Linear-Time Encodable and Decodable Error-Correcting Codes

Submitted to *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*

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. This resolves a long-standing open problem in the theory of error-correcting codes.

*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.

1. Introduction

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. This resolves a long-standing open problem in the theory of error-correcting codes [BZP77, Sav71].

1.1. Error-correcting codes

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.

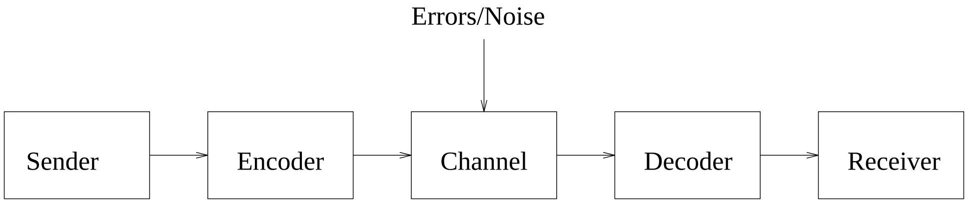

Figure 1: Communication over a noisy channel.

The sender passes a message to the encoder, which then adds redundant information to the message to form a codeword. The codewords should all be far apart 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.

Formally, we define an error correcting code of *block-length* n, *rate* $r < 1$, and *relative minimum distance* δ to be 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, *i.e.,* the number of bits in which u and v differ. The codewords of the code are the 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 $u \in \{0,1\}^n$ to the x which minimizes $d(f(x), u)$. The relative minimum distance δ implies that it is possible to correctly decode a corrupted version of $f(x)$ in which fewer than $\delta n/2$ bits have been changed.

A central problem of coding theory is to find families of codes for which δ and r remain constant as n grows large. Such families are called *asymptotically good.* Asymptotically good families are desirable because they enable one to encode data in fewer blocks of larger size, which decreases the probability that an uncorrectable number of errors will occur in any one block.

As channels become faster and codes of longer block-length are used, it becomes increasingly important to have encoding and decoding procedures that operate efficiently. If one uses encoding and decoding algorithms with super-linear complexity, then their complexity will eventually become the dominant term in the cost of sending a message. In fact, error-correcting codes with very efficient encoders and decoders are have often been implemented in favor of slightly less efficient codes with better rate-to-minimum-distance characteristics [Ima90].

1.2. Previous work

BCH codes, Reed-Solomon codes, and related asymptotically good concatenated codes can be both encoded and decoded in roughly $O(n \log n \log^* n)$ time [Bla83, page 340]. Other coding schemes that have been more efficient in one operation have suffered in the other.

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

Zyablov and Pinsker [ZP76] 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 (a somewhat more efficient algorithm for decoding these codes was presented in [SS94]). 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 either in sequential quadratic time or parallel logarithmic time with a quadratic number of processors. They presented both randomized and explicit constructions of their codes. While these codes should be useful for storing data on write-once media, their quadratic encoding time makes them unsuitable for use in on-line communications.

1.3. 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]. The decoding process is complicated by the fact that the expander graphs in superconcentrator codes do not correspond exactly to expander codes: the devices that enable one to decode an expander code correspond to parity bits in a superconcentrator code. Thus, one superconcentrator code decoding step is analogous to decoding an expander code using faulty hardware. Only by carefully orchestrating the decoding steps are we able to prevent these "faults" from propagating.

In Section 2, we present a randomized construction of superconcentrator codes, as well as efficient sequential and parallel algorithms for encoding and decoding these codes. In Section 3, we present our explicit constructions and their corresponding encoding and decoding algorithms. Our exposition follows the outline of [SS94]: for each expander code decoding algorithm, we present an analogous superconcentrator code decoding algorithm, which uses that expander code decoding algorithm as a subroutine.

2. The randomized construction

The codes that we construct will be linear codes over $GF(2)$. The encoding circuits will consist only of parity gates. Moreover, (almost) every gate in the encoding circuits will be an output gate. That is, the parity bits of the codeword corresponding to a given input will be the outputs of (almost) every gate in the encoding circuit when it is given the message bits as input.

