Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 13 (2020)

Linear-time Erasure List-decoding of Expander Codes *

Noga Ron-Zewi† Mary Wootters‡ Gilles Zémor§

May 24, 2021

Abstract

We give a linear-time erasure list-decoding algorithm for expander codes. More precisely, let $r > 0$ be any integer. Given an inner code $\mathcal{C}_0$ of length d, and a d-regular bipartite expander graph G with n vertices on each side, we give an algorithm to list-decode the code $\mathcal{C} = \mathcal{C}(G, \mathcal{C}_0)$ of length nd from approximately $\delta\delta_r nd$ erasures in time $n \cdot \text{poly}(d2^r/\delta)$, where δ and δ_r are the relative distance and the r'th generalized relative distance of $\mathcal{C}_0$, respectively. To the best of our knowledge, this is the first linear-time algorithm that can list-decode expander codes from erasures beyond their (designed) distance of approximately $\delta^2 nd$.

To obtain our results, we show that an approach similar to that of (Hemenway and Wootters, *Information and Computation*, 2018) can be used to obtain such an erasure-list-decoding algorithm with an exponentially worse dependence of the running time on r and δ; then we show how to improve the dependence of the running time on these parameters.

1 Introduction

In coding theory, the problem of *list-decoding* is to return all codewords that are close to some received word z; in *algorithmic list-decoding,* the problem is to do so efficiently. While there has been a great deal of progress on algorithmic list-decoding in the past two decades [Sud97, GS99, PV05, GR08, GW17, DL12, GX12, GX13, GK16, HRW20, KRR+21, KRSW18, GQST20, GR21], most work has relied crucially on algebraic constructions, and thus it is interesting to develop combinatorial tools to construct efficiently list-decodable codes with good parameters.

In this work, we consider the question of list-decoding *expander codes,* introduced by Sipser and Spielman in [SS96]. We define expander codes formally in Section 2, but briefly, the expander code $\mathcal{C}(G, \mathcal{C}_0)$ is a linear code constructed from a d-regular bipartite expander graph G,[1] and a linear inner code $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$. A codeword of $\mathcal{C}(G, \mathcal{C}_0)$ is a vector in $\mathbb{F}_2^{E(G)}$ which is a labeling of edges $E(G)$

*An extended abstract appeared at ISIT 2020.

†Department of Computer Science, University of Haifa. noga@cs.haifa.ac.il. Research supported in part by BSF grant 2017732 and ISF grant 735/20.

‡Department of Computer Science and Department of Electrical Engineering, Stanford University. marykw@stanford.edu. Research supported in part by NSF CAREER award CCF-1844628 and by NSF-BSF award CCF-1814629, as well as by a Sloan Research Fellowship.

§Institut de Mathématiques de Bordeaux, UMR 5251, Université de Bordeaux. zemor@math.u-bordeaux.fr. Research supported in part by the ANR project CBCRYPT: Project-ANR-17-CE39-0007.

[1] Following standard practice, we use the term 'd-regular expander graph' loosely, as a shortcut to mean a generic member of an infinite family of d-regular graphs for which the ratio $\lambda(G)/d$ is small, where $\lambda(G) := \max\{|\lambda_i|, |\lambda_i| \neq d\}$, see Section 2 for more details.

ISSN 1433-8092

in G. The constraints are that, for each vertex v of G, the labels on the d edges incident to v form a codeword in $\mathcal{C}_0$.

Expander codes are notable for their very efficient unique decoding algorithms [SS96, Zém01, LMSS01, SR03, AS06, BZ002, BZ05, BZ06, RS06, HOW15]. However, very little is known about the algorithmic list-decodability of expander codes, and it is an open problem to find a family of expander codes that admit fast, e.g., linear-time, list-decoding algorithms with good parameters. The goal of this paper is twofold. First, we aim to develop algorithms to erasure list-decode expander codes beyond the minimum distance of the code with small list size. Since a linear code can be erasure list-decoded in polynomial time by solving a linear system, this immediately implies polynomial-time algorithms for erasure list-decoding expander codes. Our second goal is to design *linear-time*[2] *algorithms* for erasure list-decoding expander codes, which are faster than the straightforward polynomial-time algorithm.

Erasure list-decoding. Erasure-list-decoding is a variant of list-decoding where the received word z may have some symbols which are "$\perp$" (erasures), and the goal is to recover all codewords consistent with z. More formally, let $\mathcal{C} \subseteq \mathbb{F}_2^N$ be a binary code of length N. For $z \in (\mathbb{F}_2 \cup \perp)^N$, define

$$\mathrm{List}_{\mathcal{C}}(z) := \{c \in \mathcal{C} \,:\, c_i = z_i \text{ whenever } z_i \neq \perp\}\,.$$

We say that $\mathcal{C}$ is *erasure-list-decodable* from e erasures with list size ℓ if for any $z \in (\mathbb{F}_2 \cup \{\perp\})^N$ with less than e symbols equal to $\perp$, $|\mathrm{List}_{\mathcal{C}}(z)| \leq \ell$.

Erasure list-decoding has been studied before [Gur03, GI02, GI04, GR06, DJX14, HW18, BDT20], motivated both by standard list-decoding and as an interesting combinatorial and algorithmic question in its own right. It is known that the erasure-list-decodability of a linear code is precisely captured by its generalized distances, defined as follows. For a linear subspace $V \subseteq \mathbb{F}_2^N$, we let N_V denote the number of coordinates which are not identically zero in V, i.e., the number of coordinates $i \in [N]$ for which there exists $v \in V$ with $v_i \neq 0$. The *r'th (relative)*[3] *generalized distance* δ_r of a linear code $\mathcal{C} \subseteq \mathbb{F}_2^N$ is defined as the minimum, over all r-dimensional linear subspaces $V \subseteq \mathcal{C}$, of the ratio $\frac{N_V}{N}$. That is,

$$\delta_r = \min_{V \subseteq \mathcal{C},\, \dim(V)=r} \frac{N_V}{N} = \min_{V \subseteq \mathcal{C},\, \dim(V)=r} \frac{\left|\{i \in [N] \,:\, \exists v \in V, v_i \neq 0\}\right|}{N}.$$

Thus, δ_1 coincides with the traditional *(relative) distance* δ of the code, which for linear codes equals the minimum relative weight of any nonzero codeword. The generalized distances of a linear code $\mathcal{C}$ characterize its erasure list-decodability:

Lemma 1.1 ([Gur03]). *Let $\mathcal{C} \subseteq \mathbb{F}_2^N$ be a linear code. Then $\mathcal{C}$ is erasure-list-decodable from e erasures with list size ℓ if and only if $\delta_r(\mathcal{C}) > e/N$, where $r = 1 + \lfloor \log_2(\ell) \rfloor$.*

As mentioned above, if $\mathcal{C}$ is linear, then it can be erasure list-decoded in polynomial time by solving a linear system. Thus, the combinatorial result of Lemma 1.1 comes with a polynomial-time algorithm.

[2] Following [SS96], when measuring running time, we consider the uniform cost computational model.

[3] Throughout this paper, we will work with the relative generalized distances (that is, measured as a fraction of coordinates). We will omit the adjective "relative" to describe these quantities in the future.

Main Results. Our main result is a linear-time erasure list-decoding algorithm for expander codes beyond the (designed) minimum distance.

Theorem 1.2. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ and r'th generalized distance δ_r. Let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices with $\lambda = \lambda(G) := \max\{|\lambda_i|, |\lambda_i| \neq d\}$. Let $\mathcal{C} = \mathcal{C}(G, \mathcal{C}_0)$, let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} \leq \frac{\varepsilon^2\delta^2}{2^{r+4}}$.*

Then there is an algorithm ListDecode *which, given a received word $z \in (\mathbb{F}_2 \cup \{\perp\})^E$ with less than $(1-\varepsilon)\delta\delta_r dn$ erasures, runs in time $n \cdot \mathrm{poly}\left(\frac{2^r d}{\varepsilon\delta}\right)$, and returns a matrix $\tilde{A} \in \mathbb{F}_2^{nd \times a}$ and a vector $\tilde{b} \in \mathbb{F}_2^{nd}$ such that $\mathrm{List}_{\mathcal{C}}(z) = \left\{\tilde{A}x + \tilde{b} : x \in \mathbb{F}_2^a\right\}$ where $a := \dim(\mathrm{List}_{\mathcal{C}}(z))$ satisfies $a \leq \frac{2^{2r+7}}{\varepsilon^4\delta^4}$.*

Because $\delta_r > \delta$ for any non-trivial linear code (any code of dimension > 1), the radius that Theorem 1.2 achieves is beyond the (designed) minimum distance of $\mathcal{C}$, which is approximately $\delta^2 dn$. To the best of our knowledge, this is the first linear-time list-decoding algorithm for expander codes that achieves this with a non-trivial list size.

In light of Lemma 1.1, the ultimate result we can hope for is an algorithm that list-decodes up to $\delta_r(\mathcal{C})$ fraction of erasures with list size 2^{r-1} for any $r \geq 1$. The quantity $\delta(\mathcal{C}_0) \cdot \delta_r(\mathcal{C}_0)$ in Theorem 1.2 may suggest it plays the role of a 'designed' r'th generalized distance, especially since for $r = 1$ it does (up to an ε term) coincide with the expander designed distance. However, we cannot expect $\delta(\mathcal{C}_0) \cdot \delta_r(\mathcal{C}_0)$ to be a general lower bound on the r'th generalized distance of an expander code, which implies in particular that the list-size in Theorem 1.2 has to be larger than 2^r. Indeed, already in the special case of *tensor codes* (i.e., when the graph G is the complete bipartite graph that has perfect expansion), the generalized distance has been shown [WY93, Sch00] to be a complicated quantity that can be lower than $\delta(\mathcal{C}_0) \cdot \delta_r(\mathcal{C}_0)$: in the general expander case, finding a reasonable description of the worst-case behavior of generalized distances seems quite challenging.

Note however that the results do imply a weak bound on the generalized distances of an expander code, namely that $\delta_r(\mathcal{C})$ is approximately at least $\delta(\mathcal{C}_0) \cdot \delta_{\Theta(\log r)}(\mathcal{C}_0)$. Moreover, for the special case of $r = 2$, we are able to show the following bound on the second generalized distance of an expander code.

