Streaming Computation of Combinatorial Objects[*]

Ziv Bar-Yossef[†]
Computer Science Division
University of California at Berkeley
387 Soda Hall, Berkeley, CA 94720, USA
zivi@cs.berkeley.edu

Omer Reingold[‡]
AT&T Labs - Research
Room A243, 180 Park Avenue, Bldg.103
Florham Park, NJ, 07932, USA
omer@research.att.com

Ronen Shaltiel[§]
Department of computer science
Weizmann Institute of Science
Rehovot, Israel
ronens@wisdom.weizmann.ac.il

Luca Trevisan
Computer Science Division
University of California at Berkeley
387 Soda Hall, Berkeley, CA 94720, USA
luca@cs.berkeley.edu

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 [24, 5]).

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.

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 [2, 10, 19]. 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., [2, 10]) make crucial use of hash functions. Error-correcting codes that admit space-efficient online encoding and decod-

[*] A full version of the paper is available at http://www.cs.berkeley.edu/∼zivi

[†] Supported by NSF Grant CCR-9820897.

[‡] Part of this research was performed while visiting the Institute for Advanced Study, Princeton, NJ.

[§] Part of this research was performed while visiting the Institute for Advanced Study, Princeton, NJ.

ing 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 [28, 29] and implicitly in [27]), 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.* [4] 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 [4] 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 [4] 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 [17].

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 [26]); error-correcting codes are used in the extractor construction of [41], and in several more recent constructions [30, 20, 31, 40, 33]; hash functions are also extractors, as follows from the Leftover Hash Lemma [18]; error-correcting codes with strong list-decodability properties can be derived from extractors [39].

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) [3, 6, 15, 13]. In fact, the results of [15, 13] 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 [4] 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 [4] 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 [17] 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. [42]). 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.* [24] and the communication-space tradeoffs of Beame *et al.* [5]. 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.* [41, 30].

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 [4]).

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 [28, 32, 37, 38].

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 $f : \Omega \to \Omega'$ 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 $f : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$; we nickname its first input "the input" and its second input "the seed".

Definition 1 (Extractor [28]) *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 [34]) *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.*

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 [4] 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 [4]) *Define $t \stackrel{\text{def}}{=} \lceil n/(n-k) \rceil$. Let $D : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ 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 $S \geq m - d - p - 1 - \log t - \log(1/1 - \epsilon t)$.*

We will be interested in a stronger notion of extractors, defined as follows. It is interesting to note that all our upper bounds for online extractors are exhibited by strong extractors.

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).*

2.2 Online Universal Hash Functions

Definition 6 (Universal hash functions [7]) *A family of functions $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^m\}$ 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. [12]) is of size 2^{n+m-1}.

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

Definition 7 (Online universal hash functions) *An algorithm A is called a* one-pass algorithm *for a family of hash functions $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^m\}$, 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 [18], yields a construction of (strong) extractors from any universal family of hash functions:

Lemma 8 (Hastad, Impagliazzo, Levin, Luby [18]) *Let $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^m\}$ be a universal family of hash functions of size 2^d. Then the function $E : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ 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, an "almost" universal family of hash functions are sufficient to construct extractors. Therefore, our lower bounds apply to such families as well.

Definition 9 (ϵ-almost universal hash functions) *A family of functions $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^m\}$ is called an* ϵ-almost universal family of hash functions, *if for every $x \neq x' \in \{0,1\}^n$, $\Pr_{h \in H}(h(x) = h(x')) \leq \epsilon$.*

Lemma 10 (Impagliazzo, Zuckerman [21]) *Let $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^m\}$ be an $((1 + \epsilon^2)/2^m)$-almost universal family of hash functions of size 2^d. Then the function $E : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ 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. [14]).

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

[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.

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 11 (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. We briefly survey them below. *Shannon's (binary) entropy* is defined as $H(X) = E_{\omega\in X}(\log_2(1/X(\omega)))$. Two basic properties of the entropy are the following (cf. [9], Chapter 2): (1) sub-additivity: $H(X, Y) \leq H(X) + H(Y)$ and (2) data processing inequality: $H(f(X)) \leq H(X)$.

The following theorem (see [9], pages 488–489) connects the variation distance between two distributions and their entropy difference:

Theorem 12 *If $||X - Y|| \leq 1/4$, then, $|H(X) - H(Y)| \leq 2||X - Y|| \cdot (\log \frac{|\Omega|}{||X-Y||} - 1)$.*

The following fact (due to Lawrence Ip [22]) 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 13 *For any distribution X on $\{0, 1\}^{\leq m}$, $H(X) \leq E(|X|) + H(|X|) \leq E(|X|) + \log(m + 1)$.*

3 Lower Bounds for Online Extractors

We present two versions of the lower bound: the first (Theorem 14) gives weaker bounds, but its proof is intuitive and utilizes known facts from information theory; the second (Theorem 15) is stronger, but its proof is more involved. We also show (Theorem 16) that an even stronger lower bound holds for strong extractors. The proofs of the two latter theorems are omitted for lack of space. They appear in the full version of the paper.

