, we show that v must always be less than αn. We assume that when the algorithm begins $v \leq \alpha n/2$ and therefore $u \leq d\alpha n/2 + \alpha n/2$. 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 equation (3) would imply that $u \geq d\alpha n/2 + 3\alpha n/2$, which would be a contradiction.

Thus, the algorithm must always maintain the condition that $v < \alpha n$. This implies that the algorithm cannot terminate unless it is in a state in which $v < \alpha n/4$. □

We have proved that $\mathcal{R}_{n,d}$ is an error-reducing code of rate 2/3 and error-reduction 1/2.

It is also possible to perform error-reduction on this code in parallel.

Parallel error-reduction algorithm

- For each input, count the number of unsatisfied parity gates to which it is an input.
- For each input that is an input to more unsatisfied than satisfied parity gates, flip the value of that input.

It is easy to implement this algorithm as a constant-depth circuit of linear size.

Lemma 2. *Let $\mathcal{R}_{n,d}$ be derived from a degree $(d, 2d)$ bipartite graph between a set of n inputs and $n/2$ parity gates such that all sets of at most αn inputs expand by a factor of at least $\left(\frac{3}{4}+\epsilon\right) d$, for any $\epsilon > 2/d$. Assume that the parallel error-reduction algorithm is given a word that resembles a codeword of $\mathcal{R}_{n,d}$ except that at most $v \leq \alpha n/2$ of the inputs have been corrupted, and at most $b \leq \alpha n/2$ of the gates have been corrupted. Then, after the execution of the parallel error-reduction algorithm, at most*

$$\frac{v + \frac{b}{d}(6 + 8\epsilon)}{1 + 4\epsilon}. \tag{4}$$

of the inputs will be corrupted.

Proof: We will let V denote the set of corrupted inputs, F the set of corrupted inputs that fail to flip, and C the set of inputs that were originally clean, but which become corrupted by the parallel error-reduction algorithm. We will let $N(V)$, the neighbors of V, denote the set of parity gates that contain inputs in V. We will let $v = |V|$, $\phi v = |F|$, $\gamma v = |C|$, and $\delta dv = |N(V)|$.

We begin by obtaining a bound on ϕ in terms of δ. Every input in F is an input to at least as many satisfied as unsatisfied parity gates. At most b of these satisfied parity gates are satisfied because they have been corrupted. Thus, at least $\frac{d\phi v}{2} - b$ of the wires leaving F end in a parity gate that contains a wire from another element of V. Thus, the set V can have at most

$$dv - \frac{d\phi v}{4} + \frac{b}{2}$$

neighbors. Because we have set the number of neighbors of V to be δdv, we obtain

$$\begin{aligned} \delta dv &\leq dv - \frac{d\phi v}{4} + \frac{b}{2} \\ \frac{dv\phi}{4} &\leq (1-\delta)dv + \frac{b}{2} \\ \phi &\leq 4(1-\delta) + \frac{2b}{dv} \end{aligned} \tag{5}$$

Next, we will bound γ in terms of δ by showing that

$$\gamma < \frac{\delta - \left(\frac{3}{4}+\epsilon\right) + \frac{b}{dv}}{\frac{1}{4}+\epsilon}. \tag{6}$$

Assume by way of contradiction that this is false, and let C' be a subset of C of size exactly $\frac{\delta - \left(\frac{3}{4}+\epsilon\right) + \frac{b}{dv}}{\frac{1}{4}+\epsilon}$. Each input in C' is an input to at least $d/2$ unsatisfied parity gates. At most b of these gates can be unsatisfied because their parity gate has been corrupted. The others must be unsatisfied because they have an input in V. Thus, the number of parity gates containing inputs in the set $V \cup C'$ is at most

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

We will set $\epsilon > 2/d$, which will imply that $|C' \cup V| < \alpha n$, so this set must expand by a factor of at least $\left(\frac{3}{4}+\epsilon\right)$. Thus, we obtain

$$\left(\frac{3}{4}+\epsilon\right)|C' \cup V| < d\delta v + \frac{d}{2}|C'| + b,$$

which contradicts our assumption about the size of C'.