Lemma 1.3. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ and second generalized distance δ_2, and let $G = (L \cup R, E)$ be a bipartite d-regular graph with $\lambda = \lambda(G)$. Let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} \leq \frac{\delta_2\delta^2\varepsilon^2}{16}$. Then the $\mathcal{C}(G, \mathcal{C}_0)$ has second generalized distance at least $(1-\varepsilon) \cdot \delta \cdot \min\{\delta_2, 2\delta\}$.*

Note that under the mild assumption that $\delta_2(\mathcal{C}_0) \leq 2\delta(\mathcal{C}_0)$ (satisfied by any code that has two minimum-weight codewords with disjoint support), the above lemma gives a lower bound of approximately $\delta_2(\mathcal{C}_0)\delta(\mathcal{C}_0)$ on the second generalized distance of expander codes.

Finally, note that while we do not know if the list size returned by the algorithm can be generally improved, the algorithm can still list-decode an expander code $\mathcal{C}$ from up to a $\delta_r(\mathcal{C})$ fraction of erasures with list size 2^{r-1} for some values of r: If r' is such that $\delta_{r'}(\mathcal{C}) < \delta(\mathcal{C}_0)\delta_r(\mathcal{C}_0)$ for some $r = O(1)$, the algorithm will run in linear time and return a list of size $2^{r'-1}$ given a $\delta_{r'}(\mathcal{C})$ fraction of erasures.

1.1 Technical Overview

In this section, we give a brief overview of the approach. The basic idea is similar to the approach in [HW18]; however, as we discuss more in Section 1.2 below, in that work the goal was *list-recovery,*

a generalization of list-decoding. In this work we can do substantially better by restricting our attention to list-decoding, as well as by tightening the analysis of [HW18].

Let $G = (L \cup R, E)$ be a bipartite d-regular expander graph, and let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ and r-th generalized distance δ_r. Since the inner code $\mathcal{C}_0$ is linear and has r'th generalized distance δ_r, there is an $O(d^3)$-time algorithm to erasure list-decode $\mathcal{C}_0$ from up to $\delta_r d$ erasures. The first step will be to do this at every vertex $v \in L \cup R$ that we can, to produce a list List_v at each such vertex.

In order to "stitch together" these lists, we define a notion of *equivalence* between edges, similar to the notion in [HW18]. Suppose that (u, v) and (w, v) are edges incident to a vertex v,[4] and that there is some $b \in \mathbb{F}_2$ such that for any $c \in \text{List}_v$, $c_{(u,v)} = b + c_{(w,v)}$. Then, even if we have not pinned down a symbol for (u, v) or (w, v), we know that for any legitimate codeword $c \in \text{List}_{\mathcal{C}}(z)$, assigning a symbol for one of these edges implies an assignment for the other. In this case, we say that $(u, v) \sim (w, v)$. Because the lists List_v are actually affine subspaces, there are not many equivalence classes at each vertex (and in particular substantially fewer equivalence classes than in the approach used in [HW18]).

With these equivalence classes defined, we actually give two algorithms, SlowListDecode and ListDecode. As the name suggests, SlowListDecode is a warm-up that has a worse dependence on ε, δ and r, but is easier to understand. We describe SlowListDecode (given in Section 3, Figure 2) here first, and then describe the changes that need to be made to arrive at the final algorithm, ListDecode (given in Section 4, Figure 4).

The main idea of SlowListDecode is to choose $s = \text{poly}(2^r, 1/\varepsilon, 1/\delta)$ large equivalence classes and generate a list of all 2^s possible labelings for these equivalence classes. For each such labeling, we now hope to uniquely fill in the rest of the codeword, to arrive at a list of size 2^s. One might hope that labeling these s large equivalence classes would leave a fraction of unlabeled symbols less than the designed distance of $\mathcal{C}$, allowing us to immediately use the known linear-time erasure unique decoding algorithm for the expander code. Unfortunately, this is not in general the case. However, we show that there are many vertices v such that the number of unlabeled edges incident to v is less than $\delta(\mathcal{C}_0)d$. Thus, we may run the unique decoder for $\mathcal{C}_0$ (in time $O(d^3)$) at each such vertex to generate yet more labels. It turns out that at this point, there *are* enough labels to run $\mathcal{C}$'s unique decoding algorithm and finish off the labeling.

Naively, the algorithm described above runs in time at least 2^s, since we must loop over all 2^s possibilities. This is exponential in ε and δ and doubly-exponential in r. The idea behind the final algorithm ListDecode is to take advantage of the linear structure of the lists List_v to find a short description of all of the legitimate labelings. We will show in Section 4 how to do this in time $n \cdot \text{poly}(d2^r/\varepsilon\delta)$ by leveraging the sparsity of $\mathcal{C}$'s parity-check matrix.

1.2 Related Work

Work on list-decoding expander codes. The work that is perhaps the most related to ours is [HW18], which seeks to *list-recover* expander codes in the presence of erasures in linear time.[5] List-recovery is a variant of list-decoding which applies to codes over a large alphabet Σ: instead of receiving as input a vector $z \in \{0, 1\}^N$, the decoder receives a collection of lists, $S_1, \ldots, S_N \subseteq \Sigma$,

[4] All graphs we consider will be undirected, and we will slightly abuse standard notation by writing (u, v) for the undirected edge between vertices u and v.

[5] We note that other works, such as [GI04], have also had this goal, but to the best of our knowledge [HW18] obtains the best known results, so we focus on that work here.

and the goal is to return all codewords $c \in \Sigma^N$ such that $c_i \in S_i$ for all i. In the setting of erasures, some lists have size $|\Sigma|$, in which case we may as well replace the whole list by a $\perp$ symbol.

List decoding from erasures is a special case of list-recovery with erasures, where the S_i that are not $\perp$ have size one. However, existing list-recovery algorithms will not immediately work in our setting, as we consider binary codes: list-recovery is only possible for codes with large alphabets.

The first observation is that the approach of [HW18] for erasure list-recovery can be used to obtain an algorithm for erasure list-decoding in linear time, even for binary codes. As described above, the first step is to erasure list-decode $\mathcal{C}_0$ at each vertex, leaving us with lists List_v that need to be "stitched together." The approach of [HW18] does precisely this, although in their context the lists that they are stitching together come from list-recovering the inner code.

However, the results of [HW18] about stitching together lists do not immediately yield anything meaningful for erasure list-decoding. More precisely, those results imply that the code $\mathcal{C}(G, \mathcal{C}_0)$ formed from a graph G with $\lambda = \lambda(G)$ and an inner code $\mathcal{C}_0$ with distance δ and r'th generalized distance δ_r is list-decodable from up to a $\delta\delta_r\left(\frac{\delta - \lambda/d}{6}\right)$ fraction of erasures in time $N \cdot \exp(\exp(\exp(r)))$. In particular, the fraction of erasures that those results tolerate is smaller than the distance of the expander code, yielding only trivial results in this setting.

Thus, while we use the same ideas as [HW18], the analysis is different and significantly tighter. This allows us to obtain a meaningful result in our setting, corresponding to the algorithm SlowList-Decode. Moreover, as described above, we are able to take advantage of the additional linear structure in our setting to improve the dependence on r in the running time.

To the best of our knowledge, there is no algorithmic work on list-decoding expander codes from errors (rather than erasures) in linear time with good parameters. We note that [MRR+20] recently showed that there are expander codes which are *combinatorially* near-optimally list-decodable from errors, but this work is non-constructive and does not provide efficient list decoding algorithms.

Work on erasure list-decoding more generally. It is known that, non-constructively, there are erasure-list-decodable codes of rate $\Omega(\varepsilon)$ which can list-decode up to a $1-\varepsilon$ fraction of erasures, with list sizes $O(\log(1/\varepsilon))$ [Gur03]. However, this proof is non-constructive and does not provide efficient linear-time algorithms, and it has been a major open question to achieve these results efficiently. Recent progress has been made by [BDT20], who provided a construction (although no linear-time decoding algorithm) with parameters close to this for ε which is polynomially small in n.

As to linear-time algorithms, [HW18] gave, for any $\rho \in (0, 1)$, an explicit construction of an erasure list-recoverable code of rate ρ which can be list-recovered from an $1 - \rho - \varepsilon$ fraction of erasures in linear-time, over large alphabets, and with large output list sizes (depending on ε and the input list size ℓ). One can use this latter construction with $\rho = 1 - \varepsilon$ as an outer code in a concatenated scheme, with a (non-explicit) binary code of rate $1-\varepsilon$ which can be list-decoded from an $\Omega(\varepsilon)$-fraction of errors as the inner code. This results in a binary erasure-list-decodable code of rate $1 - \varepsilon$ which can be list-decoded from up to a $\Omega(\varepsilon^2)$ fraction of erasures in linear time (with large list sizes, depending on ε).

Our work is somewhat orthogonal to the aforementioned work on erasure list-decoding since we are less concerned about achieving the best trade-off between rate, erasure tolerance, and list size, and more concerned with efficiently erasure-list-decoding "pure" expander codes beyond their (designed) minimum distance.

Organization. In Section 2, we formally introduce the notation and definitions we will need. In Section 3, we introduce the preliminary algorithm SlowListDecode, while in Section 4 we describe the final algorithm that has better dependence on ε, δ and r in the running time. This proves the Main Theorem 1.2. We conclude in Section 5 with the proof of Lemma 1.3, showing a bound on the second generalized distance of expander codes.

2 Preliminaries

Expander Graphs. Let $G = (L \cup R, E)$ be a bipartite graph.[6] For a vertex $v \in L \cup R$, let $\Gamma(v)$ denote the set of vertices adjacent to v. For $A \subseteq L \cup R$, let $E(A)$ denote the set of edges with both endpoints in A, and for $S \subseteq L$ and $T \subseteq R$, let $E(S,T) := E(S \cup T)$.