We now define the codes by describing the encoding circuits. For simplicity, we will only build circuits that encode inputs of size 2^k. It is easy to use similar techniques to build codes of intermediate block-lengths. When describing the construction, we will write *(d,2d)-regular expander graph* to mean a bipartite expander graph that is d-regular on one side and $2d$-regular on the other. The quality of the expansion that is required will be discussed in Sections 2.1 and 2.2. We will write C_k to denote the circuit that encodes inputs of length 2^k and we we will write $\mathcal{C}_k$ to denote the corresponding code. Our construction will be recursive: we will use C_{k-1} to build C_k. Our base

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

case will be C_b for some $b > 0$. We will defer an examination of how we should choose C_b until the end of this section. We will also defer a description of how to choose the constant d that will appear throughout the construction.

The nodes of C_k will be divided into three sets called *input nodes*, *internal nodes*, and *external nodes*. There will be 2^k input nodes, $2 \cdot 2^k$ internal nodes, and 2^k external nodes. As each node corresponds to a bit in the codeword, this circuit will produce a code of block-length $4 \cdot 2^k$ with 2^k message bits and $3 \cdot 2^k$ redundant bits. We will explain in Section 4 how to obtain codes of different rates. We will draw the circuit as in Figure 2.

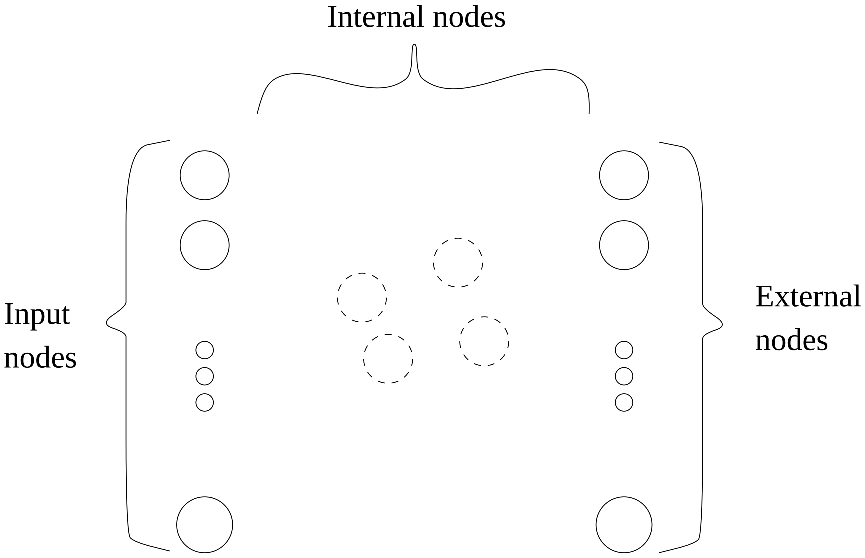

Figure 2: The circuit

When we construct circuit C_{k+1} from circuit C_k, all the nodes of C_k will become the internal nodes of C_{k+1}, and we will add 2^{k+1} new input nodes and 2^{k+1} new external nodes. The 2^{k+1} new input nodes will be connected to the 2^k input nodes of C_k by a $(d, 2d)$-regular expander graph. Each of the 2^k input nodes of C_k will become a parity gate that computes the parity of the new input nodes that are its neighbors. The external nodes of C_{k+1} will be connected to the input, internal and external nodes of C_k by another $(d, 2d)$-regular expander graph (See Figure 3). The external nodes of C_{k+1} will also be gates that compute the parities of their neighbors.

We will now discuss some of the considerations that go into choosing the constant d as well as the base circuit C_b. Other considerations can only be properly understood after one has read Sections 2.1 and 2.2.

The expander graphs needed for this construction must be very good expanders: the $(d, 2d)$-regular expanders need expansion by a factor of at least $\frac{3}{4}d$ on sets comprising a constant fraction of the vertices on the large side of the graph. We know that randomly chosen graphs have this quality of expansion, but these graphs must be fairly large before we know that we can find one with high probability. Thus, we must choose b to be large enough that we can find expanders of sufficiently high quality.

