Electronic Colloquium on Computational Complexity, Report No. 80 (2003)

Better Extractors for Better Codes?

Venkatesan Guruswami

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195-2350.
Email: venkat@cs.washington.edu.

Abstract

We present an explicit construction of codes that can be list decoded from a fraction $(1-\varepsilon)$ of errors in sub-exponential time and which have rate $\varepsilon/\log^{O(1)}(1/\varepsilon)$. This comes close to the optimal rate of $\Omega(\varepsilon)$, and is the first sub-exponential complexity construction to beat the rate of $O(\varepsilon^2)$ achieved by Reed-Solomon or algebraic-geometric codes. Beating this quadratic rate with *polynomial* decoding complexity remains one of the central open questions in the area of list decoding, and this work lends some hope to the eventual resolution of this question.

Our construction is based on recent extractor constructions with very good seed length [16]. While the "standard" way of viewing extractors as codes (as in [15]) cannot beat the $O(\varepsilon^2)$ rate barrier due to the $2\log(1/\varepsilon)$ lower bound on seed length for extractors, we use such extractor codes as a component in certain expander-based construction schemes to get our result. The $O(\varepsilon^2)$ rate barrier also arises if one argues about list decoding using the minimum distance (via the so-called Johnson bound) — so this also gives the first explicit construction that "beats the Johnson bound" for list decoding from errors.

The conceptual message from our work is that good strong extractors for low min-entropies will lead to *near-optimal* list decodable codes. Given all the progress that has been made on extractors, we view this as an optimistic avenue to look for better list decodable codes, both by looking for better extractors, as well as by importing non-trivial techniques from the extractor world in reasoning about and constructing codes.

1 Introduction

An area of algorithmic coding theory that has received much attention lately is that of *List Decoding*. List decoding deals with the situation when the amount of noise is too large to be able to identify the original transmitted codeword unambiguously. For example, when more than 1/2 of the symbols could be in error, unique decoding of the correct codeword is impossible. Under list decoding the goal is to output a small number of candidate codewords one of which is the correct one. We refer the interested reader to [12, 3] for more detailed background and history on list decoding.

Informally, a code has good list decoding properties if every Hamming ball of "a large" radius contains very "few" (say at most L) codewords of the code. This requirement only addresses the combinatorial aspect of list decoding, and in addition one would also like the code to be explicitly specified (say, constructible in polynomial time), efficiently encodable, and also have a decoding algorithm to efficiently find and output the list of at most L codewords that are close to a noisy received word. Let us quickly recap that an error-correcting code over alphabet Σ is a map $C : \Sigma^{n_1} \to \Sigma^N$ where n_1 is said to be the dimension of C, N is the block length of C, and the ratio

ISSN 1433-8092

n_1/N, which measures the amount of redundancy introduced by the code, is called the *rate* of C. Such a code C is said to be (p, L)-list decodable if for every $r \in \Sigma^N$, $|\{x \mid \Delta(r, C(x)) \leq pN\}| \leq L$, where $\Delta(y, z)$ denotes the Hamming distance between strings y and z; in other words there are at most L codewords which differ from r in at most a fraction p of symbols. The fundamental question in the subject of list decoding is then to construct (p, L)-list decodable codes for large p, small L and which have good (large) rate. These being conflicting combinatorial goals, there are natural trade-offs to how large the rate can be, and ideally we would like explicit codes with near-optimal rate, together with efficient encoding and list decoding algorithms.

For sake of simplicity, we focus on the "high noise" situation in this paper where the error fraction $p = (1 - \varepsilon)$ for $\varepsilon > 0$ treated as a very small constant. This turns out to be the cleanest setting for an initial study of the asymptotics and also represents the regime where list decoding is most beneficial. It is known that $(1 - \varepsilon, O(1/\varepsilon))$-list decodable codes of rate $\Omega(\varepsilon)$ *exist* (over an alphabet size of, say $O(1/\varepsilon^2)$), and that $\Omega(\varepsilon)$ is the best rate possible for such codes. The best explicit construction[1] achieves a rate of ε^2 — this is achieved by Reed-Solomon and algebraic-geometric codes [7] and certain expander-based constructions (that still use Reed-Solomon codes as a building block) [5]. Despite much progress in the general area of list decoding, the $O(\varepsilon^2)$ bound has been a "barrier" for the rate for codes list-decodable up to a fraction $(1 - \varepsilon)$ of errors. One explanation for this perceived barrier is that $\Omega(\varepsilon^2)$ is the best rate one can achieve if one constructs codes with good relative distance and then uses a combinatorial result called the "Johnson bound" to argue about the $(1-\varepsilon, L)$-list decodability properties of the code. (The reason is that one needs a relative distance of $(1-O(\varepsilon^2))$ for the Johnson bound argument to work, and this in turn restricts the rate to $O(\varepsilon^2)$ (by the Singleton bound). Therefore, any improvement of the rate beyond $\Omega(\varepsilon^2)$ must bypass the Johnson bound paradigm, which is the primarily employed approach including in the Reed-Solomon decoding result [7].)

The question of improving the rate beyond the $O(\varepsilon^2)$ barrier is one of the central open questions in the subject of list decoding. In [5], the authors present a randomized Monte Carlo code construction with rate $\Omega(\varepsilon)$ together with sub-exponential (of the form $2^{O(n^{1/p})}$) time algorithms to list decode them up to a fraction $(1-\varepsilon)$ of errors. However, the construction is not explicit, and there is also no efficient way to *certify* that the code, once constructed randomly, has the necessary list decoding properties.

In this paper, we present the *first explicit construction* of codes of rate better than $\Omega(\varepsilon^2)$ with non-trivial list decoding properties up to radius $(1-\varepsilon)$. In fact, the rate we achieve is $\varepsilon/\log^{O(1)}(1/\varepsilon)$ and thus close to the optimal rate of $\Omega(\varepsilon)$. The shortcoming of our result is that the list decoding algorithm is randomized and runs in sub-exponential time (of form $2^{O(N^{1/p})}$ where N is the block length) and could return sub-exponential sized lists as output. But we point out that prior to this work, even the combinatorial problem of explicit constructions with rate better than $O(\varepsilon^2)$ and sub-exponential list size for $(1 - \varepsilon)$ errors (with no requirement on decoding time) was open. Needless to say, a polynomial list decoding complexity is more desirable, but nevertheless our construction is the first to beat the naive exponential bounds and demonstrates that something non-trivial is possible in this setting, thereby lending hope to eventually beating the $\Omega(\varepsilon^2)$ rate barrier, and perhaps even achieving the optimal $\Omega(\varepsilon)$ rate, with polynomial decoding complexity.