Let G be a d-regular graph on n vertices, and let $d = \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be the eigenvalues of the adjacency matrix of G. For $n \geq 3$ we let $\lambda(G) := \max\{|\lambda_i|, \lambda_i \neq \pm d\}$. The bipartite graph version of the *Expander Mixing Lemma* reads:

Theorem 2.1 (Expander Mixing Lemma, see *e.g.* [HLW06])**.** *Suppose that $G = (L \cup R, E)$ is a d-regular graph on $2n$ vertices. Then for any $S \subseteq L$ and $T \subseteq R$,*

$$\left| E(S,T) - \frac{d}{n}|S||T| \right| \leq \lambda(G)\sqrt{|S||T|}.$$

Expander Codes. Let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices, as above. Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code, called the *inner code*. Fix an order on the edges incident to each vertex of G, and let $\Gamma_i(v)$ denote the i'th neighbor of v.

The code $\mathcal{C} := \mathcal{C}(G, \mathcal{C}_0)$ is defined as the set of all labelings of the edges of G that respect the inner code $\mathcal{C}_0$. More precisely, we have the following definition.

Definition 2.2. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code, and let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices. The code $\mathcal{C}(G, \mathcal{C}_0) \subseteq \mathbb{F}_2^E$ is the linear code of length nd, such that for $c \in \mathbb{F}_2^E$, $c \in \mathcal{C}$ if and only if, for all $v \in L \cup R$,*

$$\left(c_{(v,\Gamma_1(v))}, c_{(v,\Gamma_2(v))}, \ldots, c_{(v,\Gamma_d(v))}\right) \in \mathcal{C}_0.$$

By counting constraints, it is not hard to see that if $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ is a linear code of rate ρ, then $\mathcal{C}(G, \mathcal{C}_0) \subseteq \mathbb{F}_2^E$ is a linear code of rate at least $2\rho - 1$. Moreover, it is known that expander codes have good distance:

Lemma 2.3 ([SS96, Zém01])**.** *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ, and let $G = (L \cup R, E)$ be a bipartite d-regular graph with $\lambda = \lambda(G)$. Then the code $\mathcal{C}(G, \mathcal{C}_0)$ has distance at least $\delta(\delta - \lambda/d)$.*

Furthermore, $\mathcal{C}$ can be uniquely decoded up to this fraction of erasures in linear time.

Lemma 2.4. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ, and let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices with $\lambda = \lambda(G)$. Let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} < \frac{\delta}{2}$. Then there is an algorithm* UniqueDecode *which uniquely decodes the code $\mathcal{C}(G, \mathcal{C}_0)$ from up to $(1-\varepsilon)\delta(\delta - \lambda/d)$ erasures in time $n \cdot \mathrm{poly}(d)/\varepsilon$.*

The above lemma is by now folklore, but for completeness, we include a proof in Appendix A.

[6] In this paper we only consider undirected connected graphs.

3 A preliminary algorithm

For clarity of exposition, we begin the proof of the Main Theorem 1.2 by proving the following weaker theorem.

Theorem 3.1. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ and r'th generalized distance δ_r. Let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices with $\lambda = \lambda(G) = \max\{|\lambda_i|, |\lambda_i| \neq d\}$. Let $\mathcal{C} = \mathcal{C}(G, \mathcal{C}_0)$ be the code that results. Let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} \leq \frac{\varepsilon^2\delta^2}{2^{r+4}}$. Let $s := \frac{2^{2r+7}}{\varepsilon^4\delta^4}$. Then there is an algorithm* SlowListDecode *which erasure-list-decodes $\mathcal{C}$ from $(1-\varepsilon)\delta\delta_r dn$ erasures with list size less than 2^s in time $n \cdot \mathrm{poly}(d) \cdot \exp(s)$.*

Theorem 3.1 still provides a linear-time algorithm (provided $d, r, \varepsilon, \delta$ are all constant), but the dependence on r, ε, δ is not very good. We will prove Theorem 3.1 in this section to illustrate the main ideas, and then in Section 4, we will show how to adapt the algorithm to achieve the running times advertised in Theorem 1.2.

A formal description of the algorithm SlowListDecode is given in Figure 2. Roughly, the first step is to list decode the inner codes to obtain an inner list List_v at each vertex $v \in L \cup R$. The second and main step then is to label large equivalence classes by iterating over all possible assignments to representatives from these classes. In the third and final step we complete any such possible assignment, by first uniquely decoding at inner codes where sufficient number of edges are already labeled, followed by global unique decoding to recover the rest of the unlabeled edges. Below we elaborate on each of these steps.

In what follows, fix $r \geq 1$, and suppose that $z \in (\mathbb{F}_2 \cup \{\perp\})^E$ is a received word with less than $(1-\varepsilon)\delta\delta_r dn$ symbols that are $\perp$, and let $\mathrm{List}_{\mathcal{C}}(z)$ be the set of codewords of $\mathcal{C}$ that are consistent with z.

3.1 List decoding inner codes

The first step is to list decode all inner codes with not too many erasures. Specifically, let $B \subseteq L \cup R$ be the set of *bad* vertices v so that z has at least $\delta_r d$ erasures incident to v.

$$B = \left\{ v \in L \cup R \,:\, z_{(v,u)} = \perp \text{ for at least } \delta_r d \text{ vertices } u \right\}. \tag{1}$$

Then by the assumption on the number of erasures in z,

$$|B \cap L|\delta_r d \leq (1-\varepsilon)\delta\delta_r nd$$

and the same for $B \cap R$, which implies that

$$|B \cap L| \leq (1-\varepsilon)\delta n, \; |B \cap R| \leq (1-\varepsilon)\delta n. \tag{2}$$

The first step of the algorithm will be to list-decode the inner code $\mathcal{C}_0$ at every vertex $v \notin B$. For all such v, let

$$\mathrm{List}_v := \mathrm{List}_{\mathcal{C}_0}\left((z_{(v,\Gamma_1(v))}, z_{(v,\Gamma_2(v))}, \ldots, z_{(v,\Gamma_d(v))}) \right). \tag{3}$$

Next we shall use the following notion of *local equivalence relation* to assign labels to many of the edges. To define this notion, note first that since $\mathcal{C}_0$ has r'th generalized distance δ_r, for any $v \notin B$, List_v is an affine subspace of $\mathbb{F}_2^d$ of dimension $r_v \leq r-1$. Let $A_v \in \mathbb{F}_2^{d \times r_v}$ and $b_v \in \mathbb{F}_2^d$ be such that

$$\mathrm{List}_v = \{ A_v x + b_v \,:\, x \in \mathbb{F}_2^{r_v} \}.$$

Algorithm: FindHeavyEdges
Inputs: A description of $G = (L \cup R, E)$ and $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$, and the lists List_v for $v \notin B$.
Output: The set $E' \subseteq E$.

Initialize: $E' \leftarrow E$.

1. Remove from E' all edges incident to a vertex in B.
2. While there is some $(u, v) \in E'$, so that

$$\left|\left\{(w, v) \in E' \,:\, (w, v) \sim_v (u, v)\right\}\right| \leq \frac{\varepsilon^2 \delta^2}{2^{r+3}} \cdot d :$$

   Remove (u, v) and all edges $(w, v) \in E'$ so that $(w, v) \sim_v (u, v)$ from E'.
3. Break and return the set E'.

Figure 1: FindHeavyEdges

Note that the vectors in $\mathrm{List}_v \subseteq \mathbb{F}_2^d$ are indexed by edges (v, w) adjacent to v, and so the rows of A_v are also indexed by these edges.

Next we define, for any vertex $v \notin B$, a *local equivalence relation* $\sim_v$ at the vertex v.

Definition 3.2 (Local equivalence relation)**.** *Suppose that $v \notin B$. For $(v, u), (v, w) \in E$, say that $(v, u) \sim_v (v, w)$ if the row of A_v indexed by (v, u) equals the row of A_v indexed by (v, w).*

Notice that Definition 3.2 depends on both v and z; we suppress the dependence on z in the notation. We make the following observations.

Observation 3.3. *Suppose that $v \notin B$.*

(A) If $(v, u) \sim_v (v, w)$, then for any $c \in \mathrm{List}_{\mathcal{C}}(z)$, $c_{(v,u)}$ is determined by $c_{(v,w)}$.

(B) There are less than 2^{r-1} local equivalence classes at v, because there are less than 2^{r-1} possible vectors in $\mathbb{F}_2^{r_v}$ that could appear as rows of the matrices A_v.

3.2 Labeling large equivalence classes

The next step is to assign labels to large *global* equivalence classes, defined below. For this, we first define a new edge set $E' \subseteq E$ by first throwing out all edges touching B, and then repeatedly throwing out edges whose local equivalence classes are too small. Specifically, define E' to be the output of the following Algorithm FindHeavyEdges, given in Figure 1.

Next we define a *global* equivalence relation $\sim$ on the edges in E' as follows.

Definition 3.4 (Global equivalence relation)**.** *Suppose that $e, e' \in E'$. We say that $e \sim e'$ if there is a path $e = e_1, e_2, \ldots, e_t = e'$ so that $e_1, e_2, \ldots, e_t \in E'$, and for any pair of adjacent edges $e_i = (u, v)$, $e_{i+1} = (v, w)$ on the path it holds that $(u, v) \sim_v (v, w)$. This relation defines* global equivalence classes.

The following lemma shows that E' is partitioned into a small number of large global equivalence classes. Consequently, one can assign labels to all edges in E' by iterating over all possible assignments for a small number of representatives from these classes.