Having chosen b, we can decide on an appropriate C_b. To do this, we find a linear error-correcting code $\mathcal{C}$ of block-length $4 \cdot 2^b$, rate $\frac{1}{4}$, and as large minimum distance as possible. We could either choose $\mathcal{C}_b$ at random, or by consulting the extensive error-correcting code literature. Because $\mathcal{C}$ is a linear code, there is a circuit of size $3 \cdot (2^b)^2$ that will compute the redundant bits from the message

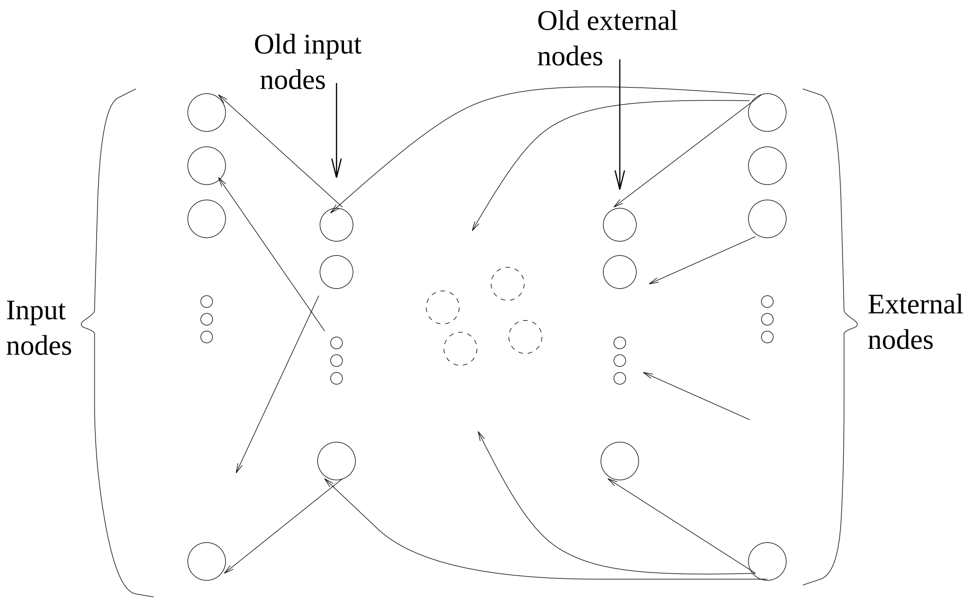

Figure 3: The recursive construction

bits.[2] We let this circuit be C_b. The output gates of C_b, those that compute the redundant bits, will be viewed as the internal and external nodes of C_b. *The other gates in C_b should be ignored by the rest of the construction.* That is, no gate in C_k, other than those in C_b, should have as input any gate of C_b that is not an output gate. Thus, the external nodes of C_{b+1} will be connected to the output nodes of C_b only. This is the exception alluded to in the first paragraph of the section by the phrase "(almost) every". Having said this, we have completed our description of the superconcentrator codes $\mathcal{C}_k$ and their encoding circuits C_k. From this discussion, it should be clear that these codes can be encoded by a linear-size circuit of logarithmic depth.

2.1. Sequential Decoding

To describe the process of decoding, we will assume that some input to a circuit C_k was chosen, that C_k computed the codeword w corresponding to the input, that this word was transmitted over a noisy channel, and that the decoder receives a message w' that differs from w in at most $\epsilon 4 \cdot 2^k$ places, for some constant ϵ. Decoding is the process whereby the decoder reconstructs the original message, w.

Each bit of w' is naturally associated with an input or a gate of C_k (except for those gates of C_b that are ignored by the rest of the circuit, and which we will ignore hereafter). To each parity gate of C_k we associate a parity check that is "satisfied" by w' if the bit that w' assigns to the parity gate is the parity of the bits that w' assigns to the inputs of the gate. Otherwise, we say that the parity check is "unsatisfied".

The reader should notice that the circuit C_k is filled with $(d, 2d)$-regular expander graphs. If, for one of these graphs, we were guaranteed that all the nodes on the small side were free of errors, and that not too many errors occurred on the large side, then we could correct the errors using the sequential expander code decoding algorithm of [SS94]. In Lemma 1, we will prove that even if some nodes on the small side have errors, most of the errors on the large side can still be corrected.

We recall the sequential expander code decoding algorithm of [SS94]:

[2] It is often possible to choose a code with a smaller encoding circuit.

Sequential expander code decoding algorithm

- If there is a variable in more unsatisfied than satisfied parity checks, then flip the value of that variable.
- Update the status of the parity checks that contain that variable.
- Repeat until no such variables remain.

Lemma 1. *[Faulty decoding lemma] Let G be 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 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 expander code decoding 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 checks containing elements of V, and s the number of satisfied parity checks containing elements of V. We will view the pair (u, v) as being 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 checks. The expansion of the graph implies that

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

Each neighbor of V accounts for at least one edge leaving V. It is possible that as many as $\alpha n/2$ of the satisfied parity checks containing V are satisfied because they have only one edge from V, but they have been corrupted. The rest must have two edges from V. By counting the dv edges leaving V, we obtain

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

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

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

Thus 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 checks.