By combining equations (5) and (6), we find that the number of corrupted inputs after an execution of the parallel error-reduction algorithm is at most

$$\begin{aligned} (\phi+\gamma)\,v &\leq v\frac{\delta - \left(\frac{3}{4}+\epsilon\right) + \frac{b}{dv}}{\frac{1}{4}+\epsilon} + 4(1-\delta) + \frac{2b}{d} \\ &= \frac{v\left(\frac{1}{4}+3\epsilon-4\epsilon\delta\right) + \frac{b}{d} + \left(\frac{1}{4}+\epsilon\right)\frac{2b}{d}}{\frac{1}{4}+\epsilon} \\ &< \frac{v + \frac{b}{d}(6+8\epsilon)}{1+4\epsilon}. \end{aligned}$$

□

We can iterate this algorithm a constant number of times in order to reduce the number of corrupted input bits.

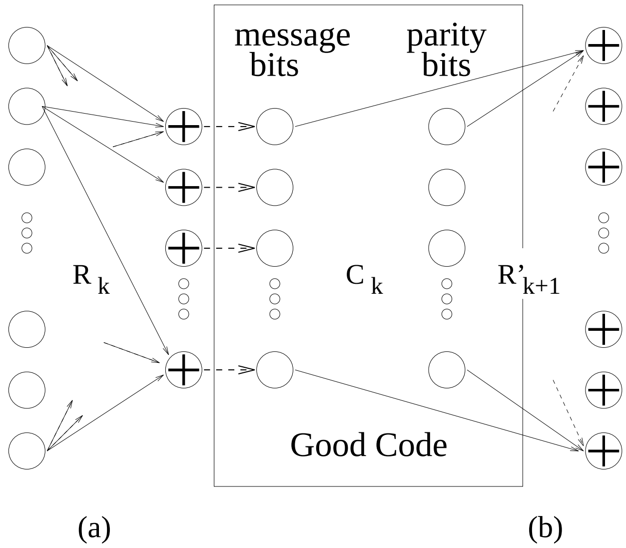

Figure 5: *The recursive construction of C_k. (a) is the error-reducing code on the new message bits. (b) is the error-reducing code placed on top of the error-correcting code.*

4. Error-correcting codes

The main idea in our construction of linear-time encodable and decodable error-correcting codes is to use the linear-time encodable and decodable error-reducing codes recursively. Imagine what happens when we take a good error-correcting code of length n and use the words of this code as the message bits of an error-reducing code (See Figure 5, part (b)). We now have an error-correcting code of greater length with a higher error-tolerance; however, it has the same number of message bits as the original code. To increase the number of message bits in the code, we will create a new set of more message bits, and encode these message bits in an error-reduction code. We then use the parity bits of this error-reduction code as the message bits of the error-correcting code we just constructed (See Figure 5).

When we apply this construction recursively, we find that not only have we defined the error-correcting code, but we have created a linear-size circuit that encodes the error-correcting code. Moreover, the output of each gate of this circuit is a parity bit of the error-correcting code. This will be true with one small exception: we should choose a better base case for our recursion. The performance of our code will be dictated by the quality of the code that we choose as our base. Thus, we may want to choose a particularly good code of small block length. Another constraint is that we don't know that we will be able to find good expander graphs of small size. However, there definitely is some constant size after which we will be able to find good expander graphs, and which we will choose to be the block length of our base code.

We will now present a formal description of one family of superconcentrator codes. We provide this description by describing the encoding circuits for these codes.

Description of superconcentrator codes:

- Choose absolute constants b and d.
- Choose a code $\mathcal{C}_b$ of length $4 \cdot 2^b$, rate 1/4, and as large minimum distance as possible.
- Let C_b be a circuit that takes 2^b bits as input and produces $3 \cdot 2^b$ bits as output so that these bits taken together form a codeword of $\mathcal{C}_b$. (The 2^b inputs bits are the message bits of the code, and the others are the parity bits.)
- For $k > b$, let R_k be a circuit that encodes an error-reduction code of rate 2/3 with 2^k message bits and $2^k/2$ parity bits, as described in Section 3.
- To form circuit C_k from C_{k-1}, take a copy of R_k and use the inputs of R_k as the inputs of C_k. Add C_{k-1} to the circuit by identifying the output gates of R_k with the inputs of C_{k-1}. Finally, attach a copy of R_{k+1} that we will call R'_{k+1} by identifying all the input and output gates of the copy of C_{k-1} with the inputs of R'_{k+1}. The output gates of C_k will be all the input and output gates of C_{k-1} along with the output gates of R'_{k+1} (See Figure 5).
- Let $\mathcal{C}_k$ be the rate 1/4 code obtained by taking 2^k message bits, feeding them into C_k, and using the $3 \cdot 2^k$ output bits as the parity bits of the code.