Lemma 3.5. *Any global equivalence class in E' has size at least $\frac{\varepsilon^4\delta^4}{2^{2r+7}}dn$. In particular, E' is partitioned into less than $s := \frac{2^{2r+7}}{\varepsilon^4\delta^4}$ different equivalence classes.*

Proof. Let F be a global equivalence class in E', and let $S \subseteq L$ and $T \subseteq R$ denote the left and right vertices touching F, respectively. By the definition of E', any vertex $v \in S \cup T$ is incident to at least $\frac{\varepsilon^2\delta^2}{2^{r+3}} \cdot d$ edges in F. Thus by the Expander Mixing Lemma (Theorem 2.1),

$$\frac{\varepsilon^2\delta^2}{2^{r+3}}d\sqrt{|S||T|} \leq \frac{\varepsilon^2\delta^2}{2^{r+3}}d\frac{|S|+|T|}{2} \leq |F| \leq \frac{d}{n}|S||T| + \lambda\sqrt{|S||T|},$$

and rearranging

$$n\left(\frac{\varepsilon^2\delta^2}{2^{r+3}} - \frac{\lambda}{d}\right) \leq \sqrt{|S||T|}.$$

This implies in turn that

$$|F| \geq \frac{\varepsilon^2\delta^2}{2^{r+3}}d\sqrt{|S||T|} \geq \frac{\varepsilon^2\delta^2}{2^{r+3}}\left(\frac{\varepsilon^2\delta^2}{2^{r+3}} - \frac{\lambda}{d}\right)dn,$$

which gives the final claim by the choice of $\frac{\lambda}{d} \leq \frac{\varepsilon^2\delta^2}{2^{r+4}}$. □

Finally, by (A) in Observation 3.3, choosing a symbol on an edge determines all the symbols in the equivalence class of that edge. Thus, we will exhaust over all choices of symbols for the equivalence classes in E'; this leads to 2^s possibilities. Next we show that any such choice determines a unique codeword in $\mathcal{C}$.

3.3 Completing the assignment

To complete the assignment we first show that many of the vertices have at least $(1-\delta)d$ incident edges in E'. For any such vertex, the inner codeword at this vertex is completely determined by the assignment to edges in E', and so can be recovered by uniquely decoding locally at this vertex. We then recover the small number of remaining edges using global unique decoding. Specifically, let

$$B' = \left\{v \in L \cup R : (v,u) \notin E \setminus E' \text{ for at least } \delta d \text{ vertices } u\right\}. \tag{4}$$

The next lemma bounds the size of B', and the number of edges in $E(B')$.

Lemma 3.6. *The following hold:*

1. $|B' \cap L| \leq \left(1 - \frac{\varepsilon}{2}\right)\delta n$, $|B' \cap R| \leq \left(1 - \frac{\varepsilon}{2}\right)\delta n$.
2. $|E(B')| \leq \left(1 - \frac{\varepsilon}{4}\right)\left(\delta - \frac{\lambda}{d}\right)\delta nd$.

Proof. For the first item, let $B_1 \subseteq (L \cup R) \setminus B$ be the subset of vertices $v \notin B$ so that at least $\left(1 - \frac{\varepsilon}{2}\right)\delta d$ edges incident to v are removed on Step 1 of FindHeavyEdges, and let $B_2 \subseteq (L \cup R) \setminus B$ be the subset of vertices $v \notin B$ so that at least $\frac{\varepsilon}{2}\delta d$ edges incident to v are removed on Step 2 of FindHeavyEdges. Note that $B' \subseteq B \cup B_1 \cup B_2$, so it suffices to show that $|(B \cup B_1 \cup B_2) \cap L| \leq$

$\left(1-\frac{\varepsilon}{2}\right)\delta n$, and similarly for R. By (2), $|B\cap L|\leq(1-\varepsilon)\delta n$ and $|B\cap R|\leq(1-\varepsilon)\delta n$. Claims 3.7 and 3.8 below show that each of the sets B_1, B_2 has size less than $\frac{\varepsilon}{4}\delta n$ which gives the desired conclusion.

For the second item, note that by the first item and the Expander Mixing Lemma,

$$\begin{aligned}\left|E(B')\right| &\leq \frac{d}{n}\left(\delta n\left(1-\frac{\varepsilon}{2}\right)\right)^2+\lambda\delta n\left(1-\frac{\varepsilon}{2}\right)\\ &\leq \left(1-\frac{\varepsilon}{2}\right)\left(\delta+\frac{\lambda}{d}\right)\delta nd\\ &\leq \left(1-\frac{\varepsilon}{4}\right)\left(\delta-\frac{\lambda}{d}\right)\delta nd,\end{aligned}$$

where the last inequality follows by the choice of $\frac{\lambda}{d}\leq\frac{\varepsilon\delta}{8}$. □

Claim 3.7. $|B_1|\leq\frac{\varepsilon}{4}\delta n$.

Proof. By the description of FindHeavyEdges, B_1 is the set of all vertices $v\in(L\cup R)\setminus B$ that are incident to at least $\left(1-\frac{\varepsilon}{2}\right)\delta d$ vertices of B. Thus by the Expander Mixing Lemma,

$$\begin{aligned}|B_1\cap L|\left(1-\frac{\varepsilon}{2}\right)\delta d &\leq |E(B_1\cap L, B\cap R)|\\ &\leq \frac{d}{n}|B_1\cap L||B\cap R|+\lambda\sqrt{|B_1\cap L||B\cap R|}\\ &\leq \frac{d}{n}|B_1\cap L|n\delta(1-\varepsilon)+\lambda\sqrt{|B_1\cap L|n\delta(1-\varepsilon)},\end{aligned}$$

where the last inequality follows by (2).

Rearranging, we have

$$\sqrt{|B_1\cap L|}\leq\frac{\lambda\sqrt{n\delta(1-\varepsilon)}}{d\delta\varepsilon/2},$$

and

$$|B_1\cap L|\leq 4n\left(\frac{\lambda}{d}\right)^2\cdot\frac{1}{\delta\varepsilon^2}\leq\frac{\varepsilon}{8}\delta n,$$

where the last inequality follows by the choice of $\frac{\lambda}{d}\leq\frac{\varepsilon^{3/2}\delta}{8}$. As the same holds for $B_1\cap R$, we conclude that B_1 has size less than $\frac{\varepsilon}{4}\delta n$. □

Claim 3.8. $|B_2|\leq\frac{\varepsilon}{4}\delta n$.

Proof. Since there are less than 2^{r-1} local equivalence classes at each vertex v, the algorithm FindHeavyEdges performs less than $2n\cdot 2^{r-1}$ iterations at Step 2. At each such iteration, less than $\frac{\varepsilon^2\delta^2}{2^{r+3}}\cdot d$ edges are removed, and so the total number of edges removed at Step 2 of FindHeavyEdges is $2n\cdot 2^{r-1}\cdot\frac{\varepsilon^2\delta^2}{2^{r+3}}\cdot d=\frac{\varepsilon^2\delta^2}{8}\cdot dn$. Finally, by averaging this implies that there are less than $\frac{\varepsilon}{4}\delta n$ vertices v so that at least $\frac{\varepsilon}{2}\delta d$ edges incident to v are removed at this step. □

Next observe that for any vertex $v\notin B'$, the choices for symbols on E' uniquely determine the codeword of $\mathcal{C}_0$ that belongs at the vertex v. This is because $\mathcal{C}_0$ has distance δ, and at least $(1-\delta)d$ edges incident to v have been labeled. Note that since $\mathcal{C}_0$ is a linear code of length d, this unique codeword can be found in time $O(d^3)$ by solving a system of linear equations. Once this is done,

the only edges that do not have labels are those in $E(B')$. By Item (2) of Lemma 3.6, there are less than $\left(1-\frac{\varepsilon}{4}\right)\left(\delta-\frac{\lambda}{d}\right)\delta nd$ such edges. By Lemma 2.4, these edges can be recovered using global unique decoding in time $n\cdot\mathrm{poly}(d)/\varepsilon$. In this way, we can recover the entire list $\mathrm{List}_{\mathcal{C}}(z)$.

The algorithm described above is given as SlowListDecode in Figure 2. Next we show that this algorithm runs in time $n\cdot\mathrm{poly}(d)\cdot\exp(s)$.

Claim 3.9. *Algorithm* SlowListDecode *runs in time* $n\cdot\mathrm{poly}(d)\cdot\exp(s)$.

Proof. The first step entails finding the kernel of a sub-matrix of a parity-check matrix $H_0\in\mathbb{F}_2^{d(1-\rho)\times d}$ for $\mathcal{C}_0$, for each vertex v, which can be done in time $O(d^3)$. Thus, the time for the first step is $n\cdot\mathrm{poly}(d)$. The second step can be done in time $O(nd)$ using Breadth-First-Search. In the third step, Step (a) can be done in time $O(nd)$ by Breadth-First-Search, Step (b) can be done in time $n\cdot\mathrm{poly}(d)$ as in Step 1, and Step (c) can be done in time $n\cdot\mathrm{poly}(d)/\varepsilon$ by Lemma 2.4. □

The above claim completes the proof of Theorem 3.1. We will show how to speed it up in Section 4, where we will conclude the proof of Theorem 1.2.

4 Final algorithm

The algorithm SlowListDecode runs in time $O_{r,\delta,\varepsilon}(n\cdot\mathrm{poly}(d))$, but the constant inside the $O_{r,\delta,\varepsilon}(\cdot)$ is exponential in poly $\left(\frac{2^r}{\varepsilon\delta}\right)$, since there are $s=$ poly $\left(\frac{2^r}{\varepsilon\delta}\right)$ equivalence classes, and we exhaust over all 2^s possible assignments to representatives from these classes. In this section, we will show how to do significantly better and obtain a running time that depends polynomially on $2^r, 1/\delta, 1/\varepsilon$, finishing the proof of Theorem 1.2. The basic idea is as follows. Instead of exhausting over all possible ways to assign values to the edges $e^{(1)},\ldots,e^{(s)}$, we will set up and solve a linear system to find a description of the ways to assign these values that will lead to legitimate codewords. Specifically, we prove the following lemma.

Lemma 4.1. *There is an algorithm* FindList *which, given the state of* SlowListDecode *at the end of Step 2, runs in time* $n\cdot\mathrm{poly}(d,s)$ *and returns* $A\in\mathbb{F}_2^{nd\times s}$, $b\in\mathbb{F}_2^{nd}$, $\hat{A}\in\mathbb{F}_2^{s\times a}$, *and* $\hat{b}\in\mathbb{F}_2^s$ *so that*

$$\mathrm{List}_{\mathcal{C}}(z)=\left\{Ax+b\,:\,x=\hat{A}\hat{x}+\hat{b}\text{ for some }\hat{x}\in\mathbb{F}_2^a\right\},$$

where $a:=\dim(\mathrm{List}_{\mathcal{C}}(z))$ *satisfies* $a\le s$.