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 $u \geq \frac{1}{2}d\alpha n + \frac{3}{2}\alpha n$, 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 where $v < \alpha n/4$. $\square$

We can now state the sequential superconcentrator code decoding algorithm for decoding C_k.

Sequential superconcentrator code decoding algorithm

- If $k = b$, then decode C_b using an arbitrary decoding algorithm.
- If $k > b$, then apply the sequential expander code decoding algorithm to the nodes in the $(d, 2d)$-regular expander graph between the external nodes and the internal nodes of C_k. Now, recursively decode the internal nodes of C_k using the sequential decoding algorithm for C_{k-1}. Finish by applying the sequential expander code decoding algorithm to the $(d, 2d)$-regular expander graph between the input and the internal nodes of C_k.

Theorem 2. *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 variables 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 in linear time.*

Proof: We assume that there are at most $\alpha 2^k/2$ errors in the input nodes, external nodes and internal nodes of C_k. By Lemma 1, after we apply the sequential expander code decoding algorithm to the graph between the internal and the external nodes, there will be at most $\alpha 2^k/4$ errors in the internal nodes. We can now assume by induction that the decoding algorithm for C_{k-1} will correct all the errors in the internal nodes. After this algorithm is run, we can assume that there are no errors in the internal nodes. As the input nodes of C_{k-1} are now parities of the original message contained at the input nodes of C_k, and there are at most $\alpha 2^k/2$ errors in these input nodes, we can use the sequential expander code decoding algorithm to correct all the errors in the input nodes of C_k (See [SS94] for proof). $\square$

It is easy to see that a constant α that satisfies the requirements of Theorem 2 exists, 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].

2.2. Parallel Decoding

Before we present our parallel algorithm for decoding superconcentrator codes, we will prove a lemma concerning the parallel expander code decoding algorithm of [SS94] that is analogous to Lemma 1. This algorithm proceeds in rounds. We first recall one round of the algorithm.

One round of the parallel expander code decoding algorithm

- For each variable, count the number of unsatisfied parity checks that the variable is in.
- For each variable that is in more unsatisfied than satisfied parity checks, flip that variable.
- Update the status of the parity checks that contain that variable.

Lemma 3. *Let G be 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 at most $v \le \alpha n/2$ of the inputs have been corrupted, and at most $b \le \alpha n/2$ of the gates have been corrupted. Then, after one round of the parallel expander code decoding 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 after one decoding round, and C the set of inputs that were originally clean, but which become corrupted after one decoding round. We will let $N(V)$, the neighbors of V, denote the set of parity checks that contain inputs in V. We will let $v = |V|$, $\phi v = |F|$, $\gamma v = |C|$, and $\delta d v = |N(V)|$.

We begin by obtaining a bound on ϕ in terms of δ. Every variable in F is contained in at least as many satisfied as unsatisfied parity checks. At most b of these satisfied parity checks are satisfied because their parity gate has been corrupted. Thus, at least $\frac{d\phi v}{2} - b$ of the edges leaving F land in a parity check that contains an edge 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

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

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

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

Assume by way of contradiction that this is false, and let C' be a subset of C of size exactly $\frac{\delta - (\frac{3}{4}+\epsilon) + \frac{b}{dv}}{\frac{1}{4}+\epsilon}$. Each variable in C' is in at least $d/2$ unsatisfied parity checks. At most b of these checks can be unsatisfied because their parity gate has been corrupted. The others must be unsatisfied because they contain a variable of V. Thus, the number of parity checks 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 one decoding round is at most

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

□

Having proved Lemma 3, we are now ready to describe the parallel superconcentrator code decoding algorithm. The first few rounds of the algorithm will be similar to the initial steps of the sequential decoding algorithm: we will use the external gates of C_i to reduce the number of errors in C_{i-1}. Lemma 3 implies that this can be done in a constant number of rounds for each i. We will then use any algorithm to decode C_b. We would then like to decode the rest of the code by using the input gates of C_{i-1} to correct all the errors in the input gates of C_i for each i. However, this could take a logarithmic number of rounds for each large i. To get around this problem, we will perform all decoding between levels of input gates *simultaneously*. Thus, while the input gates of C_{i-1} are being used to correct the errors in the input gates of C_i, the input gates in C_i are being used to correct the errors in the input gates of C_{i+1}.

