Streaming Computation of Combinatorial Objects

Ziv Bar-Yossef* Omer Reingold† Ronen Shaltiel‡ Luca Trevisan§

March 6, 2002

Abstract

We prove (mostly tight) space lower bounds for "streaming" (or "on-line") computations of four fundamental combinatorial objects: error-correcting codes, universal hash functions, extractors, and dispersers. Streaming computations for these objects are motivated algorithmically by massive data set applications and complexity-theoretically by pseudorandomness and derandomization for space-bounded probabilistic algorithms.

Our results reveal a surprising separation of extractors and dispersers in terms of the space required to compute them in the streaming model. While online extractors require space linear in their output length, we construct dispersers that are computable online with exponentially less space. We also present several explicit constructions of online extractors that match the lower bound.

We show that online universal and almost-universal hash functions require space linear in their output length (this bound was known previously only for "pure" universal hash functions [MNT93, BTY94]).

Finally, we show that both online encoding and online decoding of error-correcting codes require space proportional to the product of the length of the encoded message and the code's relative minimum distance. Block encoding trivially matches the lower bounds for constant rate codes.

*Computer Science Division, University of California at Berkeley, Berkeley, CA 94720. E-mail: zivi@cs.berkeley.edu. Supported by NSF Grant CCR-9820897.

†AT&T Labs - Research. Room A243, 180 Park Avenue, Bldg. 103, Florham Park, NJ, 07932, USA. E-mail: omer@research.att.com. Part of this research was performed while visiting the Institute for Advanced Study, Princeton, NJ.

‡Department of computer science,Weizmann Institute of Science, Rehovot, Israel. E-mail: ronens@wisdom.weizmann.ac.il. Part of this research was performed while visiting the Institute for Advanced Study, Princeton, NJ.

§Computer Science Division, University of California at Berkeley, Berkeley, CA 94720. E-mail: luca@cs.berkeley.edu.

1 Introduction

In this paper we deal with the "on-line space-bounded," or "streaming" model of computation, a model where a machine of bounded memory receives its input on a read-only tape, with one-way access. The goal is to design algorithms whose memory use is considerably shorter than the length of the input. In algorithm design, this model captures several settings where the input data is very large (hence it is infeasible to store in memory the entire input) and it is read, or "discovered" sequentially [AMS99, FKSV99, HRR99]. In complexity theory, this model captures the way probabilistic space-bounded algorithms use their randomness, so that pseudorandomness in the space-bounded setting has (non-uniform) "streaming algorithms" as "adversaries."

We consider the tasks of computing four fundamental primitives in this model: universal hash functions, error-correcting codes, randomness extractors and dispersers, and we present (mostly tight) lower bounds and explicit constructions.

Motivation

From an algorithmic perspective, streaming procedures for hash functions and error-correcting codes are basic primitives that may be useful for a variety of streaming applications. In fact, most streaming algorithms today (e.g., [AMS99, FKSV99]) make crucial use of hash functions. Error-correcting codes that admit space-efficient online encoding and decoding are important when having to transmit large amounts of data over a fast but unreliable channel. If the data is generated continuously on the fly, then online encoding eliminates the need to store the data before transmitting it. Online decoding allows one to process the received data without having to store it beforehand; this may result in large savings of space because the encoded data is frequently much larger than the message it encodes.

From a complexity-theoretic perspective, extractors and dispersers are important pseudorandomness and derandomization tools. They were used in the design of pseudorandom generators for space-bounded computations (explicitly in [NZ96, RR99] and implicitly in [Nis92]), and they are roughly equivalent to randomness-efficient procedures to reduce the error-probability in probabilistic algorithms. Randomness of space-bounded computations is assumed to arrive in a stream (written on a one-way random tape); this necessitates some of the pseudorandomness and derandomization procedures designed for such computations to admit online space-bounded implementations.

Roughly speaking, an extractor is a procedure $Ext(\cdot,\cdot)$ with two inputs, where the second input (also called the *seed*) is typically logarithmically shorter than the first input; the property is that if the first input comes from a distribution of sufficiently large entropy, and the seed is uniformly distributed, then the output of the procedure (which is shorter than the first input but still exponentially longer than the seed) is almost uniformly distributed. We think of an online extractor as a procedure with two one-way input tapes (one per input) and limited memory. To reduce error probability in space-bounded probabilistic algorithms, one needs online extractors in a slightly different model: the algorithm has one input tape for the first input x, and must produce the list of outputs $Ext(x, s)$ for all possible values of the second input s. Bar-Yossef *et al.* [BGW99] prove a strong space lower bound in this setting, but not in the more general setting where the algorithm has two tapes. If the first input has length n, the seed has length d, the output has length m, and the output is almost uniform provided the first input has entropy k, then [BGW99] prove that, in their model, the computation uses memory at least, roughly, $m - d$. A disperser $D(\cdot, \cdot)$ is a weaker type of extractor, such that if the first input has sufficiently large entropy, and the seed is uniform, then the output hits with non-zero probability every set of sufficiently large density. The [BGW99] lower bound also applies to dispersers. We remark that extractors (and, for a stronger reason, dispersers) can be computed in logarithmic space assuming one has *two-way* access to the input [HR00].

Perspective

Very strong connections are known between the combinatorial objects discussed in this paper.

To cite some examples, it is a matter of folklore that one can get hash functions with low collision probability from error-correcting codes (by encoding the input and projecting it to a small set of coordinates – an example of this approach can be seen in [Mil98]); error-correcting codes are used in the extractor construction of [Tre99], and in several more recent constructions [RRV99, ISW00, RSW00, TZS01, SU01]; hash functions are also extractors, as follows from the Leftover Hash Lemma [HILL99]; error-correcting codes with strong list-decodability properties can be derived from extractors [TZ01].

While there is no known equivalence, or transformation, between extractors and dispersers, there are several results pointing to a substantial equivalence between "randomization with one-sided error" and "randomization with two-sided error," the former being the setting of dispersers, and the latter being the setting of extractors. For example, it is known that "hitting set generators" (that are somewhat the "computational" version of dispersers) can be used to derandomize algorithms with two-sided error (a task for which it formerly appeared that "pseudorandom generators," the "computational" version of extractors, were needed) [ACR98, BF99, GW99, GVW00]. In fact, the results of [GW99, GVW00] have an information-theoretic interpretation that says that dispersers can be converted into "samplers," that are almost, but not quite, extractors. Upper and lower bounds for extractors and dispersers show that essentially the same parameters are achievable for the two objects, and even the results of [BGW99] do not differentiate between the two.

For hash functions and error-correcting codes, we were very interested in the question of whether reasonably space-efficient algorithms could exist. As we state below, the answer is affirmative for hash functions, while it is completely negative for error-correcting codes.

However, more generally, our main interest was to look at these tightly related objects under the lens of a very restrictive model of computation, and see what happens of their connections. As we state below, we give a strong (exponential, in the case of long seed) separation between the space sufficient to compute dispersers and the space necessary to compute extractors. Our results also show the extreme differences in power between seemingly similar models of space-bounded computation. We show how to construct online dispersers with exponentially less memory than in the [BGW99] setting (that had one input tape, and the outputs had to be "enumerated" over all values for the second input), and we show a lower bound for extractors with one-way tapes that is exponentially bigger than the space needed with a two-way tape in the [HR00] construction.

Finally, the generation of high-quality randomness from biased sources is a very important practical problem, that does arise in settings for which the streaming model is an appropriate formalization. In such cases, one is probably not interested in the randomness extractors satisfying the strong definition used in this paper (that need the uniform second input), but rather in faster deterministic extractors that (for special classes of distributions) directly convert a biased streaming input into an almost uniform output stream (cf. [TV00]). It would probably be very interesting to study which classes of distributions admitting polynomial time deterministic extractors also have space-efficient streaming extractors. Our work hopefully prepares the terrain for the treatment of such questions.

Our Results

For error-correcting codes mapping k bits into n bits, and that are able to correct at least δk errors, we prove that both the encoding and the decoding procedures must use memory $\Omega(\delta k)$. The bound can be matched trivially by dividing the input into $O(1/\delta)$ blocks, and encoding each block with a code of constant rate and constant relative minimum distance. These results are described in Section 7.

For universal hash functions mapping n bits into m bits, our space lower bound is roughly m. (This bound follows also from the time-space tradeoffs of Mansour *et al.* [MNT93] and the communication-space tradeoffs of Beame *et al.* [BTY94]. Our bound has the advantage of being applicable also to almost-universal hash function.) The bound can be achieved by linear hash functions. If the output is considerably shorter than the input, this can be a significant saving. Using almost-universal hash functions, which admit $O(m + \log n)$-sized descriptions, one can evaluate such hash functions on many inputs using space $O(m + \log n)$. The lower bound for hash functions follows from the lower bound for extractors and from the Leftover Hash Lemma. These results are described in Section 4 (lower bound) and Section 5 (upper bound).

Our main results are the lower bound for extractors and the construction of dispersers. Combined, they show an unusual "separation" between the two combinatorial objects, and offer fresh insights.

For extractors where the first input has length n, the seed has length d, the output has length m, and the output is uniform assuming the first input has entropy k, our lower bound has two cases.

If $k \leq n/2$, i.e. if the extractor works with inputs of relatively small entropy, then we prove that the memory has to be at least, roughly, $m - d$. This is matched by careful implementations of the extractors of Trevisan and of Raz *et al.* [Tre99, RRV99].

If $k > n/2$, then the lower bound is roughly $(m - d)(n - k)/n$, which we can match with a space-efficient implementation of random walks on expanders (using ideas from [BGW99]).

These results are described in Section 3 (lower bounds) and Section 5 (upper bounds).

The proof uses the idea that after looking at a block of the input of size $n - k$, the extractor cannot be sure it has seen any randomness (because the k bits of entropy could be "concentrated" in the remaining part of the input), and so it can only output bits from the memory or from the seed. For $k \leq n/2$, we can say that the extractor cannot output anything while looking at the first $n/2$

bits, and it can only output randomness that is in the state or in the seed afterwards. While the intuition is clear, the actual proof requires considerable technical work.

For dispersers and for $k \leq n/2$, we show that it is possible to use memory about m/ℓ and seed $d = O(\ell \log n)$ for every parameter ℓ. The idea is to randomly partition the input into ℓ blocks, in such a way that each block still contains sufficiently large entropy, and then use memory m/ℓ to extract randomness from each block. A good partition will be found with low probability, but this is compatible with the definition of disperser. The construction and its analysis, presented in Section 6, utilize ideas from previous constructions of extractors and dispersers [NZ96, SSZ98, Ta-96, Ta-98].

2 Preliminaries

In this section we define online extractors, dispersers, universal hash functions, and error-correcting codes. We then review some tools from Information Theory we use in our analysis.

2.1 Online Extractors and Dispersers

$||X-Y||$ denotes the total variation distance between two distributions on the same domain Ω: $||X-Y|| \stackrel{\text{def}}{=} \frac{1}{2}\sum_{\omega\in\Omega}|X(w)-Y(w)| = \max_{T\subseteq\Omega}|X(T)-Y(T)|$. Given a distribution X on Ω and a function $f:\Omega\rightarrow\Omega'$ we denote by $f(X)$ the distribution induced by X and f on Ω'. U_t denotes the uniform distribution on $\{0,1\}^t$. $H_\infty(X) = \min_{w\in\Omega}(\log(1/X(\omega)))$ denotes the min-entropy of a distribution X on Ω. An (n,d,m) *function* is a function of the form $f:\{0,1\}^n\times\{0,1\}^d\rightarrow\{0,1\}^m$; we nickname its first input "the input" and its second input "the seed".

Definition 1 (Extractor [NZ96]) *An (n,d,m) function E is called a (k,ϵ)-*extractor*, if for every distribution X on $\{0,1\}^n$ with $H_\infty(X)\geq k$, $||E(X,U_d)-U_m||<\epsilon$.*

Definition 2 (Disperser [Sip88]) *An (n,d,m) function D is called a (k,ϵ)-*disperser*, if for every distribution X on $\{0,1\}^n$ with $H_\infty(X)\geq k$, and for every subset $T\subseteq\{0,1\}^m$ of size at least $\epsilon 2^m$, $\Pr(D(X,U_d)\in T)>0$.*

We study *online extractors* and *online dispersers* – ones that are computable by space-bounded one-pass algorithms. We consider two variants of such algorithms: *single-seed algorithms* and *all-seeds algorithms*:

Definition 3 (Online extractors/dispersers) *An algorithm A is called a* single-seed one-pass algorithm *for an (n,d,m) function f, if given one-way access to an input $x\in\{0,1\}^n$ and one-way access to a seed $r\in\{0,1\}^d$, it outputs $f(x,r)$ on a one-way output tape.*[1] *A is called an* all-seeds one-pass algorithm for *f, if given one-way access to an input $x\in\{0,1\}^n$, it outputs $f(x,r)$ for all $r\in\{0,1\}^d$ on 2^d one-way output tapes.*

[1] The one-way access to the seed is required for the lower bound for strong extractors, where the seed may be very long. For the standard scenario, where the seed is much shorter than the input, we can relax the definition by allowing two-way access to the seed; this would change our lower bounds by only an additive factor of d.

The space of an algorithm is defined to be the binary logarithm of the number of possible configurations the algorithm has. Each configuration consists of the machine's state and the contents of the work space. Hence, the maximum number of configurations an S-space machine has is 2^S.

Bar-Yossef, Goldreich, and Wigderson [BGW99] proved space lower bounds for all-seeds one-pass algorithms for dispersers. Our results show that for extractors their space lower bound holds even for single-seed one-pass algorithms, but for dispersers there are substantially more space-efficient single-seed one-pass algorithms.