We can decode these codes in linear sequential time.

Sequential superconcentrator code decoding algorithm:

- If $k = b$, then decode C_b using an arbitrary decoding algorithm.
- If $k > b$, then apply the sequential error-reduction algorithm to the nodes in the R'_{k+1}. Now, recursively decode the nodes in the copy of C_{k-1} using the sequential decoding algorithm for C_{k-1}. Finish by applying the sequential error-reduction algorithm to the copy of R_k.

Theorem 3. *If the superconcentrator code $\mathcal{C}_k$ is constructed from degree $(d, 2d)$ graphs such that in each graph, every at most α fraction of inputs expands by a factor of at least $(\frac{3}{4}d + 1)$, and if $\mathcal{C}_b$ is chosen to be a code in which any $\alpha/2$ fraction of errors can be corrected, then the sequential superconcentrator code decoding algorithm will correct up to an $\alpha/8$ fraction of errors and will run in linear time.*

Proof: We assume that there are at most $\alpha 2^k/2$ errors in the nodes of C_k. By Lemma 1, after we apply the sequential error-reduction algorithm R'_{k+1}, there will be at most $\alpha 2^k/4$ errors in the nodes of the copy of C_{k-1}. We can now assume by induction that the decoding algorithm for C_{k-1} will correct all the errors in its input and output nodes. As the input nodes of C_{k-1} are now the parity bits of the error-reduction code $\mathcal{R}_k$ corresponding to the message originally contained at the inputs of the copy of R_k, and there are at most $\alpha 2^k/2$ errors in these message bits, we can use the sequential error-reduction algorithm to correct all the errors in the input nodes of R_k (This can be easily observed from the proof of Lemma 1, or from the analysis of the sequential expander code decoding algorithm of [SS94]). Since the inputs of R_k are the inputs of C_k, we have removed all the errors from the message bits of the code.

To see that this algorithm runs in linear time, observe that each error-reduction step runs in time that is linear in the number of bits that it is acting on, and each step acts on half as many bits as the previous step did.

$\square$

It is easy to see that there is a constant α that satisfies the requirements of Theorem 3, but we will not attempt to optimize the constant in this paper. We will note that the main constraint on the constant is in the analysis of the quality of expansion obtained by a randomly chosen graph. The best analysis of this expansion of which we are aware appears in the appendix of [SS94].

Remark 4. *We have only constructed codes of lengths 2^k where k is an integer. It is easy to use similar techniques to construct codes of other lengths.*

Remark 5. *We remind the reader that we have only shown that this algorithm corrects some constant fraction of errors. It does not produce the codeword closest to an arbitrary word, and we suspect that an efficient algorithm that does this would be difficult to construct.*

We need to be slightly trickier to decode the superconcentrator codes in parallel logarithmic time. The problem that we must overcome is that if we iterate the parallel error-reduction algorithm enough times to remove all the errors from the input bits of C_i, we will need to go through $O(i)$ iterations. If we did this for each C_i, then we would have an $O(log^2 n)$ time algorithm. To overcome this problem, we will perform the error-reductions from the input bits of C_{i-1} to the input bits of C_i *simultaneously* for all i. Thus, while the input gates of C_{i-1} are being used to reduce the errors in the input gates of C_i, the input gates in C_i are being used to reduce the errors in the input gates of C_{i+1}.

In order to show that the reduction of errors of the bits of C_i using the output bits of R'_{i+1} works, we will assume $\epsilon = \frac{4}{d} + \epsilon'$, for $\epsilon' > 0$ and $d > 16$. We now wish to observe that if $b \le \alpha n/2$, and $\alpha n/4 \le v \le \alpha n/2$, then after one round of the parallel error-reduction algorithm, v will decrease by a constant multiplicative factor. From equation (4), we can see that this constant factor will be bounded by the decrease that occurs when $v = \alpha n/4$ and $b = \alpha n/2$. By plugging these values into equation (4), we can see that this constant is less than 1. We now know that if $v \le \alpha n/2$ and $b \le \alpha n/2$, then after a constant number of rounds, we will have $v \le \alpha n/4$. Let c be this constant.

