The Computational Complexity Column

by

Lance FORTNOW

NEC Research Institute
4 Independence Way, Princeton, NJ 08540, USA
fortnow@research.nj.nec.com
http://www.neci.nj.nec.com/homepages/fortnow/beatcs

An extractor is a device that takes a distribution with low maximum probability and with a small amount of truly random bits creates a nearly uniform distribution. Extractors have surprisingly many uses in complexity theory.

Ronen Shaltiel takes us on a tour of extractors focusing on recent constructions including the amazing connection to pseudorandom generators.

# Recent Developments in Explicit Constructions of Extractors

Ronen Shaltiel
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel University
ronens@wisdom.weizmann.ac.il

**Abstract**

Extractors are functions which are able to "extract" random bits from arbitrary distributions which "contain" sufficient randomness. Explicit constructions of extractors have many applications in complexity theory and combinatorics.

This manuscript is a survey of recent developments in extractors and focuses on explicit constructions of extractors following Trevisan's breakthrough result [Tre99].

## 1 Introduction

This manuscript attempts to appeal to both experts and newcomers to extractors. It is composed of two parts. The first part gives a brief introduction to the area. More details can be found in the excellent previous survey papers [Nis96, NTS99]. The first part also presents the current "state of the art" in extractor constructions. The second part attempts to complement the previous survey papers and cover more recent work on extractors. The most exciting development in recent years is Trevisan's construction [Tre99] which opened the door to many other constructions which are surveyed here. The presentation attempts to focus on ideas on an intuitive level and the reader is referred to the original papers for precise details.

## 1.1 The initial motivation: Weak random sources

The introduction of probabilistic algorithms and protocols revolutionized complexity theory and cryptography. In some cases (most notably in cryptography) probabilistic protocols make it possible to perform tasks that are impossible deterministically. In other cases probabilistic algorithms are faster, more space efficient or simpler than known deterministic algorithms. We refer the reader to textbooks such as [MR95, Gol98] for more details. All these algorithms and protocols expect to be given "truly random bits" (that is a sequence of bits which are uniformly distributed and independent of each other). A question arises: How can we obtain truly random bits? One solution is to sample from some physical processes.[1] While there are such distributions which are believed to be "somewhat random" (a popular example are Zener Diodes which produce quantum mechanical noise) it is unlikely that these distributions produce "truly random bits". A natural approach is to use a deterministic procedure called an *extractor* to extract truly random bits from "weak random sources". Here are several examples of weak random sources. All the examples are distributions over $n$ bit strings which intuitively "contain" $k < n$ bits of randomness.

- **A Markov-chain source:** A distribution $X$ on $\{0,1\}^n$ such that $X_1, \cdots, X_n$ are obtained by taking $n$ steps on a Markov-chain with transition matrix $P = (p_{ij})$ of constant size in which for every entry $k/n < p_{ij} < 1 - k/n$. Such sources were studied by Blum [Blu84].

- **An unpredictable source:** A distribution $X$ on $\{0,1\}^n$ with the property that for every $1 \le i \le n$ and $b_1, \cdots, b_{i-1} \in \{0,1\}$

  $$k/n \le \Pr[X_i = 1 | X_1 = b_1, \ldots, X_{i-1} = b_{i-1}] \le 1 - k/n$$

  In words, every bit is slightly unpredictable given previous bits. Such sources were studied by Santha and Vazirani [SV86]

- **A bit fixing source:** A distribution $X$ on $\{0,1\}^n$ on which $n - k$ bits are fixed and the remaining $k$ bits are uniform and independent of each other. Such a distribution has an "embedded" copy of the uniform distribution on $k$ bits. Such sources were studied by Cohen and Wigderson [CW89].

In all these examples the purpose is to design a *single extractor* which extracts randomness from an arbitrary source of the prescribed type. Given an extractor, we can use such distributions when running probabilistic algorithms and protocols. (We elaborate more on the sources in the examples above in section 1.8.)

## 1.2 Preliminaries

**Probability distributions:** We use the notation $x \leftarrow X$ to denote sampling an element $x$ from a distribution $X$. We also use the notation $x \in_R T$ to denote sampling an element $x$ uniformly from a set $T$. For a function $f$ and a distribution $X$ on its domain $f(X)$ denotes the distribution of sampling $x$ from $X$ and applying $f$ to $x$. $f(\cdot)$ is used when $X$ is the uniform distribution. $U_m$ is used to denote the uniform distribution on $\{0,1\}^m$.

[1] The other common solution is to replace "truly random bits" by "pseudo-random bits" which are produced by an efficient procedure called a "pseudo-random generator". All such "pseudo-random generators" rely on unproven assumptions. The reader is referred to [Gol98] for a manuscript on pseudo-randomness.

**statistical distance:** Two distribution $P, Q$ over the same domain $T$ are *$\epsilon$-close* if the $L1$-distance between them is bounded by $2\epsilon$, namely:

$$\frac{1}{2} \cdot \sum_{x \in T} |P(x) - Q(x)| \leq \epsilon$$

An equivalent definition is that $|P(A) - Q(A)| \leq \epsilon$ for every event $A \subseteq T$.

The latter formulation explains the usefulness of this notion. The two distributions are almost indistinguishable.[2]

## 1.3 Formal definition of extractors

A general model for weak random sources (that generalizes the examples above) was given by David Zuckerman [Zuc90, Zuc96b] who suggested to "measure" the amount of randomness a probability distribution contains by its *min-entropy*.

**Definition 1 (min-entropy)** *The* min-entropy *of a distribution $X$ on $\{0,1\}^n$ (denoted by $H_\infty(x)$) is defined by:*

$$H_\infty(x) = min_{x \in \{0,1\}^n} \log \frac{1}{\Pr[X = x]}$$

In other words, a distribution has min-entropy at least $k$ if the probability of every element is bounded by $2^{-k}$. Intuitively, such a distribution *contains $k$ random bits.*

**Remark 1** *It is instructive to compare min-entropy with Shannon-entropy. Whereas Shannon's entropy measures the amount of randomness a distribution contains* on average*, min-entropy measures the amount of randomness on the* worst case*. This is suitable as we don't get multiple independent samples from the distribution and have to do with just one sample.*

Consider the goal of designing an extractor for all distributions $X$ with $H_\infty(X) \geq n-1$. That is a function $E : \{0,1\}^n \to \{0,1\}$ such that for every distribution $X$ on $\{0,1\}^n$ the distribution $E(X)$ is a random bit. (Note that here we only want to extract a single random bit.) It is easy to see that no such function $E$ exists. This is because every such function $E$ has a bit $b$ such that the set $S = \{x|E(x) = b\}$ is of size at least $2^{n-1}$. It follows that the distribution $X$ which is uniformly distributed on $X$ has $H_\infty(X) \geq n-1$ and yet $E(X)$ is fixed to the value $b$.

Thus, we will need to settle for a weaker concept of extractor. We will allow the extractor to use an additional input: A "short seed" of truly random bits. We require that for every distribution $X$ with sufficient min-entropy, the output distribution of the extractor is (close to) uniform. Naturally, we want the seed to be smaller than the output of the extractor. Extractors were first defined by Nisan and Zuckerman [NZ96].[3]

**Definition 2 (extractor)** *A* $(k, \epsilon)$-extractor *is a function*

$$Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

*Such that for every distribution $X$ on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the distribution $Ext(X, Y)$ (where $Y$ is uniformly distributed in $\{0,1\}^d$) is $\epsilon$-close to the uniform distribution on $\{0,1\}^m$.*

[2] This should not be confused with pseudo-randomness where the distributions are indistinguishable to efficient procedures. The distributions $P$ and $Q$ are *statistically indistinguishable* and cannot be distinguished even by non-efficient procedures.

[3] Actually, the object defined in that paper is a *strong extractor* see section 1.10.

Naturally, we want the seed length to be as small as possible, and the output length to be as large as possible. We have not yet addressed the question of how to obtain a short random seed for running the extractor. This issue is addressed in section 1.6.

We use the following terminology: We refer to $n$ as the *length of the source*, to $k$ as the *min-entropy threshold* and to $\epsilon$ as the *error* of the extractor. We also refer to the ratio $k/n$ as the *entropy rate of the source* $X$ and to the ratio $m/k$ as the *fraction of randomness extracted by Ext*. The *entropy loss of the extractor* is the amount of randomness "lost" in the extraction process, that is $k+d-m$.

### 1.4 What do we want to optimize?

There are five different parameters here. What do we want to achieve? The most common optimization problem is the following one. When given $n$ (the length of the source), $k$ (the min-entropy threshold) and $\epsilon$ (the required error) we want to construct an extractor with as small as possible seed length $d$ and as large as possible output length $m$. This should be achieved for all entropy-rates and for every choice of error $\epsilon$. (In actual constructions it is often impossible to optimize both seed length and output length simultaneously, and one parameter is sacrificed in order to optimize the other.)

### 1.5 Explicitness

A standard application of the probabilistic method shows that for every $n, k$ and $\epsilon$, there exists a $(k,\epsilon)$-extractor with seed length $d = \log(n-k) + 2\log(1/\epsilon) + O(1)$ and output length $m = k + d - 2\log(1/\epsilon) - O(1)$. This argument was first used by Sipser [Sip88] (for dispersers), and also appears in [RTS00]. This extractor has optimal parameters and matches the lower bounds of [RTS00] (see section 1.12). However, in most applications of extractors it not sufficient to prove the existence of an extractor and what is required is explicit construction.

**Definition 3 (Explicit extractor)** *For functions $k(n), \epsilon(n), d(n), m(n)$ a family $Ext = \{Ext_n\}$ of functions*

$$Ext_n : \{0,1\}^n \times \{0,1\}^{d(n)} \rightarrow \{0,1\}^{m(n)}$$

*is an* explicit $(k,\epsilon)$-extractor *if $Ext(x,y)$ can be computed in polynomial time (in $n$) and for every $n$, $Ext_n$ is a $(k(n),\epsilon(n))$-extractor.*

### 1.6 Simulating BPP given access to a weak random source

How does the addition of a truly random seed correspond to the motivation of obtaining random bits for probabilistic algorithms and protocols? Can we use extractors to run probabilistic algorithms and protocols using a weak random source? The direct answer is that using an extractor whenever an algorithm requires $m$ truly random bits, it can be run using an extractor spending only $d$ random bits when given access to a weak random source. The question arises: How do we get even $d << m$ truly random bits? While we do not have a general answer to this question, in some cases we can simulate probabilistic computation using a weak random source without additional random bits. The most important case is BPP algorithms. Consider a BPP algorithm which computes a boolean function $f(w)$.[4] Given access to a weak random source with sufficient min-entropy we compute $f(w)$ as follows: On an input $w$ and an element $x$ chosen from the weak random source, we run

[4] A polynomial time machine $A(w,z)$ such that for every $w$, $\Pr_z[A(w,z) = f(w)] > 2/3$.

$z = Ext(x, y)$ for all $y \in \{0,1\}^d$ and gather the answers of $A$ on $(w, z)$ for all seeds. It easy to see that with high probability (over the choice of $x$ from the weak source) the majority of these answers agree with $f(w)$.