In order to show that the decoding with the external nodes works, we will set $\epsilon = \frac{4}{d} + \epsilon'$, for $\epsilon' > 0$ and $d > 16$. We now wish to observe that if $b \leq \alpha n/2$, and $\alpha n/4 \leq v \leq \alpha n/2$, then after one round of the parallel expander code decoding 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 $b = \alpha n/2$. We now prove that this constant factor is less than 1 by plugging these values into equation (4) to learn that after one decoding round with $v = \alpha n/4$ and $b = \alpha n/2$, the number of corrupted variables is at most

$$\begin{aligned}\frac{\alpha n}{2}\left(\frac{\frac{1}{2}+\frac{1}{d}\left(6+\frac{32}{d}+8\epsilon'\right)}{1+\frac{16}{d}+4\epsilon'}\right) &= \frac{\alpha n}{4}\left(\frac{1+\frac{12}{d}+\frac{64}{d^2}+\frac{16\epsilon'}{d}}{1+\frac{16}{d}+4\epsilon'}\right)\\ &< \frac{\alpha n}{4}\left(1-\frac{3\epsilon'}{1+\frac{16}{d}+4\epsilon'}\right),\end{aligned}$$

because $d > 16$. We now know that if $v \leq \alpha n/2$ and $b \leq \alpha n/2$, then after a constant number of rounds, we will have $v \leq \alpha n/4$. Let c be this constant.

To prove the correctness of the decoding with the input nodes, we will assume that we have $v \leq w$ and $b \leq w/2$ for some $w \leq \alpha n/2$. By substituting into equation (4), we find that after one decoding round the number of corrupted variables is bounded by

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

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 $i = k$ to $b+1$: Apply c rounds of the parallel expander code decoding algorithm to the expander graph between the external nodes of C_i and the internal nodes of C_i.
- Decode the errors in C_b using any decoding algorithm.
- For $\log_{1/\gamma} 2^k$ rounds: Apply the parallel expander code decoding algorithm to the expander graph between the input nodes of C_i and the input nodes of C_{i+1}, simultaneously for all $b \leq i \leq k-1$.

Theorem 4. *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 variables 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 external and input nodes of C_k and at most $\alpha 2^k/2$ errors in the internal nodes of C_k. After we apply c rounds of the parallel expander code decoding algorithm to the expander graph between the external and internal nodes of C_k, there will be at most $\alpha 2^k/4$ errors in the internal nodes. Similarly, after we have finished the first stage of the algorithm, there will be at most $\alpha 2^k/2^{i+1}$ errors in the nodes 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 nodes. We have already observed that the input nodes of C_i have at most $\alpha 2^i/2$ errors. Thus, after we apply one round of the parallel expander code decoding algorithm to all of the graphs between the input nodes, the input nodes of C_i will have at most $\gamma\alpha 2^i/2$ errors. Similarly, after we apply the expander code decoding algorithm for $\log_{1/\gamma} 2^k$ rounds, there will be no more errors in any of the input nodes of C_k. $\square$

3. Explicit Constructions

As in [SS94], we will create explicit constructions of superconcentrator codes by using subcodes that are more complex than the parity checks used in Section 2. This will enable us to use the expanders for which we have explicit constructions, which are weaker than the randomly chosen expanders needed for the construction in Section 2. Where, in Section 2, we formed expander graphs between nodes and parity gates, we will now form expander graphs between nodes and clusters of parity gates. We will associate a small error-correcting code with each cluster. The input nodes that are the neighbors of the cluster will contain the message bits of a codeword in that code and the parity gates in the cluster will compute the corresponding parity bits.

There are an immense number of ways that one can vary the parameters in such a construction. For simplicity we will restrict ourselves to one particular type of construction in this draft.

We use the following definition to indicate when we have a family of expanders that is sufficiently rich for our construction:

Definition 5. We will say that $\mathcal{G}$ is a *dense family of expander graphs* if $\mathcal{G}$ contains graphs $G_{n,d}$ of n nodes and degree d such that

- There exist graphs $G_{n,d} \in \mathcal{G}$ for an infinite number of values of d.
- for each d, the second-largest eigenvalues of $G_{n,d}$ are bounded from above by constants λ_d such that $\lim_{d\to\infty} \lambda_d/d = 0$, and
- for each d there exists an $\epsilon > 0$ such that for every graph $G_{n,d} \in \mathcal{G}$ there is another graph $G_{m,d} \in \mathcal{G}$ such that $m > n$ and $m - n = O(n^{1-\epsilon})$.