To prove the correctness of the error-reductions on the input nodes of the C_i's, we will assume that there is a $w \le \alpha n/2$ such that $v \le w$ and $b \le w/2$. By substituting into equation (4), we find that after one decoding round the number of corrupted inputs is bounded by

$$w\left(\frac{1+\left(\frac{3}{d}+\frac{16}{d^2}+\frac{4\epsilon'}{d}\right)}{1+\frac{16}{d}+4\epsilon'}\right) \quad < \quad w\left(1-\frac{\frac{11}{d}}{1+\frac{16}{d}+4\epsilon'}\right).$$

Let $\gamma = \left(1-\frac{\frac{11}{d}}{1+\frac{16}{d}+4\epsilon'}\right)$.

We can now state the parallel superconcentrator code decoding algorithm.

Parallel superconcentrator code decoding algorithm:

- For $i = k-1$ to b: Apply c rounds of the parallel error-reduction algorithm using the inputs and outputs of C_i as the message bits, and the outputs of R'_{k+1} to which they are attached as the parity bits.
- Decode the errors in C_b using any decoding algorithm.
- For $\log_{1/\gamma} 2^k$ rounds: Apply the parallel error-reduction algorithm to the copy of R_i between the inputs of C_i and the inputs nodes of C_{i+1}, simultaneously for all $b \le i \le k-1$.

Theorem 6. *If the superconcentrator code $\mathcal{C}_k$ is constructed from degree $(d, 2d)$ graphs such that in each graph, every at most α fraction of inputs expands by a factor of at least $(\frac{3}{4}+\frac{4}{d}+\epsilon')d$, for some $\epsilon' > 0$ and $d > 16$, and if $\mathcal{C}_b$ is chosen to be a code in which any $\alpha/2$ fraction of errors can be corrected, then the parallel superconcentrator code decoding algorithm will correct up to an $\alpha/8$ fraction of errors in logarithmic time with a linear number of processors.*

Proof: We begin by assuming that there are at most $\alpha 2^k/2$ errors in the bits of C_k. After we apply c rounds of the parallel error-reduction algorithm to R_{k+1}, there will be at most $\alpha 2^k/4$ errors in the bits of the copy of C_{k-1}. Similarly, after we have finished the i-th stage of the algorithm, there will be at most $\alpha 2^k/2^{i+1}$ errors in the bits of C_{k-i}. Thus, C_b will have fewer than $\alpha 2^b/2$ errors, so the decoding algorithm for C_b will correct all the errors in C_b.

We can now move on to the decoding of the input bits of the C_i's. We have already observed that the input bits of C_i have at most $\alpha 2^i/2$ errors and that the input bits of C_b are free of error. Thus, after we apply one round of the error-reduction algorithm simultaneously to all of the R_i's, the input bits of C_i will have at most $\gamma\alpha 2^i/2$ errors. Similarly, after we apply the error-reduction algorithm for $\log_{1/\gamma} 2^k$ rounds, there will be no more errors in any of the input nodes of C_k. □

5. Explicit Constructions

The obstacle to obtaining explicit constructions of the superconcentrator codes described in Section 4 is that we are unaware of any explicit constructions of degree d expander graphs that have expansion greater than $d/2$, let alone the $3d/4$ that is required for these constructions.[3] To get around this problem, we will modify our construction so that a lower level of expansion will suffice. This means of obtaining explicit constructions is analogous to that which appeared in [SS94].

We will begin by saying a little about what types of expander graphs we can construct, and then showing how to produce explicit constructions of error-reduction codes from them.

Definition 7. We will say that a family of graphs $\mathcal{G}$ is a *family of good expander graphs* if $\mathcal{G}$ contains graphs $G_{n,d}$ of n nodes and degree d so that

- For an infinite number of values of d, there exists an infinite number of graphs $G_{n,d} \in \mathcal{G}$, and

[3] For a proof that Ramanujan graphs have expansion approaching $d/2$, see [Kah93].

- for each of these values d, the second-largest eigenvalues of $G_{n,d}$ are bounded from above by constants λ_d such that $\lim_{d \to \infty} \lambda_d / d = 0$.

The expander graphs constructed by Lubotzky, Phillips and Sarnak [LPS88] and by Margulis [Mar88] are such a family. Pippenger [Pip93] points out that one can also obtain such a family by exponentiating the expander graphs of Gabber and Galil [GG81].