This algorithm runs in time $2^d$ times the running time of the extractor and the initial algorithm. This serves as motivation to have as short as possible seed length $d$. In particular, an explicit extractor with *logarithmic* seed length can be used to simulate BPP given access to a weak random source of sufficient min-entropy.

Note that this method is not suited for interactive protocols: One cannot run the other party on many messages before deciding which message to send.

## 1.7 Discussion: why min-entropy?

We have chosen to model "weak random sources" as distributions with high min-entropy. In some sense, this is the most general choice as it is not hard to show that if randomness can be extracted from a distribution $X$ then $X$ (is close) to having high min-entropy.[5] Thus, it is impossible to extract randomness from distributions which are not close to having high min-entropy. However, we pay a price for this general choice: The need to have an additional seed of randomness. Some restricted families of sources allow *deterministic extraction* and do not require an additional seed. Before surveying these families it is important to point out that extractors (for the general definition of distributions with high min-entropy) have many other applications. In most of these applications the full generality of min-entropy is required (see section 1.13).

## 1.8 Restricted families of sources

The attempts to extract randomness from a weak random source can be traced back to von Neumann [vN51] who showed how to use many independent tosses of a biassed coin (with unknown bias), to obtain a biassed coin. Blum [Blu84] considered sources which are generated by a walk on a Markov-chain. He showed how to deterministically extract perfect randomness from such a source. Santha and Vazirani [SV86] showed that it is impossible to deterministically extract even a single random bit from an *unpredictable* source (see section 1.1). Vazirani [Vaz87a, Vaz87b] showed that it is possible to deterministically extract randomness from two such independent sources. Chor and Goldreich [CG88] showed how to simulate BPP using one *block-wise source*, and extract randomness from two independent block-wise sources. Cohen and Wigderson [CW89] considered several variants of *bit-fixing sources* and constructed a deterministic disperser for the kind defined in section 1.1. Mossel and Umans [MU01] construct a deterministic disperser for bit-fixing sources with $k > 3n/4$ (even when an adversary gets to choose the value of the bad bits as a function of the uniform ones). A different approach was taken by Trevisan and Vadhan [TV00] who consider what they call "samplable sources". These are sources of high min-entropy which can be efficiently generated from truly random bits. They construct a deterministic extractor for distributions samplable by small circuits (assuming some complexity theoretic hardness assumption).

## 1.9 Dispersers

A disperser is the one-sided analogue of an extractor. The requirement that the output of the extractor is $\epsilon$-close to uniform says that for every event $A \subseteq \{0,1\}^m$, $\Pr_{U_m}[A] - \Pr_{Ext(X,U_d)}(A)| \leq \epsilon$.

[5] More precisely, for any probability distribution $X$ on $\{0,1\}^n$ and any function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ (even one that is tailored specifically for $X$), if the distribution $E(X, U_d)$ is $\epsilon$-close to the uniform distribution then $X$ is $O(\epsilon)$-close to a distribution $X'$ with $H_\infty(X') \geq m - d - 1$.

In a disperser we omit the absolute value and only require that the probability assigned to $A$ by the output distribution of a disperser is not much smaller than that of the uniform distribution. Dispersers were first defined by Sipser [Sip88]. The following definition is equivalent to the one in the discussion above.

**Definition 4 (disperser)** *A* $(k, \epsilon)$-disperser *is a function*

$$Dis : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

*Such that for every distribution $X$ on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the support of the distribution $Ext(X,Y)$ (where $Y$ is uniformly distributed in $\{0,1\}^d$) is of size at least $(1-\epsilon)2^m$.*

Dispersers suffice for simulating RP algorithms (probabilistic algorithms with one-sided error) using a weak random source.

### 1.10 Strong extractors

The input of an extractor contains two independent sources of randomness: the source and the seed. In some applications it is required that the extractor's output will be uniform even to someone who sees the seed. A way of enforcing such a condition is to demand that even if the seed is concatenated to the output the resulting distribution is close to uniform.

**Definition 5 (Strong extractor)** *A* $(k, \epsilon)$-strong extractor *is a function*

$$Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

*Such that for every distribution $X$ on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the distribution $Y \circ Ext(X,Y)$ (where $Y$ is uniformly distributed in $\{0,1\}^d$) is $\epsilon$-close to the uniform distribution on $\{0,1\}^m$.*

### 1.11 Extractors as graphs

Extractors can also be thought of as bipartite graphs where the nodes of the left hand side are strings of length $n$ and the nodes on the right hand side are strings of length $m$. Every node $x$ on the left hand side is connected to all $2^d$ nodes $z$ for which there exists a $y \in \{0,1\}^d$ such that $E(x,y) = z$. Note that the out degree of every node on the left hand side is exponential in the seed length. The definition of extractors guarantees that for every set $S \subseteq \{0,1\}^n$ of size $2^k$ on the left hand side and for every set $T \subseteq \{0,1\}^m$ on the right hand side the number of nodes from $S$ to $T$ is close to what one expects in a random graph. More precisely:

$$\left| E(S,T) - |S||T|2^{d-m} \right| \leq \epsilon \tag{1}$$

This behavior resembles that of expander graphs. An important difference is that extractors are *unbalanced* bipartite graphs (by that we mean that the two sides differ in size). Another analogy to expander graphs is that equation (1) above entails that the set $S$ "sees" a $1-\epsilon$ fraction of the nodes in the right hand side. (Actually, the definition of dispersers suffices for this conclusion.) This can be interpreted as "relative expansion": The volume the set takes in the space it lives in is expanded from $2^{k-n}$ to $1-\epsilon$.

The effort spent to construct explicit extractors with small seed length can be understood as a continuation of the effort spent to construct explicit expander graphs with small degree. It should be noted that in contrast to expander graphs, "extractor graphs" cannot have constant degree (see section 1.12 for lower bounds on the seed length in extractors).

## 1.12 Lower bounds

Radhakrishnan and Ta-Shma [RTS00] gave tight lower bounds on the seed length and output length of extractors and dispersers (extending previous bounds by Nisan and Zuckerman [NZ96]). We say that an extractor is *non-trivial* if $m \geq d+1$. In words, the extractor extracts more bits then it receives.

**Lower bounds on seed length:** Every non-trivial $(k,\epsilon)$-extractor with $\epsilon \leq 1/2$ has seed length $d \geq \log(n-k) + 2\log(1/\epsilon) - O(1)$. An almost similar bound holds for dispersers $d \geq \log(n-k) + \log(1/\epsilon) - O(1)$.[6].

**Lower bound on entropy loss:** Recall that the entropy loss of an extractor is $k+d-m$, that is the amount of randomness "lost" in the extraction process. A surprising result by [RTS00] is that every non-trivial $(k,\epsilon)$-extractor cannot extract all the randomness present in its input and suffers an entropy loss of $2\log(1/\epsilon) - O(1)$. Here there's a substantial difference between extractors and dispersers and dispersers exhibit entropy loss of only $\log\log(1/\epsilon) - O(1)$.

It should be noted that both these lower bounds are met (except for the precise constant in the $O(1)$ term) by the non-explicit construction of extractors using the probabilistic method.

## 1.13 Applications of extractors

While simulating probabilistic algorithms given access to a weak random source was the main motivation to extractors, they turn out to have many other applications. Some of these applications don't even seem to be connected to randomness and derandomization. Applications of extractors include:

- "Randomness versus space tradeoffs" [Sip88].
- unconditional constructions of pseudo-random generators for probabilistic algorithms with small memory [NZ96, INW94, RR99].
- Randomness efficient oblivious sampling and "deterministic amplification" of the success probability of probabilistic algorithms [Zuc97].
- Hardness of approximation results for CLIQUE [Zuc96a], some $\Sigma_2^P$ problems [Uma99] and the VC-dimension [MU01].
- Explicit constructions of depth 2 superconcentrators [WZ99], nonblocking networks and certain expander graphs with expansion properties stronger than can be obtained using the eigenvalue method [WZ99].
- Protocols for sorting and selecting in rounds [WZ99] and leader election [Zuc97, RZ98].
- A different proof that BPP $\subseteq$ PH [GZ97].
- Explicit constructions of error correcting codes with strong list decoding properties [TSZ01].

The reader is encouraged to look at the excellent previous survey papers [Nis96, NTS99] which explain some of these applications. It should be noted that all these applications require explicit constructions of extractors (for some of them dispersers suffice). In some of the applications improved extractor constructions immediately translate to an in improvement in the application.

[6] For $\epsilon > 1/2$ [RTS00] give lower bounds on dispersers of the form $d = \log(n-k) - \log\log(1/(1-\epsilon)) - O(1)$.

### 1.14 Explicit constructions: state of the art

A substantial effort has been spent in recent years trying to explicitly construct optimal extractors. Table 1 reports many such extractor constructions. For simplicity all the results in the table are stated for constant error $\epsilon$. In most cases this implies an extractor with arbitrary $\epsilon$ and roughly the same parameters by the error reduction technique of [RRV99a]. (Exact details are given in section 6.2). The reader should also note that the constructions are presented in a way which optimizes the seed length. In most cases the output length of an extractor can be increased at the cost of increasing the seed length. (Exact details are given in section 6.1). Thus, for example, an extractor with $m = \Omega(k)$ gives one with optimal output length $m = k + d - O(1)$. This is achieved at the cost of increasing the seed from $d$ to $O(d \log k)$.

## 2 A brief overview of the early constructions

Before surveying more recent developments in extractor constructions we give a brief overview of the methods and ideas initially used to construct extractors. More details can be found in the previous survey papers [Nis96, NTS99]. (In fact, these ideas show up in some recent constructions too).

### 2.1 The initial hashing-based construction

The first explicit construction appears implicitly in the work of Impagliazzo, Levin and Luby [ILL89] and is based on Carter-Wegman universal hash functions.

**Definition 6 (universal family of hash functions)** *A collection $H$ of functions $h : \{0,1\}^n \to \{0,1\}^\ell$ is a* universal family of hash functions *if for any $w_1 \neq w_2 \in \{0,1\}^n$, and for any $x_1, x_2 \in \{0,1\}^\ell$,*

$$\Pr_{h \in_R H}[h(x_1) = w_1 \wedge h(x_2) = w_2] = 2^{-2\ell}$$

There are such explicit families of size $2^{2n}$ for every $\ell \leq n$. Let $H = \{h_y\}_{y \in \{0,1\}^{2n}}$ be such a family. It follows from [ILL89] that the function $Ext(x, y) = y \circ h_y(x)$ is an $(m + 2\log(1/\epsilon), \epsilon)$-extractor for every $\epsilon > 0$. Exact details can be found (for example) in [SZ99].

While this extractor has optimal entropy loss ($m = k + d - 2\log(1/\epsilon) - O(1)$), it has a very large seed length $d = O(n)$. In fact, the seed is longer than the amount of randomness in the source, and the only reason this makes "economic sense" is because this "investment" is returned with a "dividend" of random bits extracted from the source.

It was noted by Srinivasan and Zuckerman [SZ99] that universal hash functions can be replaced by "almost pairwise independent hash functions", this can reduce the seed length to $k + O(\log(1/\epsilon))$, which is still very large. The early extractor constructions worked by composing this extractor with itself in a way that reduces the length of the seed.