Extractors and codes

Our construction is based on a connection between list-decodable codes and *extractors*. Extractors are well-studied objects in the pseudorandomness literature. An extractor takes as input a sample

[1] For purposes of this paper, by an "explicit" construction we mean a deterministic polynomial time construction.

drawn from a weak-random source, and using a small seed of truly random bits as catalyst, "extracts" almost pure randomness from it. Since [15] first pointed this out, it is now well-known that (strong) extractors give list decodable codes — in fact, (strong) 1-bit extractors are equivalent to binary list-decodable codes with related parameters. To obtain a code from an extractor, one views the input from the weak-random source as the message, and its encoding consists of the extractor's output as the seed varies over all possible values. The block length (and hence rate) of the code depends on the length of the seed. The error of an extractor is the deviation of its output from the uniform distribution, and roughly speaking an extractor that outputs several bits with error ε yields a code list-decodable up to radius $(1-\varepsilon)$. Now, a lower bound of $2\log(1/\varepsilon) - O(1)$ on the seed length as a function of the extractor's error ε [9] translates into an $O(\varepsilon^2)$ upper bound on the rate of the extractor code.[2] This suggests that the above connection between extractors and codes cannot be used to meet out goal of constructing codes with near-optimal rate, or even with rate better than $\Omega(\varepsilon^2)$.

Extractor codes, however, are more than list-decodable. They have the following useful property as observed by [15]: given a set S_i of possibilities for the i'th symbol for every i, the number of codewords whose i'th symbol lies in S_i for εN more coordinates than the expected number of such coordinates for a random string, is small. When the S_i's have exactly one element, this corresponds to list decoding, but otherwise it is more general and was called "list-recovering" in [6]. Our basic idea is to use extractors with large error, say $\gamma = 1/4$, so that the $O(\gamma^2)$ upper bound on the rate of the associated codes is a constant independent of ε. But now one cannot argue that these codes are $(1-\varepsilon, L)$-list decodable. However, these codes have a very useful list-recovering property that can be used in certain expander-based constructions as in [5] to give codes with rate almost the optimal $\Omega(\varepsilon)$ bound.

Despite this connection between extractors and codes, there are certain concerns (that do not normally arise when one studies extractors for their own sake) that have to be addressed for this way of constructing codes to be useful for us. For example, the list size is related exponentially to the min-entropy of the weak-random source, and thus one is interested in the low, ideally logarithmic or lower, min-entropy regime which is not that interesting for extractors in their pseudorandomness context. Furthermore, in order to get rate that is bounded away from zero, it is crucial that the extractors used have seed length $\log n + O(1)$, i.e. with the constant in front of $\log n$ being 1 (as opposed to the $O(\log n)$ seed length which is good enough for other applications of extractors). Extractors with such seed length were not known till recently (except for some constructions in [14], but these were just obtained by viewing certain list-decodable codes as extractors and are therefore not useful to us). Fortunately, in recent work, Ta-Shma, Zuckerman, and Safra [16] construct such extractors, which we will sometime refer to as TZS extractors. The min-entropy needed for their result is $n^{\Omega(1)}$ and this dictates the sub-exponential list decoding complexity of the codes we construct from them. We recall that obtaining $O(\log n)$ seed length for min-entropies smaller than $n^{\Omega(1)}$ is a qualitatively harder problem (eg. Trevisan's celebrated extractors [17] required $n^{\Omega(1)}$ min-entropy), though it was eventually obtained in subsequent works. In light of this, improving our codes using this connection to extractors will probably require additional key insights — we should point out that even the subsequent improvements to TZS extractors by Shaltiel and Umans [10] (which work for up to polylogarithmically small min-entropy) fall short for our purposes, due to the seed length becoming at least $\alpha \log n$ for $\alpha > 1$.

Finally, we note that explicit specification of the extractor implies a polynomial time encoding procedure for the codes. As for decoding, this does not follow from any aspect of the extractor world, since the extraction property only implies a combinatorial list decoding property, namely

[2] Throughout this paper, all logarithms will be to the base 2.

small list size, and in general it does not imply anything other than a brute-force decoding procedure to find the list of codewords. Nevertheless, for the extractors we use, we are able to turn the *proof* of the extraction property into an *algorithm*, thereby giving a sub-exponential time algorithm to perform the list decoding. Ta-Shma and Zuckerman [15] did a conceptually similar thing with Trevisan's extractor construction [17] and gave a decoding algorithm for it, but since the TZS extractors are very different from Trevisan's, the parallel between our work and that of [15] ends with this high-level structural similarity.

Dispersers and Codes. While not explicitly pointed out in the literature, it is clear that the dispersers, the one-sided analogue of extractors, correspond, in the coding world, to decodability from *erasures*. We look at the quantitative aspects of this connection and point out that constructing optimal dispersers will solve another central open question concerning list decoding: constructing *binary* codes of rate $\Omega(\varepsilon)$ that can be decoded from a fraction $(1-\varepsilon)$ of erasures — currently the best known explicit construction has rate $O(\varepsilon^2)$.

Organization. In the next section, we present our main conceptual result connecting extractors and codes. In Section 3 we use extractors from [16] based on bivariate polynomials to achieve a $2^{O(\sqrt{N})}$ list decoding complexity. We then use the multivariate polynomial based extractors from [16] to improve the complexity to $2^{O(N^\delta)}$ for arbitrary $\delta > 0$ in Section 4. Dispersers and erasure codes are discussed in Section 5.

2 Codes from Extractors: The Main Connection

In this section, we will present our method of constructing good list-decodable codes from extractors. It is already known that extractors can as such be viewed as codes [15]. Our construction is a little less direct in that we uses extractors as a component in a expander-based code construction scheme (such expander-based schemes have already been used a few times, eg. in [1, 5]). We now turn to the basic definitions concerning extractors and expanders.

Definition 1 (Extractors) *A function $E : \{0,1\}^{n_1} \times \{0,1\}^d \to \{0,1\}^m$ is a* strong (k,γ)-extractor[3] *if for every $X \subseteq \{0,1\}^{n_1}$ of size at least 2^k, the distribution of $y \circ E(x,y)$ where x, y are picked uniformly from X and $\{0,1\}^d$ respectively, is γ-close to the uniform distribution on $\{0,1\}^{d+m}$.*

Viewing an Extractor as a Code. We now mention a "direct" way of getting codes from extractors first made explicit by Ta-Shma and Zuckerman [15]. Given an extractor $E : \{0,1\}^{n_1} \times \{0,1\}^d \to \{0,1\}^m$, define a code $C_E : \{0,1\}^{n_1} \to [Q]^{2^d}$ where $Q = 2^m$ as follows: $C_E(x) = \langle E(x,y) \rangle_{y \in \{0,1\}^d}$. Thus C_E is a code over alphabet $[Q]$ of dimension n_1/m (as n_1 bits can be viewed as n_1/m symbols over $[Q]$) and block length 2^d. This view of extractors as codes motivates an algorithmic problem concerning "decoding" extractors.