From a d-regular graph G on n vertices, we will derive a $(2, d)$-regular graph with $dn/2$ vertices on one side, and n vertices on the other.

Definition 8. 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

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

We can now define the circuits, $R(G, \mathcal{C})$, that will encode our explicit error-reduction codes.

Explicit error-reduction codes:

- Let G be a d-regular graph on n vertices, and let $\mathcal{C}$ be a linear error-correcting code of block-length l with d message bits. Let B be the edge-vertex incidence graph of G.
- The circuit $R(G, \mathcal{C})$ has $dn/2$ input gates and $(l - d)n$ parity gates.
- Each node on the large side of B will correspond to an input of $R(G, \mathcal{C})$. The parity gates are arranged into clusters of size $l - d$, and each cluster is identified with one of the nodes on the small side of B. The input gates that are neighbors of a cluster will be called the inputs of the cluster.
- The parity gates are connected to the input gates so that for each cluster, if the inputs of that cluster are the message bits of a codeword of $\mathcal{C}$, then the parity gates in the cluster compute the parity bits of that codeword.
- Let $\mathcal{R}(G, \mathcal{C})$ denote the code obtained by using the inputs of $R(G, \mathcal{C})$ as message bits and the outputs of the circuit as parity bits.

To prove that $\mathcal{R}(G, \mathcal{C})$ is a good error-reduction code if G is a good expander graph, we will make use of the following result due to Alon and Chung:

Lemma 9 ([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*

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

To perform error-reduction on $\mathcal{R}(G, \mathcal{C})$, we will associate each bit of a received word with an input or gate of $R(G, \mathcal{C})$, as we did in Section 3.

Parallel explicit error-reduction algorithm:

- In parallel, for each cluster, if the bits associated with the inputs and gates of a cluster are within $\frac{\epsilon}{6}$ of a codeword of $\mathcal{C}$, then send a "flip" message to every input that needs to be flipped to obtain that codeword.
- In parallel, every input that receives the message "flip", flips its value.

We can now prove a lemma for our explicit construction that is analogous to Lemma 2.

Lemma 10. *Let $\{G_{n,d}\}$ be a family of good expander graphs. There exist constants d and g such that if the parallel explicit error-reduction algorithm is given as input a word that resembles a codeword of $\mathcal{R}(G_{n,d}, \mathcal{C})$ except that for some $w \leq \frac{\epsilon^2}{g} n \frac{d}{2}$*

- *at most $\frac{w}{2} \leq v \leq 2w$ inputs are corrupted and at most $b \leq w$ parity gates are corrupted, then after the execution of the algorithm, the number of corrupted inputs will decrease by a constant multiplicative factor, and*
- *if at most $v \leq \frac{w}{2}$ of the inputs are corrupted and at most $b \leq w$ of the parity gates are corrupted, then after the execution of the algorithm, at most $\frac{w}{2}$ will be corrupted.*

Proof: Let V be the set of v corrupted inputs. Set α and β so that $v = \alpha n \frac{d}{2}$ and $b = \beta n \frac{d}{2}$.

We will say that a cluster is *confused* if it sends a "flip" message to an input that is not corrupt. In order for a cluster to be confused, it must have at least $\frac{5\epsilon}{6} d$ corrupt inputs and gates. Thus, there can be at most

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

confused clusters. Each of these can send at most $\frac{1}{6} \epsilon d$ "flip" messages, so at most

$$\frac{n \frac{d}{2} (2\alpha + \beta)}{\frac{5}{6} \epsilon d} \cdot \frac{1}{6} \epsilon d = \frac{n \frac{d}{2} (2\alpha + \beta)}{5}$$

uncorrupted inputs can receive "flip" signals.

We will call a cluster *unhelpful* if it has a node of V among its inputs, but it fails to send a "flip" signal to that node. There can be at most

$$\frac{n \frac{d}{2} (2\alpha + \beta)}{\frac{1}{6} \epsilon d}$$

unhelpful clusters. By Lemma 9, there can be at most

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

inputs both of whose neighbors are unhelpful clusters.