The expander graphs constructed by Lubotsky, 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 such a family of graphs $\mathcal{G}$, we create a family of unbalanced bipartite expander graphs $\mathcal{B}$ that contains a graph $B_{n,d}$ for each graph $G_{n,d} \in \mathcal{G}$. $B_{n,d}$ is a graph with $n\frac{d}{2}$ vertices of degree 2 on the large side and n vertices of degree d on the small side. The nodes on the large side represent the edges of $G_{n,d}$ and the nodes on the small side represent its vertices. A node representing an edge is connected to a node representing a vertex if the vertex is an end-point of that edge. This is a special case of a construction introduced in [AKS87], and the more general construction can be used to obtain superconcentrator codes with different characteristics.

We will make use of the following lemma due to Alon and Chung:

Lemma 6 ([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*

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

In our construction, we will pick one value of d and make use of the graphs $B_{n,d}$ throughout the construction. The density condition ensures that for each graph used in the construction, there is another graph in the family with roughly twice the number of vertices. To make our construction analogous to the construction in Section 2, we will use subcodes of rate $\frac{4}{5}$. Thus, each cluster will have $\frac{d}{4}$ parity gates that will compute the parity bits corresponding to the d inputs entering the cluster. This will simplify the analysis because the number of gates in the graph will be equal to half the number of inputs. 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 $H(\epsilon) < 1/5$ (where $H(\cdot)$ denotes the binary entropy function).

Our decoding algorithm for an individual graph will be similar to that used in Section 4 of [SS94].

One parallel decoding round for explicit expander codes

- In parallel, for each cluster, if the setting of the inputs in that cluster is within $\frac{\epsilon}{6}$ of a codeword, 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 3.

Lemma 7. *There exists a dense family of expander graphs $\mathcal{G}$ and a derived family of unbalanced expander graphs $\mathcal{B}$ and constants ϵ, g, and d and an error-correcting code $\mathcal{C}$ of length $\frac{5}{4}d$ and minimum relative distance ϵ such that if $B_{n,d} \in \mathcal{B}$ is a bipartite graph from $n\frac{d}{2}$ inputs of degree 2 to n clusters of degree d such that each cluster contains $\frac{d}{4}$ parity gates that compute the parity bits of $\mathcal{C}$ from the inputs that are their neighbors and if $w \leq \frac{\epsilon^2}{g} n\frac{d}{2}$ then*

- *if at most $\frac{w}{2} \leq v \leq 2w$ inputs are corrupted and at most $b \leq w$ parity gates in the clusters are corrupted, then one parallel decoding round will decrease the number of corrupted inputs 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 one decoding round the number of corrupted inputs will be at most $\frac{w}{2}$.*

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 6, 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.

Thus, the total number of corrupted inputs after one decoding round will be at most

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

We note that this function decreases as α decreases, and that the ratio of corrupted inputs before a decoding round to after a decoding round is minimized when α is as small as it can be and β is as large as it can be. Thus, we will substitute in $\beta = \epsilon^2/g$ and $\alpha = \beta/2$ and show that

$$\begin{aligned} \frac{2\alpha+\beta}{5} + \left(\frac{6\alpha+3\beta}{\epsilon}\right)^2 + \left(\frac{6\alpha+3\beta}{\epsilon}\right)\frac{\lambda_d}{d} &< \alpha \quad \Leftrightarrow \\ \frac{\beta}{2} - \frac{2\beta}{5} - \left(\frac{6\beta}{\epsilon}\right)^2 - \left(\frac{6\beta}{\epsilon}\right)\frac{\lambda_d}{d} &> 0 \quad \Leftrightarrow \\ \frac{\epsilon^2}{10g} &> \frac{36\epsilon^2}{100g^2} + \frac{6\epsilon}{10g}\cdot\frac{\lambda_d}{d}. \end{aligned}$$

We now see that we can choose g and then d so that this inequality holds. □

We now proceed by analogy with Section 2.2. For a more detailed argument, see Appendix A.