Definition 2 (γ-frequent elements) *For a statistical test $T \subseteq \{0,1\}^{m+d}$, we define an $x \in \{0,1\}^{n_1}$ to be γ-frequent for T if $\mathbf{Prob}_y[y \circ E(x,y) \in T] \geq \frac{|T|}{2^{m+d}} + \gamma$ for y picked u.a.r from $\{0,1\}^d$.*

Definition 3 (Decoding of Extractors) *A γ-decoding algorithm for a (strong) extractor $E : \{0,1\}^{n_1} \times \{0,1\}^d \to \{0,1\}^m$ takes as input a "test" subset $T \subseteq \{0,1\}^{m+d}$ and outputs all $x \in \{0,1\}^{n_1}$*

[3] We will use n_1 to denote the length of the string from the weak-random source to avoid confusion with the block length of codes for which n is usually reserved. Also all extractors we will be interested in will be *strong*, so we will often omit the qualifier strong in the sequel.

which are γ-frequent for T. An extractor is said to be γ-decodable in time $T(n_1)$ if there exists a γ-decoding algorithm for it that runs in time $T(n_1)$.

Definition 4 (Expander) *For positive integers n, d and reals $\zeta, \varepsilon > 0$, an $(n, d, \zeta, \varepsilon)$-expander is an $n \times n$ d-regular bipartite graph (X, Y, E) with the property that for every subset $S \subseteq Y$ with $|S| \geq \varepsilon|Y|$, the fraction of nodes in X that have no neighbors in S is at most ζ.*

Combining a code and an expander. We now review the basic operation by which we combine a code with an expander to "improve" its error-resilience (at the expense of increase in alphabet size). This operation is quite natural and simple and dates back to the work of Alon *et al* more than a decade ago [1].

Definition 5 *Given a code C of block length n over alphabet Σ, and a Δ-regular $n \times n$ bipartite graph G, define the code $G(C)$ over alphabet Σ^Δ which has the same number of codewords as C as follows. For each $\mathbf{c} = \langle c_1, c_2, \ldots, c_n \rangle \in C$, define a codeword $G(\mathbf{c}) \in G(C)$ by pushing the symbols of $\mathbf{c}$ along the edges of G and at each node on the right hand side of the bipartition, collecting Δ symbols from its neighbors to form a symbol from Σ^Δ. Formally, $G(c)_i = \langle c_{\Gamma_1(i)}, \ldots, c_{\Gamma_\Delta(i)} \rangle$ where $\Gamma_k(i)$ is the index of the k'th neighbor on the left of the i'th node on the right hand side of G.*

We remark that the above can be viewed as encoding each symbol of $\mathbf{c}$ by the repetition code of block length Δ and then redistributing these symbols via an expander. Clearly, one can (and should) use better codes than the repetition code, and such generalizations of above have been crucial in [2, 6]. For the noise regime we are interested in, keeping things simple as in above construction does not cost us much, and for ease of presentation, we will just use the above expander-based construction in this paper.

To avoid confusion with our use of ε for the agreement from which we list decode, we will use γ to denote the error of extractor. We will think of this error γ as an absolute constant independent of ε and this will be key to avoid the $2\log(1/\gamma)$ lower bound on seed length from ruining our final rate. The reader might want to think of γ as being fixed to, say, $1/4$ (we keep an additional parameter and don't fix it as $1/4$ throughout only because we will have to work with γ very close to 1 in Section 4). The following lemma is just an instance of converting "list-recoverable" codes into list-decodable codes implicit in [5]. For self-containment we present a proof without explicitly referring to the list-recovering terminology. For ease of notation, for an integer p, we will use (p) to stand for $\{0,1\}^p$ throughout the paper.

Lemma 1 *Let $E : (n_1) \times (d) \to (m)$ be a* strong *(k, γ)-extractor. Let G be a $(2^d, \Delta, \zeta, \varepsilon)$-expander. Then, provided $2^m > \Delta/(1 - \zeta - \gamma)$, $G(C_E)$ is a $(1 - \varepsilon, 2^k)$-list decodable code over alphabet $[2^m]^\Delta$ with block length 2^d and "dimension" $\frac{n_1}{m\Delta}$ (and thus rate $\frac{n_1}{2^d m\Delta}$). Furthermore, if the extractor is γ-decodable in time $T(n_1)$, then $G(C_E)$ can be $(1 - \varepsilon, 2^k)$-list decoded in time $O(T(n_1) + 2^d\Delta)$.*

Proof: Denote by $N = 2^d$ the block length of the code $G(C_E)$, and let $\Sigma = [2^m]^\Delta$ be its alphabet. Let $\mathbf{r} = \langle r_1, r_2, \ldots, r_N \rangle \in \Sigma^N$. We wish to prove that there are at most 2^k codewords in $G(C_E)$ that agree with $\mathbf{r}$ on εN or more locations, and that the list of all such codewords can be found in time $O(T(n_1) + N\Delta)$.

Consider the following decoding algorithm. For each $y \in \{1, 2, \ldots, 2^d\}$, let $N_G(y) \subseteq [2^d]$ be the set of neighbors on the right of the y'th node on the left in G. Now each symbol on the right votes for what it thinks the corresponding symbol at each of its neighbors on the left is. Formally, we compute the sets $T_y = \{y \circ (r_i)_k \mid i \in N_G(y) \text{ and } \Gamma_k(i) = y\} \subseteq (d + m)$ (where $(r_i)_k$ is the k'th component of the Δ-tuple r_i and can thus be viewed as an element of (m)) for each $1 \leq y \leq 2^d$

where y is viewed as an element of $\{0,1\}^d$. We then set $T = \bigcup_{y \in (d)} T_y$, and run the γ-decoding algorithm for the extractor E on input the statistical test T. Since E is a strong (k, γ)-extractor, we will output at most 2^k codewords. The claimed runtime also follows immediately.

It remains to prove that the algorithm outputs each message x whose encoding by $G(C_E)$ agrees with $\mathbf{r}$ on locations in some $S \subseteq [N]$ with $S \geq \varepsilon N$. This amounts to proving that such an x is γ-frequent for the test T computed above. Here is the key point in proving this: for such x, by the property of the expander G, we are guaranteed that $y \circ E(x,y) \in T_y$ for at least $(1-\zeta)2^d$ values of y. Hence we have

$$\mathbf{Prob}_y[y \circ E(x,y) \in T] \geq (1-\zeta) \ . \tag{1}$$

Now $|T| \leq 2^d \Delta$ and since $2^m \geq \Delta/(1-\zeta-\gamma)$ by hypothesis, we have $\frac{|T|}{2^{d+m}} \leq (1-\zeta-\gamma)$. Combining this with Equation (1), we conclude that x is γ-frequent for T, as desired. □

To use the above lemma, we use the fact that the necessary expanders of degree $\Delta = O(\frac{1}{\zeta\varepsilon})$ can be explicitly constructed. We state two corollaries which record consequences of the above lemma once we use such an expander. The first one says that very good extractors will do wonders for codes, and it captures the main conceptual message of this work. The second one is more "realistic" and notes down the parameters obtainable from possibly sub-optimal constructions.

