Simple Codes and Sparse Recovery with Fast Decoding

Mahdi Cheraghchi* João Ribeiro†‡

Abstract

Construction of error-correcting codes achieving a designated minimum distance parameter is a central problem in coding theory. In this work, we study a very simple construction of binary linear codes that correct a given number of errors K. Moreover, we design a simple, nearly optimal syndrome decoder for the code as well. The running time of the decoder is only logarithmic in the block length of the code, and nearly linear in the number of errors K. This decoder can be applied to exact for-all sparse recovery over any field, improving upon previous results with the same number of measurements. Furthermore, computation of the syndrome from a received word can be done in nearly linear time in the block length. We also demonstrate an application of these techniques in non-adaptive group testing, and construct simple explicit measurement schemes with $O(K^2 \log^2 N)$ tests and $O(K^3 \log^2 N)$ recovery time for identifying up to K defectives in a population of size N.

1 Introduction

The problem of constructing low-redundancy codes with practical decoding algorithms that handle a prescribed number of adversarial errors has been extensively studied in coding theory. We distinguish between two standard decoding settings for linear codes: *Syndrome decoding*, where one has access to the syndrome of the corrupted codeword, and *full decoding*, where one has access to the corrupted codeword itself. In both cases, the goal is to return the error pattern.

The syndrome holds extra pre-computed information about the corrupted codeword. As a result, we expect to be able to perform syndrome decoding much faster than full decoding. In fact, while full decoding has complexity at least linear in the block length of the code, syndrome decoding can be potentially accomplished in time *sublinear* in the block length. Syndrome decoding is important for various reasons: In many cases, the most efficient way we have of performing full decoding for a given linear code is to first compute the syndrome from the corrupted codeword, and then run a syndrome decoding algorithm. Furthermore, as we shall see, syndrome decoding is connected to other widely studied recovery problems.

Two examples of families of high-rate linear codes with good decoding guarantees are the classical BCH codes, which are widely used in practice, and expander codes, introduced by Sipser and Spielman [1]. The properties of BCH codes derive from the theory of polynomials over large finite fields. On the other hand, the guarantees behind expander codes follow from the combinatorial properties of the underlying expander graphs. Due to their combinatorial nature, expander codes have simple decoders, while decoding BCH codes requires algorithms which perform arithmetic over large fields. However, while BCH codes support sublinear syndrome decoding [2, 3], no such sublinear syndrome decoders are known for expander codes.

Syndrome decoding of linear codes can be interpreted as sparse recovery over a finite field. In *exact for-all* sparse recovery, we aim to recover all sparse vectors x from Wx, where W is a measurement matrix (which may be sampled with high probability from some distribution). The goal is to minimize recovery time and number of measurements (i.e., rows of W) with respect to the sparsity and length of the vectors. We say a for-all sparse recovery scheme is *approximate* if it allows recovery (within some error) of the best sparse approximation of arbitrary vectors. There has been significant interest in developing combinatorial algorithms for sparse recovery. Unlike their geometric counterparts, such procedures have the advantage of

*EECS Department, University of Michigan, Ann Arbor, MI, USA. Email: mahdich@umich.edu.

†Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA. Email: jlourenc@cs.cmu.edu.

‡This work was mainly performed while the authors were with the Department of Computing, Imperial College London, UK.

supporting sublinear time recovery. Furthermore, while sparse recovery is normally studied over the reals, such algorithms can usually be modified to perform sparse recovery over any field.

In this work, we study a simple combinatorial construction of high-rate binary linear codes that support nearly optimal syndrome decoding. While we present our decoding algorithms over $\mathbb{F}_2$, our syndrome decoder can be adapted to work over any field, improving upon previous combinatorial algorithms for exact for-all sparse recovery with the same number of measurements. This adaptation to arbitrary fields is discussed in more detail at the end of Section 4.1.1. We remark that the parity-check matrix of our code remains 0-1-valued even when we work over fields other than GF(2).

A different sparse recovery problem that has been extensively studied is that of *Non-Adaptive Group Testing* (NAGT). In this setting, our goal is to identify all defectives within a given population. To this end, we may perform tests by pooling items of our choice and asking whether there is a defective item in the pool. In NAGT, tests are fixed a priori, and so cannot depend on outcomes of previous tests. A main problem in this area consists in finding NAGT schemes supporting sublinear recovery time with few tests in the *zero-error* regime, where it is required that the scheme always succeeds in recovering the set of defectives. In the second part of this work, we present a zero-error NAGT scheme with few tests and a competitive sublinear time recovery algorithm.

1.1 Related Work

Sublinear time decoding of BCH codes was studied by Dodis, Reyzin, and Smith [2, 3], who gave a $\mathrm{poly}(K \log N)$ time syndrome decoder, where N is the block length and K is the maximum number of errors. More precisely, according to [3, Proof of Lemma 1] their decoder asymptotically requires $\Theta(K \log^2 K \cdot \log\log K + K^2 \log N)$ multiplications over the extension field of order $N+1$. In turn, we can perform multiplications over this field in time $\Theta(\log N \cdot \log\log N)$ under a believable assumption [4]. Combining both results yields asymptotic syndrome decoding complexity

$$\Theta((K \log^2 K \cdot \log\log K + K^2 \log N) \cdot \log N \cdot \log\log N).$$

In contrast, the best syndrome decoders for expander codes run in time $O(DN)$, where D is the left degree of the expander [5]. Full decoding of expander codes takes time $O(N)$ in the regime where $K = \Theta(N)$ [1], but it takes time $O(N \log N)$ when K is small and we want the rate of the expander code to be large [5].

The work on combinatorial for-all sparse recovery algorithms was initiated by Cormode and Muthukrishnan [6], and several others soon followed [7, 8, 9] with improved recovery time and number of measurements. More recently, sublinear time combinatorial algorithms for approximate for-all sparse recovery with optimal number of measurements [10] or very efficient recovery [11] were given (both with strong approximation guarantees). We compare the results obtained by these works in the context of for-all sparse recovery with the result we obtain in this work in Section 1.2.

The first zero-error NAGT schemes supporting sublinear recovery time with few tests were given independently by Cheraghchi [12] and Indyk, Ngo, and Rudra [13]. In [13], the authors present an NAGT scheme which requires $T = O(K^2 \log N)$ tests (which is order-optimal) and supports recovery in time $\mathrm{poly}(K) \cdot T \log^2 T + O(T^2)$, where N is the population size and K is the maximum number of defectives. However, their scheme is only explicit when $K = O\left(\frac{\log N}{\log\log N}\right)$. Cheraghchi [12] gives explicit schemes which require a small number of tests and handle a constant fraction of test errors. However, his schemes may output some false positives. While this can be remedied, the resulting recovery time will be worse. Later, Ngo, Porat, and Rudra [14] also obtained explicit sublinear time NAGT schemes with near-optimal number of tests that are robust against test errors. In particular, they obtain schemes requiring $T = O(K^2 \log N)$ tests with recovery time $\mathrm{poly}(T)$. Subsequently to the publication of a shortened version of the present work [15], Cheraghchi and Nakos [16] constructed explicit NAGT schemes requiring $T = O(K^2 \log N)$ tests and recovery time nearly linear in T.

1.2 Contributions and Techniques

Our binary linear codes are a *bitmasked* version of expander codes. Roughly speaking, bitmasking a binary $M \times N$ matrix W consists in replacing each entry in W by the $\log N$-bit binary expansion of its corresponding

column index, multiplied by that entry of W. This gives rise to a new bitmasked matrix of dimensions $M \log N \times N$. The bitmasking technique has already been used in [6, 7, 8, 9, 11] in several different ways to obtain sublinear time recovery algorithms for approximate sparse recovery. We provide a detailed explanation of this technique and its useful properties in Section 3.

The parity-check matrices of our codes are obtained by following the same ideas as [9, 11]: We take the parity-check matrix of an expander code, bitmask it, and stack the two matrices. Note that these codes have a blowup of $\log N$, where N is the block length, in the redundancy when compared to expander codes. However, we show that these codes support randomized and deterministic syndrome decoding in time $O(K \log K \cdot \log N)$ under a random expander, where K is the number of errors. We remark that this is within an $O(\log K)$ factor of the optimal recovery time for small K. In particular, this also leads to randomized and deterministic full decoders running in time $O(\log K \cdot N \log N)$.