### 2.2 Block-wise sources

One scenario in which extraction with shorter seed can be achieved is when the source distribution $X$ consists of two independent concatenated distributions $X = (X_1, X_2)$, where both $X_1$ and $X_2$ contain randomness. One can run an extractor to extract random bits from $X_2$ and use these bits (which are independent of $X_1$) as seed to extract randomness from $X_1$ with another (strong) extractor. More formally, an extractor is given by: $E(x_1, x_2; y) = E_1(x_1, E_2(x_2, y))$. The important

Table 1: Milestones in extractors constructions:

| Reference | min-entropy threshold | seed length | output length |
|---|---|---|---|
| [ILL89]* | any $k$ | $d = O(n)$ | $m = k + d - O(1)$ |
| [NZ96] | $k = \Omega(n)$ | $d = O(\log^2 n)$ | $m = \Omega(k)$ |
| [GW97] | $k > n/2$ | $d = O(n - k)$ | $m = k + d - O(1)$ |
| [SZ99] | any $k$ | $d = O(k + \log n)$ | $m = k + d - O(1)$ |
| [SZ99] | $k \geq n^{1/2+\delta}$ | $d = O(\log^2 n)$ | $m = k^{\Omega(1)}$ |
| [Zuc97] | $k = \Omega(n)$ | $d = O(\log n)$ | $m = \Omega(k)$ |
| [TS96] | any $k$ | $d = \log^{O(1)} n$ | $m = k$ |
| [NTS99]† | $k = n^{\Omega(1)}$ | $d = \log n \cdot \log^{(i)} n$ | $m = k^{\Omega(1)}$ |
| [Tre99] | any $k$ | $d = O(\frac{\log^2 n}{\log k}))$ | $m = k^{1-\delta}$ |
| [RRV99b] | any $k$ | $d = O(\log^2 n)$ | $m = \Omega(k)$ |
| [ISW00] | any $k$ | $d = O(\log n)$ | $m = k^{1-\delta}$ |
| [RSW00] | any $k$ | $d = O(\log n \cdot (\log\log n)^2)$ | $m = \Omega(k)$ |
| [RSW00] | any $k$ | $d = O(\log n)$ | $m = \Omega(k/\log n)$ |
| [RVW00] | $k > n/2$ | $d = \log^{O(1)}(n-k)$ | $m = k + d - O(1)$ |
| [TSUZ01] | $k \leq 2^{\log^{1/(2+\delta)} n}$ | $d = O(\log n)$ | $m = k + d - O(1)$ |
| [TSZS01] | $k \geq n^{1/2}$ | $d = \log n + O(\log(\log^* n))$ | $m = \frac{k}{n^{1/2}\log^2 n}$ |
| [TSZS01] | $k = \Omega(n)$ | $d = \log n + O(\log\log n)$ | $m = \Omega(k)$ |
| [SU01]‡ | any $k$ | $d = (1+\alpha)\log n$ | $m = \frac{k}{\log^{O(1/\alpha)} n}$ |
| optimal§ | any $k$ | $d = \log n + O(1)$ | $m = k + d - O(1)$ |

Table 2: Milestones in dispersers constructions:

| Reference | min-entropy threshold | seed length | output length |
|---|---|---|---|
| [SSZ98] | $k = n^{\Omega(1)}$ | $d = O(\log n)$ | $m = k^{\Omega(1)}$ |
| [TS98] | $k = n^{\Omega(1)}$ | $d = O(\log n)$ | $m = k - \log^{O(1)} n$ |
| [TSUZ01] | any $k$ | $d = O(\log n)$ | $m = k$ |
| optimal¶ | any $k$ | $d = \log n + O(1)$ | $m = k + d - O(1)$ |

All the results are stated for constant error $\epsilon$.

$\delta > 0$ is an arbitrary constant and constants in $O(\cdot), \Omega(\cdot)$ may depend on it.

* [ILL89] has appeared before extractors were defined.

† Here, $\log^{(i)}$ is the $i$ times iterated log function. $i$ is an arbitrary constant.

‡ Here $0 < \alpha \leq 1$ is an arbitrary function of $n$ and $k$.

§ The existence of an optimal extractor which matches the lower bounds of [RTS97] is proved using the probabilistic method.

¶ The existence of an optimal disperser which matches the lower bounds of [RTS97] is proved using the probabilistic method.

feature of this extractor is that we only require truly random bits to run $E_2$. Thus, the final extractor $E$ has a better relation between seed length and output length compared to its components. (Though, $E$ is not guaranteed to work on a general source).

Can this approach work when $X_1$ and $X_2$ are dependent? As noted implicitly in [NZ96] and explicitly in [SZ99] it is sufficient that $X_1, X_2$ form a block-wise source.

**Definition 7 (block-wise sources [CG88])** *Two (possibly correlated) distributions $X_1, X_2$ form a* block-wise source with min-entropy $k_1, k_2$ *if:*

1. $H_\infty(X_1) \geq k_1$
2. *For every $x_1$, $H_\infty(X_2|X_1 = x_1) \geq k_2$. (Here $X_2|X_1 = x_1$ is the distribution of $X_2$ conditioned on the event $\{X_1 = x_1\}$).*

Intuitively $X_1, X_2$ form a block-wise source if $X_2$ contains $k_2$ random bits which are *not* contained in $X_1$. This definition can be generalized to $t > 2$ blocks $X_1, \cdots, X_t$. It is required that the $i$'th block contains randomness even when all previous $i-1$ blocks are fixed. When the parameters are chosen correctly, $t$ extractors can be composed together requiring truly random bits only for the first one.

To get an extractor (for general sources) one needs to give a method of transforming a general source into a block-wise source using few random bits. The initial extractor construction by Nisan and Zuckerman [NZ96] introduced such a method, and following work [SZ99, SSZ98, TS96, Zuc97, TS98, NTS99] gradually improved the parameters by composing extractors with themselves in various ways. (Some of these constructions only achieve dispersers).

**Remark 2** *It should be noted that the block-wise source approach is still very promising even though by now we have other techniques to construct extractors. This is because current explicit constructions cannot optimize both seed length and output length simultaneously. However, when given a block-wise source, two extractors which optimize the two different parameters can be composed to give one which optimizes both simultaneously. Thus, to get nearly optimal extractors it is sufficient to transform a general source into a block source using very few random bits. More details are given in section 4.2.*

# 3 Trevisan's extractor

A major breakthrough was made by Trevisan in [Tre99]. He observed that methods used to construct pseudo-random generators from hard functions actually produce extractors. He went on to construct a direct and simple extractor using the pseudo-random generator constructions of [NW94, IW97].

## 3.1 Pseudo-random generators

A pseudo-random generator is a procedure which stretches a short seed of $d$ truly random bits into a long output of $m$ "pseudo-random" bits. In our setup a distribution is pseudo-random if no small circuit can distinguish it from the uniform distribution. Pseudo-random generators play an important role in complexity theory and cryptography, the reader is referred to [Gol98] for a monograph devoted to this subject.

Pseudo-random generators entail the existence of explicit hard functions, and thus, can only be constructed assuming unproven assumptions. This "hardness versus randomness" paradigm was introduced by [BM84, Yao82]. Nisan and Wigderson [NW94] initialized a sequence of such hardness versus randomness tradeoffs in the following setup:

**The Nisan-Wigderson setting:** Let $\ell < s(\ell) < 2^\ell$ be some integer function.
**Assumption:** There exists a (family of) functions $f = \{f_\ell\}$, $f_\ell : \{0,1\}^\ell \rightarrow \{0,1\}$ such that:

1. $f$ is computable in time $2^{O(\ell)}$.
2. For all $\ell$, circuits of size $s(\ell)$ cannot compute $f_\ell$.[7]

**Conclusion:** A pseudo-random generator $G : \{0,1\}^d \rightarrow \{0,1\}^m$ which is computable in time $2^{O(d)}$ and for every circuit $C$ of size $m$, $|\Pr_{y \in \{0,1\}^d}[C(G(y)) = 1] - \Pr_{z \in_R \{0,1\}^m}[C(z) = 1]| \leq \epsilon$. (Here $d, m$ and $\epsilon$ are functions of $\ell$).

As in extractors the goal is to minimize the seed length of the generator and maximize its output length. A sequence of works [NW94, BFNW93, Imp95, IW97, STV99, ISW99, ISW00, SU01, Uma02] constructed such generators with gradually improving parameters.

An important milestone was reached by Impagliazzo and Wigderson [IW97][8]. They focused on $s(l) = 2^{\Omega(l)}$ and gave a generator with $d = O(\ell)$, $m = s(\ell)^{\Omega(1)}$ and $\epsilon = 1/\mathrm{poly}(m)$. (This implies that BPP = P if the assumption above holds for $s(\ell) = 2^{Omega(\ell)}$).

## 3.2 Extractors from pseudo-random generators

Trevisan observed that the [IW97] construction gives an extractor. (It should be noted that before Trevisan's paper, pseudo-random generators and extractors were considered to be very different objects as pseudo-random generators live in the "computational realm" whereas extractors are "information theoretic" and do not rely on unproven assumptions.)

He observed that the [IW97] construction uses the hard function $f$ as a black box. Thus, we can think of the function $f$ (represented by its truth table $x_f$) as an additional input to the generator. More precisely, let $n = 2^\ell$, the [IW97] paper constructs a function $E : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ such that whenever $x$ is the truth table of a hard function $f : \{0,1\}^l \rightarrow \{0,1\}$, the distribution $E(x, \cdot)$ cannot be distinguished from uniform by size $m$ circuits. We will call hardness versus randomness tradeoffs with this property *pseudo-random generator schemes.* It turns out that all known tradeoffs are pseudo-random generator schemes.

The requirement that $G$ is computable in time $2^{O(l)} = n^{O(1)}$ gives that $E$ is computable in polynomial time, as the input length of $E$ is of length about $n$.

The [IW97] construction proves the correctness of the generator by showing that for every $f : \{0,1\}^l \rightarrow \{0,1\}$, given a circuit $B$ of size $m$ which distinguishes the output of $E(x_f, \cdot)$ from uniform there exists a circuit $C$ of size $s$ which computes $f$. Thus, if $f$ is hard then the generator is pseudo-random. Trevisan noticed that this proof uses $B$ as a black box. That is the proof shows that for every circuit $B$, (not necessarily small) which distinguishes the output of $E(x_f, \cdot)$ from uniform there exists a circuit $C$ of size $s$, which uses $B$-gates[9] and computes $f$. We call pseudo-random generator schemes with this property *pseudo-random generator schemes with black box proof.* Once again, all existing construction have this property. Trevisan showed that such hardness versus randomness tradeoffs *are* extractors.

[7] We say that a circuit $C$ computes $f$ if for every input $w$, $C(w) = f(w)$.

[8] Different constructions and proofs were later given in [STV99, SU01, Uma02].

[9] A circuit with $B$ gates, for some function $B$, is a circuit which has gates computing the function $B$ in addition to the standard boolean gates.

In the following theorem $T(s) = 2^{\Theta(s \log s)}$ denotes the number of circuits of size $s$ with constant number of gate types.