Corollary 2 (Better Extractors for Better Codes) *If near-optimal strong extractors with seed length $\log n_1 + O(1)$ and entropy loss $O(1)$ can be explicitly constructed for all min-entropies and error, say, $1/4$, then we can explicitly construct $(1-\varepsilon, O(1/\varepsilon))$-list decodable codes of rate $\Omega(\varepsilon/\log(1/\varepsilon))$. Furthermore, if the extractors can be $1/4$-decoded in time polynomial in n_1, then the resulting codes can be list decoded up to a fraction $(1-\varepsilon)$ of errors in polynomial time.*

Proof: We use Lemma 1 with a $(k, 1/4)$-extractor $E : (n_1) \times (d) \to (m)$ where $2^m \geq a/\varepsilon$, $k = \log(1/\varepsilon) + O(1)$, and $d = \log n_1 + O(1)$, and a $(2^d, b/\varepsilon, 1/2, \varepsilon)$-expander G (an explicit such expander can be constructed using Ramanujan graphs, cf. [1]) — here a, b are absolute constants. If $a > 4b$, then Lemma 1 guarantees that $G(C_E)$ is $(1-\varepsilon, 2^k)$-list decodable, and $2^k = O(1/\varepsilon)$ so the claim about list decodability follows. The rate of the code is $\frac{n_1}{2^d(b/\varepsilon)m}$ which is $\Omega(\varepsilon/m)$ and since $m = O(\log(1/\varepsilon))$, we get $\Omega(\varepsilon/\log(1/\varepsilon))$ rate. □

Corollary 3 *For arbitrary $\varepsilon > 0$, if for a function $k : \mathbb{N} \to \mathbb{N}$ a family of* strong *$(k(n_1), 1/4)$-extractors $E : (n_1) \times (d) \to (m)$ where $2^m = \Theta(1/\varepsilon)$ and $d = \log n_1 + c_\varepsilon$ can be explicitly constructed, then one can explicitly construct a family of codes of rate $\Omega(\frac{\varepsilon}{2^{c_\varepsilon} \log(1/\varepsilon)})$ such that a code of block length N in the family is $(1-\varepsilon, 2^{k(N/2^{c_\varepsilon})})$-list decodable. Furthermore, if the extractor is $1/4$-decodable in time $T(n_1)$, then the resulting code can be list decoded up to a fraction $(1-\varepsilon)$ of errors in time $T(N/2^{c_\varepsilon}) + O(N/\varepsilon)$.*

3 Construction using bivariate polynomials

As mentioned in the introduction, we need extractors whose seed length d is $\log n_1 + O(1)$, with the constant in front of $\log n_1$ being 1, so that the rate of the codes we construct using Lemma 1 will be bounded away from zero. The only currently known extractors with this property are due to Ta-Shma, Zuckerman, and Safra [16] who use Reed-Muller codes (which are based on multivariate polynomials) to construct such extractors. Naturally, we turn to these extractors in our quest for codes via Lemma 1. Specifically, in this section, we will use the bivariate polynomial based extractors from [16] to prove the following.

Theorem 4 *For every constant $\varepsilon > 0$, there is a polynomial time constructible family of codes of rate $\Omega(\varepsilon/\log^{11}(1/\varepsilon))$ such that every code in the family is $(1-\varepsilon, 2^{O(\sqrt{N \log N})})$-list decodable in* randomized $2^{\tilde{O}(\sqrt{N \log N})}$ *time where N is the block length of the code.*[4]

Improving the extractor used for the above result to work for lower min-entropies will improve the runtime and list size. In the next section, we will improve the decoding time and list size to 2^{N^δ} for any desired $\delta > 0$ at the expense of some worsening in the rate, by using the multivariate polynomial based extractors for lower min-entropies from the same paper [16].

Perspective: Since the TZS extractor is itself based on codes, our construction can be cast purely as an operation on Reed-Muller codes, with no reference to extractors. However, not only was the extractors language useful to define such a nice operation, but techniques from the domain of extractors like next-bit predictor and reconstruction paradigm seem crucial for the development of the list decoding algorithm.

3.1 The TZS extractor

We recall the extractor construction from [16]. Recall that the goal is an (k, γ) extractor $E : (n_1) \times (d) \to (m)$ with $d = \log n_1 + c_{m,\gamma}$ and k as small as possible. Define $\rho = \gamma/2m$ — we will set $\gamma = 1/4$ and $m = \log(1/\varepsilon) + O(1)$ where our target is to list decode up to radius $(1-\varepsilon)$, but retain these parameters to maintain the same notation as [16]. Pick parameters h, q that satisfy $\binom{h+1}{2} \log q \geq n_1$, specifically $h = \lceil 3\sqrt{\frac{n_1}{\log n_1}} \rceil$ and q a power of two of size $\Theta(h/\rho^4)$. Let C be a $(1/2 - \rho, O(1/\rho^2))$-list decodable binary linear code of dimension $\ell = \log q$. Such a code of block length $\bar{\ell} = O(\rho^{-2} \log q)$ (and thus rate $\Omega(\rho^2)$) exists and can be found in $q^{O(1/\rho^2)}$ time [4]. (Alternatively, such a code of rate $\Omega(\rho^4)$ can be found in $(\log q)^{O(\rho^{-2})}$ time, but we will use the stronger bound since it will still give an affordable construction time of $n_1^{O(\log(1/\varepsilon))}$ for our choice of parameters.)

The strong extractor, say $E^* : (n_1) \times (d) \to (m)$, is defined as follows. We associate with $x \in (n_1)$ a bivariate polynomial $p_x : \mathbb{F}_q^2 \to \mathbb{F}_q$ of total degree at most $(h-1)$ (the number of such polynomials is at least $q^{\binom{h+1}{2}} \geq 2^{n_1}$ so each x can be associated with a different polynomial). We set $d = 2\log q + \log \bar{\ell}$ and view the seed $y = (a, j)$ as a point $a = (a_1, a_2) \in \mathbb{F}_q^2$ and an index $j \in [\bar{\ell}]$. The extractor output is now defined to consist of the j'th bit of the encoding by C of the evaluations of p_x at m successive points along a "horizontal" line beginning with a. Formally,

$$E^*(x, ((a_1, a_2), j)) = \langle C(p_x(a_1, a_2))_j \circ C(p_x(a_1+1, a_2))_j \circ \cdots \circ C(p_x(a_1+m-1, a_2))_j \rangle \ . \qquad (2)$$

Note that $d = \log(q^2\bar{\ell}) = \log(q^2 \log q) + 2\log(1/\rho) + O(1)$. Now, by our choice of $q = \Theta(\rho^{-4}\sqrt{n_1/\log n_1})$, we have $d \leq \log n_1 + 10\log(1/\rho) + O(1)$, and thus the above has a very good seed length. The main result of [16] is the following which states that is also a good extractor for min-entropy at least about $\sqrt{n_1}$.

Theorem 5 *The function $E^* : (n_1) \times (d) \to (m)$ as defined above gives a strong (k, γ)-extractor with $d = \log n_1 + 10\log(m/\gamma) + O(1)$ provided $k \geq \Omega(m\sqrt{n_1 \log(n_1/\gamma)})$.*

For the choice $\gamma = 1/4$ and $m = \log(1/\varepsilon) + O(1)$ we can apply Corollary 3, this immediately gives codes of rate $\Omega(\varepsilon/\log^{11}(1/\varepsilon))$ which have at most sub-exponentially many codewords in any Hamming ball of (relative) radius $(1-\varepsilon)$. Thus the combinatorial aspect of list decoding is already taken care of!