Theorem 4 (Bar-Yossef, Goldreich, Wigderson [BGW99]) *Define* $t \stackrel{\text{def}}{=} \lceil n/(n-k) \rceil$. *Let* $D : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *be a* (k, ϵ)*-disperser with* $\epsilon < 1/t$. *Then, for all integers* $1 \leq p \leq 2^d$, *any all-seeds one-pass algorithm for* D *that writes to at most* p *output tapes simultaneously requires space*

$$S \geq m - d - p - 1 - \log t - \log(1/1 - \epsilon t)$$

The original definition of extractors in [NZ96] is stronger than the one in Definition 1. Loosely, E is a (k, ϵ) *strong* extractor if for any distribution X with $H_\infty(X) \geq k$, all but an ϵ fraction of the seeds are good for X. Where a seed y is good for X if the distribution $E(X, y)$ is ϵ-close to uniform. This requirement is essentially equivalent to the following definition:

Definition 5 (Strong extractor) *An* (n, d, m) *function* E *is called a* (k, ϵ) strong extractor, *if for every distribution* X *on* $\{0,1\}^n$ *with* $H_\infty(X) \geq k$, $||E(X, U_d) \circ U_d - U_{m+d}|| < \epsilon$ *(where the two occurrences of* U_d *refer to the same variable).*

In a similar manner, strong dispersers may be defined as follows:

Definition 6 (Strong disperser) *An* (n, d, m) *function* D *is called a* (k, ϵ) strong disperser, *if for every distribution* X *on* $\{0,1\}^n$ *with* $H_\infty(X) \geq k$, *and for every subset* $T \subseteq \{0,1\}^{m+d}$ *of size at least* $\epsilon 2^{m+d}$, $\Pr(D(X, U_d) \circ U_d \in T) > 0$ *(where the two occurrences of* U_d *refer to the same variable).*

It is interesting to note that all our upper bounds for online extractors are exhibited by strong extractors. In contrast, our construction of space-efficient online dispersers allows almost all of the seeds to be "bad" for a specific source.

2.2 Online Universal Hash Functions

Definition 7 (Universal hash functions [CW79]) *A family of functions* $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ *is called a* universal family of hash functions, *if for every* $x \neq x' \in \{0,1\}^n$, $\Pr_{h \in H}(h(x) = h(x')) \leq 1/2^m$.

There are explicit constructions of universal hash functions of size $2^{O(n+m)}$ that are logspace computable. For example, the Toeplitz family (cf. [Gol97]) is of size 2^{n+m-1}.

We study *online hash functions* – ones that are computable by space-bounded one-pass algorithms:

Definition 8 (Online universal hash functions) *An algorithm A is called a* one-pass algorithm *for a family of hash functions $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$, if given one-way access to a (description of a) function $h \in H$ and one-way access to an input $x \in \{0,1\}^n$, A outputs $h(x)$ on a one-way output tape.*

The Leftover Hash Lemma, due to Hastad, Impagliazzo, Levin, and Luby [HILL99], yields a construction of (strong) extractors from any universal family of hash functions:

Lemma 9 (Hastad, Impagliazzo, Levin, Luby [HILL99]) *Let $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a universal family of hash functions of size 2^d. Then the function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ defined as $E(x,h) = h(x)$ is a (k,ϵ) strong extractor for any $k \leq n$ and for $\epsilon \geq 2^{(m-k)/2-1}$.*

Therefore, our space lower bounds for online extractors will directly imply space lower bounds for online universal hash-functions. In fact, a "almost" universal family of hash functions are sufficient to construct extractors. Therefore, our lower bounds apply to such families as well.

Definition 10 (ϵ-almost universal hash functions) *A family of functions $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is called a* ϵ-almost universal family of hash functions, *if for every $x \neq x' \in \{0,1\}^n$, $\Pr_{h\in H}(h(x) = h(x')) \leq \epsilon$.*

Such ϵ-almost universal hash functions still yield (strong) extractors using the following version of the Leftover Hash Lemma.

Lemma 11 (Impagliazzo, Zuckerman [IZ89]) *Let $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ be an $((1+\epsilon^2)/2^m)$-almost universal family of hash functions of size 2^d. Then the function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ defined as $E(x,h) = h(x)$ is a (k,ϵ) strong extractor for any $k \leq n$ such that $\epsilon \geq 2^{(m-k)/2}$.*

We note that the extractors based on universal hash functions (e.g., the Toeplitz family), have seed length $O(n)$. Based on almost universal families, the seed length can be reduced to $O(m + \log n + \log 1/\epsilon)$ (see e.g. [GW94]).

2.3 Online Error-Correcting Codes

Let $\mathbb{F}_q$ be a field of size q. An $(n,k,d)_q$ *error-correcting code* (ECC) is a subset $\mathcal{C} \subseteq \mathbb{F}_q^n$ of size q^k, such that for every two distinct codewords $w, w' \in \mathcal{C}$, the Hamming distance between w and w' (i.e., $|\{i \mid w_i \neq w'_i\}|$) is at least d. n is called the *code's length*, k is its *dimension*, and d is its *minimum distance*. k/n is the code's *rate* and d/n is its *relative minimum distance*. An *encoding function*, $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$, maps every message to its encoding. A *decoding function*, $D : \mathbb{F}_q^n \to \mathbb{F}_q^k$, maps every received (possibly corrupted) message to the origin of its closest codeword.

Definition 12 (Online error-correcting codes) *Let $\mathcal{C}$ be an $(n,k,d)_q$-code. An algorithm A is called a* one-pass encoding algorithm, *if given one-way access to a message $x \in \mathbb{F}_q^k$ it outputs*

$E(x)$ on a one-way output tape. An algorithm B is called a one-pass decoding algorithm, *if given one-way access to a received message $w \in \mathbb{F}_q^n$, it outputs $D(w)$ on a one-way output tape.*

2.4 Tools from Information Theory

Throughout this paper we use several tools from information theory and coding theory. We briefly survey them below.

Shannon's (binary) entropy is defined as $H(X) = E_{w \in X}(\log(1/X(\omega)))$, where the log is to the base of 2. Two basic properties of the entropy are the following (cf. [CT91], Chapter 2):

Proposition 13 *(1) Entropy sub-additivity: for any two distributions X, Y, $H(X,Y) \leq H(X) + H(Y)$.*

(2) Data processing inequality: for any distribution X on Ω and any function $f : \Omega \to \Omega'$, $H(f(X)) \leq H(X)$.

The following theorem (see [CT91], pages 488–489) connects the variation distance between two distributions and their entropy difference:

Theorem 14 *Let X, Y be two distributions on a set Ω with $||X - Y|| \leq 1/4$. Then,*

$$|H(X) - H(Y)| \quad \leq \quad 2||X - Y|| \cdot (\log \frac{|\Omega|}{||X - Y||} - 1)$$

The following fact (due to Lawrence Ip [Ip01]) connects the entropy of a distribution on $\{0,1\}^{\leq m}$ (all the binary strings of length at most m) to the expected length of strings under the distribution:

Proposition 15 *Let X be a distribution over $\{0,1\}^{\leq m}$. Then,*

$$H(X) \leq E(|X|) + H(|X|) \leq E(|X|) + \log(m+1)$$

Proof. Since $H(X) \leq H(X \mid |X|) + H(|X|)$, it is enough to prove that $H(X \mid |X|) \leq E(|X|)$.

$$H(X \mid |X|) \;=\; \sum_{\ell=0}^{m} H(X \mid |X| = \ell) \Pr(|X| = \ell) \;\leq\; \sum_{\ell=0}^{m} \ell \Pr(|X| = \ell) \;=\; E(|X|)$$

□

A *binary code* is a subset $\mathcal{C} \subseteq \{0,1\}^*$. $\mathcal{C}$ is called *prefix-free*, if for every two codewords $w, w' \in \mathcal{C}$, w is not a prefix of w' and vice versa. Kraft's inequality (see [CT91], pages 82–83) provides a constraint on the sizes of codewords in prefix-free codes:

Theorem 16 (Kraft's Inequality) *For any finite binary prefix-free code $\mathcal{C}$, the codeword lengths $l_1, \ldots, l_m$ must satisfy the inequality*

$$\sum_{i=1}^{m} \frac{1}{2^{l_i}} \leq 1$$

For the extractor upper bounds, we will need to use binary error-correcting codes with very good rate and distance properties (polynomial rate and almost $1/2$ relative minimum distance). We next show a construction of such codes that are efficiently encodable.

The Reed-Solomon (RS) code is an $(n, k, n-k+1)_q$ code for $q \geq n$. We associate every vector in $\mathbb{F}_q^k$ with the corresponding degree $k-1$ polynomial over $\mathbb{F}_q$, whose coefficients are the entries of this vector. The encoding function is $E_{RS}(p) = (p(0), \ldots, p(n-1))$ that assigns to p its evaluations on the first n elements of $\mathbb{F}_q$ (under some canonical order). Since no two degree $k-1$ polynomials can agree on more than $k-1$ elements, the code has minimum distance $n-k+1$.

The Hadamard (H) code is an $(2^k, k, 2^{k-1})_2$ code that corresponds to the rows of the $2^k \times 2^k$ Hadamard matrix. That is, for every $\alpha \in \{0,1\}^k$, $E_H(\alpha) = (\langle \alpha, \beta_1 \rangle, \ldots, \langle \alpha, \beta_{2^k} \rangle)$, where $\beta_1, \ldots, \beta_{2^k}$ are the bitstrings in $\{0,1\}^k$ ordered lexicographically, and the inner product is modulo 2.

A *concatenation* of two codes, $(n_1, k_1, d_1)_{q^{k_2}} \circ (n_2, k_2, d_2)_q$, is an $(n_1 \cdot n_2, k_1 \cdot k_2, d_1 \cdot d_2)_q$ code obtained by encoding each message by the first ("outer") code and each letter of the obtained codewords by the second ("inner") code. Concatenation is a standard technique to construct good codes over small alphabets from good codes over large alphabets. By concatenating Reed-Solomon and Hadamard codes one can achieve good binary codes:

Lemma 17 *For every $0 < \gamma < 1/2$, and for every n which is a power of 2, the concatenation of a $(\sqrt{n}, 2\gamma\sqrt{n}+1, \sqrt{n}(1-2\gamma))_{\sqrt{n}}$ Reed-Solomon code with a $(\sqrt{n}, \frac{1}{2}\log n, \frac{\sqrt{n}}{2})_2$ Hadamard code gives a $(n, \gamma\sqrt{n}\log n + \frac{1}{2}\log n, n(\frac{1}{2}-\gamma))_2$ binary code.*

3 Lower Bounds for Online Extractors

We present two versions of the lower bound: the first (Theorem 18) gives weaker bounds, but its proof is intuitive and utilizes known facts from information theory; the second (Theorem 19) is stronger, but its proof is more involved. We also show (Theorem 22) that an even stronger lower bound holds for strong extractors.

The basic idea behind both lower bounds is the following: we split every input $x \in \{0,1\}^n$ into $t = n/(n-k)$ blocks of size $n-k$. A distribution $X_{i,\alpha}$ that is fixed to some string $\alpha \in \{0,1\}^{n-k}$ on the i-th block and uniform everywhere else has min-entropy k, and thus $E(X_{i,\alpha}, U_d)$ is ϵ-close to uniform. The algorithm, when reading the i-th block of the input, cannot know whether these bits contain some entropy or are totally fixed. Intuitively, this implies that the algorithm cannot immediately extract the entropy of the i-th block, if it exists, but rather has to wait for the $i+1$-st block. However, at that point the algorithm "remembers" only S bits of the entropy of the i-th block. It follows that the algorithm can extract at most S bits of entropy from each block, and therefore $S \geq m/t$.

Both lower bounds hold only for extractors in which $\epsilon < 1/t$. When the entropy is at most a constant fraction of n, this does not pose a strict restriction on the error. However, for extremely high entropies ($k \geq n - o(n)$) this requires the error to be $o(1)$. It remains open to check whether this limitation is inherent or just an artifact of our proof.

Theorem 18 *Let $t = \lceil n/(n-k) \rceil$, and let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k, ϵ)-extractor with $\epsilon < 1/4$. Then any one-pass algorithm A that computes E requires space*

$$S \geq \frac{m}{t} - d - 2\epsilon(m + \log\frac{1}{\epsilon} - 1) - \log(m+1)$$

Proof. Let A be a one-pass algorithm that computes E, and let S be the space of A. We split any input $x \in \{0,1\}^n$ into t blocks: $x^1, \ldots, x^t$, each (except, maybe, for the last one) of length $n - k$. In the sequel we assume for simplicity of notation that the last block is also of size $n - k$.

For any input $x \in \{0,1\}^n$ and seed $r \in \{0,1\}^d$, we divide the execution of A on x and r into t phases: the i-th phase ends just before A starts to read the first bit of the $i+1$-st block; the first phase starts at the first step of the algorithm; the last phase ends at the end of the execution. We denote by $y^i(x,r)$ the bits A outputs during the i-th phase, by $r^i(x,r)$ the bits of the seed r it reads during the i-th phase, and by $c^i(x,r)$ the configuration of A at the beginning of the i-th phase.

Varying over all inputs x and seeds r, A may have at most 2^S possible configurations at the beginning of the i-th phase for any $i \geq 2$. For $i = 1$, it has only one possible configuration at the beginning of the first phase – the initial configuration.

For any block i and string $\alpha \in \{0,1\}^{n-k}$, let $X_{i,\alpha}$ be a distribution on $\{0,1\}^n$ which is fixed to α on the i-th block and uniform everywhere else. Clearly, $H_\infty(X_{i,\alpha}) = k$.

In general, $y^i(x,r)$ is a deterministic function of $x^i, r^i(x,r)$ and $c^i(x,r)$. When picking inputs according to $X_{i,\alpha}$, all the inputs share the same i-th block, and therefore $y^i(x,r)$ is a function only of $r^i(x,r)$ and $c^i(x,r)$. We can thus write $y^i(x,r) = g_{i,\alpha}(c^i(x,r), r^i(x,r))$ for some function $g_{i,\alpha}$.