The above lemma immediately implies Theorem 1.2: We first run Steps 1 and 2 in SlowListDecode in order to find the set E' and its partition into equivalence classes. As before, this takes time $n\cdot\mathrm{poly}(d)$. Next, we run FindList in order to find a linear-algebraic description of the list $\mathrm{List}_{\mathcal{C}}(z)$, which we return. This second step takes time $n\cdot\mathrm{poly}(s,d)$, for a total running time of $n\cdot\mathrm{poly}(s,d)$. Plugging in the definition of s proves Theorem 1.2. The formal description of the final algorithm ListDecode is given in Figure 4. The rest of this section is devoted to the proof of Lemma 4.1.

First, note that every value y_e determined by SlowListDecode is some affine function of the labels on $e^{(1)},\ldots,e^{(s)}$. That is, there is some matrix $A\in\mathbb{F}_2^{dn\times s}$ and some vector $b\in\mathbb{F}_2^{dn}$ so that the list generated by SlowListDecode is

$$\{Ax+b\,:\,x\in\mathbb{F}_2^s\},$$

Algorithm: SlowListDecode
Inputs: A description of $G = (L \cup R, E)$ and $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$, and $z \in (\mathbb{F}_2 \cup \{\perp\})^E$.
Output: The list $\mathrm{List}_{\mathcal{C}}(z)$.

Initialize: $\mathrm{List}_{\mathcal{C}}(z) = \emptyset$.

1. Let $B \subseteq L \cup R$ be as in (1). For each $v \notin B$, run $\mathcal{C}_0$'s erasure list-decoding algorithm to obtain the lists List_v as in (3).
2. Run the algorithm FindHeavyEdges given in Figure 1 to find the set E', find the partition of E' into s global equivalence classes, and choose representative edges $e^{(1)}, \ldots, e^{(s)}$ from each of the equivalence classes.
3. For each $y^{(advice)} \in \mathbb{F}_2^{\{e^{(1)},\ldots,e^{(s)}\}}$:
   (a) For each $i \in [s]$, and for each $e \sim e^{(i)}$, define y_e to be the value uniquely determined by $y^{(advice)}_{e^{(i)}}$, as given in Item (A) of Observation 3.3.
   (b) Let $B' \subseteq L \cup R$ be as in (4). For each $v \notin B'$, find the unique $y|_{\{v\}\times\Gamma(v)}$ so that $y_{(v,u)}$ is consistent with existing assignments to y, or determine that no such y exists. If no such y exists for some v, continue to the next choice of $y^{(advice)}$.
   (c) Use the linear-time erasure unique decoding algorithm UniqueDecode from Lemma 2.4 to find a unique $y \in \mathbb{F}_2^E$ that agrees with all the choices of y made so far, or determine that none exists. If it exists, add y to $\mathrm{List}_{\mathcal{C}}(z)$.
4. Return $\mathrm{List}_{\mathcal{C}}(z)$.

Figure 2: SlowListDecode: Returns $\mathrm{List}_{\mathcal{C}}(z)$ in time $O_{r,\delta,\varepsilon}(n)$. However, the dependence on r, δ, ε is not good, and is improved in ListDecode, given in Figure 4.

where $x := y^{(advice)}$. The goal in FindList will thus be to find this A and b efficiently, as well as to find a description of the x's so that $Ax + b$ is actually a codeword in $\mathcal{C}$. An overview of the algorithm FindList is given in Figure 3, and the steps are described below.

4.1 Finding A and b

The first step of the algorithm will be to find A and b. To find this efficiently, we will mirror the decoding algorithm in SlowListDecode, except we will do it while keeping the choices of $y^{(advice)}$ as variables. As we will see below, this can be done in time $n \cdot \text{poly}(s, d)$. For this, we shall find a sequence $\left(A^{(t)}, b^{(t)}\right)$ for $t = 0, 1, \ldots, T$, where $(A, b) = \left(A^{(T)}, b^{(T)}\right)$ as described below. It will be convenient to use the following standard notation: for a vector $c \in \mathbb{F}_2^E$, and a subset $E' \subseteq E$, we let $c|_{E'} \in \mathbb{F}_2^{E'}$ be the vector c restricted to the coordinates of E'.

Finding $A^{(0)}$ and $b^{(0)}$. First, let $E_0 := E'$, and let $A^{(0)} \in \mathbb{F}_2^{E_0 \times s}$ and $b^{(0)} \in \mathbb{F}_2^{E_0}$ such that

$$(A^{(0)}x + b^{(0)})_e = y_e,$$

where $x := y^{(advice)}$, and y_e is as in Step (3a) in Algorithm SlowListDecode. Note that $A^{(0)}$ has rows which are 1-sparse, and that $A^{(0)}, b^{(0)}$ can be created in time $O(nd)$ given the matrices A_v and vectors b_v. Further note that, for any $c \in \mathcal{C}$, $c|_{E_0} = A^{(0)}c|_{\{e^{(1)},\ldots,e^{(s)}\}} + b^{(0)}$, where $e^{(1)}, \ldots, e^{(s)}$ are the representative edges chosen in Step (2) of Algorithm SlowListDecode.

Finding $A^{(1)}$ and $b^{(1)}$. Recalling B' from (4), let $E_1 := (E \setminus E(B')) \cup E_0$. Note that for each $e \in E_1 \setminus E_0$, the label on e can be determined in an affine way from the labels on edges in E_0. More precisely, there is some vector $k^{(e)} \in \mathbb{F}_2^{E_0}$ of weight less than d and some $h^{(e)} \in \mathbb{F}_2$ so that for any $c \in \mathcal{C}$,

$$c_e = (k^{(e)})^T \cdot c|_{E_0} + h^{(e)},$$

and moreover $k^{(e)}$ and $h^{(e)}$ can be found in time $\text{poly}(d)$ by inverting a submatrix of one of the matrices A_v.

Let $K \in \mathbb{F}_2^{(E_1 \setminus E_0) \times E_0}$ be the matrix with the $k^{(e)}$ as rows, let $h \in \mathbb{F}_2^{E_1 \setminus E_0}$ be the vector with entries $h^{(e)}$, and let

$$A^{(1)} := \begin{bmatrix} A^{(0)} \\ \hline KA^{(0)} \end{bmatrix} \qquad \text{and} \qquad b^{(1)} := \left(\begin{array}{c|} & \\ b^{(0)} & \\ & \\ \hline & \\ Kb^{(0)} + h & \\ & \end{array}\right).$$

Note that $A^{(1)}, b^{(1)}$ can be created in time $n \cdot \text{poly}(d) \cdot s$ given $A^{(0)}$, $b^{(0)}$, K, and h. Further note that for any $c \in \mathcal{C}$, $c|_{E_1} = A^{(1)}c|_{\{e^{(1)},\ldots,e^{(s)}\}} + b^{(1)}$.

Finding $A^{(t)}$ and $b^{(t)}$ for $t = 2, \ldots, T$. At this point, by the analysis above (following from Lemma 3.6), we know that there are less than $\left(1 - \frac{\varepsilon}{4}\right)\left(\delta - \frac{\lambda}{d}\right)\delta nd$ edges which are not in E_1. If we had labels for the edges in E_1, then by Lemma 2.4 we can use the algorithm UniqueDecode to recover the rest.

The algorithm UniqueDecode is given in Appendix A in Figure 5. The basic idea is to iteratively decode $\mathcal{C}_0$ at vertices in L, then R, then L, and so on, to arrive at a unique assignment for all of the edges. In order to do this with matrices, we will continue as above, creating $A^{(t)}, b^{(t)}$ from $A^{(t-1)}, b^{(t-1)}$ for larger t just as we did for $t = 1$. Note that the sets E_t in UniqueDecode play the same role that they do here; E_t represents the set of edges for which a label can be assigned in step t.

More precisely, suppose that at step $t-1$, UniqueDecode has assigned labels to E_{t-1}, and suppose that we have $A^{(t-1)} \in \mathbb{F}_2^{E_{t-1} \times s}$ and $b^{(t-1)} \in \mathbb{F}_2^{E_{t-1}}$ so that for any $c \in \mathcal{C}$,

$$c|_{E_{t-1}} = A^{(t-1)} c|_{\{e^{(1)}, \ldots, e^{(s)}\}} + b^{(t-1)}.$$

At the next step, UniqueDecode would have assigned labels to edges in $E_t \setminus E_{t-1}$. We note that the total amount of time (over all iterations) to determine the edges in $E_t \setminus E_{t-1}$ is the same as in UniqueDecode, which, with the right bookkeeping, is $n \cdot \text{poly}(d)/\varepsilon$.

Then as above, for every $e \in E_t \setminus E_{t-1}$, there is some vector $k^{(e)} \in \mathbb{F}_2^{E_{t-1}}$ of weight less than d, and some $h^{(e)} \in \mathbb{F}_2$ so that for any $c \in \mathcal{C}$,

$$c_e = (k^{(e)})^T \cdot c|_{E_{t-1}} + h^{(e)},$$

and moreover, these vectors can be found in time $\text{poly}(d)$. Then, as above, let $K \in \mathbb{F}_2^{(E_t \setminus E_{t-1}) \times E_{t-1}}$ be the matrix with the $k^{(e)}$ as rows, let $h \in \mathbb{F}_2^{E_t \setminus E_{t-1}}$ be the vector with entries $h^{(e)}$, and let

$$A^{(t)} := \begin{bmatrix} A^{(t-1)} \\ \hline KA^{(t-1)} \end{bmatrix} \quad \text{and} \quad b^{(t)} := \begin{pmatrix} | \\ b^{(t-1)} \\ | \\ \hline | \\ Kb^{(t-1)} + h \\ | \end{pmatrix}.$$

As above, $A^{(t)}, b^{(t)}$ can be created in time $|E_t \setminus E_{t-1}| \cdot \text{poly}(d) \cdot s$, and for any $c \in \mathcal{C}$,

$$c|_{E_t} = A^{(t)} c|_{\{e^{(1)}, \ldots, e^{(s)}\}} + b^{(t)}.$$

We continue this way until $E_T = E$, which happens eventually by the correctness of the algorithm UniqueDecode. Then, the amount of work that has been done to compute $A := A^{(T)}$ and $b := b^{(T)}$ is

$$n \cdot \text{poly}(d) \cdot s + \sum_{t=2}^{T} |E_t \setminus E_{t-1}| \cdot \text{poly}(d) \cdot s = n \cdot \text{poly}(d, s),$$

as claimed.