[4] By a randomized list decoding algorithm we mean a procedure that outputs the *correct* list of codewords with high probability.

3.2 Decoding the TZS Extractor

It remains to give a decoding algorithm to find and output these sub-exponentially many codewords in sub-exponential time, in other words a γ-decoding algorithm for the extractor that runs in $2^{\tilde{O}(\sqrt{n_1})}$ time and finds the set of all γ-frequent elements x for a given T — Theorem 5 guarantees that there will be at most 2^k such x's. The proof that there are not too many such x's follows a *reconstruction paradigm* by demonstrating a reconstruction procedure which outputs each γ-frequent x with good probability using a short advice string as input (the advice kick starts and guides the procedure by helping it make the right choices). If the needed advice string is short, then there cannot be too many γ-frequent x's. This proof method is by nature algorithmic so we really only need to revisit the analysis to ensure that all the steps can be performed efficiently. The dominant component in the runtime ends up being trying out the various advice strings, and this also governs the list size (or the min-entropy for which the construction works, in extractor terminology).

3.2.1 Next-element predictors

The first step is to use the distinguishing subset T to obtain a next-element predictor. This is quite standard, cf. [16, 10]. By Yao's next-bit predictor lemma, there must exist an i, $1 \leq i \leq m$, and a randomized next-bit predictor $A : \{0,1\}^{i-1} \to \{0,1\}$, such that $\mathbf{Prob}[A(E^*(x,y)_1 \cdots E^*(x,y)_{i-1}) = E^*(x,y)_i] \geq \frac{1}{2} + \frac{\gamma}{m}$ where the probability is taken over y picked u.a.r from (d) as well as A's coin tosses. In fact, A works as follows. On input $y \in (d)$ and bits $b_1, b_2, \ldots, b_{i-1}$ which are supposedly the first $(i-1)$ bits of $E^*(x,y)$, it predicts the i'th bit as follows.

1. Pick $(m-i+1)$ bits $b_i, b_{i+1}, \ldots, b_m \in \{0,1\}$ at random.
2. If $y \circ \langle b_1, b_2, \ldots, b_m \rangle \in T$, output b_i, else output $\bar{b}_i$.

Clearly there must exist a fixing of the coin tosses of A to give a *function* $g : \{0,1\}^{i-1} \to \{0,1\}$, such that $\mathbf{Prob}_y[g(E^*(x,y)_1 \cdots E^*(x,y)_{i-1}) = E^*(x,y)_i] \geq \frac{1}{2} + \frac{\gamma}{m}$. Alternatively, by going over all possible 2^{m-i+1} random choices, as well as various choices of i, we conclude the following (we can assume $i = m$, since for smaller i the next-bit predictor can simply ignore the first $(m-i)$ bits owing to the symmetry of the extractor's definition):

Lemma 6 *For every $T \subseteq (d+m)$, we can find a set $\mathcal{G}$ of at most 2^{m+1} Boolean functions on $(m-1)$ bits such that for every x that is γ-frequent for T, there is a function $g \in \mathcal{G}$ such that*

$$\mathbf{Prob}\Big[g\Big(C(p_x(a))_j, C(p_x(a+(1,0)))_j, \ldots, C(p_x(a+(m-1,0)))_j\Big) = C(p_x(a+(m,0)))_j\Big] \geq \frac{1}{2} + \frac{\gamma}{m} ,$$

where the probability is taken over random choices of $a \in \mathbb{F}_q^2$ and $j \in [\bar{l}]$.

Recalling that $\rho = \frac{\gamma}{2m}$, one can use that C is $(1/2 - \rho, O(\rho^{-2}))$-list decodable together with the next-bit predictor to get a next-element predictor for the values of p_x as stated below. This is implicit in [16] and is made explicit in Lemma 5 of [10].

Lemma 7 *For every $T \subseteq (d+m)$, we can find a set $\mathcal{F}$ of at most 2^{m+1} functions $f : \mathbb{F}_q^{m-1} \to \mathbb{F}_q^{O(\rho^{-2})}$ such that for every x that is γ-frequent for T, there is a function $f \in \mathcal{F}$ such that*

$$\mathbf{Prob}_{a \in \mathbb{F}_q^2}\Big[f\Big(p_x(a-(m-1,0))), p_x(a-(m-2,0)), \ldots, p_x(a-(1,0))\Big) \ni p_x(a)\Big] \geq \rho \ . \qquad (3)$$

Note that the number of such functions over $\mathbb{F}_q$ is in general at least $q^{q^{m-1}}$, which is way too many. The above lemma shows that a much smaller number $O(2^m)$ of functions always contains a good next-element predictors. Though its content is implicit in the previous works [16, 10], we had to make explicit the number of candidate next-element predictors that have to be tried, since we care about the complexity of the reconstruction procedure (while in the context of [16, 10] it was sufficient to show that a good reconstruction procedure *exists*).

3.2.2 Reconstruction using the next-element predictor

We can now assume we have at our disposal a good next-element predictor $f : \mathbb{F}_q^{m-1} \to \mathbb{F}_q^{O(\rho^{-2})}$ that satisfies (3) for the γ-frequent element x in question. We should now make sure that we can use this to reconstruct x. The reconstruction procedure R^f with oracle access to f and input an advice string z works as follows:

1. Pick a random line $L \subseteq \mathbb{F}_q^2$ (i.e. pick $\alpha \in \mathbb{F}_q^2$ and $\beta \in \mathbb{F}_q^2 \setminus \{(0,0)\}$ at random and set $L = \{\alpha + t\beta \mid t \in \mathbb{F}_q\}$)[5]

2. Use z to take as advice the value of p_x on the $(m-1)$ predecessor lines $L-(1,0), L-(2,0), \cdots, L-(m-1,0)$, each expressed as a degree $(h-1)$ univariate polynomial in t over $\mathbb{F}_q$ given by its h coefficients in $\mathbb{F}_q$. /* This takes $(m-1)h \log q$ bits of advice */
   Initialize $L' = L$.

3. Repeat the following for h steps to learn the values of p_x on L and its $(h-1)$ "successors" $L + (i,0)$ for $1 \leq i \leq h-1$.

   (a) For each $t \in \mathbb{F}_q$, use the next-element predictor with already known values for $p_x(L'(t) - (j,0))$ for $1 \leq j < m$ to get a list S_t of $O(\rho^{-2})$ elements one of which is hopefully $p_x(L(t))$.

   (b) Find a list of at most $O(\rho^{-3})$ univariate polynomials Q over $\mathbb{F}_q$ of degree at most $(h-1)$ which satisfy $Q(t) \in S_t$ for at least $\rho q/2$ values of t.[6]

   (c) Use at most $3\log(1/\rho) + O(1)$ bits of advice from z to pick the correct polynomial $p_{x|L'}$ from the above list.

   (d) Move onto the next line by setting $L' \leftarrow L' + (1,0)$.

4. From the predicted values for p_x, interpolate to find p_x and thus x. If no such degree $(h-1)$ bivariate polynomial exists, output a failure.