**Theorem 1 ([Tre99])** *Every pseudo-random generator scheme with black box proof (and in particular the [IW97] construction) is a* $(\log T(s) + \log(1/\epsilon), 2\epsilon)$*-extractor.*

**Proof:** Let $E$ be a pseudo-random generator scheme with black-box proof. Let $X$ be a distribution on $\{0,1\}^n$ such that $H_\infty(X) \geq \log T(s) + \log(1/\epsilon)$. To show that $E$ is an extractor, we need to prove that for every event $A$ the distributions $E(X, U_d)$ and $U_m$ assign (roughly) the same probability to $A$.

It is instructive to prove this first only for events $A$ of the form: $A = \{x | B(x) = 1\}$ where $B$ is a circuit of size $m$. These are exactly the events which cannot distinguish the output of a pseudo-random generator from uniform. The next claim shows that with high probability an element sampled from the source is a truth table of a hard function. It follows that with high probability over choosing $x$ from $X$, $E(x, \cdot)$ is a pseudo-random generator. This implies that events $A$ as above cannot distinguish $E(X, U_d)$ from uniform.

**Claim 1** *For $x \in \{0,1\}^n$, let $f_x : \{0,1\}^l \to \{0,1\}$ denote the function which truth table is $x$.*

$$\Pr_{x \leftarrow X}[f_x \textit{ cannot be computed by size } s \textit{ circuits}] > 1 - \epsilon$$

**Proof:** Since $H_\infty(X) \geq \log T(s) + \log(1/\epsilon)$ we have that for every $x \in \{0,1\}^n$, $\Pr[X = x] \leq 2^{-(\log T(s) + \log(1/\epsilon))} = \epsilon / T(s)$. Thus, with probability $1 - \epsilon$ an $x$ such that $f_x$ isn't computable by a size $s$ circuit is chosen. $\bullet$

We now extend this argument to any event $A$. Fix some event $A \subseteq \{0,1\}^m$, we let $B(x)$ be a circuit (of possibly exponential size) such that $B(x) = 1$ if and only if $x \in A$. The crucial observation is that the proof of claim 1 still holds when allowing circuits to use $B$-gates. (This is because when $B$ is fixed, the complexity of $B$ does not contribute to the size of a circuit with $B$-gates). $\bullet$

A more careful examination of the proof above shows that we proved that for every event $A$, the number of $x \in \{0,1\}^n$ such that $A$ distinguishes $E(x, \cdot)$ from uniform is small. This was shown by mapping every such "bad" $x$ to a small circuit which computes $f_x$. However, no computational features of circuits were used in the proof. We only used the fact that there are few small circuits. Circuits were only used as "descriptions" of the functions they compute. Any other one to one mapping of "bad" $x$'s into a small set would have been sufficient. The following section states this observation precisely.

## 3.3 The reconstruction proof technique

Stripping the argument of the previous section from all "computational issues", Trevisan's argument gives a new technique to construct extractors. To state it we use the following theorem by Yao [Yao82]:

**Definition 8 (prediction tests)** *A function $P : \{0,1\}^{i-1} \to \{0,1\}$ $\epsilon$-predicts a distribution $Z$ on $\{0,1\}^m$ if $\Pr[P(Z_1, \cdots, Z_{i-1}) = Z_i] > 1/2 + \epsilon$. The distribution $Z$* passes $\epsilon$-prediction tests *if there's no $P$ that $\epsilon$-predicts it.*

**Theorem 2 ([Yao82])** *A distribution which passes $\epsilon/m$-prediction tests is $\epsilon$-close to uniform.*

In order to construct extractors it is sufficient to analyze the behavior of $E(x,\cdot)$ for fixed $x$. Note that for every $x$, the distribution $E(x,\cdot)$ cannot contain more than $d$ random bits and in particular cannot be close to uniform or unpredictable. What is required is to show that no single $P$ predicts too many of these distributions.

**Definition 9 (bad strings)** *Fix some $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, $x \in \{0,1\}^n$ is $\epsilon$-bad for $P : \{0,1\}^{i-1} \to \{0,1\}$ if $P$ $\epsilon/m$-predicts the distribution $E(x, U_d)$.*

**Lemma 1** *A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a $(t + \log(m/\epsilon), 2\epsilon)$-extractor if for every $P : \{0,1\}^{i-1} \to \{0,1\}$ the number of $x \in \{0,1\}^n$ which are $\epsilon$-bad for $P$ is at most $2^t$.*

**Proof:** Let $X$ be a distribution on $\{0,1\}^n$ with $H_\infty(X) \geq t+\log(m/\epsilon)$. We will show that $E(X, U_d)$ passes $2\epsilon/m$-prediction tests. Fix some $P : \{0,1\}^{i-1} \to \{0,1\}$. There are at most $2^t$ "bad" $x$'s on which $P$ $\epsilon/m$-predicts the distribution $E(x, U_d)$. The total weight of these "bad" $x$'s according to $X$ is bounded by $2^t \cdot 2^{-(t+\log(m/\epsilon))} = \epsilon/m$. On any other $x$, $P$ predicts the $i$'th bit with probability at most $\epsilon/m$. Thus, when $x$ is chosen according to $X$, $P$ predicts the $i$'th bit of $E(X, U_d)$ with probability at most $2\epsilon/m$. •

It will be more convenient to state this lemma in a different form. We will use a specific way to bound the number of bad $x$'s. Let $T$ be some small set. We will require that for every predictor $P$, there is a mapping $F_P : \{0,1\}^n \to T$ such that $F_P$ is one to one on $x$'s which are bad for $P$. This indeed insures that there are few bad $x$'s for every $P$. A way to show that $F_P$ is one to one on bad $x$'s is to show that it has an inverse function $R_P$ which reconstructs a bad $x$ from $F_P(x)$. Thus, the following lemma follow from lemma 1.

**Definition 10 (reconstruction)** *Given a function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, a $(t,\epsilon)$-*reconstruction *for $E$ is a collection of functions $(F_P, R_P)$, where for every $1 \leq i \leq m$ and every $P : \{0,1\}^{i-1} \to \{0,1\}^i$, $F_P : \{0,1\}^n \to \{0,1\}^t$, $R_P : \{0,1\}^t \to \{0,1\}^n$ and for every $x \in \{0,1\}^n$ which is $\epsilon$-bad for $P$:*

$$R_P(F_P(x)) = x$$

**Corollary 1 (Reconstruction is sufficient for extractors)** *A function $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ that has a $(t,\epsilon)$-reconstruction is a $(t + \log(m/\epsilon), 2\epsilon)$-extractor*

To summarize, given a function $E$ with an $(t,\epsilon)$-reconstruction we obtain an extractor with min-entropy threshold $k \approx t$. We will use this approach to describe many extractor constructions.

**Remark 3** *In corollary 1 it is necessary to have a reconstruction for predictors which $\epsilon/m$-predict the distribution $E(x,\cdot)$ to get an extractor with error $\epsilon$. This is caused by the hybrid argument in Yao's theorem (theorem 2). Ta-Shma, Zuckerman and Safra [TSZS01] give a way to avoid this loss and construct extractors using reconstruction for predictors which only $\epsilon$-predict $E(x,\cdots)$. Stating their result in a general form is somewhat complicated. The main idea is to show that given an $\epsilon$-reconstruction for $E$, the output distribution of $E$ is $O(\epsilon)$-close to having min-entropy $\Omega(m)$.*

### 3.4 The connection to list-decodable error correcting codes

Consider the following function $\hat{E} : \{0,1\}^{\hat{n}} \times \{0,1\}^{log\hat{n}} \to \{0,1\}^{\log \hat{n}+1}$, $\hat{E}(\hat{x}, y) = y \circ \hat{x}_y$. Let's try to show a reconstruction for $E$. As the first $\log n$ bits of the output of $E$ are truly random, we only need to handle predictors $P(y)$ which attempt to predict $\hat{x}_y$ from $y$. Such a predictor $P$ can be thought of as a string $p \in \{0,1\}^{\hat{n}}$, where $p_y = P(y)$. $P$ $\epsilon$-predicts $E(x, \cdot)$ if and only if the relative Hamming distance[10] between $p$ and $\hat{x}$ is smaller than $1/2 - \epsilon$. The task of the "reconstruction function" $R_P$ is very similar to that of decoding an error correcting code: It needs to reconstruct $\hat{x}$ given a string $p$ which is close to it. Consider the following modification: $E(x, y) = \hat{E}(\hat{x}, y)$ where $\hat{x}$ is an encoding of $x$ using an error correcting code. By the discussion above decoding the "corrupted" $p$ to obtain $x$ gives an $(0, \epsilon)$-reconstruction.

A complication is that if $\epsilon < 1/4$ then $p$ is a too noisy version of $\hat{x}$ and it is impossible to correct so many errors. This leads to a weaker notion of decoding, which will be sufficient for our purposes.

**Definition 11 (List decodable codes [Sud97])** *A mapping $Ecc : \{0,1\}^n \to \{0,1\}^{\hat{n}}$ is an $(\epsilon, \ell)$-list decodable error correcting code if for every $p \in \{0,1\}^{\hat{n}}$, the set*

$$L_p = \{x | Ecc(x) \textit{ and } p \textit{ have relative hamming distance smaller than } 1/2 - \epsilon\}$$

*is of size at most $\ell$. (We refer to the set $L_p$ as the* list of decodings *of $p$).*

In the standard notion of decoding error correcting codes, $L_p$ is a singleton. In other words, given a corrupted codeword $p$, it is possible to uniquely decode and identify the sent message. List-decodable codes only guarantee that given the corrupted codeword $p$, the sent message $x$ appears in the list of possible decodings of $p$. There are explicit constructions of $(\epsilon, 1/\epsilon^2)$-list decodable codes with $\hat{n} = n/\epsilon^{O(1)}$.

List decoding suffices for our purposes as our final goal is to bound the number of "bad" $x$'s. More precisely, an $(\epsilon, \ell)$-list decodable code gives an $(\log \ell, \epsilon)$-reconstruction for the function $E$ above: We define $F_P(x)$ to be the index of $x$ in the list of decodings of $p$. Thus, $t = \log |L_p|$. The function $R_P$ works by first "computing" $L_p$, and then using $F_P(x)$ to output $x$.

The construction sketched here already gives a non-trivial extractor, however it is only able to extract one bit more than it spends.

### 3.5 Extractors using the Nisan-Wigderson generator

A way to extract more bits is to output the content of $\hat{x}$ in many positions. We will use a $(\epsilon/m, (m/\epsilon)^2)$-list decodable codes $Ecc : \{0,1\}^n \to \{0,1\}^{\hat{n}}$. Let $\hat{x} = Ecc(x)$ denote the encoding of $x$. There are explicit constructions of such codes with $\hat{n} = (n/\epsilon)^{O(1)}$. Consider the following function:

$$E(x; y_1, \cdots, y_m) = y_1, \cdots, y_m \circ \hat{x}_{y_1}, \cdots, \hat{x}_{y_m}$$

This approach produces an extractor with very long seed $(y_1, \cdots, y_m)$. We will fix this problem later and show how to "generate" $y_1, \cdots, y_m$ from a short string $y$. We start by giving a reconstruction for $E$.