Our syndrome decoders can be made to work over any field with almost no modification. As a result, we obtain a recovery algorithm for exact for-all sparse recovery over any field with nearly optimal recovery time $O(K \log K \cdot \log N)$ from $O(K \log^2 N)$ measurements. This improves upon the recovery time of previous schemes using the same number of measurements in the exact for-all sparse recovery setting [9, 11]. Although sublinear time recovery is possible with fewer measurements (the optimal number of measurements is $O(K \log(N/K))$), the dependency on N in that case is generally worse than what we obtain, which is optimal. A detailed comparison between our work and previous results in the for-all sparse recovery setting can be found in Table 1.

Our randomized decoders have several advantages over their deterministic counterparts that make them more practical. First, the hidden constants in the runtime are smaller. Second, the runtime is independent of the degree of the underlying expander. As a result, we can instantiate our codes under explicit expanders with sub-optimal degree without affecting the decoding complexity. Third, the failure probability of the algorithm has a negligible effect on its runtime for large block lengths. Therefore, it can be set to an arbitrarily small constant of choice with limited effect on the runtime. In particular, our randomized full decoder is faster than the expander codes decoder even under a random expander if K is small.

	Recovery time	Approximate? (Y/N)
[6]	$K^2 \log^2 N$	N
[7]	$K \log^2 K \cdot \log^2 N$	Y
[8]	$K^2 \cdot \text{polylog}(N)$	Y
[9]	$K \log K \cdot \log^2 N$	N
[11]	$K \log K \cdot \log^2 N$	Y
This work (see Remark 1)	$K \log K \cdot \log N$	N
$< K \log^2 N$ measurements [10]	$\text{poly}(K \log N)$	Y
BCH codes [3, 4]	$K^2 \log^2 N \cdot \log \log N$	N

Table 1: Summary of best known previous results and the result obtained in this work on sublinear recovery in the for-all sparse recovery setting. Here, K denotes the sparsity and N the vector length. We omit the $O(\cdot)$ notation in recovery times for simplicity, and in the third column we distinguish between schemes that work for approximate sparse recovery (i.e., arbitrary vectors), versus those that work only in the exact for-all setting. Since it is not relevant for us, we do not distinguish between the approximation guarantees obtained in each work for approximate sparse recovery. All works except the last two rows require at least $O(K \log^2 N)$ measurements, which is the number of measurements our scheme requires. While sublinear decoding is possible in those cases, the dependency on N is generally worse than what we obtain. For a more detailed description of such schemes, see [10, Table 1] and [7, Table 1]. As mentioned before, our measurement matrices remain 0-1-valued even when we work over fields other than GF(2).

Remark 1. *We assume that reading an integer from memory takes time $O(1)$. If instead we assume that reading an L-bit integer takes time $O(L)$, then we incur an extra $\log K$ factor in our deterministic recovery time, for a total runtime of $O(K \log^2 K \cdot \log N)$.*

In the second part of our work, we present a simple, explicit zero-error NAGT scheme with recovery time $O(K^3 \log^2 N)$ from $O(K^2 \log^2 N)$ tests, where N is the population size and K is the maximum number

of defectives. Such a scheme is obtained by bitmasking a *disjunct* matrix. At a high-level, the recovery algorithm works as follows: First, we use the bitmask to obtain a small superset of the set of defectives. Then, we simply apply the naive recovery algorithm for general disjunct matrices to this superset to remove all false positives. The recovery time of this scheme is better than of those presented in Section 1.1, albeit we are an $O(\log N)$ factor away from the optimal number of tests. Moreover, unlike our scheme, those schemes make use of algebraic list-decodable codes and hence require sophisticated recovery algorithms with large constants. Finally, we note that the bitmasking technique has been used in a different way to obtain efficient NAGT schemes which recover a large fraction of defectives, or even all defectives, with high probability [17].

1.3 Organization

In Section 2, we introduce several concepts and results in coding and group testing that will be relevant for our work in later sections. Then, in Section 3 we present the bitmasking technique and its original application in sparse recovery. We present our code construction and the decoding algorithms in Section 4. Finally, our results on non-adaptive group testing can be found in Section 5.

2 Preliminaries

2.1 Notation

We denote the set $\{0, \dots, N-1\}$ by $[N]$. Given a vector x, we denote its support $\{i : x_i \neq 0\}$ by $\mathsf{supp}(x)$. We say a vector is K-sparse if $|\mathsf{supp}(x)| \leq K$. We index vectors and matrix rows/columns starting at 0. Sets are denoted by calligraphic letters like $\mathcal{S}$ and $\mathcal{X}$. In general, we denote the base-2 logarithm by log. Given a graph G and a set of vertices $\mathcal{S}$, we denote its neighborhood in G by $\Gamma(\mathcal{S})$. For a matrix W, we denote its i-th row by $W_{i\cdot}$ and its j-th column by $W_{\cdot j}$.

2.2 Unbalanced Bipartite Expanders

In this section, we introduce unbalanced bipartite expander graphs. We will need such graphs to define our code in Section 4.

Definition 2 (Bipartite Expander)**.** *A bipartite graph $G = (\mathcal{L}, \mathcal{R}, \mathcal{E})$ is said to be a* (D, K, ϵ)-bipartite expander *if every vertex $u \in \mathcal{L}$ has degree D (i.e., G is left D-regular) and for every $\mathcal{S} \subseteq \mathcal{L}$ satisfying $|\mathcal{S}| \leq K$ we have $|\Gamma(\mathcal{S})| \geq (1-\epsilon)D|\mathcal{S}|$.*

Such a graph is said to be layered *if we can partition $\mathcal{R}$ into disjoint subsets $\mathcal{R}_1, \dots, \mathcal{R}_D$ with $|\mathcal{R}_i| = |\mathcal{R}|/D$ for all i such that every $u \in \mathcal{L}$ has degree 1 in the induced subgraph $G_i = (\mathcal{L}, \mathcal{R}_i, \mathcal{E})$. We call such $i \in [D]$* seeds *and denote the neighborhood of $\mathcal{S}$ in G_i by $\Gamma_i(\mathcal{S})$.*

The graph G can be defined by the function $C : \mathcal{L} \times [D] \to \mathcal{R}$ which maps $(u, i) \in \mathcal{L} \times [D]$ to the neighbor of u in $\mathcal{R}_i$. For convenience, we also define $C_i = C(\cdot, i)$, which defines the subgraph G_i.

Informally, we say that a bipartite expander graph is *unbalanced* if $|\mathcal{R}|$ is much smaller than $|\mathcal{L}|$. The next lemma follows immediately from Markov's inequality.

Lemma 3. *Fix some $c > 1$ and a set $\mathcal{S}$ such that $|\mathcal{S}| \leq K$, and let G be a (D, K, ϵ)-layered bipartite expander. Then, for at least a $(1 - 1/c)$-fraction of seeds $i \in [D]$, it holds that*

$$|\Gamma_i(\mathcal{S})| \geq (1 - c\epsilon)|\mathcal{S}|.$$

We will also need the following lemma that bounds the number of right vertices with a single neighbor in a given subset of left vertices.

Lemma 4. *Let $G = (\mathcal{L}, \mathcal{R}, \mathcal{E})$ be a layered bipartite graph. If $\mathcal{S} \subseteq \mathcal{L}$ satisfies $|\Gamma_i(\mathcal{S})| \geq (1-\delta)|\mathcal{S}|$, then the number of right vertices in $\Gamma_i(\mathcal{S})$ with only one neighbor in $\mathcal{S}$ (with respect to the subgraph G_i) is at least $(1 - 2\delta)|\mathcal{S}|$.*

Proof. Suppose that there are fewer than $(1-2\delta)|\mathcal{S}|$ vertices in $\Gamma_i(\mathcal{S})$ with only one neighbor in $\mathcal{S}$. Since G_i has left-degree 1, this means that there are more than $2\delta|\mathcal{S}|$ vertices in $\mathcal{S}$ which are adjacent to right vertices with degree at least 2. Therefore, we can upper bound $|\Gamma_i(\mathcal{S})|$ as

$$|\Gamma_i(\mathcal{S})| < (1-2\delta)|\mathcal{S}| + \frac{1}{2}\cdot 2\delta|\mathcal{S}| = (1-\delta)|cS|,$$

which contradicts our assumption. □

The next theorem states we can sample nearly-optimal layered unbalanced bipartite expanders with high probability.