4.2 Finding $\hat{A}$ and $\hat{b}$

Once we have found A and B, the goal is to find the set of $x \in \mathbb{F}_2^s$ so that

$$HAx + Hb = 0, \tag{5}$$

where $H \in \mathbb{F}_2^{2nd(1-\rho) \times nd}$ is the parity-check matrix for $\mathcal{C}$.

First, notice that given the parity-check matrix for $\mathcal{C}_0$, $H_0 \in \mathbb{F}_2^{d(1-\rho)\times d}$, where ρ denotes the rate of the code, and a description of G, we can access any entry of H in time $O(1)$: for each vertex v, there is a parity check for each row of H_0 on the edges incident to v. Next, notice that we can compute HA in time $O(nd^2s)$: each row of H has less than d nonzeros, so for each of the $O(nd)$ rows of H, we take time $O(ds)$ to compute the corresponding row of HA. Similarly we can compute Hb in time $O(nd^2)$.

The goal then is to find the space of x's which lead to legitimate codewords, which is

$$\mathcal{W} = \{x \in \mathbb{F}_2^s \,:\, HAx = Hb\}\,.$$

To find a description of $\mathcal{W}$, we first find a basis for the row space of HA, which we can do in time $O(nds^3)$: we iterate through the $O(nd)$ rows of HA, and check (in time $O(s^3)$) to see if they are linearly independent from the rows we have already found. If so, we add the new row to our basis and continue. Suppose that $t \le s$ is the dimension of the row space of HA, and let $j_1, \ldots, j_t \le 2nd(1-\rho)$ be the indices of the rows in the basis; let $J \in \mathbb{F}_2^{t\times s}$ be the submatrix of HA with these rows.

Let $\hat{A} \in \mathbb{F}_2^{s\times(s-t)}$ be a matrix so that the columns of $\hat{A}$ span $\mathrm{Ker}(J) = \mathrm{Ker}(HA)$. Note that we can compute such $\hat{A}$ in time $\mathrm{poly}(s)$ given J. Next suppose there is some $\hat{b} \in \mathbb{F}_2^s$ so that

$$HA\hat{b} = Hb. \tag{6}$$

Then the space we are after is

$$\mathcal{W} = \left\{\hat{A}\hat{x} + \hat{b} \,:\, \hat{x} \in \mathbb{F}_2^{s-t}\right\}.$$

If there is no such $\hat{b}$, then $\mathrm{List}_{\mathcal{C}}(z) = \emptyset$ and we should return $\perp$. If such a $\hat{b}$ exists, we may compute it by finding a solution to the system

$$J\hat{b} = (Hb)_{j_1,\ldots,j_t} \tag{7}$$

which can be done in time $\mathrm{poly}(s)$. Indeed, suppose that there is some $\hat{b}$ satisfying (6). Then $\hat{b}$ satisfies (7), and for any b' which also satisfies (7), $b' \in \hat{b} + \mathrm{Ker}(J) = \hat{b} + \mathrm{Ker}(HA)$, and hence b' satisfies (6) as well. Then we check to see if this $\hat{b}$ satisfies $HA\hat{b} = Hb$, which can be done in time $O(nds)$. If so, we return A, b and $\hat{A}, \hat{b}$. If not (or if no $\hat{b}$ satisfying (7) exists), then we return $\perp$.

The formal description of the algorithm FindList is given below in Figure 3.

Claim 4.2. *Algorithm* FindList *runs in time* $n \cdot \mathrm{poly}(d, s)$.

Proof. As described above, one can form $A^{(1)}, b^{(1)}$ and find E_1 in Step 1 in time $n \cdot \mathrm{poly}(d, s)$. By Lemma 2.4, the total running time (over all iterations) for Steps 2(a-d) is $n \cdot \mathrm{poly}(d)/\varepsilon$. Moreover, as described in the text, Step 2(e) can be done in time $|E_t \setminus E_{t-1}| \cdot \mathrm{poly}(s, d)$, and so the total running time of this step (over all iterations) is $n \cdot \mathrm{poly}(d, s)$. In Step 4, one can compute HA and Hb in time $O(nd^2s)$, and one can find J in time $O(nds^3)$. Finally, note that Step 5 can be done in time $\mathrm{poly}(s)$. □

Algorithm: FindList
Inputs: The state of SlowListDecode after step 2.
Output: $A \in \mathbb{F}_2^{nd \times s}, b \in \mathbb{F}_2^{nd}, \hat{A} \in \mathbb{F}_2^{s \times a}, \hat{b} \in \mathbb{F}_2^s$ so that

$$\mathrm{List}_{\mathcal{C}}(z) = \left\{ Ax + b \,:\, x = \hat{A}\hat{x} + \hat{b} \text{ for some } \hat{x} \in \mathbb{F}_2^a \right\}$$

or returns $\perp$ if $\mathrm{List}_{\mathcal{C}}(z) = \emptyset$.

1. Form $A^{(1)}, b^{(1)}$, and find E_1 as described in the text. Let $P_0 \subseteq R, P_1 \subseteq L$ be the sets of vertices incident to an edge in $E \setminus E_1$.
2. For $t = 2, 3, \ldots$:
   (a) If $P_{t-1} = \emptyset$, break.
   (b) Initialize $P_t \leftarrow \emptyset$ and $E_t \leftarrow E_{t-1}$.
   (c) For each vertex $v \in P_{t-1}$ so that $|(\{v\} \times \Gamma(v)) \cap E_{t-1}| > (1-\delta)d$:
      - Remove v from P_{t-1}.
      - For any vertex u so that $(v,u) \notin E_{t-1}$, add (v,u) to E_t.
   (d) For each vertex $v \in P_{t-1}$, for any $(v,u) \notin E_{t-1}$, add u to P_t.
   (e) Find $A^{(t)}$ and $b^{(t)}$ given $A^{(t-1)}, b^{(t-1)}$ so that for all $c \in \mathcal{C}$,
   $$c|_{E_t} = A^{(t)} c|_{e^{(1)},\ldots,e^{(s)}} + b^{(t)}.$$
3. Let $A = A^{(t)}$ and $b = b^{(t)}$.
4. Compute HA and Hb, let $t \leq s$ be the dimension of the row space of HA, and find $j_1, \ldots, j_t$ so that the rows of HA indexed by $j_1, \ldots, j_t$ form a basis for the row space of HA. Let $J \in \mathbb{F}_2^{t \times s}$ be the submatrix of HA with these rows.
5. Find $\hat{A} \in \mathbb{F}_2^{s \times (s-t)}$ whose columns are a basis for the kernel of J, and find $\hat{b} \in \mathbb{F}_2^s$ so that $J\hat{b} = (Hb)|_{j_1,\ldots,j_t}$.
6. If $HA\hat{b} \neq Hb$, return $\perp$. In this case, $\mathrm{List}_{\mathcal{C}}(z) = \emptyset$.
7. Otherwise, return $A, b, \hat{A}, \hat{b}$.

Figure 3: FindList: prunes the list of advice strings $y^{(advice)}$ in SlowListDecode to a space $\mathrm{List}_{\mathcal{C}}(z) = \left\{ \tilde{A}x + \tilde{b} \,:\, x \in \mathbb{F}_2^a \right\}$ and returns this description.

Algorithm: ListDecode
Inputs: A description of $G = (L \cup R, E)$ and $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$, and $z \in (\mathbb{F}_2 \cup \{\perp\})^E$.
Output: A matrix $\tilde{A} \in \mathbb{F}_2^{nd \times a}$ and a vector $\tilde{b} \in \mathbb{F}_2^{nd}$ so that

$$\mathrm{List}_{\mathcal{C}}(z) = \{Lx + \ell \,:\, x \in \mathbb{F}_2^a\}$$

for some integer a (which does not depend on n), or else $\perp$ if $\mathrm{List}_{\mathcal{C}}(z)$ is empty.

1. Run Steps 1 and 2 from SlowListDecode (Figure 2).
2. Run FindList (Figure 3).
3. If FindList returns $\perp$, return $\perp$.
4. Otherwise, FindList returns $A, b, \hat{A}, \hat{b}$.
5. Compute $\tilde{A} = A\hat{A}$ and $\tilde{b} = A\hat{b} + b$.
6. Return $\tilde{A}, \tilde{b}$.

Figure 4: ListDecode: Returns a description of $\mathrm{List}_{\mathcal{C}}(z)$ in time $n \cdot \mathrm{poly}(d, 2^r, 1/\delta, 1/\varepsilon)$.

5 Second generalized distance of expander codes

In this section we prove Lemma 1.3, restated below.

Lemma 1.3 (restated). *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ and second generalized distance δ_2, and let $G = (L \cup R, E)$ be a bipartite d-regular graph with $\lambda = \lambda(G)$. Let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} \le \frac{\delta_2 \delta^2 \varepsilon^2}{16}$. Then the code $\mathcal{C}(G, \mathcal{C}_0)$ has second generalized distance at least $(1-\varepsilon)\cdot\delta\cdot\min\{\delta_2, 2\delta\}$.*

We first note the following simple claim which provides an equivalent definition of generalized distance. For a vector $x \in \mathbb{F}_2^N$, let $\mathrm{Supp}(x) := \{i \in [n] \mid x_i \neq 0\}$.

Claim 5.1. *Let $C \subseteq \mathbb{F}_2^N$ be a linear code. The r'th generalized distance of C is*

$$\frac{1}{N} \min_{c_1, c_2, \ldots, c_r} \left|\mathrm{Supp}(c_1) \cup \mathrm{Supp}(c_2) \cup \cdots \cup \mathrm{Supp}(c_r)\right|,$$

where the minimum is taken over all r-tuples $c_1, c_2, \ldots, c_r$ of linearly independent codewords in C.