Theorem 8. *There exists an infinite family of easily computable asymptotically good error-correcting codes that can be encoded and decoded in linear sequential time and logarithmic parallel time with a linear number of processors.*

Proof: [Sketch] Our construction is like the construction in Section 3 except that we use the parameters and graphs discussed in this section. The one catch is that we are not guaranteed that we will always be able to find an expander graph that is twice the size of the previous graph. However, because we have a dense family of expander graphs, we can find a graph that is just slightly larger. The impact of this difference on the analysis is negligible (although it might mean that our base circuit has to be really big so that we can use the asymptotics in the definition of dense).

Lemma 7 implies that there is a constant c such that if $v \leq w$ and $b \leq w$, where $w \leq \epsilon^2/g$, then after c parallel decoding rounds 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 one parallel decoding round there will be at most γv corrupted inputs. Thus, to decode these codes in parallel, we use an algorithm identical to that of Section 2.2 except that we use the parallel decoding round for explicit expanders presented in this section.

We conclude by observing that it is possible to simulate this parallel decoding algorithm in sequential linear time. □

4. Conclusion

For simplicity, we have presented recursive constructions in which each level is twice the size of the preceding level. It is possible to vary the construction by changing these ratios, and even making them vary from one level to the next. It is not necessary that the relation between the internal and external nodes be the same as the relation between the input and internal nodes. By varying these parameters, one can vary the rate and error-correction capabilities of these codes. We would also like to note that the actual minimum distance achieved by the randomized construction probably meets the Gilbert-Varshmov bound [MS77]. However, the actual number of errors that we know how to effectively correct is much less than half this minimum distance.

5. Acknowledgement

I would like to thank Michael Sipser for many helpful conversations.

References

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

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

[Bla83] Richard E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1983.

[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.

[Ima90] Hideki Imai. *Essentials of Error-Control Coding Techniques*. Academic Press, Inc., San Diego, CA, 1990.

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

[Mar88] G. A. Margulis. Explict 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 *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 355–361, 1993.

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

[SS94] Michael Sipser and Daniel A. Spielman. Expander codes. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 566–576, 1994.

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

A. Details of the explicit construction

We begin with any dense family of expander graphs, $\mathcal{G}$. We then take the derived family of bipartite expander graphs $\mathcal{B}$ as described in Section 3. Next, we find parameters ϵ and d and an error-correcting code $\mathcal{C}$ that satisfy the conditions of Lemma 7.

We will now describe the encoding circuits of our explicit codes. We will call the kth encoding circuit E_k and the associated code $\mathcal{E}_k$.

The circuit E_k will be composed of n_k input gates, i_k internal gates, and e_k external gates. We will arrange that $e_k = n_k$ and $i_k = n_{k-1} + i_{k-1} + e_{k-1}$. We want to choose the n_k's so that n_{k+1} is approximately twice n_k. That is, having chosen n_k, we choose a graph $B_{n,d} \in B$ with the maximal number of vertices such that $n\frac{d}{4} < n_k$. We then set $n_{k+1} = n\frac{d}{2}$. Because $\mathcal{G}$ is dense, we are guaranteed that n_{k+1} will not be much less than twice n_k (will can choose our base case large enough to insure this). We then group some $n\frac{d}{4}$ input nodes of E_k into clusters of size $d/4$ and connect the clusters to the n_{k+1} input nodes of E_{k+1} by the graph $B_{n,d}$. The nodes in the clusters will become parity gates that compute parities of input nodes that are their neighbors so that the input nodes of E_{k+1} that are neighbors of a cluster and the parity gates in that cluster form a codeword of the code $\mathcal{C}$.

When forming circuit E_{k+1}, we will set to zero the input nodes of E_k that are not in the clusters. We note that this only improves our ability to decode errors in E_k.

We must take similar measures when we choose the expander graph between the internal nodes of E_k and the external nodes of E_k. The internal nodes of E_k are the entire circuit of E_{k-1}, except that some of the input gates of E_{k-1} have been set to zero as was just described. We now need to choose another expander graph, $B_{m,k}$ such that $m\frac{d}{2} > i_k$. We then set $e'_k = m\frac{d}{4}$, and create e'_k external nodes for E_k which are connected to the internal nodes of E_k in the same fashion as the input nodes of E_k were connected to the input nodes of E_{k-1}. As there are more nodes on the large side of $B_{m,d}$ than there are internal nodes of E_k, we will arbitrarily identify the extra nodes with the constant zero. Again, we observe that this doesn't hurt our ability to decode using this expander.