We first show that there exists some block i and an assignment α to this block, such that A outputs at least m/t bits in the i-th phase when running on $(X_{i,\alpha}, U_d)$:

Claim 1 *There exists a block i and a string $\alpha \in \{0,1\}^{n-k}$, such that $E(|y^i(X_{i,\alpha}, U_d)|) \geq m/t$.*

Proof. Since the output of A is always of length m, then for any input x and any seed r, $\sum_{i=1}^t |y^i(x,r)| = m$. In particular, if we choose $x \in U_n$ and $r \in U_d$, we have, $E(\sum_{i=1}^t |y^i(U_n, U_d)|) = m$. Therefore, there exists some $i \in \{1, \ldots, t\}$, such that $E(|y^i(U_n, U_d)|) \geq m/t$.

We can think of the choice of $x \in U_n$, as first picking $x^i \in U_{n-k}$, and then picking the rest of the bits, x^{-i}, from U_k. We denote by $x^i \circ x^{-i}$ the n-bit string that has x^i in its i-th block and x^{-i} in the rest of the blocks. Then, $m/t \leq E(|y^i(U_{n-k} \circ U_k, U_d)|) = E_{x^i \in U_{n-k}}(E(|y^i(x^i \circ U_k, U_d)|))$. It follows that there exists some assignment $\alpha \in \{0,1\}^{n-k}$ for which $E(|y^i(\alpha \circ U_k, U_d)|) \geq m/t$. $\square$

Define $Y^i \stackrel{\text{def}}{=} y^i(X_{i,\alpha}, U_d)$ and $C^i \stackrel{\text{def}}{=} c^i(X_{i,\alpha}, U_d)$. The discussion above implies that there exists some function $g_{i,\alpha}$ such that $Y^i = g_{i,\alpha}(C^i, U_d)$. Therefore, $H(Y^i) = H(g_{i,\alpha}(C^i, U_d)) \leq H(C^i, U_d) \leq H(C^i) + H(U_d) \leq S + d$.

Let $Y^{-i} \stackrel{\text{def}}{=} y^{-i}(X_{i,\alpha}, U_d)$, where $y^{-i}(x,r)$ are all the bits A outputs on x and r *not* during the i-th phase. Note that $|y^{-i}(x,r)| = m - |y^i(x,r)|$, and thus by Claim 1, $E(|Y^{-i}|) \leq m(1 - 1/t)$. In order to bound $H(Y^{-i})$, we use Proposition 15 (see Section 2.4): $H(Y^{-i}) \leq E(|Y^{-i}|) + \log(m+1) \leq m(1-1/t) + \log(m+1)$. We can now bound the entropy of $Y \stackrel{\text{def}}{=} E(X_{i,\alpha}, U_d)$:

$$H(Y) = H(Y^i, Y^{-i}) \leq H(Y^i) + H(Y^{-i}) \leq S + d + m(1-1/t) + \log(m+1)$$

Since $||Y - U_m|| < \epsilon \leq 1/4$, and since the the function $x\log(1/x)$ is monotone increasing for $x \leq 1/2$, we can apply Theorem 14 (see Section 2.4) and obtain:

$$|H(Y) - H(U_m)| \leq 2\epsilon(\log(2^m/\epsilon) - 1) = 2\epsilon(m + \log(1/\epsilon) - 1)$$

Since $H(U_m) = m$, this implies that: $H(Y) \geq m(1-2\epsilon) - 2\epsilon(\log(1/\epsilon) - 1)$. Combining the upper and lower bounds on $H(Y)$, we obtain the desired lower bound on S. □

Theorem 19 *Let $t \stackrel{\text{def}}{=} \lceil n/(n-k) \rceil$, and let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ϵ)-extractor with $\epsilon \leq 1/(2t^2)$. Then any one-pass algorithm A that computes E requires space*

$$S \geq \frac{m-d}{t-1} - 2\left(1 + \frac{1}{t-1}\right)\log t$$

Proof. Let A be any one-pass algorithm that computes E, and let S be the space of A. We will use the same notations as in the proof of Theorem 18.

We define a seed r to be i-bad for an input x, if the algorithm, when running on x and r, outputs many more bits during the i-th phase than what it has in memory at the beginning of the phase and what it reads from the seed during the phase:

Definition 20 (Good seeds and Bad seeds) *A seed $r \in \{0,1\}^d$ is called* i-bad *for an input $x \in \{0,1\}^n$, if $|y^i(x,r)| \geq S^i + |r^i(x,r)| + 2\log t + 1$, where $S^i = 0$ for $i = 1$, and $S^i = S$ for $i = 2, \ldots, t$. Otherwise, r is* i-good *for x.*

We denote by $B^i(x)$ the set of seeds that are i-bad for x. $B(x) = \bigcup_{i=1}^t B^i(x)$ is the set of seeds that are i-bad for x for some i. Equivalently, $G^i(x)$ is the set of seeds that are i-good for x, and $G(x) = \bigcap_{i=1}^t G^i(x)$ is the set of seeds that are i-good for x for all i.

For any block i and string $\alpha \in \{0,1\}^{n-k}$, let $X_{i,\alpha}$ be a distribution on $\{0,1\}^n$ which is fixed to α on the i-th block and uniform everywhere else. Clearly, $H_\infty(X_{i,\alpha}) = k$. The following lemma shows that for all i and for all α, the expected number of i-bad seeds for inputs chosen according to $X_{i,\alpha}$ is small:

Lemma 21
For all $i \in \{1, \ldots, t\}$, and for all $\alpha \in \{0,1\}^{n-k}$, $E_{x \in X_{i,\alpha}}(|B^i(x)|) < (1/(2t^2) + \epsilon)2^d$.

Proof. In the following we use the notation $X_{i,\alpha}$ interchangeably both for the distribution defined above and for the support of this distribution (i.e., the subset of $\{0,1\}^n$ consisting of all strings whose i-th block is α).

Since $H_\infty(X_{i,\alpha}) \geq k$, then $||E(X_{i,\alpha}, U_d) - U_m|| < \epsilon$. Therefore, for all subsets $T \subseteq \{0,1\}^m$, $|\Pr_{y \in U_m}(y \in T) - \Pr_{x \in X_{i,\alpha}, r \in U_d}(E(x,r) \in T)| < \epsilon$.

Define $T \stackrel{\text{def}}{=} \{E(x,r) \mid x \in X_{i,\alpha}, r \in B^i(x)\}$ to be the set of all strings obtained by applying the extractor on an input from $X_{i,\alpha}$ and a bad seed for this input. We will show that $|T| \leq 2^m/(2t^2)$. This would imply that $\Pr_{y \in U_m}(y \in T) \leq 1/(2t^2)$, and therefore $\Pr_{x \in X_{i,\alpha}, r \in U_d}(E(x,r) \in T) < 1/(2t^2) + \epsilon$. Thus, $E_{x \in X_{i,\alpha}}(|B^i(x)|) = 2^d \cdot \Pr_{x \in X_{i,\alpha}, r \in U_d}(r \in B^i(x)) \leq 2^d \cdot \Pr_{x \in X_{i,\alpha}, r \in U_d}(E(x,r) \in T) < (1/(2t^2) + \epsilon) \cdot 2^d$, completing the proof.

Let $c^i_1, \ldots, c^i_{\ell_i}$ be all the possible configurations of A at the beginning of the i-th phase (note that $\ell_i \leq 2^{S^i}$). Recall that for inputs chosen according to $X_{i,\alpha}$, the output of A during the i-th phase, $y^i(x,r)$, is a function only of $c^i(x,r)$ and $r^i(x,r)$: $y^i(x,r) = g_{i,\alpha}(c^i(x,r), r^i(x,r))$.

For any configuration c^i_j, let us denote by $R^i(c^i_j)$ the set of seed sub-strings that A can read during the i-th phase on inputs $x \in X_{i,\alpha}$, given that A is in c^i_j at the beginning of the phase (that is, $R^i(c^i_j) = \{r^i(x,r) \mid x \in X_{i,\alpha}, r \in \{0,1\}^d, c^i(x,r) = c^i_j\}$). Note that $R^i(c^i_j)$ is a prefix-free code, since for any $r^i \in R^i(c^i_j)$, A, when being at c^i_j and reading r^i, starts its $i+1$-st phase; in particular, any further bits it reads from the seed cannot belong to the i-th phase.

We denote by $B^i(c^i_j)$ the seed sub-strings on which A's execution during the i-th phase is "bad"; that is, $B^i(c^i_j) = \{r^i \in R^i(c^i_j) \mid |g_{i,\alpha}(c^i_j, r^i)| \geq S^i + |r^i| + 2\log t + 1\}$. Since $B^i(c^i_j)$ is a subset of $R^i(c^i_j)$ it is also a prefix-free code. For every $u = 0, \ldots, d$, let $B^i_u(c^i_j)$ be the set of strings in $B^i(c^i_j)$ of length u.

For every $j = 1, \ldots, \ell_i$, let T_j be the set of strings in T obtained from pairs (x,r) for which $c^i(x,r) = c^i_j$; that is, $T_j = \{E(x,r) \mid x \in X_{i,\alpha}, r \in \{0,1\}^d, c^i(x,r) = c^i_j, r^i(x,r) \in B^i(c^i_j)\}$. Clearly, $|T| \leq \sum_{j=1}^{\ell_i} |T_j|$.

We further decompose T_j into $d+1$ sets, corresponding the possible lengths of the seed sub-string read during the i-th phase: $T_{j,u} = \{E(x,r) \mid x \in X_{i,\alpha}, r \in \{0,1\}^d, c^i(x,r) = c^i_j, r^i(x,r) \in B^i_u(c^i_j)\}$. Clearly, $|T_j| \leq \sum_{u=0}^{d} |T_{j,u}|$.

On every pair (x,r) that creates a string $y(x,r)$ in $T_{j,u}$, A outputs at least $S^i + u + 2\log t + 1$ bits during the i-th phase. We decompose each string $y \in T_{j,u}$ into two parts: y_1 – the first $S^i + u + 2\log t + 1$ bits output during the i-th phase, and y_2 – all the rest. This induces the following decomposition of $T_{j,u}$: $T^1_{j,u} = \{y_1 \mid y \in T_{j,u}\}$ and $T^2_{j,u} = \{y_2 \mid y \in T_{j,u}\}$. Clearly, $|T_{j,u}| \leq |T^1_{j,u}| \cdot |T^2_{j,u}| \leq |B^i_u(c^i_j)| \cdot 2^{m - S^i - u - 2\log t - 1}$.

We thus obtain:

$$\begin{aligned}
|T| &\le \sum_{j=1}^{\ell_i} |T_j| \;\le\; \sum_{j=1}^{\ell_i}\sum_{u=0}^{d} |T_{j,u}| \\
&\le \sum_{j=1}^{\ell_i}\sum_{u=0}^{d} |B^i_u(c^i_j)| \cdot 2^{m-S^i-u-2\log t-1} \\
&= 2^{m-S^i-2\log t-1} \cdot \sum_{j=1}^{\ell_i}\sum_{u=0}^{d} |B^i_u(c^i_j)| \cdot 2^{-u} \\
&\le 2^{m-S^i-2\log t-1} \cdot \ell_i \;\le\; \frac{2^m}{2t^2}
\end{aligned}$$

where the next to the last inequality follows from Kraft's inequality and the fact that $B^i(c^i_j)$ is a prefix-free code, and the last inequality follows from the fact that $\ell_i \le 2^{S^i}$. □

Since the lemma holds for every fixing α of the i-th block, it follows that it holds also when picking $x \in U_n$. That is, $E_{x \in U_n}(|B^i(x)|) < (1/(2t^2) + \epsilon)2^d$. Therefore, by Markov's inequality, for more than a $1 - 1/t$ fraction of the $x \in \{0,1\}^n$, $|B^i(x)| < (1/(2t) + \epsilon t)2^d$. Applying the union bound, there is an $x \in \{0,1\}^n$ for which $|B^i(x)| < (1/(2t) + \epsilon t)2^d$ for all $i = 1, \ldots, t$. Therefore, $|B(x)| \le \sum_{i=1}^t |B^i(x)| < (1/2 + \epsilon t^2)2^d \le 2^d$. This implies that $|G(x)| = 2^d - |B(x)| \ge 1$; that is, x has at least one good r. We now obtain for $y = E(x, r)$:

$$\begin{aligned}
m = |y| &= \sum_{i=1}^{t} |y^i(x,r)| \;\le\; \sum_{i=1}^{t} S^i + |r^i(x,r)| + 2\log t \\
&= (t-1)S + \sum_{i=1}^{t} |r^i(x,r)| + 2t\log t \;=\; (t-1)S + d + 2t\log t
\end{aligned}$$

The theorem follows. □

For strong extractors a stronger lower bound holds:

Theorem 22 *Let $t \stackrel{\text{def}}{=} \lceil n/(n-k) \rceil$, and let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k, ϵ) strong extractor with $\epsilon \le 1/(2t^2)$. Then any one-pass algorithm A that computes E requires space*

$$S \ge \frac{m}{t-1} - 2\left(1 + \frac{1}{t-1}\right)\log t$$

Proof sketch. The proof is almost identical to the proof of Theorem 19. The main difference is in the definition of bad seeds: a seed $r \in \{0,1\}^d$ is i-bad for an input $x \in \{0,1\}^n$, if $|y^i(x,r)| \ge S^i + 2\log t + 1$ (i.e., without the $|r^i(x,r)|$ part).

We next show that Lemma 21 holds for strong extractors even with this weaker definition of bad seeds. The proof is practically the same, except for the following: we define $T \stackrel{\text{def}}{=} \{E(x,r) \circ r \mid x \in$

$X_{i,\alpha}, r \in B^i(x)\}$, and show that $|T| \leq 2^{m+d}/(2t^2)$. Since $||E(X_{i,\alpha}, U_d) \circ U_d - U_{m+d}|| < \epsilon$, an identical argument to the one in the proof of Lemma 21 shows that this implies that $E_{x \in X_{i,\alpha}}(|B^i(x)|) < (1/(2t^2) + \epsilon)2^d$.