The total number of corrupted inputs after the algorithm is run will be at most

$$n\frac{d}{2}\left(\frac{2\alpha+\beta}{5}+\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d}\right).$$

Set $w=\frac{\epsilon^2}{g}n\frac{d}{2}$. We want to show that

$$\frac{2\alpha+\beta}{5}+\left(\frac{6\alpha+3\beta}{\epsilon}\right)^2+\left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d}<\alpha,$$

for $\beta\in[0,\frac{\epsilon^2}{g}]$ and $\alpha\in[\frac{\epsilon^2}{2g},\frac{2\epsilon^2}{g}]$. Because the function in consideration is decreasing in β, and quadratic in α with positive coefficients, it suffices to consider the function at the points $\alpha\in\left\{\frac{\epsilon^2}{2g},\frac{2\epsilon^2}{g}\right\}$ and $\beta=\frac{\epsilon^2}{g}$. For $\alpha=\frac{\epsilon^2}{2g}$, we need

$$\frac{\epsilon^2}{10g}>\left(\frac{6\epsilon}{g}\right)^2+\frac{6\epsilon}{g}\cdot\frac{\lambda_d}{d},$$

and for $\alpha=\frac{2\epsilon^2}{g}$, we need

$$\frac{\epsilon^2}{g}>\left(\frac{15\epsilon}{g}\right)^2+\frac{15\epsilon}{g}\cdot\frac{\lambda_d}{d}.$$

We now see that, given ϵ, we can choose g and then d so that both of these inequalities hold. For lower values of w, the claim follows from a similar analysis.

To prove the second claim, it suffices to observe that the function in question is decreasing in α for $\alpha>0$. □

We will find it convenient to let $\mathcal{C}$ be a code of length $5d/4$ and rate $4/5$. By the Gilbert-Varshmov bound [MS77], there exists a code with length $\frac{5}{4}d$, rate $\frac{4}{5}$, and minimum distance ϵ where $H(\epsilon)\geq 1/5$ (where $H(x)=-x\log_2 x-(1-x)\log_2(1-x)$ denotes the binary entropy function).

To construct superconcentrator codes as we did in Section 4, we will need error-reduction codes of very specific sizes. To produce these, we first observe that explicit constructions of good expander graphs meet the following definition.

Definition 11. Let $\mathcal{G}=\{G_i\}_i$ be a sequence of graphs such that G_i has n_i vertices, and $n_{i+1}>n_i$. We say that $\mathcal{G}$ is *dense* if $n_{i+1}-n_i=o(n_i)$.

One can observe that by exponentiating the expander graphs of Gabber and Galil, one obtains a dense family of expander graphs. The results of [Hei33] can be used to prove that the expander graphs constructed by Lubotzky, Phillips and Sarnak [LPS88] and by Margulis [Mar88] are dense.

We can use a dense family of d-regular good expander graphs to construct a family of good expander graphs of every sufficiently large number of vertices. Let G be a d-regular graph on n vertices. Let S be an independent subset of the vertices of G such that no two vertices in S have a common neighbor. Sets with this property can be found in any size up to $n/(d^2+1)$. Consider the graph obtained by removing the vertices of S from G. Every node has degree d or $d-1$, and the induced subgraphs in this graph are all induced subgraphs of G. The fact that the vertices in these graphs have degree either d or $d-1$ has only a negligible impact on the analysis of the error-reduction codes that we derive from them. Thus, we will assume hereafter that we can obtain good error-reduction codes of any sufficiently large size, and that they perform as described in Lemma 10.

Explicit superconcentrator codes:

- Choose a family of good expander graphs, $\mathcal{G}=\{G_{n,d}\}$ such that the subfamily for each d contains a dense subsequence.
- Choose a constant d and a code C_d of rate $4/5$, length $5d/4$ and minimum relative distance ϵ where $\epsilon\geq H(1/5)$. (As described in the previous paragraph, we will assume that $\mathcal{G}$ contains a graph of every sufficiently large number of vertices in which every vertex has degree d or $d-1$.)
- Choose a constant b and a code $\mathcal{C}_b$ of length $4\cdot 2^b$, rate $1/4$, and as large minimum distance as possible. Let C_b be a circuit that takes 2^b bits as input and produces $3\cdot 2^b$ bits as output so that these bits taken together form a codeword of $\mathcal{C}_b$. (The 2^b inputs bits are the message bits of the code, and the others are the parity bits.)
- For $k>b$, let R_k be the circuit $R(G_{2^k,d},\mathcal{C}_d)$ defined earlier in this section. We have chosen the code $\mathcal{C}_d$ so that this circuit has 2^k input gates and $2^k/2$ parity gates (If the graph isn't d-regular, then we might have to modify the codes at the clusters slightly to make sure that we have $2^k/2$ parity gates, but this effect is insignificant).
- To form circuit C_k from C_{k-1}, take a copy of R_k and use the inputs of R_k as the inputs of C_k. Add C_{k-1} to the circuit by identifying the output gates of R_k with the inputs of C_{k-1}. Finally, attach a copy of R_{k+1} that we will call R'_{k+1} by identifying all the input and output gates of the copy of C_{k-1} with the inputs of R'_{k+1}. The output gates of C_k will be all the input and output gates of C_{k-1} along with the output gates of R'_{k+1}.