Theorem 5 ([18, Lemma 4.2])**.** *Given any N, K, and ϵ, we can sample a layered (D, K, ϵ)-bipartite expander graph $G = (\mathcal{L} = [N], \mathcal{R} = [MD], \mathcal{E})$ with high probability for $D = O\Big(\frac{\log N}{\epsilon}\Big)$ and $M = O\big(\frac{K}{\epsilon}\big)$.*

The graph in Theorem 5 can be obtained by sampling a random function $C\colon [N] \times [D] \to [M]$, and choosing $(x;(s,y))$ as an edge in G if $C(x,s) = y$.

2.3 Coding Theory

In this section, we define some basic concepts from coding theory that we use throughout our paper. We point the reader to [19] for a much more complete treatment of the topic.

Given an alphabet Σ, a *code over Σ of length N* is simply a subset of Σ^N. We will be focusing on the case where codes are *binary*, which corresponds to the case $\Sigma = \{0,1\}$. For a code $\mathcal{C} \subseteq \Sigma^N$, we call N the *block length* of $\mathcal{C}$. If $|\Sigma| = q$, the *rate* of $\mathcal{C}$ is defined as

$$\frac{\log_q |\mathcal{C}|}{N}.$$

We will study families of codes indexed by the block length N. We do not make this dependency explicit, but it is always clear from context.

If Σ is a field, we say $\mathcal{C}$ is a *linear code* if $c_1 + c_2 \in \mathcal{C}$ whenever $c_1, c_2 \in \mathcal{C}$. In other words, $\mathcal{C}$ is a linear code if it is a vector space over Σ. To each linear code $\mathcal{C}$ we can associate a unique *parity-check matrix* H such that $\mathcal{C} = \ker H$. Given such a parity-check matrix H and a vector $x \in \Sigma^N$, we call Hx the *syndrome* of x. Clearly, we have $x \in \mathcal{C}$ if and only if its syndrome is zero.

2.4 Group Testing

As mentioned in Section 1, in group testing we are faced with a set of N items, some of which are defective. Our goal is to correctly identify all defective items in the set. To this end, we are allowed to test subsets, or pools, of items. The result of such a test is 1 if there is a defective item in the pool, and 0 otherwise. Ideally, we would like to use as few tests as possible, and have efficient algorithms for recovering the set of defective items from the test results.

In Non-Adaptive Group Testing (NAGT), all tests are fixed a priori, and so cannot depend on the outcome of previous tests. While one can recover the defective items with fewer tests using adaptive strategies, practical constraints preclude their use and make non-adaptive group testing relevant for most applications.

It is useful to picture the set of N items as a binary vector $x \in \{0,1\}^N$, where the 1's stand for defective items. Then, the T tests to be performed can be represented by a $T \times N$ test matrix W satisfying

$$W_{ij} = \begin{cases} 1, & \text{if item } j \text{ is in test } i \\ 0, & \text{else.} \end{cases}$$

The outcome of the T tests, which we denote by $W \odot x$, corresponds to the bit-wise union of all columns corresponding to defective items. In other words, if $\mathcal{S}$ denotes the set of defective items, we have

$$W \odot x = \bigvee_{j \in \mathcal{S}} W_{\cdot j},$$

where the bit-wise union of two N-bit vectors, $x \vee y$, is an N-bit vector satisfying

$$(x \vee y)_i = \begin{cases} 1, \text{ if } x_i = 1 \text{ or } y_i = 1 \\ 0, \text{ else.} \end{cases}$$

We say an NAGT scheme is *zero-error* if we can always correctly recover the set of defectives. Zero-error NAGT schemes are equivalent to *disjunct* matrices.

Definition 6 (Disjunct matrix)**.** *A matrix W is said to be* d-disjunct *if the bit-wise union of any up to d columns of W does not contain any other column of W.*

The term *contains* used in Definition 6 is to be interpreted as follows: A vector x is contained in a vector y if $y_i = 1$ whenever $x_i = 1$, for all i. If we know there are at most K defective items, taking the rows of a K-disjunct $T \times N$ matrix as the tests to be performed leads to a NAGT algorithm with T tests and a simple $O(TN)$ recovery algorithm: An item is not defective if and only if it is part of some test with a negative outcome, so one can just check whether each item participates in a negative test.

As a result, much effort has been directed at obtaining better randomized and explicit[1] constructions of K-disjunct matrices, with as few tests as possible with respect to number of defectives K and population size N. The current best explicit construction of a K-disjunct matrix due to Porat and Rothschild [20] requires $O(K^2 \log N)$ rows (i.e., tests), while a probabilistic argument shows that it is possible to sample K-disjunct matrices with $O(K^2 \log(N/K))$ rows with high probability. We remark that both these results are optimal up to a $\log K$ factor [21].

Theorem 7 ([21, 20])**.** *There exist explicit constructions of K-disjunct matrices with $T = O(K^2 \log N)$ rows. Moreover, it is possible to sample a K-disjunct matrix with $T = O(K^2 \log(N/K))$ rows with high probability. Both these results are optimal up to a $\log K$ factor, and the matrix columns are $O(K \log N)$-sparse in the two constructions.*

3 Bitmasked Matrices and Exact Sparse Recovery

In this section, we describe the bitmasking technique, along with its application in sparse recovery presented by Berinde et al. [9]. As already mentioned, later on Cheraghchi and Indyk [11] modified this algorithm to handle approximate sparse recovery with stronger approximation guarantees, and gave a randomized version of this algorithm that allows for faster recovery.

Fix a matrix W with dimensions $M \times N$. Consider another matrix B of dimensions $\log N \times N$ such that the j-th column of B contains the binary expansion of j with the least significant bits on top. We call B a *bit-test matrix*. Given W and B, we define a new *bitmasked* matrix $W \otimes B$ with dimensions $M \log N \times N$ by setting the i-th row of $W \otimes B$ for $i = q \log N + t$ as the coordinate-wise product of the rows $W_{q\cdot}$ and $B_{t\cdot}$ for $q \in [M]$ and $t \in [\log N]$. This means that we have

$$(W \otimes B)_{i,j} = W_{q,j} \cdot B_{t,j} = W_{q,j} \cdot \mathsf{bin}_t(j),$$

for $i = q \log N + t$ and $j \in [N]$, where $\mathsf{bin}_t(j)$ denotes the t-th least significant bit of j.

Let W be the adjacency matrix of a (D, K, ϵ)-bipartite expander graph $G = (\mathcal{L} = [N], \mathcal{R} = [M], \mathcal{E})$. We proceed to give a high level description of the sparse recovery algorithm from [9]. Suppose we are given access to $(W \otimes B)x$ for some unknown K-sparse vector x of length N. Recall that our goal is to recover x from this product very efficiently. A fundamental property of the bitmasked matrix $W \otimes B$ is the following: Suppose that for some $q \in [M]$ we have

$$\mathsf{supp}(x) \cap \mathsf{supp}(W_{q\cdot}) = \{u\} \tag{1}$$

for some $u \in [N]$. We claim that we can recover the binary expansion of u directly from the $\log N$ products

$$(W \otimes B)_{q \log N} \cdot x, (W \otimes B)_{q \log N + 1} \cdot x, \ldots, (W \otimes B)_{q \log N + \log N - 1} \cdot x.$$

[1] By an *explicit* construction, we mean one in which we can construct the matrix in time polynomial in N.