One of the key steps in the above is the list decoding step where all the polynomials Q for which $Q(t) \in S_t$ quite often are found. By a well-known combinatorial result (cf. [11]), for the concerned choice of parameters (specifically $q = \Theta(h/\rho^4)$), the number of such polynomials is $O(\rho^{-3})$ (see [10] for an explicit calculation in this setting). Moreover, it is also known how to find the list of all such univariate polynomials Q in time polynomial in q — this lets us also conclude that R^f can be has an efficient implementation.

[5]For the various steps to predict the value of new points, L must not be parallel to the x axis, i.e. β should not be of the form $(i,0)$. But this happens with probability at most $1/q$ which is $o(1)$, so we implicitly assume this pathological case does not occur.

[6]The proof that the reconstruction procedure works from [16] proceeds by showing that for a random choice of L, owing to the pairwise independence of points on L', the next-element predictor will work correctly for at least a fraction $\rho/2$ of the points in L' with high probability (around $1 - O(1/q)$). Hence this list decoding step will succeed in finding $p_{x|L'}$ as one of its solutions.

Lemma 8 (Implicit in [16]) *Given an f which is a good next-element predictor for x and satisfies (3), there is an advice string z of at most $(m-1)h\log q + 3h\log(1/\rho) + O(h)$ bits for which the above reconstruction procedure R^f on input z outputs x with probability at least $1/2$ over the coin tosses of R (i.e. the choice of the random line L). Furthermore, given oracle access to f, the procedure R can be implemented to run in time polynomial in n_1.*

3.2.3 Obtaining a Decoding algorithm

We now indicate how the above reconstruction procedure can be turned into a list decoding algorithm that outputs *every* γ-frequent x with high probability.

On input the distinguisher $T \subseteq \{0,1\}^{d+m}$:
Repeat the following steps for each of the at most 2^{m+1} next-element predictor functions f.

1. For each possible input advice string z which is $(m-1)h\log q + 3h\log(1/\rho) + O(h)$ bits long, do the following:
   (a) Run the reconstruction procedure R^f $2n_1$ times with independent coin tosses, and let X be the set of x's reconstructed in the various iterations.
   (b) Check whether each $x \in X$ is indeed γ-frequent for T, and if so output x.

Since an x_0 that γ-frequent for T is output by R^f for some choice of f on some input advice z_0 with probability $1/2$, except with probability $1/2^{2n_1}$, this x_0 will be output in one of the $2n_1$ runs of R^f on input z_0. By a union bound, the probability that the list output by the above algorithm is exactly the list of all γ-frequent elements for T is at least $(1 - 2^{-n_1})$.

3.2.4 Complexity analysis of decoding procedure

Since each run of R^f can be completed in $\mathrm{poly}(n_1)$ time, the dominant component in the runtime ends up being the time to go over all possible advice strings. The number of different advice strings is $2^{O(h(m\log q + \log(1/\rho)))}$ (the number of next-element predictors to try is much smaller than this).

Let us recall our setting of parameters, namely $h = \Theta(\sqrt{n_1/\log n_1})$, $m = O(\log(1/\varepsilon))$, $\rho = \frac{\gamma}{2m} = \frac{1}{8m}$, and $q = \Theta(h/\rho^4)$. The overall running time is thus $2^{O(\sqrt{n_1 \log n_1}\log(1/\varepsilon))}$, which is certainly $2^{O(\sqrt{N})}$ where $N = 2^d = \Omega(n_1 \log^{10}(1/\varepsilon))$ (recall the value of the seed length d from Theorem 5) is the block length of the extractor viewed as a code. Therefore the decoding time for the extractor and number of γ-frequent elements to be output are both $2^{O(\sqrt{N\log N})}$.

Now combining this with Theorem 5 and Corollary 3, gives us our main result of this section, namely Theorem 4.

4 Construction using multivariate polynomials

To improve the min-entropies for which their extractors work below the $\sqrt{n_1}$ bound achieved by the above construction, Ta-Shma *et al* [16] also give a construction based on D-variate polynomials which works for min-entropy about $n_1^{1/D}$. We restate their result in this setting below (this is Theorem 3 in their paper):

Theorem 9 *[16] For every n_1, m and D, there is an explicit family of strong (k,γ) extractors $E_{n_1}^{(D)} : (n_1) \times (d) \to (m)$ with $k = \Omega(m^{D-1} n_1^{1/D} \log n_1)$, $d \le \log n + O(D^2 \log m)$, and $\gamma = 1 - \frac{1}{8D}$.*

The proof of the above theorem is deferred in the conference version of [16], but appears in the full version. We sketch the basic proof idea in a manner which will hopefully convince the reader that one can indeed obtain a decoding algorithm from the proof of the extractor property in [16]. The input $x \in (n_1)$ is viewed as a D-variate polynomial p_x of total degree at most h over $\mathbb{F}_q$. The parameters are chosen so that $\binom{h+D}{D} \log q \geq n_1$, so this is possible; specifically, $h = D(n_1/\log h)^{1/D}$ and q is the smallest power of 2 larger than $\Omega(m^{\max\{4,D-1\}}h)$ (the field size q is chosen large enough compared to h so that the error probability of the reconstruction procedure works out to be small). The number of output bits m will be chosen to satisfy $2^m = \Theta(D^2/\varepsilon)$. We can then use this extractor in Lemma 1 with $\gamma = 1 - \frac{1}{8D}$, $\zeta = \frac{1}{16D}$ along with an explicit $(2^d, O(\frac{1}{\zeta\varepsilon}), \zeta, \varepsilon)$-expander, to construct a code of rate $\Omega(\varepsilon/\log(D^2/\varepsilon)^{O(D^2)})$ that is $(1-\varepsilon, 2^k)$-list decodable for k as in Theorem 9.[7]

Now to the extractor definition itself. The random seed $y \in (d)$ consists of three parts: $a \in \mathbb{F}_q^D$, $i \in \{1, 2, \ldots, D-1\} = [D-1]$, $j \in [\bar{l}]$ (where $\bar{\ell}$ is the block length of a binary linear code of dimension $\log q$ which is $(1/2 - \rho, O(\rho^{-2}))$-list decodable as in the previous section; we will set $\rho = \frac{1}{8m}$). The extractor $E^{(D)}$ is defined as $E^{(D)}(x; (a,i,j)) = \langle C(p_x(a + e_i))_j C(p_x(a + 2e_i))_j \cdots C(p_x(a + me_i))_j \rangle$ where e_i denotes the unit vector in m dimensions with 1 in the i'th coordinate and 0's elsewhere.

The proof that this construction is an extractor again follows the reconstruction paradigm which lets us turn it into a γ-decoding algorithm, for $\gamma = 1 - \frac{1}{8D}$. Let a distinguisher $T \subseteq (d+m)$ be given and the goal is to find all x which are γ-frequent for T. Since γ is very close to 1, a simple averaging argument is used to show that for *each* $i \in [D-1]$, if U^i denotes the uniform distribution on $\{(a,i,j) \mid a \in \mathbb{F}_q^D,\ j \in [\bar{\ell}]\}$, then $\mathbf{Prob}[U^i \circ E^{(D)}(x, U^i) \in T] - \frac{|T|}{2^{m+d}} \geq 7/8$. One can then use Yao's next-bit predictor argument to convert T into a predictor T_i for each i. Using arguments as in the bivariate case, for *each* $i \in [D-1]$, we can find a collection $\mathcal{F}_i$ of at most $O(2^m)$ q-ary next-element predictor functions satisfying a property similar to Lemma 7, with $\rho = \frac{7}{16m}$ and the vector $(1,0)$ replaced by e_i.