We now have what appears to be a problem: n_{k+1} is slightly less than twice n_k, but e'_{k+1} will be slightly more than half i_{k+1}, and thus could be slightly more than twice n_k. If we were to let this continue, then the ratio of e'_k to n_k would grow to be unacceptably large. To get around this problem, we throw away $e'_k - n_k$ of the external nodes of E_k and set $e_k = n_k$. This doesn't hurt our analysis because we can make this number an arbitrarily small fraction of e'_k, and then we can treat the discarded gates as if they were faulty gates. One should note that in our decoding algorithm, we do not need to eliminate *all* the errors from any level of external gates, we just need to eliminate enough so that those gates can be used to halve the number of errors in the next level.

We choose our base case b large enough that we can find the graphs that we need to perform this construction. We then choose the code $\mathcal{E}_b$ to be a good code as we did in Section 2. As in Section 2, we will choose an encoding circuit E_b, and all the nodes of E_b except the nodes that correspond to bits in the code $\mathcal{E}_b$ will be ignored by the rest of the construction.

From this discussion, we see that $\mathcal{E}_k$ has rate $\frac{1}{4}$ and has an encoding circuit of linear size and logarithmic depth.

We now state the decoding algorithm for these codes (we choose c and γ as described in Section 3).

Parallel explicit superconcentrator code decoding algorithm

- For $i = k$ to $b+1$: Apply c rounds of the parallel expander code decoding algorithm to the expander graph between the external nodes of E_i and the internal nodes of E_i.
- Decode the errors in E_b using any decoding algorithm.
- For $\log_{1/\gamma} n_k$ rounds: Apply the parallel expander code decoding algorithm to the expander graph between the input nodes of E_i and the input nodes of E_{i+1}, simultaneously for all $b \leq i \leq k-1$.

Theorem 9. *If the superconcentrator code $\mathcal{E}_k$ is constructed as described in this section, then 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.*

Proof: We begin by assuming that there are at most $\epsilon^2 n_k/g$ errors in the input and external nodes of C_k and at most $\epsilon^2 n_k/g$ errors in the internal nodes of C_k. After we apply c rounds of the parallel expander code decoding algorithm to the expander graph between the external and internal nodes of E_k, there will be at most $\epsilon^2 n_k/2g$ errors in the internal nodes. Similarly, after we have finished the first stage of the algorithm, there will be at most $\epsilon^2 n_{k-i}/g$ errors in the nodes of E_{k-i}. Thus, E_b will have fewer than $\epsilon^2 n_b/g$ errors, so the decoding algorithm for E_b will correct all the errors in E_b.

We can now move on to the decoding of the input nodes. We have already observed that the input nodes of E_i have at most $\epsilon^2 n_i/g$ errors. Thus, after we apply one round of the parallel expander code decoding algorithm to all of the graphs between the input nodes, the input nodes of E_i will have at most $\gamma\epsilon^2 n_i/g$ errors. Similarly, after we apply the expander code decoding algorithm for $\log_{1/\gamma} n_k$ rounds, there will be no more errors in any of the input nodes of E_k. □

We now need to show that this parallel algorithm can be simulated on a sequential machine in linear time.

Theorem 10. *The parallel explicit superconcentrator code decoding algorithm can be simulated in sequential linear time.*

Proof: [Sketch] We begin by observing that it is easy to simulate the initial rounds in which the external nodes are used to decode errors: for each level the algorithm is only repeated c times, so linear work is being used at each level. As each level is smaller than the next by a constant factor, the total work is linear.

Decoding E_b requires only constant time because b is a constant.

To simulate the decoding of the input nodes is slightly more tricky. The key is that the algorithm should keep lists of which parity checks are unsatisfied on each level. It can then use this information to figure out which bits are going to be flipped after one decoding round. For each level i, the algorithm should update in two passes the list of which parity checks corresponding to gates in level i are unsatisfied. On the first pass, the algorithm should change the status of every parity check that is affected by the flipping of a bit in the input gates of E_{i+1}. It should then perform a pass in which it changes the status of every parity check in which the bit associated with the gate corresponding to the parity check is flipped. Because the graph has constant degree, the time required to simulate one round of this part of the parallel decoding algorithm will be linear in the number of unsatisfied parity checks. Because the number of unsatisfied parity checks decreases by a constant factor after each iteration, the total time required to simulate the algorithm will be linear. □