In fact, if (1) holds, then for $i = q \log N + t$ it is the case that

$$(W \otimes B)_i \cdot x = \sum_{j=1}^{N} W_{q,j} \cdot \mathsf{bin}_t(j) \cdot x_j = \mathsf{bin}_t(u),$$

since $W_{q,j} \cdot x_j \neq 0$ only if $j = u$. In words, if u is the unique neighbor of q in $\mathsf{supp}(x)$, then the entries of $(W \otimes B)x$ indexed by $q \log N, \ldots, q \log N + \log N - 1$ spell out the binary expansion of u.

By the discussion above, if the edges exiting $\mathsf{supp}(x)$ in G all had different neighbors in the right vertex set, we would be able to recover x by reading off the binary expansion of the elements of $\mathsf{supp}(x)$ from the entries of $(W \otimes B)x$. However, if there are edges (u, v) and (u', v) for $u, u' \in \mathsf{supp}(x)$ in G, it is not guaranteed that we will recover u and u' as elements of $\mathsf{supp}(x)$. While such collisions are unavoidable, and thus we cannot be certain we recover x exactly, the expander properties of G ensure that the number of collisions is always a small fraction of the total number of edges. This means we will make few mistakes when reconstructing $\mathsf{supp}(x)$. As a result, setting ϵ to be a small enough constant and using a simple voting procedure (which we refrain from discussing as it is not relevant to us), we can recover a sparse vector y that approximates x in the sense that

$$\|x - y\|_0 \leq \frac{\|x\|_0}{2}.$$

We can then repeat the procedure on input $(W \otimes B)(x - y)$ to progressively refine our approximation of x. As a result, we recover a K-sparse vector x exactly in at most $\log K$ iterations.

4 Code Construction and Decoding

In this section, we define our code that is able to tolerate a prescribed number of errors, and analyze efficient syndrome decoding and full decoding algorithms.

Let N be the desired block-length of the code, K an upper bound on the number of adversarial errors introduced, and $\epsilon \in (0, 1)$ a constant to be determined later. We fix an adjacency matrix W of a (D, K, ϵ)-layered unbalanced bipartite expander graph $G = (\mathcal{L}, \mathcal{R}, \mathcal{E})$ with $\mathcal{L} = [N]$ and $\mathcal{R} = [D \cdot M]$, where

$$D = O\left(\frac{\log N}{\epsilon}\right) \quad \text{and} \quad M = O\left(\frac{K}{\epsilon}\right).$$

Such an expander G can be obtained with high probability by sampling a random function $C\colon [N] \times [D] \to [M]$ as detailed in Section 2.2.

We define our code $\mathcal{C} \subseteq \{0, 1\}^N$ by setting its parity-check matrix H as

$$H = \begin{bmatrix} W \\ W \otimes B \end{bmatrix}.$$

It follows that H has dimensions $(D \cdot M(1 + \log N)) \times N$, and so $\mathcal{C}$ has rate at least

$$1 - \frac{D \cdot M(1 + \log N)}{N} = 1 - O\left(\frac{K \cdot \log^2 N}{\epsilon^2 N}\right).$$

In particular, if K and ϵ are constants, then $\mathcal{C}$ has rate at least $1 - O\left(\frac{\log^2 N}{N}\right)$.

Note that instead of storing the whole parity-check matrix H in memory, one can just store the function table of C, which requires space $ND \log M$, along with a binary lookup table of dimensions $\log N \times N$ containing the $\log N$-bit binary expansions of $0, \ldots, N - 1$.

4.1 Syndrome Decoding

In this section, we study algorithms for syndrome decoding of $\mathcal{C}$. Fix some codeword $c \in \mathcal{C}$, and suppose c is corrupted by some pattern of at most K (adversarially chosen) errors. Let x denote the resulting corrupted codeword. We have $x = c + e$, for a K-sparse error vector e. The goal of syndrome decoding is to recover e

from the syndrome Hx as efficiently as possible. Our decoder is inspired by the techniques from [9] presented in Section 3, and also the sparse recovery algorithms presented in [11].

For the sake of exposition, we consider only the case of decoding over GF(2) – the adaptation to arbitrary fields is simple and is discussed at the end of Section 4.1.1.

4.1.1 A Deterministic Algorithm

In this section, we present and analyze our deterministic decoder which on input Hx recovers the error vector e in sublinear time. Before we proceed, we fix some notation: For $s \in [D]$, let G_s denote the subgraph of G induced by the function C_s (recall Definition 2), and let W_s be its adjacency matrix. Informally, our decoder receives

$$Hx = \begin{bmatrix} Wx \\ (W \otimes B)x \end{bmatrix}$$

as input, and works as follows:

1. Estimate the size of $\mathsf{supp}(e)$. This can be done by computing $\|W_s \cdot x\|_0$ for all seeds $s \in [D]$ and taking the maximum;
2. Using information from $W_s \cdot x$ and $(W_s \otimes B)x$ for the good seed fixed in Step 2, recover a K-sparse vector y which approximates the error pattern e following the ideas from Section 3;
3. If needed, repeat these three steps with $x - y$ in place of x.

A detailed description of the deterministic decoder can be found in Algorithm 1. We will proceed to show the procedure detailed in Algorithm 1 returns the correct error pattern e, provided ϵ is a small enough constant.

Algorithm 1 Deterministic decoder

procedure ESTIMATE(Wx) ▷ Estimates $|\mathsf{supp}(e)|$ and outputs best seed
  for $s \in [D]$ **do**
    Compute $L_s = \|W_s \cdot x\|_0$
  Output $(\max_s L_s, \arg\max_s L_s)$
procedure APPROXIMATE($W_s \cdot x, (W_s \otimes B)x$) ▷ Computes a good approximation of $\mathsf{supp}(e)$
  Set $y = \mathbf{0}$
  for $q = 0, 1, \ldots, M-1$ **do**
    if $(W_s \cdot x)_q \neq 0$ **then**
      Let $u \in [N]$ be the integer with binary expansion

$$(W_s \otimes B)_{q \log N} \cdot x, (W_s \otimes B)_{q \log N + 1} \cdot x, \ldots, (W_s \otimes B)_{q \log N + \log N - 1} \cdot x,$$

      Set $y_u = 1$.
  Output y
procedure DECODE(Hx) ▷ The main decoding procedure
  Set $(L, s) =$ ESTIMATE(Wx)
  if $L = 0$ **then** ▷ If no errors found, stop and return the zero vector
    Output $y = \mathbf{0}$
  else
    Set $y =$ APPROXIMATE($W_s \cdot x, (W_s \otimes B)x$)
    Set $z =$ DECODE($H(x - y)$)
    Output $y + z$

We begin by showing that procedure ESTIMATE in Algorithm 1 returns a good approximation of the size of $\mathsf{supp}(e)$ along with a good seed.

Lemma 8. *Procedure* ESTIMATE(Wx) *in Algorithm 1 returns a seed $s \in [D]$ satisfying*

$$(1-2\epsilon)|\mathsf{supp}(e)| \leq |\Gamma_s(\mathsf{supp}(e))| \leq |\mathsf{supp}(e)|,$$

where the parameter ϵ comes from the underlying expander graph.

Proof. Recall that we defined $L_s = \|W_s \cdot x\|_0$ and $L = \max_{s\in[D]} \|W_s \cdot x\|_0$. First, observe that $W_s \cdot x = W_s \cdot e$ for all s. Then, the upper bound $L \leq |\mathsf{supp}(e)|$ follows from the fact that

$$\|W_s \cdot e\|_0 \leq |\mathsf{supp}(e)|$$

for all seeds s and all vectors e, since each column of W_s has Hamming weight 1 by the fact that the underlying graph G is layered.