We now state use the above to prove the following, which is our main technical result.

Theorem 10 *For every integer $D \geq 2$, for every $\varepsilon > 0$, there is a polynomial time constructible family of codes of rate $\Omega(\varepsilon/\log^{O(D^2)}(D^2/\varepsilon))$ such that every code in the family is $(1-\varepsilon, 2^{O(N^{1/D}\log N)})$-list decodable in* randomized $2^{O(N^{1/D}\log N)}$ *time where N is the block length of the code.*

Proof Sketch: As argued above, one can compute, based on the distinguishing set T, a small collection of next-element predictors for every direction e_i, $1 \leq i \leq D-1$. These are then used in a reconstruction procedure to determine all γ-frequent elements for T. The goal is to take as advice the value of p_x on a small collection of points in $\mathbb{F}_q^D$ and then use the next-element predictors to deduce the value of p_x on enough points to be able to interpolate and determine p_x, and thus also x.

Recall that for the bivariate case, we reconstructed p_x by finding its values on a collection of "successive" lines. This approach as such is not suitable for the D-variate case since we will need to make about q^{D-1} line prediction steps, which is too many (the error probability for the procedure R^f adds up for each step, and also we need advice to pick the right polynomial at each Reed-Solomon list decoding step which amounts to requiring a longer advice string than before!). Instead the approach of [16] is to recursively predict the value along "cubes" of larger and larger dimension. We pick a line L at random and the goal is to predict the value on all of the D-dimensional affine subspace $L + \mathrm{span}_h\{e_1, e_2, \ldots, e_{D-1}\}$ where span_h refers to the fact that the

[7] This is why we did not fix $\gamma = 1/4$ throughout the paper.

coefficients are all bounded by h (predicting the value of p_x on such a subcube suffices to identify it, and thus x, uniquely).[8]

Once a line L is picked, for $s = 0, 1, 2, \ldots, D-1$, the values of p_x on the $s+1$-dimensional affine subspace $U = L + \mathrm{span}_h\{e_1, \ldots, e_s\}$ are computed recursively as follows. If $s = 0$, $U = L$, and we determine $p_{x|L}$ by just consulting the advice string (this takes $(h+1)\log q$ bits of advice since $p_{|L}$ is a univariate degree h polynomial). If $s \geq 1$, we recursively compute values on $U_r = L + re_s + \mathrm{span}_h\{e_1, e_2, \ldots, e_{s-1}\}$ for $0 \leq r \leq m-2$ (since $L + re_s$ is also just another line). Then for each $u \in U_r$ for $m-1 \leq r \leq h-1$, we run the next-element predictor from $\mathcal{F}_s$ for direction s using the already computed values for the $(m-1)$ predecessors of u (i.e. the points $u - e_s, u - 2e_s, \ldots, u - (m-1)e_s$). We then perform a "Reed-Muller list decoding step" using these predicted values to compute a small list of candidate s-variate polynomials one of which is likely to compute the correct value of p_x on U_r.[9] As in the bivariate extractor from the previous section, the advice string is used for purposes of "tie-breaking" to determine the correct polynomial at each of these intermediate list decoding steps. As before, the crucial aspect is that the number of such candidate polynomials is small and can also be found efficiently. This follows from a known result on list decoding Reed-Muller codes, stated below.

Lemma 11 (Follows from [13]) *For each $u \in \mathbb{F}_q^s$, let $S_u \subset \mathbb{F}_q$ be a set of size at most A. Then, provided $\delta \geq 2\sqrt{hA/q}$, there are at most $4A/\delta$ s-variate polynomials p over $\mathbb{F}_q$ of total degree h such that $p(u) \in S_u$ for at least a fraction δ of the points u. Moreover, there is a randomized algorithm that runs in time* $\mathrm{poly}(q^s)$ *and for any such input collection of sets $\{S_u\}_{u \in \mathbb{F}_q^s}$, outputs exactly the list of* all *such polynomials p with overwhelming probability.*

In analyzing the extractor, the above lemma is used with $A = O(\rho^{-2})$ (the number of outputs of the next-element predictors), and thus one only needs $O(\log(1/\rho) + \log(q/h))$ bits of advice for each tie-breaking step. Given an advice string, therefore, the reconstruction procedure is very efficient and in fact can be implemented to run in polynomial time. One can obtain a γ-decoding algorithm from the reconstruction procedure as in Section 3.2.3 by running through all the choices for next-element predictor functions — there will be $2^{O(mD)}$ choices in all since one has to pick a function from $\mathcal{F}_i$ for each of the directions e_i, $1 \leq i \leq D-1$ — as well as all choices of the advice strings.

Let us now argue about the number of advice bits needed as that governs the running time of the decoding algorithm. Unwinding the recursion, the number of different lines L' for which we determine $p_{x|L'}$ using the advice string in the computation of p_x on all of $L + \mathrm{span}_h\{e_1, \ldots, e_{D-1}\}$ equals $(m-1)^{D-1}$. Thus at most $m^{D-1}(h+1)\log q$ bits of advice are needed to figure out p_x on all the lines needed to kick-start the next-element prediction steps. The number of Reed-Muller list decoding steps, say N_s, involved for predicting p_x on $L + \mathrm{span}_h\{e_1, e_2, \ldots, e_s\}$ satisfies the recurrence $N_s = (m-1)N_{s-1} + (h-m+1)$ and $N_0 = 0$. Therefore we have $N_{D-1} \leq (m-1)^{D-2}(h-m+1) \leq m^{D-1}h$, and thus the total number of advice bits needed to disambiguate between candidate polynomials for all intermediate list decoding steps is at most $O(m^{D-1}h(\log(1/\rho) + \log(q/h)))$. Recalling that $\rho = \frac{7}{16m}$, the overall decoding time is thus $2^{O(m^{D-1}h\log(mq))}$, which for the choice of parameters $m = \log(D^2/\varepsilon) + O(1)$, $h \leq n_1^{1/D}$, and $q \leq m^{\max\{4, D-1\}}h \leq n_1$ for large enough n_1, is at most $2^{O(m^{D-1}n_1^{1/D}\log n_1)}$. Since the block length of the extractor $E^{(D)}$ viewed as a code equals $N = n_1 \cdot 2^{\Theta(D^2 \log m)} = n_1 m^{\Theta(D^2)}$, the runtime of the decoding procedure, expressed as a function of the block length of the constructed code, is at most $2^{O(N^{1/D}\log N)}$. □

[8] In case $L + \mathrm{span}_h\{e_1, e_2, \ldots, e_{D-1}\}$ is not D-dimensional, which happens with negligible probability for a random line L, the algorithm gives up and reports a decoding failure.