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 14 *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 $S \geq m/t - d - 2\epsilon(m + \log(1/\epsilon) - 1) - \log(m + 1)$.*

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 $E(|y^i(\alpha \circ U_k, U_d)|) \geq m/t$. □

Define $Y^i \overset{\text{def}}{=} y^i(X_{i,\alpha}, U_d)$ and $C^i \overset{\text{def}}{=} c^i(X_{i,\alpha}, U_d)$. 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 $Y^{-i} \overset{\text{def}}{=} y^{-i}(X_{i,\alpha}, U_d)$, 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 13 (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 $Y \overset{\text{def}}{=} E(X_{i,\alpha}, U_d)$: $H(Y) = H(Y^i, Y^{-i}) \leq H(Y^i) + H(Y^{-i}) \leq S + d + m(1-1/t) + \log(m+1)$.

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 12 (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 15 *Let $t \overset{\text{def}}{=} \lceil n/(n-k) \rceil$, 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 $S \geq (m-d)/(t-1) - 2\log t(1 + 1/(t-1))$.*

For strong extractors a stronger lower bound holds:

Theorem 16 *Let $t \overset{\text{def}}{=} \lceil n/(n-k) \rceil$, and let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be a (k,ϵ) strong extractor with $\epsilon \leq 1/(2t^2)$. Then any one-pass algorithm A that computes E requires space $S \geq m/(t-1) - 2\log t(1 + 1/(t-1))$.*

4 Lower Bounds for Online Hashing

Since universal and almost universal hash functions imply strong extractors (Lemma 8 and Lemma 10) we can immediately deduce space lower bounds for online hashing from Theorem 16 (the proofs are straightforward and appear in the full version of the paper).

Theorem 17 *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 $S \geq m/(t-1) - 2\log t(1 + 1/(t-1))$.*

Theorem 18 *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 $S \geq m/(t-1) - 2\log t(1 + 1/(t-1))$.*

Remark 19 *A somewhat strange aspect of Theorem 17 and Theorem 18 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 Extractors and 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). For lack of space, we omit most of the technical details of the section from this extended abstract.

5.1 Upper Bound for Low Entropies

Universal Hashing 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.

Theorem 20 *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 $m + O(\log n)$.*

The extractor we use to prove Theorem 20 employs the family of universal hash functions that is based on matrix multiplication. That is, we define $E(x,r) = M_r \circ (M_r \cdot x)$, (where the seed r of E is interpreted as an m by n Binary matrix M_r). That E is an extractor follows from Lemma 8.

We define a space efficient one-pass algorithm A that evaluates E. The algorithm A works in phases. After phase i it has in memory the product of the first i columns of M_r with the first i bits of x. Almost all of the space used by A is dedicated to holding this partial product.

Remark 21 *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., that 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. Many such families can be evaluated by space efficient one-pass algorithms, implying the following theorem (details are deferred to the full version):

Theorem 22 *For all integers $k < n$, and every $\epsilon > 0$, there exists a (k, ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^{d+m}$, 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 $O(d)$.*

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 [41] and their extensions by Raz, Reingold, and Vadhan [30] 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 [30] using the weak designs of Hartman and Raz [17], 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 [30] 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 [17]; 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 $\overline{n} = \mathrm{poly}(n/\epsilon)$ 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.

In the full version, we 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 [35].

Theorem 23 *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.*

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 [14]. The "heavy" computation in their extractors is a random walk on an expander graph [1]. Bar-Yossef, Goldreich, and Wigderson [4] presented a space-efficient one-pass algorithm for computing neighborhoods in Margulis-Gabber-Galil expanders [25, 11, 23]. Using this algorithm, we show in the full version how to evaluate the extractors of [14] by an $O(n-k) + O(\log(1/\epsilon))$ space one-pass algorithm. This is optimal up to a constant factor in the case where $m = \Omega(n)$ (which is the interesting setting of parameters for [14]).

Upper bound for smaller $\mathbf{m}$ To match the lower bound for large k and smaller m, we use a watered down variant of [14] (that does not use expanders and has a smaller output length).