- Let $\mathcal{C}_k$ be the code obtained by taking 2^k message bits, feeding them into C_k, and using the $3 \cdot 2^k$ output bits as the parity bits of the code.

Lemma 10 implies that there is a constant c such that if $v \leq w$ and $b \leq w$, where $w \leq 2^k \epsilon^2/g$, then after c executions of the parallel explicit error-reduction algorithm, we will have at most $w/2$ corrupted inputs. Similarly, there exits a constant $\gamma < 1$ such that if $v \leq 2w$ and $b \leq w$, then after the execution of the error-reduction algorithm there will be at most γv corrupted inputs.

Parallel explicit error-correction algorithm:
• Identical to the algorithm presented in Section 4, but using the definitions of C_k and R_k presented in this section.

Theorem 12. *There exist settings of the parameters of the constructions of the rate* $1/4$ *superconcentrator codes* $\mathcal{C}_k$ *presented in this section, as well as settings of* ϵ *and* g*, such that the parallel explicit superconcentrator code decoding algorithm will correct up to an* $\epsilon^2/4g$ *fraction of errors in logarithmic time with a linear number of processors. Moreover, this algorithm can be simulated in linear sequential time.*

Proof: [Sketch] The first claim follows from the discussion in this section and the proof of Theorem 6. By keeping track of which clusters of parity gates are "unsatisfied", it is fairly simple to simulate this algorithm in linear time on a sequential machine (assuming that pointer references have unit cost). The idea is similar to that used in Theorem 9 of [SS94]. □

6. Open question

There are many ways to vary the parameters of our constructions. However, we are not aware of any way of modifying our construction to produce codes that approach the Gilbert-Varshmov bound. We would like to know if it possible to produce linear-time encodable and decodable error-correcting codes that can be decoded up to the Gilbert-Varshmov bound.

7. Acknowledgments

I would like to thank Michael Sipser for many helpful conversations. I would like to thank Brendan Hassett for pointing me to [Hei33].

References

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

[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.

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

[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.

[Hei33] Hans Heilbronn. Uber den primzahlsatz von Herrn Hoheisel. *Mathematische Zeitschrift*, 36:394–423, 1933.

[Jus76] Jorn Justesen. On the complexity of decoding Reed-Solomon codes. *IEEE-TIT*, 22(?):237–238, March 1976.

[Kah93] Nabil Kahale. On the second eigenvalue and linear expansion of regular graphs. In *Proc. of the 33rd FOCS*, pages 296–303, 1993.

[Kuz73] A. V. Kuznetsov. Information storage in a memory assembled from unreliable components. *Problems of Information Transmission*, 9(3):254–264, 1973.

[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.

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

[Pip93] Nicholas Pippenger. Self-routing superconcentrators. In *Proc. of the 25th STOC*, pages 355–361, 1993.

[Sar77] Dilip V. Sarwate. On the complexity of decoding Goppa codes. *IEEE-TIT*, 23(?):515–516, July 1977.

[Sav71] John E. Savage. The complexity of decoders–part II: Computational work and decoding time. *IEEE-TIT*, 17(1):77–85, January 1971.

[SS94] Michael Sipser and Daniel A. Spielman. Expander codes. In *Proc. of the 35th FOCS*, pages 566–576, 1994.

[Val76] L. G. Valiant. Graph-theoretic properties in computational complexity. *Journal of Computer and System Sciences*, 13:423–432, 1976.

[Ziv67] Jacob Ziv. Asymptotic performance and complexity of a coding scheme for memoryless channels. *IEEE-TIT*, 13(3):356–359, July 1967.

[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.