In order to prove the bound on $|T|$, we go along the same lines of the proof of Lemma 21. The bound on the size of $T_{j,u}$ is done differently: we decompose each string $y \circ r \in T_{j,u}$ into four parts: y_1 – the first $S^i + 2\log t + 1$ bits of y output during the i-th phase, y_2 – the rest of the bits in y, r_1 – the u bits of the seed read during the i-th phase, and r_2 – the rest of the seed bits. We define $T^1_{j,u} = \{y_1 \circ r_1 \mid y \circ r \in T_{j,u}\}$ and $T^2_{j,u} = \{y_2 \circ r_2 \mid y \circ r \in T_{j,u}$. Note that $|T^1_{j,u}| \leq |B^i_u(c^i_j)|$ and that $|T^2_{j,u}| \leq 2^{m - S^i - 2\log t - 1 + d - u}$. Continuing the argument of Lemma 21, we obtain $|T| \leq 2^{m+d}/(2t^2)$, as desired.

An identical averaging argument shows that there exists at least one input $x \in \{0,1\}^n$ and at least one seed $r \in \{0,1\}^d$, such that r is i-good for x for all i (under the stronger definition of goodness). This implies that:

$$m = |E(x,r)| = \sum_{i=1}^{t} |y^i(x,r)| \leq \sum_{i=1}^{t} S^i + 2\log t = (t-1)S + 2t\log t$$

The theorem follows. ☐

4 Lower Bounds for Online Hashing

Since universal and almost universal hash functions imply strong extractors (Lemma 9 and Lemma 11) we can immediately deduce space lower bounds for online hashing from Theorem 22:

Theorem 23 *Let $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a 2^d-sized universal family of hash functions with $m \leq n - 4\log n$. Define $k = \lceil m + 4\log n \rceil$ and $t = \lceil n/(n-k) \rceil$. Then any one-pass algorithm A for H requires space*

$$S \geq \frac{m}{t-1} - 2(1 + \frac{1}{t-1})\log t$$

Proof. By Lemma 9, H can be viewed as a (k, ϵ) strong extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ for any $\epsilon \geq 2^{(m-k)/2-1}$. We choose $\epsilon = 2^{(m-k)/2-1}$ and obtain for our choice of k, $\epsilon \leq 1/(2n^2) \leq 1/(2t^2)$. Every one-pass algorithm A for H is effectively also a one-pass algorithm for E, because A gets $x \in \{0,1\}^n$ (the input) and $h \in \{0,1\}^d$ (the seed) on one-way inputs tapes and outputs $h(x)$ (the output) on a one-way output. We can thus apply Theorem 22, and obtain:

$$S \geq \frac{m}{t-1} - 2(1 + \frac{1}{t-1})\log t$$

☐

Theorem 24 *Let $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ be a 2^d-sized $(1+\epsilon^2)/2^m$-almost universal family of hash functions with $\epsilon < 1/2$ satisfying $\lceil m + 2\log(1/\epsilon)\rceil \leq n(1-\sqrt{2\epsilon})$. Define $k = \lceil m + 2\log(1/\epsilon)\rceil$ and $t = \lceil n/(n-k)\rceil$. Then any one-pass algorithm A for H requires space*

$$S \geq \frac{m}{t-1} - 2(1 + \frac{1}{t-1})\log t$$

Proof. The proof is almost identical to the previous one. By our choice of k, $\epsilon \geq 2^{(m-k)/2}$. Therefore, we can apply Lemma 11 to view H as a (k, ϵ) strong extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$. Since $k = \lceil m + 2\log(1/\epsilon)\rceil \leq n(1-\sqrt{2\epsilon})$, then $\epsilon \leq 1/(2t^2)$. Thus, we can apply the space lower bound for strong extractors, and obtain:

$$S \geq \frac{m}{t-1} - 2(1 + \frac{1}{t-1})\log t$$

□

Remark 25 *A somewhat strange aspect of Theorem 23 and Theorem 24 is that the bounds on S may deteriorate when the output size m increases. Such anomalies disappear when we consider stronger notions of hashing such as pair-wise independence: $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is a family of pair-wise independent hash functions if for every $x \neq x' \in \{0,1\}^n$, and for every $y, y' \in \{0,1\}^m$, $\Pr_{h\in H}(h(x) = y \wedge h(x') = y') = 1/2^{2m}$. Observe that for every $m' < m$, the family $H' = \{h' : \{0,1\}^n \to \{0,1\}^{m'}\}$, where $h'(x)$ is the m'-bit prefix of $h(x)$, is also a family of pair-wise independent hash functions. The space required for online evaluation of H' is a lower bound for the space required for online evaluation of H.*

5 Upper Bounds for Online Extractors and Online Hashing

We give several constructions of extractors that can be computed by one-pass algorithms in space that almost matches our lower bounds. These constructions cover many settings of the parameters (i.e. the input min-entropy, the output length and the seed length).

The simplest bounds on the online evaluation of extractors for "low" min-entropies (e.g., $k \leq n/2$) can be obtained by universal hashing and almost universal hashing. In the "right model" (e.g., that of branching programs), the universal hashing extractors may use space that is optimal up to an additive factor of $O(1)$! The disadvantage of these extractors is their large seed length. We therefore show a careful implementation of the extractors of Trevisan and of Raz *et al.* [Tre99, RRV99], that can be evaluated in almost optimal space. Note that these extractors may simultaneously have a large output and a relatively small seed length.

To match the lower bounds for "high" min-entropy extractors, we give a space-efficient implementation of random walks on expanders based on the ideas of [BGW99].

5.1 Upper Bound for Low Entropies

Universal Hashing The simplest bound on the online evaluation of extractors for "low" min-entropies (e.g., $k \leq n/2$) can be obtained by universal hashing.

Theorem 26 *For all integers $m < k < n$, and $\epsilon \geq 2^{(m-k)/2-1}$, there exists a (k,ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{d+m}$ that can be evaluated by a one-pass algorithm with space $m + O(\log n)$.*

Proof. Set $d = m \cdot n$ and interpret the seed r of E as an m by n Binary matrix M_r. The extractor E is defined by $E(x,y) = M_r \circ (M_r \cdot x)$. Since matrix multiplication is a family of universal hash functions, Lemma 9 implies that E is indeed a (k,ϵ)-extractor. The one-pass algorithm A that evaluates E starts by initializing m one-bit variables $y_1, \ldots, y_m$ to 0. These variables are to hold the m-bit value $(M_r \cdot x)$ by the end of the computation. A reads the bits of x one by one. After reading the i'th bit x_i (and storing it instead of x_{i-1}), A reads the column $M_r(i,\cdot)$. When $M_r(i,j)$ is read, A set $y_j \leftarrow y_j + (M_r(i,j) \cdot x_i)$, writes $M_r(i,j)$ to the output and erases it from the memory. Finally, after reading x and M_r completely, A writes $y_1, \ldots, y_m$ to the output. □

Remark 27 *Note that for $k \leq n/2$ (assuming that $\epsilon \leq 1/8$) the space of A is optimal up to the additive $O(\log n)$ factor. In fact, in the "right model" (e.g., of branching programs), A would only use space $m + O(1)$ that would be optimal up to an additive factor of $O(1)$.*

As mentioned above, the seed length of extractors that are based on hashing can be reduced using almost universal hashing.

Theorem 28 *For all integers $k < n$, and every $\epsilon > 0$, there exists a (k,ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{d+m}$, with $m = k - 2\log 1/\epsilon - O(1)$ and $d = O(m + \log n/\epsilon)$ that can be evaluated by a one-pass algorithm with space $O(d)$.*

Proof. The extractor E, required to prove the theorem is defined by $E(x,h) = h \circ h(x)$, where h is selected from a $((1+\epsilon^2)/2^m)$-almost universal family of hash functions, H. Lemma 11, implies that E is indeed a (k,ϵ)-extractor. It remains to show that there exists such a family of hash functions such that: (1) Each function is defined by a d-bit key with $d = O(m + \log n/\epsilon)$. (2) The family can be evaluated by a one-pass algorithm with space $O(d)$. Many such families are already known in the literature. For completeness, we sketch the definition of a family that is based on the "evaluation" functions of Shoup [Sho96]:

The family $H = \{h : \{0,1\}^n \to \{0,1\}^m\}$ is defined using two families $H' = \{h' : \{0,1\}^n \to \{0,1\}^{m'}\}$ and $H'' = \{h'' : \{0,1\}^{m'} \to \{0,1\}^m\}$. The family H contains all the functions h that are composed of two functions $h' \in H'$ and $h'' \in H''$ (that is $\forall x \in \{0,1\}^n, h(x) = h''(h'(x))$).

The family H'' is an (essentially arbitrary) family of universal family of hash functions. For example, we can take the Toeplitz family. To conclude that H is $((1+\epsilon^2)/2^m)$-almost universal it is enough that H' is $(\epsilon^2/2^m)$-almost universal. For example, using the "evaluation" functions of Shoup [Sho96] we can define $h'(x)$ for any $x \in \{0,1\}^n$ as follows: First interpret x as a degree $n-1$ polynomial p_x over $GF(2^{m'})$. The m'-bit description of h' is interpreted as a value $\alpha = \alpha(h') \in GF(2^{m'})$. The output of h' on x is $\alpha \cdot p_x(\alpha) \in GF(2^{m'})$. By setting $m' = m + \log n + 2\log(1/\epsilon)$ we obtain the desired properties of H'.

A one-pass algorithm for H works as follows. Given $h = h'' \circ h'$ and x, the algorithm reads h and stores it in memory ($O(m'+m)$ bits). It also allocates two m'-bit variables: (1) a variable a to store powers of $\alpha = \alpha(h')$; (2) a variable p to store the evaluation $p_x(\alpha)$. a is initially set to 1 and p is set to 0. The algorithm then starts to read the bits of x one by one. When reading x_i, if it is 1, the algorithm sets $p \leftarrow p + a$, where the addition is in the field $GF(2^{m'})$. The algorithm then sets $a \leftarrow a \cdot \alpha$, where again the multiplication is in $GF(2^{m'})$. When done reading x, p contains $p_x(\alpha)$. The algorithm can then compute $\alpha \cdot p_x(\alpha)$ and $h''(\alpha \cdot p_x(\alpha))$. Note that the evaluation of h'' requires only $O(m')$ space because its domain is of size m'. The total space used by this algorithm is $O(m') = O(m + \log(n/\epsilon))$. □

Trevisan/RRV Extractors The disadvantage of the hashing based extractors defined above is that their seed length is rather large. Here we show that Trevisan's extractors [Tre99] and their extensions by Raz, Reingold, and Vadhan [RRV99] can also be computed by a one-pass algorithm with space that is close to match our upper bounds. More specifically, we show that the extractors of [RRV99] using the weak designs of Hartman and Raz [HR00], can be computed by a one-pass algorithm with space $m + O(d)$. For $k \leq n/2$, this matches the lower bound up to an additive factor of $O(d)$. We note that these extractors can extract the entire entropy of the source using $poly(\log(n/\epsilon))$ bits. We also note that these extractors are *strong extractors*. This is true for all the extractors used for upper bounds in this paper (that is, the extractors based on hashing described above and the extractors for high min-entropy described below).

Trevisan's construction is based on error-correcting codes and *designs*. Raz, Reingold and Vadhan [RRV99] improve Trevisan's construction for some setting of the parameters by using *weak designs* instead of designs. A family of m subsets $S_1, \ldots, S_m \subseteq [d]$ is called a weak (ℓ, ρ)-design, if for all i, $|S_i| = \ell$, and $\sum_{j<i} 2^{|S_i \cap S_j|} \leq \rho(m-1)$. We will use the explicit family of weak designs of Hartman and Raz [HR00]; for these designs there is an $O(\log m)$-space algorithm that given an index $1 \leq i \leq m$, outputs S_i.

Given n, $\epsilon > 0$, $k \leq n$, and $m \leq k/2$, the Trevisan/RRV construction uses any efficiently encodable $(\overline{n}, n, (\frac{1}{2} + \frac{\epsilon}{m})\overline{n})$ binary code $\mathcal{C}$ with $\overline{n} = \text{poly}(n/\epsilon)$ and a $(\log \overline{n}, (k - O(\log(m/\epsilon) + d))/m)$ weak design $S_1, \ldots, S_m$ with $d = O(\log^2(n/\epsilon)\log k)$. For an input $x \in \{0,1\}^n$ and a seed $r \in \{0,1\}^d$, the value of the extractor is $E(x,r) = (\overline{x}(r|_{S_1}), \ldots, \overline{x}(r|_{S_m}))$, where $\overline{x}$ is the encoding of x under $\mathcal{C}$, $\overline{x}(j)$ is the j-th coordinate of $\overline{x}$, and $r|_{S_i}$ is the projection of r on the coordinates designated by S_i.

We will show that if the binary code used in the construction is the concatenation of Reed-Solomon and Hadamard and the weak designs are those of Hartman-Raz, then the Trevisan/RRV extractor can be computed space efficiently in one pass. A similar (slightly weaker) construction was suggested by D. Sivakumar [Siv01].

Theorem 29 *The Trevisan/RRV construction with concatenated Reed-Solomon and Hadamard code and the Hartman-Raz weak designs is computable by an $m + O(d)$ space one-pass algorithm.*

Note that for $k \le n/2$, this construction is optimal up to an additive factor of $O(d)$.