One of the main observations of [14] is that any high min-entropy source is also a "block source" [8]. When the source is divided into blocks of length a, then each of them contains roughly $a - (n-k)$ bits of "independent" randomness. Since extracting randomness from block sources is much easier than from general sources [28], we can use them for the following simplified variant of [14]:

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 $a = O((n-k) + \log n/\epsilon)$. The output of E is now defined as: $E(x, r) \overset{\mathrm{def}}{=} r \circ E'(x_1, r) \circ E'(x_2, r) \circ \ldots E'(x_{t'}, r)$, where $E' : \{0,1\}^a \times \{0,1\}^d \rightarrow \{0,1\}^{m/t'}$ is an $(a - (n-k) - O(\log n/\epsilon), \epsilon)$ strong extractor. Since the space for computing E online is roughly the space needed for computing E' online, then by taking E' to be one of the space-efficient extractors for small min-entropy, E requires space roughly $m/t' + O(\log(n/\epsilon))$.

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 (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.

More precisely, we construct $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ given a parameter t and a $(\bar{k}, \bar{\epsilon})$-extractor $\bar{E} : \{0,1\}^{\bar{n}} \times \{0,1\}^{\bar{d}} \to \{0,1\}^{\bar{m}}$ which can be evaluated by a one pass algorithm with space $\bar{s}$. It gets an input $x \in \{0,1\}^n$ and a seed $r = (\ell_1, \cdots, \ell_{t-1}; y_1, \cdots, y_t)$, where $\ell_j \in \{0,1\}^{\log n}$ and $y_j \in \{0,1\}^{\bar{d}}$, and outputs $z_1, \cdots, z_t \in \{0,1\}^{\bar{m}}$, where for every j, $z_j = \bar{E}(x_{[\ell_{j-1}, \ell_j]}, y_j)$.[2] Here $x_{[i,j]}$ stands for $x_i, x_{i+1}, \cdots, x_{j-1}$.

E can be evaluated by a one pass algorithm with space $\bar{s} + O(\log n)$ if the seed is given in the following order: $\ell_1, y_1, \ell_2, y_2, \cdots, \ell_{t-1}, y_{t-1}, y_t$. At step j we read ℓ_j and run $\bar{E}$ on the j'th block using y_j as seed.

Theorem 24 *E is a (k, ϵ)-disperser 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 23.

Corollary 25 *For every n, k, ϵ and t, there exists a (k, ϵ)-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 $s = m/t + O(\log n)$.*

To extend the argument above from bit-fixing sources to general sources we use methods from [28, 32, 37, 38] to argue that any (general) source contains a sub-source on which it resembles a bit fixing source in the sense that E works for such a sub-source. Exact details appear in the full version[3].

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. We provide only sketches of the proofs. The full proofs appear in the full version of the paper.

Theorem 26 *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 $S \geq k \cdot d/n \cdot \log q - \log(k \log q + 1)$.*

Proof sketch. For simplicity, we assume the code is binary. 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

[2] 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.

[3] Many previous extractors and dispersers constructions [28, 36, 32, 37, 43, 38, 31] take this route and start by "partitioning" a source into a "block source". It should be noted that whereas most of the previous works were interested in $t = O(\log n)$, we are also interested in larger t's, (like $\sqrt{k}$) and our proof works by tailoring techniques from previous work to this setup.

that differ only in the last $d-1$ bits of the i-th from the last block, contradicting the distance property of the code. □

Theorem 27 *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 $S \geq k \cdot (d-1)/(2n) \cdot \log q - \log(k \log q + 1)$.*

Proof sketch. 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 26. 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. □

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 [16]. 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 $\gamma \overset{\text{def}}{=} (d/n) \cdot 1/(1/2 - \epsilon)$. Our code uses $1/\gamma$ blocks, each of size $k' = \gamma k$. It is easy to verify that this is indeed an $(n,k,d)_2$ code. Our encoding algorithm encodes each block at a time; this encoding runs in linear time and therefore also in linear space, and thus requires space $O(k') = O(k \cdot (d/n))$. The decoding algorithm decodes each block at a time, and thus requires space $O(n') = O((n/k) \cdot k') = O(d)$. If we choose $k = \Omega(n)$, then this decoding is optimal.

8 Conclusions and Open Problems

An interesting open problem is whether the space–seed length tradeoff in our online disperser construction ($S \cdot d \approx m$) is the best possible. In fact, we have no lower bound for online dispersers. Two natural approaches to reduce the seed length in the construction (recall that we use $(t-1)\log n$ random bits in "partitioning" the source) are: (1) use randomness extracted from the source in the partitioning process; this randomness, however, 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 [32, 38], 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 bounds deteriorate as the output length increases. This may be an artifact of the 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 is not tight for small rates. It remains open to find a better construction in this sense.

Acknowledgments

We thank D. Sivakumar for helpful discussions, and Lawrence Ip for pointing out Proposition 13.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

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

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

[10] 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.

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

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

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

[26] 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.

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

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

[29] 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.

[30] 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.

[31] 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.

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

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

[34] M. Sipser. Expanders, randomness, or time versus space. *J. Comput. Syst. Sci.*, 36(3):379–383, June 1988.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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