Let $x \in \{0,1\}^n$ be an arbitrary string. Fix $i$ and a predictor $P : \{0,1\}^{i-1} \to \{0,1\}$ which is $\epsilon$-bad for $x$. In other words, $P$ $\epsilon/m$-predicts $E(x; Y_1, \cdots, Y_m)$ (where $x$ is fixed and $Y_1, \cdots, Y_m$

[10] The relative Hamming distance between two strings of the same length is the fraction of indices in which they differ.

are uniformly distributed). In order to use corollary 1, we need to show the existence of functions $F_P, R_P$ with $R_P(F_P(x)) = x$.

As $Y_1, \cdots, Y_m$ are independent there exists fixings $y'_1, \cdots, y'_{i-1}, y'_{i+1}, \cdots, y'_m$ to $Y_1, \cdots, Y_{i-1}, Y_{i+1}, \cdots, Y_m$ such that $P$ $\epsilon/m$-predicts $E(x, y'_1, \cdots, y'_{i-1}, Y_i, y'_{i+1}, \cdots, y'_m)$.[11] In other words, on input

$$(y'_1, \cdots, y'_{i-1}, y, y'_{i+1}, \cdots, y'_m \circ \hat{x}_{y'_1}, \cdots, \hat{x}_{y'_{i-1}})$$

$P$ predicts $\hat{x}_y$ correctly for a $1/2 + \epsilon/m$ fraction of $y$'s. We let

$$F_p(x) = (y'_1, \cdots, y'_{i-1}, y'_{i+1}, \cdots, y'_m \circ \hat{x}_{y'_1}, \cdots, \hat{x}_{y'_{i-1}})$$

To complete the argument we need to reconstruct $x$ from $F_P(x)$. This is done just like in the previous section by using $P$. we construct a string $p \in \{0,1\}^{\hat{n}}$,

$$p_y = P(y'_1, \cdots, y'_{i-1}, y, y'_{i+1}, \cdots, y'_m \circ \hat{x}_{y'_1}, \cdots, \hat{x}_{y'_{i-1}})$$

It follows that the relative Hamming distance between $p$ and $\hat{x}$ is small and we can use list-decoding to reconstruct $x$. (As in the previous section we need to also "append" another short string to $F_P(x)$: the index of $x$ in the list $L_p$ of decodings of $p$.

**The Nisan-Wigderson generator:** To reduce the seed length we will generate "sufficiently independent" $y_1, \cdots, y_m$ from much fewer bits using the Nisan-Wigderson generator [NW94].

**Definition 12 (weak design [RRV99b])** [12] *A collection of sets $S_1, \cdots, S_m \subseteq [d]$ is a $(l, \rho)$-weak design if for every $i$:*

1. $|S_i| = l$.
2. $\sum_{j<i} 2^{|S_i \cap S_j|} \leq \rho(m-1)$.

Given a $(\log \hat{n}, \rho)$-weak design, a seed $y \in \{0,1\}^d$ is used to compute $y_1, \cdots, y_m$ by $y_i = y|_{S_i}$. More formally:

$$E(x, y) = y \circ \hat{x}_{y|_{S_1}}, \cdots, \hat{x}_{y|_{S_m}}$$

Note that now we don't include $y_1, \cdots, y_m$ in the output. Instead, we put $y$ in the output. This is because $y_1, \cdots, y_m$ are no longer uniformly distributed.

**Theorem 3 ([Tre99])** *$E$ is a $(\rho m + O(\log(m/\epsilon)), O(\epsilon))$-extractor*

The proof follows the argument given above and gives an $(\rho m + O(\log(m/\epsilon)), \epsilon)$-reconstruction for $E$. Unlike the situation in the beginning of the section, $Y_1, \cdots, Y_m$ are dependent. Mow, it isn't possible to fix all the $Y_1, \cdots, Y_{i-1}, Y_{i+1}, \cdots, Y_m$ and still have that $Y_i$ is uniformly distributed. Nevertheless, we can fix the value of the $\ell$'th coordinate of $y$ (for all $\ell \notin S_i$). This leaves $Y_i$ uniformly distributed. While the remaining $Y_j$'s are not completely fixed, it can be shown that they are limited to a small set of values. This follows from the properties of weak designs.

[11] Actually, this argument shows more than existence. A random fixing $y'_1, \cdots, y'_{i-1}, y'_{i+1}, \cdots, y'_m$ is good with non-negligible probability. More precisely, for an $\epsilon/2m$ fraction of the fixings, $P$ $\epsilon/2m$-predicts the fixed distribution. We will make use of this observation in section 4.1.

[12] Nisan and Wigderson defined a slightly different object called a *design*, with a stronger second property saying that for every $i \neq j$, $2^|S_i \cap S_j| \leq \rho$. The weaker notion suffices for their argument and was introduced in [RRV99b].

More precisely, as $Y_i$ varies, the random variable $Y_1, \cdots, Y_{i-1}, \cdots, Y_{i+1}, \cdots, Y_m$ takes values in a set of size $\rho m$. This follows from the second requirement in the definition of weak designs. (Note that for every $j < i$, the fixing has left only $|S_i \cap S_j|$ indices of $Y_j$ free, and thus $Y_j$ takes only $2^{|S_i \cap S_j|}$ different values). $F_P(x)$ is now constructed as follows: For all $j < i$, and for all possible values $a$ of $Y_j$ after the fixing, we append $\hat{x}_a$ to $F_P(x)$. The argument can continue just as before, as $F_P(x)$ and $P$ uniquely define the string $p$.

**Remark 4** *What is the relation between the presentation of Trevisan's extractor given here, and the pseudo-random generator based extractor presented in section 3.2? The Impagliazzo-Wigderson pseudo-random generator construction is exactly the construction presented here, except for the fact that Ecc is a very specific error correcting code which allows "sub-linear time list-decoding". (It should be noted that this is not the way the construction is presented in the Impagliazzo-Wigderson paper [IW97]. The observation that [IW97] constructs a list-decodable error correcting codes was explicitly pointed out by Sudan, Trevisan and Vadhan [STV99] who showed that any list-decodable code with "sub-linear time list-decoding" suffices, and gave a "sub-linear time" algorithm for list-decoding the "low-degree extension" code). In [Tre99] (which actually appeared before [STV99]) Trevisan pointed out that the very complicated error correcting code constructed by Impagliazzo and Wigderson (which involved the "low-degree extension" as well as two "derandomized XOR-lemmas") can be replaced by any list-decodable code for the purpose of constructing extractors.*

### 3.6 Choosing the parameters

The quality of the extractor in theorem 3 depends on the quality of the weak design. It is desired to have weak designs in which $d \geq l$ and $\rho \geq 1$ are as small as possible.

Raz, Reingold and Vadhan [RRV99b] gave an explicit construction of $(\ell, \rho)$-weak with $d = O(\frac{\log^2 l}{\log \rho})$, as well as almost matching lower bounds.

Note that in the construction $\ell = \log \hat{n} = O(\log(n/\epsilon))$. For simplicity, let us fix $\epsilon$ to be some small constant.

**Optimizing the seed length:** As $d = O(\log^2 n/\log \rho)$, in order to get $d = O(\log n)$ we have to set $\rho = n^{\Omega(1)}$. This makes $t \geq n^{\Omega(1)}$ (as $t \approx \rho m$) and gives an extractor in which $m = k/n^{\Omega(1)}$ and in particular the extractor only works when $k = n^{\Omega(1)}$.

To get an extractor for $k = n^{o(1)}$ we choose $\rho = k^{\delta'}$ for some small constant $\delta' > 0$. This leads to a larger seed of length $O(\log^2 n/\log k)$ and output length $k^{1-\delta}$ for some constant $\delta > 0$. Both these results were achieved by Trevisan in [Tre99].

By the lower bound of [RRV99b], it is impossible to get a seed of length $O(\log n)$ when $k = n^{o(1)}$ by improving the quality of the weak-design. Nevertheless, a more sophisticated application of the reconstruction argument achieves this goal as we'll see in section 4.1.

**Optimizing the output length:** As $k \approx t \approx \rho m$, in order to get $m = \Omega(k)$ we have to set $\rho = O(1)$. We now have to set $d = O(\log^2 n)$. This was first achieved by Raz, Reingold and Vadhan [RRV99b] who introduced weak-designs to replace a more restricting notion which did not allow $\rho = O(1)$ when $m$ is large.

# 4 Condensers

A condenser is a weakening of an extractor. Whereas an extractor outputs a distribution which is close to uniform, a condenser is only required to output a distribution (which is close to having) high min-entropy. Naturally, we want the entropy-rate of the output distribution to be larger than that of the input distribution. Several variants of condensers appeared in different works. The following definition is taken from [RSW00].

**Definition 13 (condenser)** *A $(k, k', \epsilon)$-condenser is a function $Con : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{n'}$, such that for every distribution $X$ on $\{0,1\}^n$, the distribution $Con(X,Y)$ (where $Y$ is uniformly distributed in $\{0,1\}^d$) is $\epsilon$-close to a distribution $X'$ with $H_\infty(X') \geq k'$.*

Note that an extractor is a special case of a condenser, when $n' = k'$.

## 4.1 Using the reconstruction technique

Corollary 1 shows that a $t$-reconstruction gives an extractor for $k$ slightly greater than $t$. As noted in section 3.6, this is problematic when $k$ is small (say $k = n^{o(1)}$) as it requires that $t$ is also small. Impagliazzo, Shaltiel and Wigderson [ISW99, ISW00] and later Ta-Shma, Umans and Zuckerman [TSUZ01] used a $t$-reconstruction to construct extractors for $k < t$. The presentation here follows the [TSUZ01] paper.

The main idea is to use the reconstruction technique to construct condensers. Let $E : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ be the extractor construction of the previous section. In the previous section we gave a $t$-reconstruction for $E$ (with $d = O(\log n)$ and $t = n^\delta m$ for a small constant $\delta$). An examination of the proof reveals that the functions $F_P$ depends on the predictor $P$ in a very weak way: For every $P$, the existence of the function $F_P$ was shown by selecting $F_v$ at random from some small collection of functions $\{F_v\}_{v \in \{0,1\}^d}$ in a way which does not depend on $P$. (Recall that the function $F_P$ was determined by fixing some of the indices of the seed $y$. For every predictor $P$, with high probability a random fixing $v$ yields a function $F_v : \{0,1\}^n \to \{0,1\}^t$ which is "good" for $P$. Here "good" means that there exists a function $R_P : \{0,1\}^t \to \{0,1\}^n$ such that on every $x$ that is bad for $P$, $R_P(F_v(x)) = x$.) We define $Con(x,v) = F_v(x)$. We will use the reconstruction technique to show that $Con$ is a condenser.

Let $X$ be a source with $H_\infty(X) = k$, for $k < m \leq t$. Choosing $k < m$ makes little sense in extractors. How can an extractor output more random bits than present in its input? The answer is that it cannot. Therefore, there must be a predictor $P$ which predicts $E(X, U_d))$ and therefore with high probability predicts $E(x, U_d)$ when randomly choosing $x$ from $X$. Thus, with high probability (over choosing $x$ from $X$ and $v$ from $V$) $R_P(F_v(x)) = x$ which means that $F_v(x)$ "contains all the information of $x$". This entails that $Con(X, U_d)$ is (close to) having all the randomness of $X$.[13]

**Remark 5** *There's a slight cheating in this presentation. We claimed that $F_P$ doesn't depend on $P$. However, the index of $x$ in the list of decodings of $p$, which appears in $F_P(x)$,* does *depend on $P$. Nevertheless, in this setup we can use (standard) unique decoding instead of list decoding. It can be shown that when $m >> k$ there exists a predictor $P$ with success greater than* 3/4*. Such a predictor defines a string $p$ which is relatively close to $\hat{x}$, and uniquely specifies $\hat{x}$.*

[13] The same argument also shows that $Y \circ Con(X,Y)$ is close to having $k + d$ random bits. Note, that there's no entropy loss here. All the randomness is "extracted".