Proof. Define $\gamma \stackrel{\text{def}}{=} \epsilon/m$, and let $\overline{n} \ge \Omega(n^2/(\gamma^2 \log^2 n))$ be a power of 2. Let $q \stackrel{\text{def}}{=} \sqrt{\overline{n}}$. The code we use is a concatenation of a $(q, 2\gamma q+1, q(1-2\gamma))_q$ Reed-Solomon code with a $(q, \log q, q/2)_2$ Hadamard code. Every input x is associated with a polynomial of degree $h \stackrel{\text{def}}{=} n/(\log q) - 1$ over $\mathbb{F}_q$. Let us denote by $x_0, \ldots, x_h \in \mathbb{F}_q$ the coefficients of this polynomial. Every coordinate of the codeword $\overline{x}$ is associated with a pair (a, β) where $a \in \mathbb{F}_q$ and $\beta \in \{0,1\}^{\log q}$. $\overline{x}(a,\beta) = \langle \sum_{j=0}^{h} x_j a^j, \beta \rangle$, where the summation and the product are in the field $\mathbb{F}_q$ and the inner product is modulo 2. Since q is a power of 2, the summation in $\mathbb{F}_q$ is equivalent to bitwise xor in $\{0,1\}^{\log q}$. This implies that $\langle \sum_{j=0}^{h} x_j a^j, \beta \rangle = \sum_{j=0}^{h} \langle x_j a^j, \beta \rangle$.

The algorithm starts by initializing m one-bit variables $y_1, \ldots, y_m$ to 0. These variables are to hold the m-bit output of the extractor by the end of the computation.

The algorithm then reads the seed r and stores it in memory. At any given point, our algorithm can use the Hartman-Raz algorithm to compute in space $O(\log m)$ the projection $r|_{S_i}$ for any given $i \in \{1, \ldots, m\}$. This projection specifies a pair (a_i, β_i), which determines the coordinate of $\overline{x}$ that is supposed to constitute the i-th bit of the output.

The algorithm reads the coefficients of the polynomial x one by one. Upon reading x_j, it stores it in memory ($O(\log q)$ bits), and enumerates on $i = 1, \ldots, m$. For each such i, the algorithm computes (a_i, β_i) from r, calculates $\langle x_j a_i^j, \beta_i \rangle$ and adds the result to y_i (modulo 2). Note that the computation can be carried out in $O(\log m + \log d + \log q)$ space.

In the end of the computation y_i contains $\sum_{j=0}^{h} \langle x_j a_i^j, \beta_i \rangle$, which is exactly the i-th bit of the extractor's output. The total space used is m (for storing $y_1, \ldots, y_m$) plus $O(d + \log m + \log d + \log q) = O(d)$ (for storing the seed and for the computations at each iteration). $\square$

5.2 Upper Bound for High Entropies

For the case where both k and m are large, our upper bound is obtained using the Goldreich–Wigderson construction [GW94] which is based on the Ajtai-Komlós-Szemerédi expander ideas [AKS87]. We then discuss an upper bound for smaller output lengths using a watered down variant of [GW94] (in particular, this variant does not employ expander graphs). In both cases we do not attempt to minimize the seed length (which can be made much shorter than what we state in the constructions).

Fix n, $\epsilon > 0$, and $\frac{c}{c+1}n + (8+c)\log(1/\epsilon) \le k \le n$, where c is some universal constant (in our construction, $c \approx 34$). Set $m = (c+1)k - cn - (8+c)\log(1/\epsilon)$.

We use a D-regular expander graph G on 2^m nodes, whose second eigenvalue (i.e., the second largest eigenvalue of its adjcancy matrix) λ satisfies $\lambda/D \le \epsilon^2/(4 \cdot 2^{(n-k)/2})$. We obtain G by taking a power $p = O(n-k) + O(\log(1/\epsilon))$ of the Margulis-Gabber-Galil expander graph [Mar73, GG81, JM87]. The degree D will thus be $D = 2^{O(n-k)+O(\log(1/\epsilon))}$.

We further use a Toeplitz 2-universal family of hash functions (cf., [Gol97]), $H = \{h : \{0,1\}^{n-m} \to \{1, \ldots, D\}\}$. Each function in the family is represented by $n - m + 2\log D$ bits, and the functions are logspace computable.

Each input $x \in \{0,1\}^n$ of the extractor corresponds to a pair (u, i) where $u \in \{0,1\}^m$ is a node in G and $i \in \{0,1\}^{n-m}$. Each seed corresponds to a function $h \in H$. $E((u,i),h)$ is defined to be the $h(i)$-th neighbor of u in G. The length of the seed is thus $d = O(n-k) + O(\log(1/\epsilon))$.

We next present a space-efficient one-pass algorithm for E:

Theorem 30 *The Goldreich–Wigderson extractor is computable by an $O(n-k)+O(\log(1/\epsilon))$ space one-pass algorithm.*

It is easy to verify that for the setting of parameters in this theorem, $O(n-k) = O(m/t)$. Therefore, the space used by this algorithm is optimal up to a constant factor.

Proof. The algorithm we present is based on the space-efficient one-pass algorithm of Bar-Yossef *et al.* [BGW99] for computing neighborhoods in the Margulis-Gabber-Galil expander.

We encode each input pair (u, i) as follows: we first put the index i and then the node u. The algorithm starts by reading i and the seed h and stores them in memory. Note that $|i| + |h| = O(n-k) + O(\log(1/\epsilon))$. It then computes $h(i)$ (with only $O(\log(n-k)) + O(\log\log(1/\epsilon))$ space) and stores it in memory.

We now apply the one-pass algorithm of [BGW99] that computes neighborhoods in (powers of) the Margulis-Gabber-Galil expander graph using small space. Specifically, for computing a neighbor in the p-th power of the graph, their algorithm uses $O(p)$ space. In our case $p = O(n-k)+O(\log(1/\epsilon))$, and therefore the total space used by our algorithm is $O(n-k) + O(\log(1/\epsilon))$. □

Upper bound for smaller m The theorem above shows that the Goldreich–Wigderson extractor can be computed with optimal space (up to a constant factor). In this construction the output length m is $\Omega(n)$. Can we match the lower bound even for smaller m (which should imply smaller space)? As it turns out, a space-efficient construction with smaller length is even easier to obtain.

One of the main observations of [GW94] is that any high min-entropy source is also a "block source" (as defined by Chor and Goldreich [CG88]). When the source is divided into blocks of length a then each one of these values contains roughly $a - (n-k)$ bits of "independent" randomness.[2] Extracting randomness from block sources is much easier than from general sources [NZ96] – since the randomness in each one of the blocks is independent, it can also be extracted independently. This can be used for the following simplified variant of [GW94] (that does not use expanders and has a smaller output length):

On input $x \in \{0,1\}^n$ and seed $r \in \{0,1\}^d$, the $(k, O(\epsilon))$ extractor E first divides the input into blocks $x = x_1 \circ x_2 \circ \ldots x_{t'}$, where each block is of length $a = O((n-k) + \log n/\epsilon)$ (we assume wlog. that all of the blocks are of equal length). The output of E is now defined as:

$$E(x,r) \stackrel{\text{def}}{=} r \circ E'(x_1, r) \circ E'(x_2, r) \circ \ldots \circ E'(x_{t'}, r),$$

[2] The notion of a block source is is formally defined in Section 6. In fact the construction sketched here is very related to the construction of dispersers in Section 6.

where $E' : \{0,1\}^a \times \{0,1\}^d \to \{0,1\}^{m/t'}$ is an $(a - (n-k) - O(\log n/\epsilon), \epsilon)$ strong extractor. Note that the space needed for online computation of E is roughly the one needed for online computation of E'. Therefore the space needed to evaluate E can be roughly $m/t' + O(\log(n/\epsilon))$, by taking E' to be one of the space-efficient extractors for small min-entropy.

6 Upper Bound for Online Dispersers

In this section we construct online dispersers which beat the lower bounds we give in Section 3 on online extractors. Recall that when $k < n/2$ any online extractor has to have space roughly $m - d$. The space in our disperser construction can be arbitrarily smaller (roughly m/t for any t). However, to achieve such small space we use seeds of length $O(t \log n)$.

The idea of the construction is best explained if we assume that the source is a *bit-fixing source*. By that we mean that $n - k$ of the bits are fixed, and the remaining k bits are uniformly distributed. The first kind of bits are called "bad bits" and the second kind "good bits". For a parameter t, let $1 = i_0 < i_1 < \cdots < i_{t-1} < i_t = n+1$ be indices such that in any interval $[i_j, i_{j+1})$ there are exactly k/t good bits. If we know such i's we can easily extract many bits online using small space. We simply run an online extractor on each one of the blocks and concatenate the outputs. Each application of an online extractor outputs only m/t bits, (where m is the total number of bits we output). Thus, the space used by any application of the online extractor can be about m/t. We can reuse this space for the next application and thus the total space of this construction is about m/t. By increasing t we can arbitrarily reduce the space.

In practice, we do not know the indices that yield a partition with k/t good bits in each block. We therefore add more bits to the seed and use them to choose random $t-1$ indices. The rational is that we will hit a "correct" choice of indices with positive probability, and when that happens we extract randomness from the source. It follows that our construction is a disperser as we hit almost all outputs when choosing "correct" indices.

To extend the argument above from bit-fixing sources to general sources we use methods from [NZ96, SSZ98, Ta-96, Ta-98] to argue that any (general) source "contains" a sub-source on which it resembles a bit fixing source in the sense that the construction outlined above works for such a sub-source. Many previous extractors and dispersers constructions [NZ96, SZ99, SSZ98, Ta-96, Zuc97, Ta-98, RSW00] take this route and start by "partitioning" a source into a block source.

6.1 Statement of Results

We will use the following notation. Given a distribution P and an event A with $P(A) > 0$ we define the distribution $(P|A)$ to be the distribution P conditioned on A, that is:

$$(P|A)(x) = \begin{cases} \frac{P(x)}{P(A)} & x \in A \\ 0 & x \notin A \end{cases}$$

In the following definition we consider a variant of an extractor which is only required to work correctly for some subset of its input.

Definition 31 (Sub-source extractor) *An (n, d, m) function E is called a (k, ϵ)-sub-source extractor, if for every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$ there are subsets $X' \subseteq \{0,1\}^n$ and $R' \subseteq \{0,1\}^d$ such that $\|E((X|X'), (U_d|R')) - U_m\| < \epsilon$.*

A sub-source extractor is not necessarily an extractor. For example it may be the case that some element in $\{0,1\}^m$ gets a very large probability under $E(X, U_d)$. However, a sub-source extractor is a disperser as no set of size $\epsilon 2^m$ is missed when conditioning on X' and R', and removing this conditioning can only decrease the size of the missed sets. We conclude that:

Lemma 32 *Every (k, ϵ)-sub-source extractor is a (k, ϵ)-disperser.*

The main theorem of this section constructs online sub-source extractors which beat the lower bounds for online extractors proved in Section 3.

Theorem 33 *Given any $(\bar{k}, \bar{\epsilon})$-extractor $\bar{E} : \{0,1\}^{\bar{n}} \times \{0,1\}^{\bar{d}} \to \{0,1\}^{\bar{m}}$ that can be evaluated by a one-pass algorithm with space $\bar{s}$ and for every integer t, there exists a (k, ϵ)-sub-source extractor $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with $k = t(\bar{k} + 2\log n + O(1))$, $\epsilon = t\bar{\epsilon}$, $n = \bar{n}$, $d = (t-1)\log n + t\bar{d}$ and $m = t\bar{m}$. Furthermore, E can be evaluated by a one-pass algorithm with space $s = \bar{s} + O(\log n)$.*

The following corollary follows by plugging in the upper bound from Theorem 29.

Corollary 34 *For every n, k, ϵ and integer t, there exists a disperser $E : \{0,1\}^n \times \{0,1\}^{O(t \log^2(n/\epsilon) \log k)} \to \{0,1\}^m$ for $m = k - O(t \log(n/\epsilon))$. Furthermore, E can be evaluated by a one-pass algorithm with space $s = m/t + O(\log n)$.*

Remark 35 *Using the improvements suggested in Remark 42 we can reduce d to $(t-1)\log n + O(\log^2(n/\epsilon) \log k)$ at the cost of enlarging s to $m/t + O(\log^2(n/\epsilon) \log k)$.*

In the remainder of this section we describe the construction, and prove its correctness in Lemma 39. For the analysis, we will need the concept of block sources.

6.2 Block Sources

Definition 36 (Block sources [CG88]) *A $(k_1, \cdots, k_t)$-block source is a distribution $\bar{X} = (X_1, \cdots, X_t)$, where $X_j \in \{0,1\}^{n_j}$ and for every $j \in \{1, \cdots, t\}$ and $x_{j+1} \in \{0,1\}^{n_{j+1}}, \cdots, x_t \in \{0,1\}^{n_t}$:*

$$H_\infty(X_j \mid X_{j+1} = x_{j+1}, \cdots, X_t = x_t) \geq k_t$$

A $(t; k)$-block-source is a $(k, k, \cdots, k)$ (t times) block source.

We call each X_j a *block*. Intuitively this definition means that each X_j contains k bits of randomness which are not contained in any of the other blocks. In particular, any $(t;k)$-block source has tk bits of min-entropy.

Given a string $x \in \{0,1\}^n$ we use the notation $x_{[i,j]}$ for $i < j$ to denote the sequence $x_i, x_{i+1}, \cdots, x_{j-1}$.

Definition 37 *For a distribution X on $\{0,1\}^n$, we say that $i_1 < i_2 < \cdots < i_{t-1}$* partition *$X$ into a $(t;k)$-block source if the distribution $X_{[1,i_1]}, X_{[i_1,i_2]}, X_{[i_2,i_3]}, \cdots, X_{[i_{t-1},n+1]}$ is a $(t;k)$-block source.*

The following lemma shows that every source of min-entropy k has a sub-source which can be partitioned into a block source, where each block has min-entropy which is not much smaller than k/t. Such lemmas are key components in some extractor and disperser constructions [NZ96, SZ99, SSZ98, Ta-96, Zuc97, Ta-98, RSW00] and we use similar techniques in the proof of the lemma.