It remains to lower bound L. Since e is K-sparse, we know that

$$|\Gamma(\mathsf{supp}(e))| \geq (1-\epsilon)D|\mathsf{supp}(e)|.$$

By an averaging argument, it follows there is at least one seed $s \in [D]$ such that

$$|\Gamma_s(\mathsf{supp}(e))| \geq (1-\epsilon)|\mathsf{supp}(e)|.$$

As a result, by Lemma 4 the number of vertices in $\Gamma_s(\mathsf{supp}(e))$ adjacent to a single vertex in $\mathsf{supp}(e)$ is at least

$$(1-2\epsilon)|\mathsf{supp}(e)|,$$

and thus $|\Gamma_s(\mathsf{supp}(e))| \geq \|W_s \cdot x\|_0 \geq (1-2\epsilon)|\mathsf{supp}(e)|$. □

We now show that, provided a set $\mathcal{X} \subseteq \mathcal{L} = [N]$ has good expansion properties in G_s, then procedure APPROXIMATE in Algorithm 1 returns a good approximation of $\mathcal{X}$.

Lemma 9. *Fix a vector $x \in \{0,1\}^N$ and denote its support by $\mathcal{X}$. Suppose that*

$$|\Gamma_s(\mathcal{X})| \geq (1-2\epsilon)|\mathcal{X}| \tag{2}$$

for a given seed $s \in [D]$. Then, procedure APPROXIMATE($W_s \cdot x, (W_s \otimes B)x$) *returns an $|\mathcal{X}|$-sparse vector $y \in \{0,1\}^N$ such that*

$$\|x - y\|_0 \leq 5\epsilon\|x\|_0.$$

Proof. First, it is immediate that the procedure returns an $|\mathcal{X}|$-sparse vector y. This is because $|\Gamma_s(\mathcal{X})| \leq |\mathcal{X}|$ as G_s is 1-left regular, and the procedure adds at most one position to y per element of $\Gamma_s(\mathcal{X})$.

In order to show the remainder of the lemma statement, observe that we can bound $\|x - y\|_0$ as

$$\|x - y\|_0 \leq A + B,$$

where A is the number of elements of $\mathcal{X}$ that the procedure does not add to y, and B is the number of elements outside $\mathcal{X}$ that the procedure adds to y.

First, we bound A. Let $\Gamma'_s(\mathcal{X})$ denote the set of neighbors of $\mathcal{X}$ in G_s that are adjacent to a single element of $\mathcal{X}$. Then, the lower bound in (2) and Lemma 4 ensure that

$$|\Gamma'_s(\mathcal{X})| \geq (1-4\epsilon)|\mathcal{X}|.$$

Note that, for each right vertex $q \in \Gamma'_s(\mathcal{X})$, the bits

$$(W \otimes B)_{q\log N} \cdot x, (W \otimes B)_{q\log N+1} \cdot x, \ldots, (W \otimes B)_{q\log N+\log N-1} \cdot x$$

are the binary expansion of u for a distinct $u \in \mathcal{X}$, and u is added to $\mathsf{supp}(y)$. As a result, we conclude that

$$A \leq 4\epsilon|\mathcal{X}|.$$

It remains to bound B. Note that the procedure may only potentially add some $u \notin \mathcal{X}$ if the corresponding right vertex q is adjacent to at least three elements of $\mathcal{X}$. This is because right vertices q that are adjacent

to exactly two elements of $\mathcal{X}$ satisfy $(W_s \cdot x)_q = 0$, and so are easily identified by the procedure and skipped. Then, the lower bound in (2) ensures that there are at most

$$\frac{1}{2} \cdot 2\epsilon|\mathcal{X}| = \epsilon|\mathcal{X}|$$

such right vertices. As a result, we conclude that

$$B \leq \epsilon|\mathcal{X}|,$$

and hence

$$\|x - y\|_0 \leq 5\epsilon|\mathcal{X}|. \qquad \square$$

We combine the lemmas above to obtain the desired result.

Corollary 10. *Suppose that $\epsilon < 1/10$. Then, on input Hx for $x = c + e$, $\textsc{Decode}(Hx)$ in Algorithm 1 recovers the error vector e after at most $1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}$ iterations.*

Proof. Under the conditions in the corollary statement, combining Lemmas 8 and 9 guarantee that in the first iteration of DECODE we obtain a K-sparse vector y_1 such that

$$\|e - y_1\|_0 \leq 5\epsilon\|e\|_0 \leq 5\epsilon K.$$

Recursively applying this result shows that after ℓ iterations we have a vector $y = y_1 + y_2 + \cdots + y_\ell$ with sparsity at most $\sum_{i=1}^{\ell}(5\epsilon)^{i-1}K \leq 2K$ satisfying

$$\|e - y\|_0 \leq (5\epsilon)^\ell K.$$

Setting $\ell = 1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}$ ensures that $\|e - y\|_0 < 1$, and hence $e = y$. $\square$

To conclude this section, we give a detailed analysis of the runtime of the deterministic syndrome decoder. We have the following result.

Theorem 11. *On input Hx for $x = c+e$ with $c \in \mathcal{C}$ and e a K-sparse error vector, procedure $\textsc{Decode}(Hx)$ in Algorithm 1 returns e in time*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}(K + M)(D + \log N)\right).$$

In particular, if ϵ is constant, $M = O(K/\epsilon)$, and $D = O(\log N/\epsilon)$, procedure DECODE takes time

$$O(K \log K \cdot \log N).$$

Proof. We begin by noting that, for a K-sparse vector y and seed $s \in [D]$, we can compute $W_s \cdot y$ and $(W_s \otimes B)y$ in time $O(K)$ and $O(K \log N)$, respectively, with query access to the function table of C and the lookup table of binary expansions. We now look at the costs incurred by the different procedures. We will consider an arbitrary iteration of the algorithm. In this case, the input vector is of the form $x + y$, where x is the corrupted codeword and y is a $2K$-sparse vector.

- The procedure ESTIMATE in Algorithm 1 requires computing D products of the form $W_s(x+y)$, which in total take time $O(D(K + M))$, along with computing the 0-norm of all resulting vectors. Since $W_s(x+y)$ has length M, doing this for all seeds takes time $O(DM)$. In total, the ESTIMATE procedure takes time $O(D(K + M))$;
- The procedure APPROXIMATE in Algorithm 1 requires the computation of $W_s(x+y)$ and $(W_s \otimes B)(x+y)$ for a fixed seed s, which take time $O(K + M)$ and $O((K + M)\log N)$, respectively. The remaining steps can be implemented in time $O(M \log N)$ for a total time of $O((K + M)\log N)$;

Note that the time required to compute the sum of sparse vectors in Line 19 is absorbed into the $O((K + M)(D + \log N))$ complexity of previous procedures. The desired statements now follow by noting that there are at most $1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}$ iterations. □

Remark 12. *In the proof of Theorem 11, we assume that reading an integer from memory (e.g., from the support of a sparse vector y or the function table of C) takes time $O(1)$. If instead we assume that reading an L-bit integer from memory takes time $O(L)$, then we obtain runtime*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}(K \log M + M)(D + \log N)\right).$$

instead.

We conclude this section by briefly describing how to adapt Algorithm 1 to perform sparse recovery over any field. There are several ways to do this. One possibility is to replace the check in Line 8 of Algorithm 1 by the following: $(W_s \cdot x)_q \neq 0$ *and* all non-zero entries of

$$(W_s \otimes B)_{q \log N} \cdot x, \ldots, (W_s \otimes B)_{q \log N + \log N - 1} \cdot x$$

equal $(W_s \cdot x)_q$. As a result, right vertices in $\Gamma_s(\mathsf{supp}(e))$ with exactly two neighbors in $\mathsf{supp}(e)$ are skipped by the algorithm. Observe that this additional condition is trivially satisfied over GF(2) if $(W_s \cdot x)_q \neq 0$. Then, in Line 17 one should set $y_u = (W_s \cdot x)_q$ instead.

We remark that the condition in Line 8 could be simplified in general to only checking whether $(W_s \cdot x)_q \neq 0$, at the expense of obtaining worse constants in the lemmas from this section. In particular, we would have to make ϵ smaller. Since we care about the practicality of our algorithms, we made an effort to have ϵ be as large as possible.

4.1.2 A Randomized Algorithm

In this section, we analyze a randomized version of Algorithm 1 that is considerably faster. The main idea behind this version is that in procedure ESTIMATE we can obtain a good estimate of $|\mathsf{supp}(e)|$ with high probability by relaxing ϵ slightly and sampling $\|W_s \cdot x\|_0$ for a few i.i.d. random seeds only.