**Getting an extractor** The argument above only gives a condenser. Setting the parameters as in section 3.6 gives a $(k, k, \epsilon)$-condenser $Con : \{0,1\}^n \times \{0,1\}^{O(\log n)} \rightarrow \{0,1\}^{k^{O(1)}}$ (for constant $\epsilon > 0$).[14] Thus, by running the condenser on the given source, we reduces the problem of constructing extractors for threshold $k = n^{o(1)}$ into that of constructing extractors for threshold $k = n^{\Omega(1)}$. In this range, Trevisan's extractor has short seed length. This gives an extractor with seed length $O(\log n)$ for $k = n^{o(1)}$ and was first achieved in [ISW00].

Using this reduction, we actually get better extractors when $k$ is small, (say $k \le 2^{log^{1/3} n}$). This is because after running the condenser we can use extractors with $d = O(\log^3 k) = O(\log n)$ to extract *all* the randomness from the source, without paying a penalty in the seed length, this was achieved in [TSUZ01] improving on a weaker result by [ISW00].

One of the applications of extractors for small min-entropy threshold is a construction of very good dispersers (for general min-entropy threshold) [TSUZ01]. These dispersers have seed length $O(\log n)$ and entropy-loss $O(\log n)$.

## 4.2 Condensers achieved by attempting to construct block wise sources

In section 2.2 (see remark 2.2) we've seen that it is sufficient to "transform" a (general) source into a block-wise source to construct extractors. (The quality of the constructed extractors constructed depend on the quality of the transformation, and on the quality of the extractors used to extract from the block-wise source.) Nisan and Zuckerman [NZ96] gave such a transformation. That is an explicit construction of a function

$$T : \{0,1\}^n \times \{0,1\}^u \rightarrow \{0,1\}^{n/r} \times \{0,1\}^n$$

which on a source $X$ and random seed $y \in \{0,1\}^u$ produces distributions $X_1, X_2$ which form a block-wise source.[15] The proof works by showing that for every source $X$ with $H_\infty(X) \ge k$:

1. $X_1$ (is close to) containing approximately $k/r$ random bits. (The exact bound is $\Omega(k/r \log(n/k))$).
2. The joint random variable $(X_1, X_2)$ contains $k$ random bits.

Loosely speaking, $X_1, X_2$ form a block source with min-entropy $k_1, k_2$ if $X_1$ contains $k_1$ random bits and $X_2$ contains $k_2$ random bits which are not contained in $X_1$. To conclude that $X_1, X_2$ form a block source, Nisan and Zuckerman set up the parameters so that $k$ is sufficiently larger than $n/r$. By the first property of $T$, $X_1$ contains approximately $k/r$ random bits. $X_2$ must contain $k - n/r$ bits which are not contained in $X_1$ as $X_1$ is too short to "steal" all the $k$ bits of randomness. Note that we want both $k/r > 1$ and $k - n/r > 1$ which forces $k > \sqrt{n}$ and this method is not applicable when $k$ is small.

The following idea appeared in [SZ99, NTS99]. (For simplicity let's set $r = 2$ and thus $X_1$ is of length $n/2$.) Suppose that $k$ is small: $k << n/r = n/2$. We are not guaranteed that $X_1, X_2$ form a block source. Nevertheless, if they don't, this is because $X_1$ "stole" all the randomness of $X$. Thus, $X_1$ is more condensed than $X$ as it contains all the randomness of $X$ in half the length. This suggests the following condenser construction: When given a source element $x$, use a short

[14]Actually, as $t = mn^\delta$, one only gets a condenser with $n' \approx kn^\delta$. The condenser reported above is achieved by running the condenser $\log\log n$ times. Note that the seed lengths are exponentially decreasing, and thus the the total length for the $\log\log n$ seeds is still $O(\log n)$.

[15]The function $T$ chooses $n/r$ indices of $x$ in a random way to give $X_1$, and $X_2 = X$ to make sure that no randomness is lost in this process. Intuitively, one hopes that taking a $1/r$-fraction of the indices of the source gives a source with $k/r$ random bits.

seed to run $T$ and get $x_1$ and $x_2$. Use an additional short seed to run a block-wise source extractor on $x_1$ and $x_2$ to obtain $z$, and output $(z, x_1)$ which is of length at most $n/2 + k \approx n/2$. The rational is that if $X_1, X_2$ form a block-wise source then $z$ is close to uniform and the output is more condensed than the initial distribution. On the other hand, if $X_1$ and $X_2$ do not form a block-wise source, then $X_1$ "stole" all the initial randomness and the output is also more condensed. This idea enabled [SZ99] to give an almost polynomial time simulation of RP with a weak random source of low min-entropy threshold, and was later used in [NTS99] to construct extractors.

Intuitively, a weakness of this method, is that the condenser only condenses the initial distribution by a little. (In the presentation above $n' \approx n/2$). To get a better condenser (say with $n' = O(k)$) one has to run the basic condenser many times, and use many independent seeds for $T$.

Reingold, Shaltiel and Wigderson [RSW00] improved the construction of Nisan and Zuckerman, giving an explicit function $T$ which uses a much shorter seed $u$, ($O(\log\log n)$ instead of $O(\log n)$) and is able to maintain a constant fraction of the initial randomness ($\Omega(k)$ instead of $\Omega(k/\log(n/k))$. This allows repeated condensing at a smaller cost which translates into a better extractor construction.

# 5 Extractor based on multivariate polynomial codes

In section 3.4 we observed that using the reconstruction method to construct extractors is related to (list)-decoding error-correcting codes. It turns out that using properties of specific error-correcting codes (based on multivariate-polynomials) can be helpful in extractors constructions.[16] Ta-Shma, Zuckerman and Safra [TSZS01] suggested this approach and gave an extremely clean and simple extractor construction based on Reed-Muller codes.

## 5.1 The intuition

The reconstruction proof technique suggests the following construction: We denote

$$N(y) = (y+1)(\bmod n)$$

$N^{(j)}(y)$ denotes $N$ successive applications of $N$ and $E : \{0,1\}^n \times \{0,1\}^{\log n} \to \{0,1\}^m$ is defined by:

$$E(x,y) = x_y, x_{N(y)}, x_{N(N(y))}, \cdots, x_{N^{(m-1)}(y)}$$

Intuitively, this should allow a very simple reconstruction: For every predictor $P : \{0,1\}^{i-1} \to \{0,1\}$, let $F_P(x) = x_1, \cdots, x_{i-1}$. For every $x$, saying that $P$ 1-predicts the distribution $E(x,\cdot)$ means that for every $z$:

$$P(x_{N^{-(i-1)}(z)}, x_{N^{-(i-2)}(z)}, \cdots, x_{N^{-(1)}(z)}) = x_z$$

Thus, given $F_P(x)$ we can use $P$ to compute $x_i$, and then use $F_P(x)$ and $x_i$ to compute $x_{i+1}$, and gradually reconstruct $x$ at every point $z$. This gives a function $R_P$ such that $R_P(F_P(x)) = x$ as required.

This argument fails because we need to give a reconstruction for predictors $P$ which predict the next bit only with probability $1/2 + \epsilon/m$. Thus, the predictor is only successful on a $1/2 + \epsilon/m$-fraction of points $z$, and once we reach a point $z$ when it fails the reconstruction process cannot continue. Ta-Shma, Zuckerman and Safra [TSZS01] used ideas from [STV99] to "error correct" the predictor's output and reconstruct $x$ at every point.

[16] In addition to the constructions covered here, the first use of specific properties of multivariate polynomial codes was made in [RRV99b] to reduce the error in Trevisan's extractor.

### 5.2 Extractors using polynomials with two variables

**$q$-ary extractors:** We will be using multivariate polynomials over a field $F$ of size $q$. It will be more natural to construct a variant of extractors (called $q$-ary extractors) and then transform them into regular extractors.

**Definition 14 ($q$-ary extractor)** *Let $F$ be a field with $q$ elements. A $(k, \epsilon)$ $q$-ary extractor is a function $E : \{0,1\}^n \times \{0,1\}^d \to F^m$ Such that for every distribution $X$ on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the distribution $Ext(X, U_d)$ is $\epsilon$-unpredictable.*

*(In this setup a predictor $P$ is a function $P : F^{i-1} \to F$ and $P$ $\epsilon$-predicts a distribution $Z_1, \cdots, Z_m$ if $\Pr[P(Z_1, \cdots, Z_{i-1}) = Z_i] \geq \epsilon$. A distribution is $\epsilon$ unpredictable if no $P$ $\epsilon$-predicts it.)*

Binary list-decodable error correcting codes can be used to explicitly transform a $(k, \epsilon/m)$ $q$-ary extractor $E : \{0,1\}^n \times \{0,1\}^d \to F^m$ into a (regular) $(k, O(\epsilon))$ extractor $E' : \{0,1\}^n \times \{0,1\}^{d'} \to \{0,1\}^m$ where $d' = d + \log\log q + O(\log(m/\epsilon)$. In other words, given a $q$-ary extractor for sufficiently small $\epsilon$ we get a regular extractor with essentially the same parameters.

We will be using the reconstruction proof technique, which also works in this setup. In analogy to corollary 1 it follows that if $E$ has a $(t, \epsilon)$-reconstruction then $E$ is a $(t + \log(1/\epsilon), O(\epsilon))$ $q$-ary extractor.

**The construction:** We now present the construction of Ta-Shma, Zuckerman and Safra [TSZS01]. Let $F$ be a field with $q$ elements. The Reed-Muller code maps strings $x \in \{0,1\}^n$ into $\ell$-variate polynomials $\hat{x}$ of degree $h$ over $F$. This is done by using $x$ to define the coefficients of $\hat{x}$. As an $\ell$-variate polynomial of degree $h$ has $\binom{h+\ell}{\ell}$ coefficients, the mapping $x \to \hat{x}$ is one to one if $\binom{h+\ell}{\ell} \log q \geq n$. In this section we fix $\ell = 2$ and assume that $q$ is relatively small (say $q = n^{O(1)}$), this gives $h \approx \sqrt{n}$.

Consider the following function: $E : \{0,1\}^n \times F^2 \to F^m$.

$$E(x, y) = \hat{x}(y), \hat{x}(N(y)), \hat{x}(N(N(y))), \cdots, \hat{x}(N^{(m-1)}(y))$$

(Here $N : F^2 \to F^2$ is given by $N(a_1, a_2) = (a_1 + 1, a_2)$ and again $N^{(j)}$ denotes $j$ successive applications of $N$). We refer to $N$ as the successor function.