Lemma 38 *For every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$ and every integer $t = O(k/\log n)$ there is a subset $X' \subseteq \{0,1\}^n$ and indices $i_1, \cdots, i_{t-1} \in \{1, \cdots, n\}$ which partition $(X|X')$ into a $(t;k')$-block source, for $k' > k/t - 2\log n - 3$*

It should be noted that whereas most of the previous work [NZ96, SSZ98, Ta-96, Ta-98] concentrated on $t = O(\log n)$, we are also interested in larger t's, and the lemma is applicable for $t = O(k/\log n)$. We prove Lemma 38 in Section 6.5

6.3 The Construction

We now show how to construct an online (k, ϵ)-sub-source extractor which uses small space. The construction is given an integer t as a parameter, and let k' be as in Lemma 38. Our construction will use as an ingredient a $(\bar{k}, \bar{\epsilon})$-extractor $\bar{E} : \{0,1\}^{\bar{n}} \times \{0,1\}^{\bar{d}} \to \{0,1\}^{\bar{m}}$,(for $\bar{k} = k'$ and $\bar{n} = n$) that can be evaluated by a one-pass algorithm with space $\bar{s}$. (We will later plug-in one of our extractor constructions from Section 5 to obtain a concrete construction).

Algorithm `sub-source extractor`
Input:

- source element: $x \in \{0,1\}^n$
- seed: $r \in \{0,1\}^d$ for $d = (t-1)\log n + t\bar{d}$. We choose to interpret this r as $\ell_1, \cdots, \ell_{t-1} \in \{0,1\}^{\log n}$ and $y_1, \cdots, y_t \in \{0,1\}^{\bar{d}}$ where the order in which these strings are given is

  $$\ell_1, y_1, \ell_2, y_2, \cdots, \ell_{t-1}, y_{t-1}, y_t$$

  We define $\ell_0 = 1$ and $\ell_t = n+1$

Output: $z_1, \cdots, z_t \in \{0,1\}^{\bar{m}}$, where for every j, $z_j = \bar{E}(x_{[l_{j-1}, l_j]}, y_j)$.[3] (If for some j, $l_{j-1} \geq l_j$, the output z_j is fixed arbitrarily).

We now show that this computation can be carried out by a one-pass algorithm with space $\bar{s} + O(\log n)$. The algorithm maintains a counter $j \in \{1, \cdots, t\}$. It also uses variables $p, \ell \in \{0,1\}^{\log n}$ which we think of as representing numbers in $\{1, \cdots, n\}$ for temporary storage. The algorithm has additional $\bar{s}$ bits of memory which it uses to run $\bar{E}$.

Online implementation:
Set $p \leftarrow 1$, $j \leftarrow 0$. While $j < t$ the algorithm performs the following steps:

- $j \leftarrow j + 1$
- If $j = t$ set $\ell = n + 1$, otherwise read $\log n$ bits from the seed (representing ℓ_j) and store them in ℓ.
- If $\ell \leq p$, pad the current output with arbitrary values to get an m-bit output, and halt.
- Run $\bar{E}$ reading $\ell - p$ bits from the source (representing $X_{[\ell_{j-1}, \ell_j]}$) and $\bar{d}$ bits from the seed (representing y_j). Whenever $\bar{E}$ outputs a bit, we immediately output it.
- $p \leftarrow \ell$

The reader can verify that this algorithm computes $z_1, \cdots, z_t$ defined above when $\ell_0 < \ell_1 < \cdots < \ell_t$.

6.4 The Analysis

We now show that E is a $(k, t\bar{\epsilon})$-sub-source extractor. For any distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$, let X' and $i_1, \cdots, i_{t-1}$ be as in Lemma 38. Let $R' = \{\ell_1, \cdots, \ell_{t-1}; y_1, \cdots, y_t \mid \ell_1 = i_1, \cdots, \ell_{t-1} = i_{t-1},\ y_1, \ldots, y_t \in \{0,1\}^{\bar{d}}\}$. R' is a subset of $\{0,1\}^d$.

Lemma 39 $\|E((X|X'), (U_d|R')) - U_m\| < t\bar{\epsilon}$

Let Z_j denote the distribution over z_j's when x is chosen according to $(X|X')$ and r is chosen according to $(U_d|R')$. Lemma 39 follows from the following lemma when $j = t$.

Lemma 40 *For every $j \in \{0, \ldots, t\}$ and for every $z_{j+1}, \cdots, z_t \in \{0,1\}^{\bar{m}}$,*

$$\|(Z_1, \cdots, Z_j \mid Z_{j+1} = z_{j+1}, \cdots, Z_t = z_t) - U_{j\bar{m}}\| < j\bar{\epsilon}$$

[3]Formally, we cannot use $\bar{E}$ on strings x of length smaller than n. In such a case we assume that the too short x is padded with zeroes. Note that this can be done online, and that if x is chosen from a source with k bits of min-entropy, the extractor will extract randomness from the source.

Proof. We prove the lemma using induction on j. The statement trivially holds for $j = 0$, because both distributions are distributions on a singleton set (the set that contains the empty string). Therefore, the distance between the two distributions is 0.

Assume, then, that the claim holds for some j; we will show it also holds for $j+1$. Let $z_{j+2}, \cdots, z_t \in \{0,1\}^{\bar{m}}$ be arbitrary fixings. We will use the following easy fact.

Fact 41 *Let P and P' be distributions over a domain S, and let Q and $\{Q'_s\}_{s\in S}$ be distributions over a domain T. Let $Q'_{P'}$ denote the distribution in which an element $t \in T$ is chosen by choosing s according to P' and choosing t according to Q'_s. The following inequality holds:*

$$\|(P,Q) - (P', Q'_{P'})\| \leq \|P - P'\| + max_{s\in S}\|Q - Q'_s\|$$

(in $(P', Q'_{P'})$ the two occurrences of P' refer to the same variable).

In our setup we take P' to be the distribution $(Z_{j+1} \mid Z_{j+2} = z_{j+2}, \cdots, Z_t = z_t)$ and P is $U_{\bar{m}}$, thus the domain S is $\{0,1\}^{\bar{m}}$. For every $z \in \bar{m}$ we take Q'_z to be the distribution

$$(Z_1, \cdots, Z_j \mid Z_{j+1} = z, Z_{j+2} = z_{j+2}, \cdots, Z_t = z_t)$$

The distribution Q is $U_{j\bar{m}}$. The lemma now follows from the following two claims since the distribution $(Q'_{P'}, P')$ is the same distribution as $(Z_1, \cdots, Z_{j+1} \mid Z_{j+2} = z_{j+2}, \cdots, Z_t = z_t)$.

Claim 1 $\|P - P'\| < \bar{\epsilon}$

Claim 2 *For every $z \in \{0,1\}^{\bar{m}}$, $\|Q - Q'_z\| < j\bar{\epsilon}$.*

The first claim follows because for every $z_{j+2}, \cdots, z_t$, let $x_{j+2}, \cdots, x_t$ and $y_{j+2}, \cdots, y_t$ be arbitrary values such that for $j + 2 \leq a \leq t$, $\bar{E}(x_a, y_a) = z_a$. We have that:

$$\begin{aligned}
&H_\infty(X_{[i_j,i_{j+1}]} \mid X', Z_{j+2} = z_{j+2}, \cdots, Z_t = z_t) \geq \\
&\quad H_\infty(X_{[i_j,i_{j+1}]} \mid X', X_{[i_{j+1},i_{j+2}]} = x_{j+2}, Y_{j+2} = y_{j+2}, \cdots, X_{[i_{t-1},n+1]} = x_t, Y_t = y_t) = \\
&\quad H_\infty(X_{[i_j,i_{j+1}]} \mid X', X_{[i_{j+1},i_{j+2}]} = x_{j+2}, \cdots, X_{[i_{t-1},n+1]} = x_t) \ \geq \ k'
\end{aligned}$$

The equality follows because $(X|X')$ is independent of $(Y_{j+2}, \cdots, Y_t)$. The last inequality follows because $(X|X')$ is a block source. Since $\bar{E}$ is a $(k', \bar{\epsilon})$-extractor we have that:

$$\|(Z_{j+1} \mid Z_{j+2} = z_{j+2}, \cdots, Z_t = z_t) - U_{\bar{m}}\| < \bar{\epsilon}$$

The second claim follows from the induction hypothesis. □

Remark 42 *Our construction gives a tradeoff between the seed length and space. (We get roughly $sd \leq m \log^{O(1)} n$). Suppose we fix s; can we reduce d below m/s? While we do not know the answer to this question in general, we can reduce the seed in the following ways:*

Instead of including $y_1, \cdots, y_t$ we can include only y_1 in the seed, and take each y_j for $j > 1$ from z_j. In fact, this is the standard way in which block-sources are used: One extracts bits from the first block and uses (some of) them to extract bits from the next block. The proof will go on almost unchanged, with the exception that the seed we are using in step j is only close to uniform. Still, this error is only added to the final error. We get $d = \bar{d} + (t-1)\log n$. This adds $O(\bar{d})$ bits to the space as we have to store $\bar{d}$ bits of the output of step j, to use them as seed for the extractor of step $j+1$. An alternative solution to this problem is to require that $\bar{E}$ is a strong extractor and use the same y_1 for all blocks.

To further reduce the seed we will have to use less random bits to choose the partition. In [SSZ98, Ta-98] it was shown that any source with min-entropy $n^{\Omega(1)}$ can be partitioned into a $(\Theta(\log n); n^{\Omega(1)})$-block source by a family of only $n^{O(1)}$ different partition. (Note that there are $n^{\Omega(\log n)}$ different partitions of n bits into $\Theta(\log n)$ blocks). However, the methods of [SSZ98, Ta-98] do not work when $t >> \log n$, and we need to split into $t >> \log n$ blocks to get $sd = o(m)$.

Another natural strategy is to use randomness from the source to get $\ell_2, \cdots, \ell_{t-1}$ in a way similar to our first suggestion. This does not seem to work as each such ℓ_j can be only guaranteed to be close to uniform, and this does not suffice for our proof.

6.5 Proof of Partitioning Lemma

Partitioning a general source into a block source is a common strategy to construct extractors and dispersers [NZ96, SZ99, SSZ98, Ta-96, Zuc97, Ta-98, RSW00]. We are interested in "large" t, and do not care if X' is small. We prove Lemma 38 by tailoring techniques from previous work to this setup.

We need the following definition.

Definition 43 *Let X and X' be subsets of $\{0,1\}^n$ such that $X' \subseteq X$. We call X' an i-*projection *of X if there is a set $A \subseteq \{0,1\}^{n+1-i}$ such that $X' = \{x \in X | x_{[i,n+1]} \in A\}$. We will say that A* defines *X'* in *X.*

If X' is an i-projection of X, then if we already know that $x \in X$, whether or not $x \in X'$ only depends on the last $n+1-i$ indices of x.

We start by proving Lemma 38 for two blocks. We will use the following lemma recursively to partition to many blocks.

Lemma 44 *For every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$ and every integer $u < k - 2\log n - 3$, there is a subset $X' \subseteq \{0,1\}^n$ and index $i \in \{1, \cdots, n\}$, such that:*

- *$(X_{[1,i]}, X_{[i,n+1]} | X')$ is a $(u, k - u - 2\log n - 3)$-block source.*
- *X' is an i-projection of X. (Here we use X to also denote the support of the distribution X).*

Proof. For a string $x \in \{0,1\}^n$, we define $P_i^X(x) = \Pr_X(X_{[i,n+1]} = x_{[i,n+1]})$. We refer to this quantity as *the probability of x at i*. Our first step is to identify an i and a large subset of X for which all strings have $P_i^X(x) \approx 2^{-(k-u)}$.

Claim 3 *There is an index i and a set $X' \subseteq X$ such that X' is an i-projection of X, $\Pr_X(X') \geq 1/2n$ and for every $x \in X'$, $2^{-(k-u)} < P_i^X(x) \leq 2^{-(k-u-\log n-2)}$.*

Proof. (of claim) Note that for every $x \in X$, the probability of x at 1 is at most 2^{-k}. We now define:

$$\begin{aligned} S_i &= \{x \in X \mid \ i \text{ is the largest index such that } P_i^X(x) \leq 2^{-(k-u-\log n-2)}\} \\ &= \{x \in X \mid P_i^X(x) \leq 2^{-(k-u-\log n-2)}, P_{i+1}^X(x) > 2^{-(k-u-\log n-2)}\} \end{aligned}$$

The sets $\{S_i\}_{i\in\{1,\cdots,n\}}$ partition X into n disjoint subsets. It follows that there exists an i such that $\Pr_X(S_i) \geq 1/n$. We define:

$$B = \{x \in S_i \mid P_i^X(x) \leq 2^{-(k-u)}\}$$

Let $X' = S_i \setminus B$. Note that $P_i^X(x)$ and $P_{i+1}^X(x)$ depend only on $x_{[i,n+1]}$ and therefore S_i, B and X' are i-projections of X. We show that $\Pr_X(X') \geq 1/2n$ by showing that $\Pr_X(B) < 1/2n$. Intuitively strings in B have low probability because their probability "jumped" from above $2^{-(k-u-\log n-2)}$ at $i+1$ to below $2^{-(k-u)}$ at i. Formally, let A_B be the subset of $\{0,1\}^{n-i+1}$ that defines B in X, and for each $a \in A_B$, let $x^a \in B$ be a "representative" for which $x^a_{[i,n+1]} = a$. Then,

$$\begin{aligned} \Pr_X(B) &= \sum_{x\in B} \Pr_X(X = x) \ = \sum_{a\in A_B} \Pr_X(X_{[i,n+1]} = a) \ = \sum_{a\in A_B} \Pr(X_{[i,n+1]} = x^a_{[i,n+1]}) \\ &= \sum_{a\in A_B} \Pr_X(X_i = x^a_i \mid X_{[i+1,n+1]} = x^a_{[i+1,n+1]}) \cdot \Pr_X(X_{[i+1,n+1]} = x^a_{[i+1,n+1]}) \\ &= \sum_{a\in A_B} \frac{P_i^X(x^a)}{P_{i+1}^X(x^a)} \cdot \Pr_X(X_{[i+1,n+1]} = x^a_{[i+1,n+1]}) \\ &< \frac{2^{-(k-u)}}{2^{-(k-u-\log n-2)}} \sum_{a\in A_B} \Pr_X(X_{[i+1,n+1]} = x^a_{[i+1,n+1]}) \ \leq \ \frac{1}{4n} \cdot 2 = \frac{1}{2n} \end{aligned}$$

where the last inequality follows from the fact that each suffix $[i+1, n+1]$ is shared by at most two a's in A_B. □

To conclude the proof of the lemma we need to compute the min-entropy of $((X_{[1,i]} \mid X') \mid X_{[i,n+1]} = a)$ for every a and of $(X_{[i,n+1]} \mid X')$. Note that for every distribution X and events E_1, E_2, $((X|E_1)|E_2) = (X|E_1, E_2)$.