In Algorithm 2, we present the randomized decoder, which uses an extra slackness parameter δ. We will show that the procedure detailed in Algorithm 2 returns the correct error pattern e with probability at least $1 - \eta$, provided ϵ is a small enough constant depending on δ.

We begin by showing that procedure ESTIMATE in Algorithm 2 returns a good approximation of the size of $\mathsf{supp}(e)$ with high probability.

Lemma 13. *Procedure* ESTIMATE(r, Wx) *in Algorithm 2 returns a seed $s \in [D]$ satisfying*

$$(1 - 2(1 + \delta)\epsilon)|\mathsf{supp}(e)| \leq |\Gamma_s(\mathsf{supp}(e))| \leq |\mathsf{supp}(e)| \tag{3}$$

with probability at least $1 - (1 + \delta)^{-r}$.

Proof. Similarly to the proof of Lemma 8, the desired result will follow if we show that with probability at least $1 - (1 + \delta)^{-r}$ over the choice of the seeds $s_1, \ldots, s_r$ it holds that

$$L_s \geq (1 - 2(1 + \delta)\epsilon)|\mathsf{supp}(e)| \tag{4}$$

for some seed $s \in \{s_1, \ldots, s_r\}$. We note that (4) holds if $|\Gamma_s(\mathsf{supp}(e))| \geq (1 - (1 + \delta)\epsilon)|\mathsf{supp}(e)|$. The probability that a random seed fails to satisfy this condition is at most $\frac{1}{1+\delta}$ by Lemma 3. Therefore, the probability that none of the seeds $s_1, \ldots, s_r$ satisfy the condition is at most $(1 + \delta)^{-r}$, as desired. □

After we obtain a good seed via the ESTIMATE procedure, invoking Lemma 9 with $\epsilon(1 + \delta)$ in place of ϵ ensures that we can obtain a good sparse approximation of $\mathsf{supp}(e)$ with high probability. We thus obtain the following corollary.

Algorithm 2 Randomized decoder

procedure ESTIMATE(r, Wx) ▷ Estimates $|\mathsf{supp}(e)|$
  Sample r i.i.d. random seeds $s_1, \ldots, s_r$ from $[D]$
  For each s_i, compute $L_i = \|W_{s_i} \cdot x\|_0$
  Output $(\max_{i \in \{1,\ldots,r\}} L_i, \arg\max_{i \in \{1,\ldots,r\}} L_i)$
procedure APPROXIMATE($W_s \cdot x, (W_s \otimes B)x$) ▷ Computes a good approximation of $\mathsf{supp}(e)$
  Set $y = \mathbf{0}$
  for $q = 0, 1, \ldots, M-1$ **do**
    if $(W_s \cdot x)_q \neq 0$ **then**
      Let $u \in [N]$ be the integer with binary expansion

$$(W_s \otimes B)_{q \log N} \cdot x, (W_s \otimes B)_{q \log N + 1} \cdot x, \ldots, (W_s \otimes B)_{q \log N + \log N - 1} \cdot x,$$

      Set $y_u = 1$
  Output y
procedure DECODE($Hx, \eta, \delta, \epsilon$) ▷ The main decoding procedure
  Set $r = 1 + \frac{\log(1/\eta) + \log\log K - \log\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}{\log(1+\delta)}$
  Set $(L, s) = \text{ESTIMATE}(r, Wx)$
  if $L = 0$ **then**
    Output $y = \mathbf{0}$
  else
    Set $y = \text{APPROXIMATE}(W_s \cdot x, (W_s \otimes B)x)$
    Set $z = \text{DECODE}(H(x - y), \eta, \delta, \epsilon)$
    Output $y + z$

Corollary 14. *Suppose that $\epsilon(1+\delta) < 1/10$. Then, on input Hx for $x = c + e$, procedure* DECODE *in Algorithm 2 returns the error vector e with probability at least $1 - \eta$ in at most $1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}$ iterations.*

Proof. The statement follows by repeating the proof of Corollary 10 but replacing Lemma 8 with Lemma 13 and by invoking Lemma 9 with $\epsilon(1+\delta)$ in place of ϵ. Since each iteration succeeds with probability at least $1 - (1+\delta)^{-r}$ and there are at most $1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}$, a union bound guarantees that decoding fails with probability at most

$$\left(1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}\right) \cdot (1+\delta)^{-r} \leq \eta$$

by the choice of r in Algorithm 2. □

We conclude this section by analyzing the runtime of the randomized decoder.

Theorem 15. *On input Hx for $x = c + e$ with $c \in \mathcal{C}$ and e a K-sparse error vector, procedure* DECODE *Algorithm 2 returns e with probability at least $1 - \eta$ in time*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}(K+M)(r + \log N)\right),$$

where

$$r = 1 + \frac{\log(1/\eta) + \log\log K - \log\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}{\log(1+\delta)}.$$

Proof. The proof is analogous to that of Theorem 11, except that in the procedure ESTIMATE in Algorithm 2 we only test r seeds. This means procedure ESTIMATE now takes time $O(r(K+M))$. □

We note that the runtime of the randomized decoder in Theorem 15 is independent of the degree D of the expander. This has two advantages: First, it means the hidden constants in the runtime are considerably smaller than in the deterministic case from Theorem 11, even assuming we use a near-optimal expander with degree $D = O(\frac{\log N}{\epsilon})$. Second, it means that replacing the near-optimal non-explicit expander graph by an explicit construction with sub-optimal parameters will affect the runtime of the randomized decoder only marginally. Furthermore, the failure probability η only affects lower order terms of the runtime complexity. Therefore, we can (for example) set η to be any arbitrarily small constant with only negligible effect in the runtime for large block lengths. Finally, we observe that computing the 0-norm of vectors can be sped up with a randomized algorithm. One can simply sample several small subsets of positions and estimate the true 0-norm with small error and high probability by averaging the 0-norm over all subsets. As mentioned before, we will consider instantiations of our code with an explicit expander in Section 4.3.

Remark 16. *As in the proof of Theorem 11, we assume in this section that reading an integer from memory takes time $O(1)$. If instead we assume that reading an L-bit integer from memory takes time $O(L)$, then we obtain runtime*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}(r\cdot(K(\log M+\log D)+M)+(K+M)\log N)\right)$$

instead. The preceding arguments still stand, as even for explicit expanders it holds that $\log M$ and $\log D$ are negligible compared to $\log N$.

4.2 Full Decoding

In this section, we study the decoding complexity of our code in the setting where we only have access to the corrupted codeword $x = c + e$. This mean that if we want to perform syndrome decoding, we must compute the parts of the syndrome that we want to use from x.

Recall that we have access to the function table of C, as well as a lookup table of $\log N$-bit binary expansions. As a result, we can compute products of the form $W_s \cdot x$ and $(W_s \otimes B)x$ in time $O(N)$ and $O(N \log N)$, respectively. This is because all columns of W_s (resp. $W_s \otimes B$) have 1 nonzero entry (resp. at most $\log N$ nonzero entries), and the nonzero entry (resp. entries) of the j-th column are completely determined by $C_s(j) = C(s, j)$ and the j-th column of the lookup table of binary expansions.

We now analyze the runtimes of both the deterministic and randomized decoders from Algorithms 1 and 2 in this alternative setting. We have the following results.

Theorem 17. *On input $x = c + e$ with $c \in \mathcal{C}$ and e a K-sparse error vector, procedure* Decode *from Algorithm 1 returns e in time*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}(N+K+M)(D+\log N)\right).$$

In particular, if ϵ is constant, $M = O(K/\epsilon)$, and $D = O(\log N/\epsilon)$, procedure Decode *takes time*

$$O(K \log K \cdot N \log N).$$

Proof. The proof is analogous to that of Theorem 11, except we now must take into account the time taken to compute products of the form $W_s \cdot x$ and $(W_s \otimes B)x$:

- The procedure Estimate in Algorithm 1 requires computing D products of the form $W_s(x+y)$, which in total take time $O(D(N+K+M))$, along with computing the 0-norm of all resulting vectors. Since $W_s(x+y)$ has length M, doing this for all seeds takes time $O(DM)$. In total, the Estimate procedure takes time $O(D(N+K+M))$;
- The procedure Approximate in Algorithm 1 requires the computation of $W_s(x+y)$ and $(W_s \otimes B)(x+y)$ for a fixed seed s, which take time $O(N+K+M)$ and $O((N+K+M)\log N)$, respectively. The remaining steps can be implemented in time $O(M \log N)$ for a total time of $O((N+K+M)\log N)$.