We proceed to the proof of Lemma 1.3. Let c, c' be two distinct non-zero codewords in $\mathcal{C}(G, \mathcal{C}_0)$. Letting $F := \mathrm{Supp}(c)$ and $F' := \mathrm{Supp}(c')$, by Claim 5.1 it suffices to show that

$$|F \cup F'| \ge (1 - \varepsilon) \cdot \delta \cdot \min\{\delta_2, 2\delta\} dn.$$

Let W denote the subset of vertices $v \in L \cup R$ which satisfy that $c|_{\{v\}\times\Gamma(v)}, c'|_{\{v\}\times\Gamma(v)}$ are two distinct non-zero codewords in $\mathcal{C}_0$. Let $\varepsilon_0 := \frac{\delta^2 \varepsilon}{8}$. Below we divide into cases.

Case 1: $|F \cap F'| \leq \varepsilon_0 dn$**.** In this case,

$$\begin{aligned} |F \cup F'| &= |F| + |F'| - |F \cap F'| \\ &\geq (2\delta(\delta - \lambda/d) - \varepsilon_0)\, dn \\ &\geq (1-\varepsilon)2\delta^2 dn, \end{aligned}$$

where the first inequality follows since by Lemma 2.3, the code $\mathcal{C}(G, \mathcal{C}_0)$ has relative distance at least $\delta(\delta - \lambda/d)$, and the second inequality follows by choice of $\frac{\lambda}{d} \leq \frac{\delta\varepsilon}{2}$ and $\varepsilon_0 \leq \delta^2\varepsilon$.

Case 2: $|W| \geq \varepsilon_0^2 n$**.** Without loss of generality, we may assume that $|W \cap L| \geq |W \cap R|$, so $|W \cap L| \geq \frac{\varepsilon_0^2}{2} n$. We apply the Expander Mixing Lemma with $S_1 := W \cap L$, and $T_1 \subseteq R$ the set of all right vertices that are incident to an edge from $F \cup F'$.

Recall that for any vertex $v \in S_1$ it holds that $c|_{\{v\}\times\Gamma(v)}, c'|_{\{v\}\times\Gamma(v)}$ are two distinct non-zero codewords in $\mathcal{C}_0$, and so

$$\left|\mathrm{Supp}\left(c|_{\{v\}\times\Gamma(v)}\right) \cup \mathrm{Supp}\left(c'|_{\{v\}\times\Gamma(v)}\right)\right| \geq \delta_2 \cdot d.$$

Therefore, any vertex $v \in S_1$ is incident to at least $\delta_2 d$ edges in $F \cup F'$, and so $|E(S_1, T_1)| \geq \delta_2 d|S_1|$.

By the Expander Mixing Lemma, the above implies in turn that

$$\delta_2 d|S_1| \leq |E(S_1, T_1)| \leq \frac{d}{n}|S_1||T_1| + \lambda\sqrt{|S_1||T_1|},$$

and rearranging gives

$$|T_1| \geq \left(\delta_2 - \frac{\lambda}{d}\sqrt{\frac{|T_1|}{|S_1|}}\right) n \geq \left(\delta_2 - \frac{\lambda}{d} \cdot \frac{2}{\varepsilon_0}\right) \cdot n,$$

where the second inequality follows by assumption that $|S_1| \geq \frac{\varepsilon_0^2}{2} n$.

Finally, note that any vertex $v \in T_1$ has at least δd incident edges in $F \cup F'$, and so

$$|F \cup F'| \geq \delta d|T_1| \geq \delta\left(\delta_2 - \frac{\lambda}{d} \cdot \frac{2}{\varepsilon_0}\right) dn \geq \delta\delta_2(1-\varepsilon)dn,$$

where the last inequality follows by choice of $\frac{\lambda}{d} \leq \frac{\delta_2\delta^2\varepsilon^2}{16} = \frac{\delta_2\varepsilon_0\varepsilon}{2}$.

Case 3: $|F \cap F'| \geq \varepsilon_0 dn$ **and** $|W| \leq \varepsilon_0^2 n$**.** Under these assumptions, Claims 5.2 and 5.3 below show that both the intersection $F \cap F'$ and the symmetric difference $F \triangle F'$ are of size at least $(1-\varepsilon)\delta^2 dn$. This implies in turn that

$$|F \cup F'| = |F \cap F'| + |F \triangle F'| \geq (1-\varepsilon)2\delta^2 dn.$$

Claim 5.2. $|F \cap F'| \geq (1-\varepsilon)\delta^2 dn$.

Proof. We apply the Expander Mixing Lemma with S_2 (T_2, resp.) being the set of all vertices $v \in L \setminus W$ ($v \in R \setminus W$, resp.) that are incident to an edge from $F \cap F'$. Without loss of generality we may assume that $|S_2| \geq |T_2|$.

Next observe that since $\mathcal{C}_0$ has relative distance at least δ, any vertex $v \in S_2$ is incident to at least δd edges in $F \cup F'$. We claim that these edges are in fact contained in $F \cap F'$; otherwise, v is incident to some edge from $F \cap F'$ and another edge from $F \triangle F'$ which means that $c|_{\{v\}\times\Gamma(v)}, c'|_{\{v\}\times\Gamma(v)}$ are two distinct non-zero codewords in $\mathcal{C}_0$, contradicting the assumption that $v \notin W$. We conclude that any vertex $v \in S_2$ has at least δd incident edges that are incident to either T_2 or W.

Consequently, we have that

$$|E(S_2, T_2)| \geq \delta d|S_2| - d|W| = \left(\delta - \frac{|W|}{|S_2|}\right) d|S_2| \geq \left(\delta - \frac{\varepsilon_0}{1-\varepsilon_0}\right) d|S_2|,$$

where the last inequality uses the assumptions that $|W| \leq \varepsilon_0^2 n$ and $|F \cap F'| \geq \varepsilon_0 dn$, implying in turn that

$$|S_2| \geq \frac{|F \cap F'|}{d} - |W| \geq \varepsilon_0(1-\varepsilon_0)n.$$

On the other hand, by the Expander Mixing Lemma we have that

$$|E(S_2, T_2)| \leq \frac{d}{n}|S_2||T_2| + \lambda\sqrt{|S_2||T_2|},$$

Combining the above, rearranging, and recalling the assumption that $|S_2| \geq |T_2|$, gives

$$|T_2| \geq \left(\delta - \frac{\varepsilon_0}{1-\varepsilon_0} - \frac{\lambda}{d}\right) n.$$

Finally, note that, similarly to the above, we have that any vertex $v \in T_2$ is incident to at least δd edges in $F \cap F'$. We conclude that

$$|F \cap F'| \geq \delta d|T_2| \geq \delta\left(\delta - \frac{\varepsilon_0}{1-\varepsilon_0} - \frac{\lambda}{d}\right) dn \geq \delta^2(1-\varepsilon)\, dn,$$

where the last inequality follows by choice of $\varepsilon_0 \leq \frac{\delta\varepsilon}{4}$ and $\frac{\lambda}{d} \leq \frac{\delta\varepsilon}{2}$. □

Claim 5.3. $|F \triangle F'| \geq (1-\varepsilon)\delta^2 dn$.

Proof. Similarly to the previous claim, we apply the Expander Mixing Lemma with S_3 (T_3, resp.) being the set of all vertices $v \in L \setminus W$ ($v \in R \setminus W$, resp.) that are incident to an edge from $F \triangle F'$, and we may assume that $|S_3| \geq |T_3|$.

Once more we observe that any vertex $v \in S_3$ is incident to at least δd edges in $F \triangle F'$, and we conclude that any vertex $v \in S_3$ has at least δd incident edges that are incident to either T_3 or W. Consequently, as before we have that

$$|E(S_3, T_3)| \geq \left(\delta - \frac{|W|}{|S_3|}\right) d|S_3| \geq \left(\delta - \frac{\varepsilon_0^2}{\delta\left(\delta - \frac{\lambda}{d}\right) - \varepsilon_0^2}\right) d|S_3|,$$

where the last inequality uses the assumption that $|W| \leq \varepsilon_0^2 n$, and the fact that $|F \triangle F'| \geq \delta\left(\delta - \frac{\lambda}{d}\right) dn$ (since $F \triangle F'$ is the support of the non-zero codeword $c + c' \in \mathcal{C}(G, \mathcal{C}_0)$), implying in turn that

$$|S_3| \geq \frac{|F \cap F'|}{d} - |W| \geq \delta\left(\delta - \frac{\lambda}{d}\right) n - \varepsilon_0^2 n.$$

On the other hand, by the Expander Mixing Lemma we have that

$$|E(S_3, T_3)| \leq \frac{d}{n}|S_3||T_3| + \lambda\sqrt{|S_3||T_3|},$$

and combining with the above, rearranging, and recalling the assumption that $|S_3| \geq |T_3|$, this gives

$$|T_3| \geq \left(\delta - \frac{\varepsilon_0^2}{\delta\left(\delta - \frac{\lambda}{d}\right) - \varepsilon_0^2} - \frac{\lambda}{d}\right) n.$$

Finally, note that, similarly to the above, we have that any vertex $v \in T_3$ is incident to at least δd edges in $F \cap F'$. We conclude that

$$|F \triangle F'| \geq \delta d|T_3| \geq \delta\left(\delta - \frac{\varepsilon_0^2}{\delta(\delta - \lambda/d) - \varepsilon_0^2} - \frac{\lambda}{d}\right) dn \geq \delta^2\left(1 - \varepsilon\right) dn,$$

where the last inequality follows by choice of $\varepsilon_0^2 \leq \frac{\delta^3\varepsilon}{8}$ and $\frac{\lambda}{d} \leq \frac{\delta\varepsilon}{2}$. □

Acknowledgements

Most of this work was done while the authors were participating in the Summer Cluster on Error-correcting Codes and High-dimensional Expansion at the Simons Institute for the Theory of Computing at UC Berkeley. We thank the Simons Institute for the hospitality.

References

[AS06] Alexei Ashikhmin and Vitaly Skachek. Decoding of expander codes at rates close to capacity. *IEEE Transactions on Information Theory*, 52(12):5475–5485, 2006.

[BDT20] Avraham Ben-Aroya, Dean Doron, and Amnon Ta-Shma. Near-optimal erasure list-decodable codes. In *proceedings of the Computational Complexity Conference (CCC)*, volume 169 of *LIPIcs*, pages 1:1–1:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[BZ002] Error exponents of expander codes. *IEEE Transactions on Information Theory*, 48(6):1725–1729, 2002.