Claim 4 *Let A' be the set that defines X' in X. For every $a \in A'$, $H_\infty((X_{[1,i]}|X') \mid X_{[i,n+1]} = a) \geq u$.*

Proof. (of claim) For every $w \in \{0,1\}^{i-1}$

$$\Pr_X(X_{[1,i]} = w \mid X', X_{[i,n+1]} = a) = \frac{\Pr_X(X_{[1,i]} = w, X', X_{[i,n+1]} = a]}{\Pr_X[X', X_{[i,n+1]} = a)}$$

Since A' defines X' in X and $a \in A'$ then given that $X_{[i,n+1]} = a$ we already know that $X \in X'$ we thus have:

$$= \frac{\Pr_X[X_{[1,i]} = w, X_{[i,n+1]} = a]}{\Pr_X[X_{[i,n+1]} = a]} = \frac{\Pr_X[X = (w \circ a)]}{\Pr_X[X_{[i,n+1]} = a]} \le \frac{2^{-k}}{2^{-(k-u)}} = 2^{-u}$$

The inequality follows because $H_\infty(X) \ge k$ and because $a \in A'$. $\square$

Claim 5 *For every* $a \in A'$

$$\Pr_X[X_{[i,n+1]} = a|X'] = \frac{\Pr_X[X_{[i,n+1]} = a, X']}{\Pr_X[X']} = \frac{\Pr_X[X_{[i,n+1]} = a]}{\Pr_X[X']} \le \frac{2^{-(k-u-\log n-2)}}{1/2n} = 2^{-(k-u-2\log n-3)}$$

The lemma follows. $\square$

We now prove Lemma 38. We get the final partition by repeatedly partitioning the source into two blocks. In every step we partition the last block we generated. The argument is more subtle than one expects at first glance. Every time we partition the last block we also induce a new conditioning that affects all previous blocks. Thus, we have to verify that conditioning on a smaller set does not spoil the blocks we previously generated.

Proof. (of Lemma 38) We will use Lemma 44 repeatedly. At step r we have $i_1, \cdots, i_r$ and $X'_r \subseteq \{0,1\}^n$ such that $(X_{[1,i_1]}, X_{[i_1,i_2]}, \cdots, X_{[i_r,n+1]}|X'_r)$ is a $(u, \cdots, u, k - r(u + 2\log n + 3))$-block source.

For $r = 1$, this is the case after we applied Lemma 44 once on X, choosing $i_1 = i$ and $X'_1 = X'$.

We now apply Lemma 44 on $(X_{[i_r,n+1]}|X'_r)$ to obtain $i_r < i \le n$ and a subset X'. We want to view X' as a subset of $\{0,1\}^n$. To do that we define $X'_{r+1} = \{x \in X'_r \mid x_{[i_r,n+1]} \in X'\}$. We set $i_{r+1} = i$ and $X'_{r+1} = X'$. From Lemma 44 we have that X'_{r+1} is an i_{r+1}-projection of X'_r. It also follows from the lemma that $(X_{[i_r,i_{r+1}]}, X_{[i_{r+1},n+1]} \mid X'_{r+1})$ is a $(u, k - (r+1)(u + 2\log n + 3))$-block source. However, we are not done. We have to verify that conditioning on X'_{r+1} does not spoil the r previous blocks. We only know these blocks to be good when conditioning on X'_r. The crucial observation is that we measure the randomness in these blocks conditioned on values of the next blocks, and in particular conditioned on values of $X_{[i_{r+1},n+1]}$. We are interested only in values of $X_{[i_{r+1},n+1]}$ which are in the set A_{r+1} which defines X'_{r+1} in X'_r. Thus, having conditioned on these values we already know that we are in X'_{r+1} and the conditioning on X'_{r+1} rather than X'_r changes nothing.

More formally, consider an arbitrary block $X_{[i_j,i_{j+1}]}$ for $j < r$. We have that for $x \in X'_r$:

$$H_\infty(X_{[i_j,i_{j+1}]} \mid X'_r, X_{[i_{j+1},n+1]} = x_{[i_{j+1},n+1]}) \ge u$$

We would like this to hold also for X'_{r+1}. We know that X'_{r+1} is an i_{r+1}-projection of X'_r and thus, for $x \in X'_{r+1}$ the event $X'_r, X_{[i_{j+1},n+1]} = x_{[i_{j+1},n+1]}$ is equal to the event $X'_{r+1}, X_{[i_{j+1},n+1]} = x_{[i_{j+1},n+1]}$. We conclude that for every $x \in X'_{r+1}$:

$$H_\infty(X_{[i_j,i_{j+1}]} \mid X'_{r+1}, X_{[i_{j+1},n+1]} = x_{[i_{j+1},n+1]}) \ge u$$

At step $r = t-1$ we have $i_1, \cdots, i_{t-1}$ and $X'_{t-1} \subseteq \{0,1\}^n$ such that $(X_{[1,i_1]}, X_{[i_1,i_2]}, \cdots, X_{[i_{t-1},n+1]} | X'_{t-1})$ is a $(u, \cdots, u, k-(t-1)(u+2\log n+3))$-block source. Choosing $u = k/t - 2\log n - 3$ we get that $k-(t-1)(u+2\log n+3) \geq u$ and thus $i_1, \cdots, i_{t-1}$ partition $(X|X'_{t-1})$ into a $(t;u)$-block source. $\square$

7 Online Error-Correcting Codes

We prove space lower bounds for both online encoding and online decoding of error-correcting codes. We then show simple (almost) matching upper bounds.

7.1 Lower Bounds

Theorem 45 *Let $\mathcal{C}$ be an $(n,k,d)_q$-ECC with encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$. Then any one-pass algorithm A for E requires space*

$$S \geq k \cdot \frac{d}{n} \cdot \log q - \log(k \log q + 1)$$

The intuition behind the proof is as follows: we divide each codeword into n/d blocks of size d. We iteratively construct a chain of codeword sets $\mathcal{C} = F_0 \supseteq F_1 \supseteq \cdots \supseteq F_{n/d}$, such that all the codewords in F_i share their last i blocks and such that $|F_i|/|F_{i+1}| \leq (k+1)2^S$. It follows that $|F_{n/d}| \geq |\mathcal{C}|/((k+1)2^S)^{n/d} = 2^{k-(S+\log(k+1))(n/d)}$. On the other hand, necessarily $|F_{n/d}| \leq 1$, implying the lower bound.

The construction of the chain works by inductively extracting F_i from F_{i-1} as follows. The encoding algorithm has at most $(k+1)2^S$ configurations (s,t), where s is a state of the algorithm and t is a location on the input tape. Consider all the configurations of the algorithm right before starting to write the i-th from the last block of the output on each of the codewords in F_{i-1}. A fraction of at least $1/((k+1)2^S)$ of them share the same configuration. We define these to be the codewords in F_i. By induction, all the codewords in F_i share their last $i-1$ blocks. To show they share also the i-th from the last block we use a cut & paste argument to show that otherwise there are two codewords that differ only in the last $d-1$ bits of the i-th from the last block, contradicting the distance property of the code.

Proof. We will prove the theorem for binary codes ($q=2$); the proof for general q is very similar.

Let A be any one-pass algorithm for $\mathcal{C}$, and let S be the space used by A.

We divide every codeword $w \in \mathcal{C}$ into $t \stackrel{\text{def}}{=} \lceil n/d \rceil$ blocks, $w^1, \ldots, w^t$, each (except, maybe, for the first one) is of size d. For simplicity of exposition, we will assume that the first block is also of size d.

For any input message $x \in \mathbb{F}_2^k$, we divide the execution of A on x into $t+1$ phases: the i-th phase $(i = 0, \ldots, t-1)$ ends just before A outputs the first bit of the $i+1$-st block of $E(x)$; the first phase starts at the first step of the algorithm; the last phase ends at the end of the execution. We denote by x^i the i-th block of the input – these are the input bits read from x during the i-th phase. Also, denote by $x^{<i}$ the portions of x read prior to the i-th phase, and by $x^{>i}$ the parts of x read after the i-th phase.

For an input $x \in \mathbb{F}_2^k$, let $c^i(x)$ be the configuration of A at the beginning of the i-th phase when running on x.

We next prove that for each i we can find a large subset of the codewords that agree in their last i blocks:

Lemma 46 *For each $i = 0, \ldots, t$, there exists a subset $F_i \subseteq \mathbb{F}_2^k$ of size at least $2^{k-(S+\log(k+1))i}$, such that all the codewords in $E(F_i)$ agree in their last i blocks.*

Proof. The proof will work by induction on i.

For $i = 0$, the statement trivially holds with $F_0 = \mathbb{F}_2^k$. Assume, then, that the statement holds for $i-1$; we will show that it holds also for i. In the sequel we define $j \stackrel{\text{def}}{=} t - i + 1$. Note that the j-th block is the i-th block from the end.

The set F_i will be a subset of F_{i-1}. Consider the at most 2^S possible configurations of A at the beginning of the j-th phase. Let F_{i-1}^c be the set of messages $x \in F_{i-1}$ for which $c^j(x) = c$. Further divide F_{i-1}^c into (at most) $k+1$ sets,$F_{i-1}^{c,0}, \ldots, F_{i-1}^{c,k}$, where $F_{i-1}^{c,r}$ consists of the inputs $x \in F_{i-1}^c$ for which $|x^{<j}| = r$ (that is, exactly r bits are read from x prior to the j-th phase).

We choose F_i to be the largest among the sets $\{F_{i-1}^{c,r}\}, c = 1, \ldots, 2^S,\ r = 0, \ldots, k$. Since $|F_{i-1}| \geq 2^{k-(S+\log(k+1))(i-1)}$, then $|F_i| \geq 2^{k-(S+\log(k+1))(i-1)}/(2^S \cdot (k+1)) = 2^{k-(S+\log(k+1))i}$.

Since all the codewords in $E(F_{i-1})$ agree in their last $i-1$ blocks, and since F_i is a subset of F_{i-1}, then also all the codewords in $E(F_i)$ agree in their last $i-1$ blocks. We are left to show that all the codewords in $E(F_i)$ agree in their j-th block (the i-th block from the end).

Assume, to the contradiction, there are two inputs $x, y \in F_i$, such that $x^j \neq y^j$. Define the input $z = x^{<j} \circ y^{\geq j}$; that is, z shares its first $j-1$ blocks with x and its last $t-j+1$ blocks with y. Since $|x^{<j}| = |y^{<j}|$, z is indeed a legal input of length k. Since $z^j = y^j \neq x^j$, x and z are distinct, and therefore $E(z)$ and $E(x)$ should differ in at least d locations.

It is easy to see that the first $j-1$ blocks of $E(z)$ are identical to the first $j-1$ blocks of $E(x)$ (because x and z share the first $j-1$ blocks). Furthermore, $c^j(z) = c^j(x) = c^j(y)$. Therefore, since z and y share the same configuration at the beginning of the j-th phase and the same bits read from this phase and on, then they also share the same output from this point on: $E(z)^{\geq j} = E(y)^{\geq j}$. Recall that $E(y)^{>j} = E(x)^{>j}$, implying that $E(z)$ and $E(x)$ can differ only in the j-th block.

By our definition of phases, the j-th phase begins immediately before the algorithm outputs the first bit of the j-th block of the codeword. Therefore, all the inputs that share the same configuration at the beginning of the j-th phase, share also the same first output bit. It follows that the first bit in the j-th block of $E(x)$ and $E(z)$ should be the same. This implies that $E(x)$ and $E(z)$ can differ in at most $d-1$ bits (the rest of the bits of the j-th block), which is a contradiction. □

Applying Lemma 46 to $i = t$, we obtain that there is a subset $F_t \subseteq \mathbb{F}_2^k$ of size at least $2^{k-(S+\log(k+1))t}$, such that all the codewords in $E(F_t)$ agree in their last t blocks. Since the last t blocks of any codeword is the whole codeword, then $|F_t| = |E(F_t)| = 1$. We thus obtain,

$$1 \geq 2^{k-(S+\log(k+1))t}$$

The theorem follows. $\square$

Theorem 47 *Let $\mathcal{C}$ be an $(n, k, d)_q$-ECC with encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$ and decoding function $D : \mathbb{F}_q^n \to \mathbb{F}_q^k$. Then any one-pass algorithm for D requires space*

$$S \geq k \cdot \frac{d-1}{2n} \cdot \log q - \log(k \log q + 1)$$

The proof is similar to the previous one. We divide each received word $w \in \mathbb{F}_q^n$ into $2n/(d-1)$ blocks of size $(d-1)/2$. We divide the decoding of w, $D(w)$, into $2n/(d-1)$ corresponding blocks (not necessarily of equal size): the i-th block of $D(w)$ is the part of $D(w)$ written to the output tape while the decoding algorithm reads the i-th block of w.