[9] In Section 3, we had $s = 1$, so that each of the list decoding steps was for univariate polynomials (Reed-Solomon codes).

5 Dispersers and Erasure List-Decodable Codes

In this section we make explicit some simple facts that connect dispersers, which are one-sided analogues of extractors, and list decoding from erasures. We recall that erasures are a noise model where a certain fraction of symbols are adversarially erased (replaced with "blanks", say) and the remaining are received intact (and it is known to the receiver which positions have been erased).

Definition 6 (Dispersers) *A function $D : (n_1) \times (d) \to (m)$ is a* strong (k, γ)-disperser *if for every $X \subseteq \{0,1\}^{n_1}$ of size at least 2^k, at most a γ fraction of points in $(d) \times (m)$ have zero probability under $U_d \circ D(X, U_d)$ where U_d (resp. X) denotes the uniform distribution on (d) (resp. X).*

The requirement is weaker than the extraction property, and dispersers exist with $d = \log(n-k) + \log(1/\gamma) + O(1)$ and entropy loss $\log\log(1/\gamma) + O(1)$, both of which are better than what is possible for extractors.

Definition 7 (Erasure List-decodability) *A code $C \subseteq [q]^n$ is said to be (ρ, L)-erasure list-decodable if for every $\mathbf{r} \in [q]^{n(1-\rho)}$ and every set $T \subseteq \{1, 2, \ldots, n\}$ of size $(1-\rho)n$, we have $|\{c \in C \mid c_{|T} = \mathbf{r}\}| \leq L$ where for $y \in [q]^n$, $y_{|T} \in [q]^{|T|}$ denotes the projection of y onto the coordinates in T.*

An interesting open problem in the area of list decoding, mentioned for example in [6], is obtaining an explicit construction of a family of $(1-\varepsilon, L)$-erasure list-decodable binary codes of rate $\Omega(\varepsilon)$ (which is optimal up to constant factors). The following simple lemma shows that explicit dispersers with optimal seed length will imply such codes. It is interesting that though dispersers are more general objects than erasure codes (just like extractors are more general than list-decodable codes), there is no asymptotic loss in rate due to their extra generality. In contrast, even optimal extractors, when interpreted as codes as in [15], give only codes of rate $O(\varepsilon^2)$ which is worse than the optimal $\Theta(\varepsilon)$ rate.

Lemma 12 *Suppose $D : (n_1) \times (d) \to (m)$ is a strong (k, γ)-disperser. Then the* binary *code $C : \{0,1\}^{n_1} \to \{0,1\}^{2^d}$ defined as $C(x)_{|i} = D(x, i)_1$ for $1 \leq i \leq 2^d$, where $D(x, i)_1$ denotes the first bit of the m-bit string $D(x, i)$, is $(1 - 2\gamma, 2^k)$-erasure list-decodable code. If the seed length of D is optimal with $d \leq \log n_1 + \log(1/\gamma) + O(1)$, then choosing $\gamma = \varepsilon/2$, we get a binary code C with rate $\Omega(\varepsilon)$ that is $(1-\varepsilon, 2^k)$-erasure list-decodable.*

Proof: Define $N = 2^d$ and let $T = \{t_1, t_2, \ldots, t_{2\gamma N}\}$ be a subset of $\{1, 2, \ldots, N\}$ of size $2\gamma N$ (with $1 \leq t_1 < t_2 < \cdots < t_{2\gamma N} \leq N$), and let $\mathbf{r} \in \{0,1\}^{2\gamma N}$. Let $x_1, x_2, \ldots, x_M$ be all the messages in (n_1) for which $C(x_s)_{|T} = \mathbf{r}$, $1 \leq s \leq M$. We need to show that $M \leq 2^k$.

Define $S \subseteq (d) \times (m)$ as follows: $S = \{(i, \alpha) \mid i \notin T; \alpha \in (m)\} \cup \{(t_j, \alpha) \mid 1 \leq j \leq 2\gamma N; \alpha_1 = \mathbf{r}_j\}$. Note that $|S| = (1-2\gamma)2^{d+m} + 2\gamma \cdot 2^d \cdot 2^{m-1} = (1-\gamma)2^{d+m}$. Now, for each x_s, $1 \leq s \leq M$, and each $y \in (d)$, we have $D(x_s, y) \in S$. By the (k, γ)-disperser property of D, it follows that $M \leq 2^k$, as desired. Finally, the rate of C equals $n_1/2^d$, and therefore is $\Omega(\varepsilon)$ if $d \leq \log n_1 + \log(1/\varepsilon) + O(1)$. □

6 Concluding Remarks

The obvious open question that our work highlights is that of constructing better extractors with $\log n_1 + O(1)$ seed length with error $\gamma = 1/2$ (say), in particular those which work for lower min-entropies (say $n_1^{o(1)}$). The improvements to the TZS extractors by Shaltiel and Umans [10] definitely

gives hope for such a pursuit. One would also need decoding algorithms for the extractors, but the success in turning proofs of the extraction property for the Trevisan and TZS extractors into decoding algorithms lends hope for also achieving the algorithmic goal if and when the extractors are constructed. Our interest in such extractors stems mainly from the quest for better list-decodable codes which can then be constructed using the connection presented in Lemma 1. Note however that extractors are stronger objects than list-decodable codes and there may well be simpler, less challenging ways to construct near-optimal list-decodable codes. But even in such a case, some of the techniques from the extractor world like next-element predictors and advice based reconstruction could be useful new additions to the toolkit of coding theory.

References

[1] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ronny Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[2] Noga Alon, Jeff Edmonds and Michael Luby. Linear time erasure codes with nearly optimal recovery. *Proceedings of FOCS'95*, pages 512-519.

[3] Venkatesan Guruswami. *List Decoding of Error-Correcting Codes*. Ph.D thesis, Massachusetts Institute of Technology, August 2001.

[4] Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Combinatorial Bounds for List Decoding. *IEEE Transactions on Information Theory*, 48(5):1021-1035, May 2002.

[5] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, October 2001, pages 658-667.

[6] Venkatesan Guruswami and Piotr Indyk. Near-optimal Linear-Time Codes for Unique Decoding and New List-Decodable Codes Over Smaller Alphabets. *Proceedings of STOC'02*, pages 812-821.

[7] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[8] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 181–190, 2000.

[9] Jaikumar Radhakrishnan and Amnon Ta-Shma. Bounds for Dispersers, Extractors, and Depth-Two Superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2-24, 2000.

[10] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudo-random generator. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, October 2001, pages 648-657.

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

[12] Madhu Sudan. List decoding: Algorithms and applications. *SIGACT News*, 31:16–27, 2000.

[13] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[14] Aravind Srinivasan and David Zuckerman. Computing with Very Weak Random Sources. *SIAM J. Comput.*, 28(4): 1433-1459, 1999.

[15] Amnon Ta-Shma and David Zuckerman. Extractor Codes. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 193–199, July 2001.

[16] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, October 2001, pages 638-647.

[17] Luca Trevisan. Extractors and pseudorandom generators. *Journal of the ACM*, 48(4): 860-879, 2001.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'