The desired statements now follow by noting that there are at most $1 + \frac{\log K}{\log\left(\frac{1}{5\epsilon}\right)}$ iterations. □

Theorem 18. *On input $x = c + e$ with $c \in \mathcal{C}$ and e a K-sparse error vector, procedure* Decode *from Algorithm 2 returns e with probability at least $1 - \eta$ in expected time*

$$O\left(\frac{\log K}{\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}(N + K + M)(r + \log N)\right),$$

where

$$r = 1 + \frac{\log(1/\eta) + \log\log K - \log\log\left(\frac{1}{5\epsilon(1+\delta)}\right)}{\log(1+\delta)}.$$

Proof. The proof is analogous to that of Theorem 17, except that in the procedure Estimate in Algorithm 2 we only need to test r seeds. This means procedure Estimate now takes time $O(q(N + K + M))$. □

There are important properties that are not explicit in the proof of Theorem 18. Observe that only one computation takes $O(N \log N)$ per iteration of the Algorithm 2; Namely, the computation of $(W_s \otimes B)x$ for a fixed seed s. Consequently, the hidden constant in the computation time is small. Moreover, as already discussed for the randomized syndrome decoder, the runtime is independent of the degree of the expander, and the effect of the failure probability on the runtime is negligible for large block lengths. This means that the decoder described in Algorithm 2 is also faster in the full decoding setting than the $O(ND)$ expander codes decoder adapted from [5], whose running time depends on the degree of the expander graph, as long as the number of iterations is not too large. This is the case if the number of errors K allowed is a small constant (e.g., $K \leq 5$) and we set ϵ to be not too large. Furthermore, observe that, unlike our decoder, the runtime of the expander codes decoder is affected by a sub-optimal choice of unbalanced bipartite expanders. Finally, if we want a faster decoder for an arbitrary but fixed error threshold K, we can also set ϵ to be small enough so that the maximum number of iterations is sufficiently small for our needs. In this case, the rate of our code becomes smaller since we must make ϵ smaller.

4.3 Instantiation with Explicit Expanders

In this section, we analyze how instantianting our construction with an explicit layered unbalanced expander with sub-optimal parameters affects the properties of our codes.

More precisely, we consider instantiating our code with the GUV expander introduced by Guruswami, Umans, and Vadhan [22], and an explicit highly unbalanced expander constructed by Ta-Shma, Umans, and Zuckerman [23]. For simplicity, in this section we will assume that all parameters not depending on N (such as K and ϵ) are constants.

Fix constants $\alpha, \epsilon, K > 0$. Then, the GUV graph is a (D, K, ϵ)-layered bipartite expander with degree

$$D = O\left(\frac{\log N \cdot \log K}{\epsilon}\right)^{1+1/\alpha},$$

and, for each layer, a right vertex set of size

$$M = D^2 \cdot K^{1+\alpha}.$$

Observe that, although the GUV expander is unbalanced, the size of its right vertex set grows with the degree. Ta-Shma, Umans, and Vadhan [23] provided explicit constructions of highly unbalanced layered expanders. In particular, they give a construction of a (D, K, ϵ)-layered bipartite expander with degree

$$D = 2^{O(\log\log N)^3},$$

and, for each layer, a right vertex set of size

$$M = K^{O(1/\epsilon)}.$$

Syndrome decoding	Deterministic	Randomized
Graphs from [22]	$O(\log N)^{3+3/\alpha}$	$O(\log N)^{3+2/\alpha}$
Graphs from [23]	$2^{O(\log\log N)^3}$	$O(\log N)$

Table 2: Complexity of deterministic and randomized syndrome decoding for different explicit graphs when the number of errors K and expander error ϵ are constants.

Full decoding	Deterministic	Randomized
Graphs from [22]	$O(N \log^{1+1/\alpha} N)$	$O(N \log N)$
Graphs from [23]	$O(N 2^{O(\log\log N)^3})$	$O(N \log N)$

Table 3: Complexity of deterministic and randomized full decoding for different explicit graphs when the number of errors K and expander error ϵ are constants.

Plugging the parameters of both graphs presented in this section into the runtimes in Theorems 11, 15, 17, and 18 and treating K, ϵ, and α as constants immediately yields explicit high-rate codes with syndrome and full decoding complexity displayed in Table 2 (for syndrome decoding), and in Table 3 (for full decoding).

Observe that for both graphs there is a substantial decrease in complexity for randomized decoding versus deterministic decoding for the same setting. This is due to the fact that the decoding complexity of our randomized decoding algorithms is independent of the degree of the underlying expander, which we have already discussed before, and that the degree of explicit constructions is sub-optimal. Using the highly unbalanced explicit graphs from [23], the decoding complexity of our randomized algorithms essentially matches that of the case where we use a random expander with near-optimal parameters (we are ignoring the contribution of K, which we assume to be small).

We conclude by noting that the full decoding complexity of expander codes under the explicit graphs from this section matches the second column of Table 3. In comparison, our randomized algorithm performs better under both graphs.

5 Group Testing

In this section, we show how we can easily obtain a scheme for non-adaptive group testing with few tests and sublinear time recovery. More precisely, we will prove the following:

Theorem 19. *Given N and K, there is an explicit test matrix W of dimensions $T \times N$, where $T = O(K^2 \log^2 N)$, such that it is possible to recover a K-sparse vector x from $W \odot x$ in time $O(K^3 \log^2 N)$.*

We begin by describing the test matrix W. Let W' be an explicit K-disjunct matrix of dimensions $M \times N$ with $M = O(K^2 \log N)$. Such explicit constructions exist as per Theorem 7. Then, our test matrix W is defined as

$$W = \begin{bmatrix} W' \\ W' \otimes B \end{bmatrix},$$

where B is the $\log N \times N$ bit-test matrix from Section 3. It follows immediately that W has dimensions $T \times N$ with $T = M \log N = O(K^2 \log^2 N)$.

It remains to describe and analyze the recovery algorithm that determines x from

$$W \odot x = \begin{bmatrix} W' \odot x \\ (W' \otimes B) \odot x \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \end{bmatrix},$$

whenever x is K-sparse. At a high-level, the algorithm works as follows:

1. For $q \in [M]$, let s_q be the integer in $[N]$ with binary expansion

$$y^{(2)}_{q \log N}, y^{(2)}_{q \log N + 1}, \ldots, y^{(2)}_{q \log N + \log N - 1}.$$

   Recover the (multi) set $\mathcal{S} = \{s_0, s_1, \ldots, s_{M-1}\}$. The disjunctness property of the underlying matrix W' ensures that $\mathsf{supp}(x) \subseteq \mathcal{S}$;

2. Similarly to the original recovery algorithm for disjunct matrices, run through all $s \in \mathcal{S}$ and check whether W'_s is contained $y^{(1)}$. Again, the fact that W' is disjunct ensures that this holds if and only if $s \in \mathsf{supp}(x)$.

A rigorous description of this recovery procedure can be found in Algorithm 3. We now show that procedure $\textsc{Recover}(W \odot x)$ indeed outputs $\mathsf{supp}(x)$, provided that x is K-sparse. First, we prove that $\mathsf{supp}(x) \subseteq \mathcal{S}$.