We show that there exists a chain of codeword sets $\mathcal{C} = F_0 \supseteq F_1 \supseteq \cdots \supseteq F_{2n/(d-1)}$, such that the decodings of all the codewords in F_i share their first i blocks and such that $|F_i|/|F_{i+1}| \leq (k+1)2^S$. As before, this implies the lower bound, since $|F_{2n/(d-1)}| \leq 1$.

The construction of the chain of codeword sets is similar in flavor to the one outlined in the proof of Theorem 45. The difference is that in order to reach a contradiction, we use a cut & paste argument to construct a corrupted received word that decodes *not* to its closest codeword.

Proof. As before, we prove the theorem for binary codes ($q = 2$). The proof for the general case is very similar. We also assume that $d = 2e + 1$ is odd, so decoding is unique.

Let A be any one-pass algorithm for D, and let S be the space used by A.

We divide every received input message $w \in \mathbb{F}_2^n$ into $t \stackrel{\text{def}}{=} \lceil n/e \rceil$ blocks, $w^1, \ldots, w^t$, each (except, maybe, for the last one) is of size e. For simplicity of exposition, we will assume that the last block is also of size e.

For any received input message $w \in \mathbb{F}_2^n$, we divide the execution of A on w into $t+1$ phases: the i-th phase ($i = 0, \ldots, t-1$) ends just before A reads the first bit of the $i+1$-st block of w; the first phase starts at the first step of the algorithm; the last phase ends at the end of the execution. For $x = D(w)$, we denote by x^i the i-th block of the decoded message – these are the bits of x output during the i-th phase. Also, denote by $x^{<i}$ the portions of x output prior to the i-th phase, and by $x^{>i}$ the parts of x output after the i-th phase.

For a received input message $w \in \mathbb{F}_2^n$, let $c^i(w)$ be the configuration of A at the beginning of the i-th phase when running on w.

We next prove that for each i we can find a large subset of the codewords that agree in their first i blocks:

Lemma 48 *For each $i = 0, \ldots, t$, there exists a subset $F_i \subseteq \mathbb{F}_2^k$ of size at least $2^{k-(S+\log(k+1))i}$, such that all the decoded messages in $D(E(F_i))$ agree in their first i blocks (i.e., $\forall x, y \in F_i,\ D(E(x))^{\leq i} = D(E(y))^{\leq i}$) and such that the configuration of A at the beginning of the $i+1$-st phase on all the codewords in $E(F_i)$ is the same (i.e, $\forall x, y \in F_i,\ c^{i+1}(E(x)) = c^{i+1}(E(y))$).*

Proof. The proof will work by induction on i.

For $i = 0$, the statement trivially holds with $F_0 = \mathbb{F}_2^k$ and the initial configuration of A. Assume, then, that the statement holds for $i-1$; we will show that it holds also for i.

Denote by c_i the configuration of A at the beginning of the i-th phase on the codewords in $E(F_{i-1})$. The set F_i will be a subset of F_{i-1}. Consider the at most 2^S possible configurations of A at the beginning of the $i+1$-st phase. Let F_{i-1}^c be the set of messages $x \in F_{i-1}$ for which $c^{i+1}(E(x)) = c$. Define c_{i+1} to be the configuration c for which the size of F_{i-1}^c is maximal. F_i will be a subset of F_{i-1}^c, and therefore $c^{i+1}(E(x)) = c_{i+1}$ for all $x \in F_i$.

We further divide F_{i-1}^c into (at most) $k+1$ sets, $F_{i-1}^{c,0}, \ldots, F_{i-1}^{c,k}$, where $F_{i-1}^{c,r}$ consists of the messages $x \in F_{i-1}^c$ for which $|D(E(x))^{\leq i}| = r$ (that is, exactly r bits of x are output prior to the $i+1$-st phase when running on the codeword $E(x)$). We choose F_i to be the largest among the sets $\{F_{i-1}^{c,r}\}, r = 0, \ldots, k$. Since $|F_{i-1}| \geq 2^{k-(S+\log(k+1))(i-1)}$, then $|F_i| \geq 2^{k-(S+\log(k+1))(i-1)}/(2^S \cdot (k+1)) = 2^{k-(S+\log(k+1))i}$.

Since all the decoded messages in $D(E(F_{i-1}))$ agree in their first $i-1$ blocks, and since F_i is a subset of F_{i-1}, then also all the decoded messages in $D(E(F_i))$ agree in their first $i-1$ blocks. We are left to show that all the decoded messages in $D(E(F_i))$ agree in their i-th block.

Assume, to the contradiction, there are two messages $x, y \in F_i$, such that $D(E(x))^i \neq D(E(y))^i$. Since $D(E(x))^{<i} = D(E(y))^{<i}$ and $|D(E(x))^{\leq i}| = |D(E(y))^{\leq i}|$, then necessarily $|D(E(x))^i| = |D(E(x))^i|$. This means that neither of the two strings $D(E(x))^i$ and $D(E(y))^i$ is a prefix of the other.

Define the following corrupted input message for the decoding algorithm: $w = E(x)^{<i} \circ E(y)^i \circ E(x)^{>i}$. Note that the Hamming distance between w and $E(x)$ is at most $e < d/2$, and therefore $D(w) = x$. $c^i(w) = c^i(E(x)) = c_i$, because w and $E(x)$ share the first $i-1$ blocks. It follows that $c^i(w) = c^i(E(y))$. But now, since w and $E(y)$ share the same configuration at the beginning of the i-th phase and the same bits read during the i-th phase, then they also share the same output during the i-th phase. That is, $D(w)^i = D(E(y))^i \neq D(E(x))^i$. It follows that $D(w) \neq D(E(x)) = x$ (note that here we crucially use the fact that neither of $D(w)^i$ and $D(E(x))^i$ is a prefix of the other). This is a contradiction. □

Applying Lemma 48 to $i = t$, we obtain that there is a subset $F_t \subseteq \mathbb{F}_2^k$ of size at least $2^{k-(S+\log(k+1))t}$, such that all the codewords in $E(F_t)$ agree in their first t blocks. Since the first t blocks of any codeword is the whole codeword, then $|F_t| = |E(F_t)| = 1$. We thus obtain,

$$1 \geq 2^{k-(S+\log(k+1))t}$$

The theorem follows. □

7.2 Upper Bounds

We now give a construction of an error-correcting code that matches the encoding lower bound and almost matches the decoding lower bound. Our construction is a block code, in which each block is encoded by the expander code of Guruswami and Indyk [GI01]. The Guruswami-Indyk code is a linear-time encodable and decodable binary code that has relative minimum distance $\delta' = 1/2 - \epsilon$ and rate $R' \geq \epsilon^4/c$ for any $\epsilon > 0$ (c is a universal constant).

For our construction, we fix n, k, d as parameters, with the restriction that k/n is not too large. We set $\epsilon = \sqrt[4]{ck/n}$ and define $\gamma \stackrel{\text{def}}{=} (d/n) \cdot 1/(1/2 - \epsilon)$. Our code uses $1/\gamma$ blocks, each of size $k' = \gamma k$.

Proposition 49 *For every n, k, d with $k/n \leq \alpha < 1/(16c)$, the above construction is an $(n, k, d)_2$ code that has an $O(k \cdot (d/n))$-space one-pass encoding algorithm and an $O(d)$-space one-pass decoding algorithm.*

Proof. The rate of each block is $k'/n' = \epsilon^4/c = k/n$. Therefore, our code indeed has code length n and message length k.

The minimum distance in each block is $(1/2 - \epsilon) \cdot n' = (1/2 - \epsilon) \cdot k' \cdot (n/k) = d$. Note that the total minimum distance equals the minimum distance of a single block, and therefore it is also d.

The one-pass encoding algorithm simply reads each block at a time, stores it in main memory, and applies the Guruswami-Indyk encoding algorithm. Since their algorithm runs in linear space (it runs in linear time), then the space used by our algorithm is $O(k') = O(\gamma k) = O(k \cdot (d/n) \cdot 1/(1/2 - \epsilon)) = O(k \cdot (d/n) \cdot 1/(1/2 - \sqrt[4]{ck/n}))$. When $k/n \leq \alpha < 1/(16c)$, this space is $O(k \cdot (d/n))$.

The one-pass decoding algorithm reads each block (of length n') of the received word at a time, stores it in main memory, and applies the Guruswami-Indyk decoding algorithm. This algorithm runs in linear space (it runs in linear time), and therefore the total space of our algorithm is $O(n') = O((n/k) \cdot k') = O(d)$ when k/n is small enough. □

8 Conclusions and Open Problems

The most burning open problem following our work is whether the tradeoff exhibited in our online disperser construction between the space and the seed length ($S \cdot d \approx m$) is the best possible. In fact, we have no lower bound on online dispersers. There are two natural approaches to reduce the seed length in the upper bound (recall that we spend $(t-1)\log n$ random bits on "partitioning" the source): (1) use some of the randomness extracted from the source in the partitioning process; the main problem, however, is that this randomness is only close to uniform, which is not sufficient for the current proof; (2) derandomize the choice of the partition; this has already been achieved for $t = O(\log n)$ by [SSZ98] and later [Ta-98], and a disperser with $Sd << m$ would follow if we could do it for $t >> \log n$.

We have not resolved so far the space complexity of online extractors for high entropies ($k > n/2$) that have a large error ($\epsilon > (n-k)/n$). Our lower bound proofs fail to imply anything for this range of parameters, possibly because they apply to the stronger model of extractors that work against block fixing sources (sources of the form $X_{i,\alpha}$).

Our online hashing lower bound has the somewhat weird anomaly that it deteriorates as the output length increases. This may be an artifact of our proof, which uses a reduction from the extractor lower bound. A direct lower bound for universal hashing might circumvent this problem.

Finally, our online decoding upper bound for error-correcting codes does not match the lower bound for small rates. It remains open to find a better construction in this sense.

Acknowledgments

We would like to thank D. Sivakumar for helpful discussions, and Lawrence Ip for pointing out Proposition 15.

References

[ACR98] A. E. Andreev, A. E. F. Clementi, and J. D. P. Rolim. A new general derandomization method. *J. of the ACM*, 45(1):179–213, 1998.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 132–140, 1987.

[AMS99] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided randomness. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–109, 1999.

[BGW99] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic amplification of space-bounded probabilistic algorithms. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity (CCC)*, pages 188–198, 1999.

[BTY94] P. Beame, M. Tompa, and P. Yan. Communication-space tradeoffs for unrestricted protocols. *SIAM J. on Computing*, 23(3):652–661, 1994.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17(2):230–261, April 1988. Special issue on cryptography.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., 1991.

[CW79] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. of Computer and System Sciences (JCSS)*, 18(2):143–154, 1979.

[FKSV99] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1-difference algorithm for massive data streams. In *Proceedings of the 40th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 501–511, 1999.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *J. of Computer and System Sciences (JCSS)*, 22:407–420, 1981.

[GI01] V. Guruswami and P. Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 658–667, 2001.

[Gol97] O. Goldreich. A sample of samplers – a computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, TR97-020, 1997.

[GVW00] O. Goldreich, S. Vadhan, and A. Wigderson. Simplified derandomization of BPP using a hitting set generator. *Electronic Colloquium on Computational Complexity (ECCC)*, TR00-004, 2000.

[GW94] O. Goldreich and A. Wigderson. Tiny families of functions with random properties. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 574–583, 1994.

[GW99] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In *Proceedings of RANDOM 99*, pages 131–137, 1999.

[HILL99] J. Hastad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM J. on Computing*, 28(4):1364–1396, 1999.

[HR00] Z. Hartman and R. Raz. On the distribution of the number of roots of polynomials and logspace explicit extractors. In *Proceedings of the ICALP 2000 Satellite Workshops*, pages 3–22, 2000.

[HRR99] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, volume 50, pages 107–118, 1999.

[Ip01] L. Ip. Personal communication, 2001.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2000.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 248–253, 1989.

[JM87] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987.

[Mar73] G. A. Margulis. Explicit construction of concentrators. *Problemy Peredači Informacii*, 9(4):71–80, 1973. English translation in Problems Inform. Transmission (1975).

[Mil98] P. B. Miltersen. Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In *Proceedings of the 12th Annual Symposium on Discrete Algorithms (SODA)*, pages 556–563, 1998.

[MNT93] Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107:121–133, 1993.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. of Computer and System Sciences (JCSS)*, 52(1):43–52, 1996.

[RR99] R. Raz and O. Reingold. On recycling the randomness of states in space bounded computation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 159–168, 1999.

[RRV99] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 149–158, 1999.

[RSW00] O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness via repeated condensing. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[Sho96] V. Shoup. On fast and provably secure message authentication based on universal hashing. In *Advances in Cryptology – CRYPTO '96, Lecture Notes in Computer Science*, volume 1109, pages 74–85, 1996.

[Sip88] Michael Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36(3):379–383, June 1988.

[Siv01] D. Sivakumar. Personal communication, 2001.

[SSZ98] M. Saks, A. Srinivasan, and S. Zhou. Explicit OR-dispersers with polylogarithmic degree. *Journal of the ACM*, 45(1):123–154, January 1998.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. In *Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 648–657, 2001.

[SZ99] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. *SIAM Journal on Computing*, 28(4):1433–1459, August 1999.

[Ta-96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 276–285, 1996.

[Ta-98] A. Ta-Shma. Almost optimal dispersers. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 196–202, New York, May 23–26 1998. ACM Press.

[Tre99] L. Trevisan. Construction of extractors using pseudo-random generators. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pages 141–148, 1999.

[TV00] L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 2000.

[TZ01] A. Ta-Shma and D. Zuckerman. Extractor codes. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 193–199, 2001.

[TZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 638–647, 2001.

[Zuc97] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11:345–367, 1997.