**The reconstruction:** We will give a reconstruction for $E$ following the outline in section 5.1. For every $x \in \{0,1\}^n$, we think of a predictor $P : F^{i-1} \to F$ which $\epsilon$-predicts $E(x, \cdot)$ as if for every $z$, $P$ attempts to predict $\hat{x}(z)$ given $N^{-(i-1)}(z), N^{-(i-2)}(z)), \cdots, N^{-1}(z)$ and succeeds for an $\epsilon$-fraction of the $z$'s. We call those strings "good" $z$'s.

We choose a random line $L$ in $F^2$, and let $L(1), \cdots, L(q)$ denote the $q$ points on that line. As $L(1), \cdots, L(q)$ are pairwise independent, Chebichev's inequality guarantees that the fraction of "good" points on the line is with high probability very close to the fraction of good points in $F^2$. More precisely, with probability roughly $1/q$ (over the choice of $L$) there are $q\epsilon/2$ "good" points on the line.

We define $F_P(x)$ to be the evaluation of $\hat{x}$ on the $i-1$ lines $N^{-1}(L), \cdots, N^{-(i-1)}(L)$. (More formally, $F_P(x)$ contains the evaluation of $\hat{x}$ on all points $N^{-j}(L(u))$ for $1 \leq j \leq i-1$ and $1 \leq u \leq h+1$. Note that it is enough to use only $h+1 < q$ points as $\hat{x}$ restricted to a line is a univariate polynomial of degree $h$ and is determined by $h+1$ points.)

As in section 5.1, given $F_P(x)$ we can use $P$ to attempt to predict the evaluation of $\hat{x}$ on $L(1), \cdots, L(q)$. The advantage of our current setup is that the polynomial $\hat{x}$ restricted to $L$ is a low-degree univariate polynomial $p$. Intuitively, when given "noisy" evaluations of a low-degree univariate polynomial it is possible to uniquely decode and find the correct polynomial. Thus, we can now "learn" the *correct* evaluation of $\hat{x}$ on all points on $L$. By applying this "decoding step" we overcome the errors of the predictor $P$. We can now continue and "learn" the evaluations of $\hat{x}$ on the line $N(L)$, (The line which consists of $N(L(1)), \cdots, N(L(q))$). This line is by itself uniformly distributed (Although it is uniquely determined from $L$) and the same argument applies. By "learning" the evaluations of $\hat{x}$ on $L, N(L), N(N(L)), \ldots$ we reconstruct $\hat{x}$ at every point. Note that $\hat{x}$ uniquely specifies $x$. We have a small probability (roughly $1/q$) of failing on each individual line, and thus by choosing a large enough $q$, we can do a union bound on all these failure probabilities and conclude that there exists a line $L$ on which the reconstruction process never fails.

There's a slight problem in the presentation above. In our setting the evaluations we get on $L$ are extremely noisy and it is (information theoretically) impossible to uniquely decode. Nevertheless, this is solved in an analogous way to what we did in the previous section by using the list-decoding properties of low-degree univariate polynomials. Sudan [Sud97] showed that the Reed-Solomon code is list-decodable, which in our setting translates into saying that the answers of the predictor $P$ on $L$ specify a small list of univariate polynomials $p_1, \cdots, p_\ell$ such that one of them is $p$. We append the index of $p$ in this list to $F_P(x)$. (Note that we have to append such an index for all lines $L, N(L), N(N(L)), \cdots$). (Jumping ahead, in the next section it will be convenient to use a slightly different strategy, and append to $F_P(x)$ the evaluation of $\hat{x}$ on $L(w)$ where $w$ is a random point on $L$. With high probability, $p(w) \neq p_j(w)$ for all the "incorrect" $p_j$'s.)

In addition to simplicity, an advantage of this construction is that the seed length is extremely short: Not $O(\log n)$ but rather $(1 + o(1)) \log n$. It is possible to also get $m = \Omega(k)$ (for $k = \Omega(n)$) by combining this extractor with the transformation $T$ of regular sources into block-wise sources given in [RSW00] (see section 4.2).

### 5.3 Extractors using polynomials with many variables

The extractor of the previous section used polynomials $\hat{x}$ with $\ell = 2$ variables. Encoding an $n$ bit string by such a polynomial requires degree $h \approx \sqrt{n}$. (Recall that we had to satisfy $\binom{h+\ell}{\ell} \log q = n$). The final reconstruction has $t \geq hm$ as $F_P(x)$ contains the evaluation of $\hat{x}$ on $m$ successive lines. A $t$-reconstruction gives an extractor with $k \approx t$. The bottom line is that the extractor we get has $m \approx k/\sqrt{n}$ and in particular only works when $k \geq \sqrt{n}$.

A natural way to improve this extractor is to increase $\ell$: the number of variables of $\hat{x}$. In general, $h$ (the degree of $\hat{x}$) reduces to $h \approx n^{1/\ell-1}$ when using $\ell$-variate polynomials. We may hope to decrease $t \geq mh$ and construct better extractors by increasing $\ell$.

It is important to notice that the seed $y$ is an input to the polynomial $\hat{x}$ and is of length $\ell \log q$. Thus, when increasing $\ell$ we have to decrease $q$, and in particular to have seed $O(\log n)$ we must have $q = h^{O(1)} \approx n^{1/\ell}$.

A construction using polynomials with many variables was given by Shaltiel and Umans [SU01].[17] The straightforward way of increasing $\ell$ in the construction of [TSZS01] actually *increases* $t$. The following modifications are made:

**An algebraic approach.** What does "successive" mean when $\ell$ is large? In the previous section $N(a_1, a_2)$ is $(a_1 + 1, a_2)$. The rational is that starting from a line $L$ and taking successive steps one

[17] The [TSZS01] paper also has a construction for large $\ell$ which achieves weaker extractors.

covers the whole space. However, this geometric approach is problematic.[18] It is replaced by an algebraic ingredient. The vector-space $F^\ell$ is viewed as the extension field of $F$. The multiplicative group of this field has a generator $g$, and $N(v)$ is defined to be $g \cdot v$. This indeed has the essential property that by repeatedly taking successors, we cover the whole space. $N$ is also a linear transform (over the vector-space) and thus lines are mapped into lines.

We can use the construction and reconstruction of the previous section with this new function $N$. The argument can be carried out just the same. The difficulty, is to make the proof work when increasing $\ell$. The main problem is that in every "prediction step" we learn the evaluation of $\hat{x}$ on one line. Each line consists of $q$ points, and thus to learn $n$ bits of information we must have at least $n/q$ predictions steps. As $q \approx n^{1/l}$, the number of prediction steps we need is huge when $\ell$ is increased.

**Curves instead of lines.** Recall that every time we "learn" a new line, the probability of failure is roughly $1/q \approx 1/n^{1/\ell}$. When $\ell$ is increased, we have too many lines, and thus to many "bad events" in the union bound.

This is overcome by replacing lines $L$ with *degree $r$ curves* $C$.[19] The points $C(1), \cdots, C(q)$ are $(r+1)$-wise independent and Chebichev's inequality can be replaced by higher moment tail inequalities in which the failure probability decreases exponentially in $r$. The failure probability of each event in the union bound can be decreased by increasing $r$, and the union bound holds. Curves can replace lines in the proof as the multivariate polynomial $\hat{x}$ restricted to a low-degree curve is also a low-degree univariate polynomial, and $N$ maps low-degree curves to low-degree curves.

**Interleaved reconstruction procedures.** There's an additional cost to using more prediction steps. Recall that $F_P(x)$ contains an evaluation of $\hat{x}$ at a random point on the curve for any successive curve of $C$. As we need at least $n/q \approx n^{1-1/\ell}$ such successive curves to cover the whole space, we get that $t$ (the range of $F_P$) is huge, and we haven't gained anything.

To overcome this problem we run two "interleaved" reconstruction procedures. Each uses its own random curve but we arrange it so that the two curves intersect at a few random points. The two reconstruction procedures work on their own. However, when one needs the value of the polynomial at a random point on its curve, it can use the value *already calculated* by the other reconstruction procedure instead of relying on $F_P(x)$. Thus, $F_P(x)$ contains only the initial evaluations of $\hat{x}$ needed to get the two interleaved reconstruction procedures started.

More formally, let $C_1$ be a random degree $r$ curve chosen by choosing $r+1$ random values in $F^\ell$ and passing a curve through them. The curve $C_2$ is defined to be the degree $r$ curve that coincides with $C_1$ in half of the $r$ points, and coincides with $N(C_1)$ on the other half of the $r$ points. Note that $C_2$ is also a random curve. We also get that $C_1$ intersects $C_2$ at $r/2$ random points, and $C_2$ intersects $N(C_1)$ at $r/2$ random points. Thus, every time one of the curves is advanced, we already know the evaluation of $\hat{x}$ on (many) random points on the curve.

[18] When successive points are on a line, then the $m$ output evaluations are evaluations of a low-degree univariate polynomial. Thus, it is impossible to reduce the degree of the polynomial $\hat{x}$ below $m$ (as otherwise a predictor for the extractor will interpolate given some prefix of the output of the extractor and will be able to predict the next evaluations.)

[19] A line is a degree one polynomial $L : F \to F^\ell$, and an $r$ degree curve $C$ is a degree $r$ polynomial $C : F \to F^\ell$.

# 6 Transformations and tradeoffs

## 6.1 Increasing the output length and achieving optimal entropy loss

Consider an extractor which extracts $m < k$ bit from a source $X$. Intuitively, as the extractor did not extract all the randomness there is still some randomness left in the source. Formally, conditioned on the extractor's output the source still contains $k - m$ bits of randomness. This randomness can be extracted using another extractor. (It is important to note that this extractor should have a min-entropy threshold of $k - m$ and not $k$.) These bits are independent of the ones initially extracted. The penalty of this method is that two independent seeds are needed for the two extractors. This method was first used by Wigderson and Zuckerman [WZ99]. (Better analysis of this method is given in [RRV99b]).

Applying this idea gives the following tradeoffs between seed length and output length: For any constant $\delta > 0$, this transformation can be used to transform a $(\delta k, \epsilon)$-extractor with $m = k/r$ into a $(k, O(r\epsilon))$-extractor with $m' = (1-\delta)k$ at the cost of increasing the seed to $d' = O(dr)$ (here the hidden constant depends on $\delta$). Thus, an extractor with $m = \Omega(k)$ can be transformed into one with $m' = (1-\delta)k$ multiplying the seed length by a constant.

This method can also be used to go from $m = (1-\delta)k$ to $m' = k$ multiplying the seed length by $O(\log k)$, (this step requires a family extractors with varying min-entropy threshold, as the amount of randomness in the source vanishes during the repeated extraction process).

Using this method in conjunction with optimal entropy loss extractors for low min-entropy given by [GW97, SZ99], Raz, Reingold and Vadhan [RRV99b] showed how to transform any extractor with entropy loss $\Delta$ into one with optimal entropy loss (that is $m = k + d - 2\log(1/\epsilon) - O(1)$) adding $O(\Delta + \log(1/\epsilon)$ bits to the seed.