Lemma 20. *Suppose that x is K-sparse. Then, if $\mathcal{S} = \textsc{SuperSet}(W \odot x)$, we have $\mathsf{supp}(x) \subseteq \mathcal{S}$.*

Proof. It suffices to show that if $\mathsf{supp}(x) = \{s_0, \ldots, s_{t-1}\}$ for $t \leq K$, then there is $q \in [M]$ such that $W'_{q,s_0} = 1$ and $W'_{q,s_j} = 0$ for all $1 \leq j \leq t-1$. If this is true, then

$$y^{(2)}_{q \log N}, y^{(2)}_{q \log N + 1}, \ldots, y^{(2)}_{q \log N + \log N - 1}$$

would be the binary expansion of s_0. The desired property follows since W' is K-disjunct. In fact, if this was not the case, then $W'_{\cdot s_0}$ would be contained in $\bigvee_{i=1}^{t-1} W'_{\cdot s_i}$, and hence W' would not be K-disjunct. □

The next lemma follows immediately from the fact that W' is K-disjunct.

Lemma 21. *If x is K-sparse and $\mathsf{supp}(x) \subseteq \mathcal{S}$, then $\textsc{Remove}(W \odot x, \mathcal{S})$ returns $\mathsf{supp}(x)$.*

Combining Lemmas 20 and 21 with Algorithm 3 leads to the following result.

Corollary 22. *If x is K-sparse, then $\textsc{Recover}(W \odot x)$ returns $\mathsf{supp}(x)$.*

We conclude this section by analyzing the runtime of procedure $\textsc{Recover}(W \odot x)$. We have the following result.

Theorem 23. *On input a K-sparse vector x, procedure $\textsc{Recover}(W \odot x)$ returns $\mathsf{supp}(x)$ in time $O(K^3 \log^2 N)$.*

Proof. We analyze the runtime of procedures SuperSet and Remove separately:

- In procedure $\textsc{SuperSet}(W \odot x)$, for each $q \in [M]$ we need $O(\log N)$ time to compute s_q and add it to $\mathcal{S}$. It follows that $\textsc{SuperSet}(W \odot x)$ takes time $O(M \log N) = O(K^2 \log^2 N)$;
- In procedure $\textsc{Remove}(W \odot x, \mathcal{S})$, it takes time $O(|\mathsf{supp}(W'_{\cdot j})|)$ to decide whether $W'_{\cdot j}$ is contained in $W' \odot x = y^{(1)}$. Therefore, in total the procedure takes time $O(|\mathcal{S}| \cdot \max_{j \in \mathcal{S}} |\mathsf{supp}(W'_{\cdot j})|)$. Noting that $|\mathcal{S}| \leq M$ and that $|\mathsf{supp}(W'_{\cdot j})| = O(K \log N)$ for all j implies that the procedure takes time

$$O(M \cdot K \log N) = O(K^3 \log^2 N).$$ □

Acknowledgments

The authors thank Shashanka Ubaru and Thach V. Bui for discussions on the role of the bitmasking technique in sparse recovery. M. Cheraghchi's research was partially supported by the National Science Foundation under Grants No. CCF-2006455 and CCF-2107345. J. Ribeiro's research was partially supported by the NSF grants CCF-1814603 and CCF-2107347, the NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

References

[1] M. Sipser and D. A. Spielman, "Expander codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, Nov 1996.

[2] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology - EUROCRYPT 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540.

Algorithm 3 Recovery algorithm for non-adaptive group testing scheme

procedure SUPERSET($W \odot x$) ▷ Recover superset of $\mathsf{supp}(x)$
  Set $\mathcal{S} = \{\}$
  for $q = 0, 1, \ldots, M-1$ **do**
    Let s_q be the integer with binary expansion

$$y^{(2)}_{q \log N}, y^{(2)}_{q \log N + 1}, \ldots, y^{(2)}_{q \log N + \log N - 1}$$

    Add s_q to $\mathcal{S}$
  Output $\mathcal{S}$
procedure REMOVE($W \odot x, \mathcal{S}$) ▷ Finds all false positives in the superset and removes them
  for $s \in \mathcal{S}$ **do**
    if $W'_{:s}$ is not contained in $y^{(1)}$ **then**
      Remove s from $\mathcal{S}$
  Output $\mathcal{S}$
procedure RECOVER($W \odot x$) ▷ Main recovery procedure
  Set $\mathcal{S} = \text{SuperSet}(W \odot x)$
  Output Remove($W \odot x, \mathcal{S}$)

[3] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Syndrome encoding and decoding of BCH codes in sublinear time," 2006, available at https://www.cs.bu.edu/~reyzin/code/bch-excerpt.pdf.

[4] D. Harvey and J. van der Hoeven, "Polynomial multiplication over finite fields in time $O(n \log n)$," *J. ACM*, vol. 69, no. 2, mar 2022.

[5] S. Jafarpour, W. Xu, B. Hassibi, and R. Calderbank, "Efficient and robust compressed sensing using optimized expander graphs," *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 4299–4308, Sep. 2009.

[6] G. Cormode and S. Muthukrishnan, "Combinatorial algorithms for compressed sensing," in *Proceedings of the 13th Colloquium on Structural Information and Communication Complexity (SIROCCO 2006)*. Springer, 2006, pp. 280–294.

[7] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "Algorithmic linear dimension reduction in the ℓ_1 norm for sparse vectors," *arXiv preprint cs/0608079*, 2006.

[8] ——, "One sketch for all: Fast algorithms for compressed sensing," in *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing (STOC 2007)*. ACM, 2007, pp. 237–246.

[9] R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. J. Strauss, "Combining geometry and combinatorics: A unified approach to sparse signal recovery," in *46th Annual Allerton Conference on Communication, Control, and Computing*, Sept 2008, pp. 798–805.

[10] A. C. Gilbert, Y. Li, E. Porat, and M. J. Strauss, "For-all sparse recovery in near-optimal time," *ACM Trans. Algorithms*, vol. 13, no. 3, pp. 32:1–32:26, Mar. 2017.

[11] M. Cheraghchi and P. Indyk, "Nearly optimal deterministic algorithm for sparse Walsh-Hadamard transform," *ACM Trans. Algorithms*, vol. 13, no. 3, pp. 34:1–34:36, Mar. 2017.

[12] M. Cheraghchi, "Noise-resilient group testing: Limitations and constructions," *Discrete Applied Mathematics*, vol. 161, no. 1, pp. 81 – 95, 2013.

[13] P. Indyk, H. Q. Ngo, and A. Rudra, "Efficiently decodable non-adaptive group testing," in *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*. SIAM, 2010, pp. 1126–1142.

[14] H. Q. Ngo, E. Porat, and A. Rudra, "Efficiently decodable error-correcting list disjunct matrices and applications," in *Proceedings of the International Colloqium on Automata, Languages, and Programming (ICALP 2011)*. Springer, 2011, pp. 557–568.

[15] M. Cheraghchi and J. Ribeiro, "Simple codes and sparse recovery with fast decoding," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 156–160.

[16] M. Cheraghchi and V. Nakos, "Combinatorial group testing and sparse recovery schemes with near-optimal decoding time," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, 2020, pp. 1203–1213.

[17] K. Lee, R. Pedarsani, and K. Ramchandran, "SAFFRON: A fast, efficient, and robust framework for group testing based on sparse-graph codes," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2873–2877.

[18] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson, "Randomness conductors and constant-degree lossless expanders," in *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (STOC 2002)*. ACM, 2002, pp. 659–668.

[19] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[20] E. Porat and A. Rothschild, "Explicit non-adaptive combinatorial group testing schemes," in *International Colloquium on Automata, Languages, and Programming (ICALP 2008)*. Springer, 2008, pp. 748–759.

[21] A. G. D'yachkov and V. V. Rykov, "Bounds on the length of disjunctive codes," *Problemy Peredachi Informatsii*, vol. 18, no. 3, pp. 7–13, 1982.

[22] V. Guruswami, C. Umans, and S. Vadhan, "Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes," *J. ACM*, vol. 56, no. 4, pp. 20:1–20:34, Jul. 2009.

[23] A. Ta-Shma, C. Umans, and D. Zuckerman, "Lossless condensers, unbalanced expanders, and extractors," *Combinatorica*, vol. 27, no. 2, pp. 213–240, Mar 2007.