[BZ05] Alexander Barg and Gilles Zémor. Concatenated codes: serial and parallel. *IEEE Transactions on Information Theory*, 51(5):1625–1634, 2005.

[BZ06] Alexander Barg and Gilles Zémor. Distance properties of expander codes. *IEEE Transactions on Information Theory*, 52(1):78–90, 2006.

[DJX14] Yang Ding, Lingfei Jin, and Chaoping Xing. Erasure list-decodable codes from random and algebraic geometry codes. *IEEE Transactions on Information Theory*, 60(7):3889–3894, 2014.

[DL12] Zeev Dvir and Shachar Lovett. Subspace evasive sets. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 351–358. ACM Press, 2012.

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 812–821. ACM Press, 2002.

[GI04] Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings. In *proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3142 of *Lecture Notes in Computer Science*, pages 695–707. Springer, 2004.

[GK16] Venkatesan Guruswami and Swastik Kopparty. Explicit subspace designs. *Combinatorica*, 36(2):161–185, 2016.

[GQST20] Fernando Granha Jeronimo, Dylan Quintana, Shashank Srivastava, and Madhur Tulsiani. Unique decoding of explicit ε-balanced codes near the gilbert-varshamov bound. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2020.

[GR06] Philippe Gaborit and Olivier Ruatta. Efficient erasure list-decoding of Reed-Muller codes. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 148–152. IEEE, 2006.

[GR08] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[GR21] Zeyu Guo and Noga Ron-Zewi. Efficient list-decoding with constant alphabet and list sizes. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2021.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[Gur03] Venkatesan Guruswami. List decoding from erasures: Bounds and code constructions. *IEEE Transactions on Information Theory*, 49(11):2826–2833, 2003.

[GW17] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017.

[GX12] Venkatesan Guruswami and Chaoping Xing. Folded codes from function field towers and improved optimal rate list decoding. In *Proceedings of the 44th annual ACM symposium on Theory of computing (STOC)*, pages 339–350. ACM Press, 2012.

[GX13] Venkatesan Guruswami and Chaoping Xing. List decoding Reed-Solomon, algebraic-geometric, and Gabidulin subcodes up to the Singleton bound. In *Proceedings of the 45th annual ACM symposium on Theory of Computing (STOC)*, pages 843–852. ACM Press, 2013.

[HLW06] Shlomo Hoory, Nati Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of AMS*, 43(4):439–561, 2006.

[HOW15] Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. Local correctability of expander codes. *Information and Computation*, 243:178–190, 2015.

[HRW20] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes and applications. *SIAM Journal on Computing*, 49(4):157–195, 2020.

[HW18] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018.

[KRR+21] Swastik Kopparty, Nicolas Resch, Noga Ron-Zewi, Shubhangi Saraf, and Shashwat Silas. On list recovery of high-rate tensor codes. *IEEE Transactions on Information Theory*, 67(1):296–316, 2021.

[KRSW18] Swastik Kopparty, Noga Ron-Zewi, Shubhangi Saraf, and Mary Wootters. Improved decoding of folded Reed-Solomon and multiplicity codes. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 212–223. IEEE Computer Society, 2018.

[LMSS01] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[MRR+20] Jonathan Mosheiff, Nicolas Resch, Noga Ron-Zewi, Shashwat Silas, and Mary Wootters. LDPC codes achieve list-decoding capacity. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2020.

[PV05] Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the Guruswami–Sudan radius in polynomial time. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–294. IEEE Computer Society, 2005.

[RS06] Ron M Roth and Vitaly Skachek. Improved nearly-MDS expander codes. *IEEE Transactions on Information Theory*, 52(8):3650–3661, 2006.

[Sch00] Hans Georg Schaathun. The weight hierarchy of product codes. *IEEE Transactions on Information Theory*, 46(7):2648–2651, 2000.

[SR03] Vitaly Skachek and Ron M Roth. Generalized minimum distance iterative decoding of expander codes. In *Proceedings of the IEEE Information Theory Workshop (ITW)*, pages 245–248. IEEE, 2003.

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[Sud97] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[WY93] Victor K.-W. Wei and Kyeongcheol Yang. On the generalized Hamming weights of product codes. *IEEE Transactions on Information Theory*, 39(5):1709–1713, 1993.

[Zém01] Gilles Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

A Erasure unique decoding of expander codes

In this appendix we prove Lemma 2.4, which we repeat here:

Lemma 2.4 (restated). *Let $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$ be a linear code with distance δ, and let $G = (L \cup R, E)$ be a bipartite d-regular graph on $2n$ vertices with $\lambda = \lambda(G)$. Let $\varepsilon > 0$, and suppose that $\frac{\lambda}{d} < \frac{\delta}{2}$. Then there is an algorithm* UniqueDecode *which uniquely decodes the expander code $\mathcal{C}(G, \mathcal{C}_0)$ from up to $(1-\varepsilon)\delta(\delta - \lambda/d)$ erasures in time $n \cdot \mathrm{poly}(d)/\varepsilon$.*

We note that this lemma is well-known and follows from the techniques of [SS96, Zém01]. However, we include its proof for completeness, because the algorithm FindList mirrors its structure.

The proof of the lemma follows from the algorithm UniqueDecode, given in Figure 5.

To see that UniqueDecode is correct, first notice that on any iteration $t = 2, 3, \ldots$, the set E_{t-1} is the subset of edges that have already been labeled before this iteration, and $P_{t-2} \cup P_{t-1}$ is the set of vertices touching an edge in $E \setminus E_{t-1}$ that we yet need to decode. The following claim bounds the size of P_t, and consequently the number of steps the algorithm runs until it terminates on Step 1.

Claim A.1. *The following hold:*

1. *For any $t \geq 1$, $|P_{t+1}| \leq (1-\varepsilon)\left(\delta - \frac{\lambda}{d}\right) n$.*
2. *For any $t \geq 2$, $|P_{t+1}| \leq \left(\frac{1}{1+\varepsilon}\right)^2 |P_t|$.*

Proof. For $t = 1, 2, 3, \ldots$, let $B_{t-1} \subseteq P_{t-1}$ be the subset of vertices $v \in P_{t-1}$ that are incident to less than $(1-\delta)d$ edges in E_{t-1}. Then we have

$$P_{t+1} \subseteq B_{t-1} \subseteq P_{t-1}, \tag{8}$$

as all vertices $v \in P_{t-1} \setminus B_{t-1}$ are removed from P_{t-1} on Step (3b), and consequently will not be present in P_{t+1}.

For the first item, note that $|P_3| \leq |B_1|$ by (8), and that $|B_1| \leq \left(\delta - \frac{\lambda}{d}\right)(1-\varepsilon)n$ since there are less than $(1-\varepsilon)\delta\left(\delta - \frac{\lambda}{d}\right) nd$ erasures to begin with. Moreover, we have that $|P_3| \geq |P_5| \geq |P_7| \geq \cdots$, and consequently $|P_{t+1}| \leq (1-\varepsilon)\left(\delta - \frac{\lambda}{d}\right) n$ for any even $t \geq 1$. Similar reasoning shows that the same holds for any odd $t \geq 1$.

For the second item, note that by the Expander Mixing Lemma, for $t = 2, 3, \ldots$,

$$\delta d |B_{t-1}| \leq |E(B_{t-1}, P_t)| \leq \frac{d}{n}|B_{t-1}||P_t| + \lambda\sqrt{|B_{t-1}||P_t|},$$

as any vertex $v \in B_{t-1}$ has at least δd unlabeled incident edges, and those edges are incident to P_t. Rearranging, we have

$$|B_{t-1}| \leq \left(\frac{\lambda/d}{\delta - |P_t|/n}\right)^2 |P_t| \leq \left(\frac{1}{1+\varepsilon}\right)^2 |P_t|,$$

Algorithm: UniqueDecode
Inputs: A description of $G = (L \cup R, E)$ and $\mathcal{C}_0 \subseteq \mathbb{F}_2^d$, and $z \in (\mathbb{F}_2 \cup \{\perp\})^E$.
Output: The unique $c \in \mathcal{C}(G, \mathcal{C}_0)$ so that c agrees with z on all un-erased positions.

Initialize:

- $E_1 := \{e \in E | z_e \neq \perp\}$
- $P_0 := \{v \in R \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$
- $P_1 := \{v \in L \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$

For $t = 2, 3, \ldots$:

1. If $P_{t-1} = \emptyset$, return the fully labeled codeword.
2. Initialize $P_t \leftarrow \emptyset$ and $E_t \leftarrow E_{t-1}$.
3. For each vertex $v \in P_{t-1}$ so that $|(\{v\} \times \Gamma(v)) \cap E_{t-1}| > (1 - \delta)d$:
   (a) Run $\mathcal{C}_0$'s erasure-correction algorithm to assign labels to the edges incident to v.
   (b) Remove v from P_{t-1}.
   (c) For any $(v, u) \notin E_{t-1}$, add (v, u) to E_t.
4. For each vertex $v \in P_{t-1}$, for any $(v, u) \in E \setminus E_{t-1}$, add u to P_t.

Figure 5: UniqueDecode: Uniquely decodes an expander code from up to $\delta(\delta - \lambda/d)(1 - \varepsilon)$ erasures.

where the last inequality follows by assumption that $\frac{\lambda}{d} \leq \frac{\delta}{2}$, and since $\frac{|P_t|}{n} \leq (1-\varepsilon)(\delta - \lambda/d)$ by the first item. Finally, by (8) this implies in turn that

$$|P_{t+1}| \leq |B_{t-1}| \leq \left(\frac{1}{1+\varepsilon}\right)^2 |P_t|.$$

□

Using the above claim we conclude that after $O((\log n)/\varepsilon)$ iterations the set P_{t-1} is empty, and so the algorithm terminates. Moreover, the amount of work done is less than

$$\mathrm{poly}(d) \cdot \sum_{t=1}^{\infty} |P_t| = \mathrm{poly}(d) \cdot n \sum_{t=1}^{\infty} \left(\frac{1}{1+\varepsilon}\right)^{2t} = \mathrm{poly}(d) \cdot \frac{n}{\varepsilon},$$

which proves the lemma.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il