## 6.2 Error reduction in extractors

Raz, Reingold and Vadhan [RRV99a] gave a general way to transform extractors with large error into extractors with smaller error. Given a $(k, 1/m)$-extractor $Ext$ with seed length $d$ and output length $m$, for every $\epsilon' > 0$ (not necessarily a constant) they explicitly construct a $(k+O(\log(1/\epsilon'), \epsilon')$-extractor $Ext'$ with seed length $d' = d+O(\log(1/\epsilon'))$ and output length $m' = \Omega(m)-O(\log(1/\epsilon'))$.[20] Note that by the lower bounds of [RTS00] any extractor with error $\epsilon'$ must have $d \geq \log(1/\epsilon')$ and thus, it is sufficient to construct extractors with small seed for large to achieve a seed length with "correct dependance" on the error.

**A rough sketch of the construction:** A key ingredient is the existence of explicit extractors with "correct dependance" on the error for very large $k$ [Zuc97] and for very small $k$ [SZ99, GW97]. When given an extractor $Ext$ with large error $\epsilon$ the main idea is to differentiate between error caused by "bad source elements" and error caused by "bad seeds". The contribution of "bad source elements" to the final error can be made arbitrarily small by requiring that the source have larger min-entropy. (Intuitively, the fraction which bad elements take in a "larger" source is smaller.) To decrease the error caused by "bad seeds" the extractor is run with two independent seeds, thus the probability of obtaining a "bad seed" in both attempts is about $\epsilon^2$. Intuitively, in one of the two trials a distribution which is $\epsilon^2$-close to uniform is obtained. By concatenating the two outputs we get a distribution of length $2m$ which is $\epsilon^2$-close to having min-entropy $m$. We can now use an extractor for large $k$ ($k = n/2$) to extract this randomness at the "correct dependance" on $\epsilon^2$. The

[20] Their construction also works starting from constant $\epsilon$ (rather than $\epsilon = 1/m$) however, in that case $d' = (d + O(\log(1/\epsilon')))(\log\log n)^{O(1)}$.

process described above doubles the seed length (as two independent seeds were chosen). However by using an extractors for small $k$, two (sufficiently independent) seeds can be chosen from much fewer random bits. The error can be reduced to an arbitrary $\epsilon'$ by repeatedly squaring the error.

### 6.3 Transforming (regular) extractors into strong extractors

Reingold, Shaltiel and Wigderson [RSW00] gave an explicit transformation of (regular) extractors into strong extractors. A $(k, \epsilon)$-extractor with seed length $d$ and output length $m$ is explicitly transformed into a strong extractor with seed length $d' = d + \text{polylog}(d/\epsilon)$ and $m = m - d - 2\log(1/\epsilon) - O(1)$.

The main idea is that when given a (non-strong) extractor with $m >> d$, the output of the extractor contains $m - d$ bits which *do not* depend on the seed. This is because fixing the $d$ bit long seed can reduce the amount of randomness in the output by at most $d$ bits. More formally, it follows that for a large fraction of seeds $y$, the distribution $E(X, y)$ contains $m - d$ random bits. This randomness can be extracted with a very short seed using strong extractors for high min-entropy threshold, and is (close to) being independent from the initial seed.

### 6.4 Extractors using small space

When using extractors to derandomize bounded-space algorithms it is sometimes beneficial to have extractors which are computable in small space. Hartman and Raz [HR00] showed how to construct weak designs in Logspace. It follows that Trevisan's extractor (as well as its later modifications) can be computed in Logspace.

Bar-Yossef, Reingold, Shaltiel and Trevisan [BYRST02] showed that extractors cannot be computed online in space significantly smaller than their output length. In contrast, they constructed dispersers which beat this lower bound.

## 7 Extractors for high min-entropy

The lower bounds of [RTS00] (see section 1.12) allow a very short seed when $k$ is large. The seed length in the lower bound is $d \geq \log(n-k)$ whereas all the constructions we've seen have $d \geq \log n$. We use $D$ to denote $n - k$, the *entropy deficiency* of the source. Goldreich and Wigderson [GW97] used random walks on explicit expander graphs to get an expander with seed length $O(D+\log(1/\epsilon))$ and optimal entropy loss. Reingold, Vadhan and Wigderson [RVW00] construct an extractor with seed length $d = \text{polylog}(D/\epsilon)$. The idea is to split the source into two parts: $X_1$ which contains the first $n - 2D$ indices of $X$, and $X_2$ which contains the remaining $2D$ indices. It follows that $X_1, X_2$ form a block-wise source with min-entropy $n - 3D, D$. The reasoning is similar to that done in section 4.2: $X_1$ must contain $n - 3D$ random bits (even if it suffers the $D$ bit deficiency), and is too short to steal all the randomness from $X_2$. Randomness can be extracted from $X_1, X_2$ by extracting $D$ random bits from $X_2$ and then using these as seed to the Goldreich-Wigderson extractor which is run on $X_1$.

Using this approach loses gives an entropy loss of $D$. The main contribution of [RVW00] is a way to perform this argument without suffering this loss.

## 8 Open problems

The most interesting open problem is to construct optimal extractors. The next milestone to achieve seems to be achieving seed length $d = O(\log n)$ and output length $m = \Omega(k)$ for all thresholds $k$. (This has already been achieved for $k \le 2^{\log^{1-o(1)} n}$ by [TSUZ01] and for $k = \Omega(n)$ by [Zuc97]). Concrete directions are:

- Construct a transformation which uses $O(\log n)$ random bits to transform general sources into block wise sources where the first block contains $\Omega(k)$ random bits (see remark 2).
- Actually, using [RSW00] it is sufficient to construct a function $B : \{0,1\}^n \times \{0,1\}^{O(\log \log n)} \rightarrow \{0,1\}^{n/\log n}$ such that for every $X$ with $H_\infty(X) \ge k$ $B(X, \cdot)$ is $1/2 \log n$-close to containing $\Omega(k)$ random bits. (To see that this suffices, requires a more careful examination of [RSW00] than the one given in section 4.2).
- Reduce $t$ in the reconstruction of the multivariate-polynomials extractors. In particular, can we do better than $t \approx mh$?

## Acknowledgements

I am grateful to my advisor Avi Wigderson for introducing me to this area and for countless conversations. I also want to thank Omer Reingold, Amnon Ta-Shma and Chris Umans for many discussions. Some of the presentation was "rehearsed" on students in a course I taught at the Weizmann Institute and I want to thank the students for their feedback. I thank Alon Rosen for a very careful reading of this manuscript and Boaz Barak for suggesting some of the notation. I also want to thank Lance Fortnow who asked me to write this manuscript.

## References

[BFNW93] Laszlo Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[Blu84] Manuel Blum. Independent unbiased coin flips from a correlated biased source: a finite state Markov chain. In *25th Annual Symposium on Foundations of Computer Science*, pages 425–433, Singer Island, Florida, 24–26 October 1984. IEEE.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[BYRST02] Ziv Bar-Yossef, Omer Reingold, Ronen Shaltiel, and Luca Trevisan. Streaming computation of combinatorial objects. In *Seventeenth Annual IEEE Conference on Computational Complexity*, 2002.

[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.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, 1989.

[Gol98] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer-Verlag, Algorithms and Combinatorics, 1998.

[GW97] Oded Goldreich and Avi Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. *Random Structures & Algorithms*, 11(4):315–343, 1997.

[GZ97] O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). Technical Report TR97-045, Electronic Colloquium on Computational Complexity, 1997.

[HR00] Tzvika Hartman and Ran Raz. On the distribution of the number of roots of polynomials and explicit logspace extractors. In *Proceedings of RANDOM*, 2000.

[ILL89] R. Impagliazzo, L.A. Levin, and M. Luby. Pseudorandom generation from one-way functions. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, 1989.

[Imp95] Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 October 1995. IEEE.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, 1994.

[ISW99] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[ISW00] R. Impagliazzo, R. Shaltiel, and A. Wigderson. Extractors and pseudo-randomn generators with optimal seed-length. In *Proceedings of the Thirty-second Annual ACM Symposium on the Theory of Computing*, 21–23 May 2000.

[IW97] Russell Impagliazzo and Avi Wigderson. $P = BPP$ if $E$ requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.

[MR95] R. Motwani and P. Rgahavan. *Randomized algorithms.* Cambridge University press, 1995.

[MU01] E. Mossel and C. Umans. On the complexity of approximating the vc dimension. In *Sixteenth Annual IEEE Conference on Computational Complexity*, pages 220–225, 2001.

[Nis96] Noam Nisan. Extracting randomness: How and why: A survey. In *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, pages 44–58, Philadelphia, Pennsylvania, 24–27 May 1996. IEEE Computer Society Press.

[NTS99] N. Nisan and A. Ta-Shma. Extracting randomness: A survey and new constructions. *JCSS: Journal of Computer and System Sciences*, 58, 1999.

[NW94] N. Nisan and A. Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

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

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

[RRV99a] R. Raz, O. Reingold, and S. Vadhan. Error reduction for extractors. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[RRV99b] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, 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.

[RTS97] Jaikumar Radhakrishnan and Amnon Ta-Shma. Tight bounds for depth-two superconcentrators. In *38th Annual Symposium on Foundations of Computer Science*, pages 585–594, Miami Beach, Florida, 20–22 October 1997. IEEE.

[RTS00] J. Radhakrishnan and A. Ta-Shma. Bounds for dispersers, extractors, and depth-two superconcentrators. *SIAM Journal on Discrete Mathematics*, 13(1):2–24, February 2000.

[RVW00] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[RZ98] A. Russell and D. Zuckerman. Perfect-information leader election in $\log^* n + O(1)$ rounds. *Journal of Computer and System Sciences*, 1998. To appear. Preliminary version in *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 576–583.

[Sip88] M. Sipser. Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36, 1988.

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

[STV99] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the xor lemma. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

[SU01] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudo-random generator. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001.

[Sud97] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13, 1997.

[SV86] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.

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

[Tre99] L. Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, 1999.

[TS96] 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.

[TS98] A. Ta-Shma. Almost optimal dispersers. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 196–202, New York, May 23–26 1998. ACM Press.

[TSUZ01] A. Ta-Shma, C. Umans, and D. Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[TSZ01] A. Ta-Shma and D. Zuckerman. Extractor codes. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[TSZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of the 42th Annual IEEE Symposium on Foundations of Computer Science*, 2001.

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

[Uma99] C. Umans. Hardness of approximating $\Sigma_2^p$ minimization problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 465–474, 1999.

[Uma02] Chris Umans. Pseudo-random generators for all hardnesses. In *Proceedings of the Thirty-fourth Annual ACM Symposium on the Theory of Computing*, 2002.

[Vaz87a] Umesh Vazirani. Efficient considerations in using semi-random sources. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, 1987.

[Vaz87b] Umesh Vazirani. Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources. *Combinatorica*, 7:375–392, 1987.

[vN51] John von Neumann. Various techniques used in connection with random digits. *Applied Math Series*, 12:36–38, 1951.

[WZ99] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. *Combinatorica*, 19(1):125–138, 1999.

[Yao82] Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

[Zuc90] David Zuckerman. General weak random sources. In *31st Annual Symposium on Foundations of Computer Science*, volume II, pages 534–543, St. Louis, Missouri, 22–24 October 1990. IEEE.

[Zuc96a] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25:1293–1304, 1996.

[Zuc96b] D. Zuckerman. Simulating BPP using a general weak random source. *Algorithmica*, 16(4/5):367–391, October/November 1996.

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