Extracting Randomness : A Survey and New Constructions *

Noam Nisan †   Amnon Ta-Shma ‡

Abstract

Extractors are boolean functions that allow, in some precise sense, extraction of randomness from somewhat random distributions, using only a small amount of truly random bits. Extractors, and the closely related "Dispersers", exhibit some of the most "random-like" properties of explicitly constructed combinatorial structures.

In this paper we do two things. First, we survey extractors and dispersers: what they are, how they can be designed, and some of their applications. The work described in the survey is due to a long list of research papers by various authors – most notably by David Zuckerman. Then, we present a new tool for constructing explicit extractors, and give two new constructions that greatly improve upon previous results.

The new tool we devise, a "merger", is a function that accepts d strings, one of which is uniformly distributed, and outputs a single string that is guaranteed to be uniformly distributed. We show how to build good explicit mergers, and how mergers can be used to build better extractors.

Using this, we present two new constructions. The first construction succeeds in extracting *all* of the randomness from *any* somewhat random source. This improves upon previous extractors that extract only *some* of the randomness from somewhat random sources with "enough" randomness. The amount of truly random bits used by this extractor, however, is not optimal. The second extractor we build extracts only some of the randomness, and works only for sources with enough randomness, but uses a near-optimal amount of truly random bits.

Extractors and dispersers have many applications in "removing randomness" in various settings, and in making randomized constructions explicit. We survey some of these applications, and note whenever our new constructions yield better results, e.g., plugging our new extractors into a previous construction we achieve the first explicit N–superconcentrators of linear size and $polyloglog(N)$ depth.

*This paper is a combination of the paper "On Extracting Randomness From Weak Random Sources" [Ta-96] and the paper "Refining Randomness: Why and How" [Nis96]. This work was supported by BSF grant 92-00043 and by a Wolfeson award administered by the Israeli Academy of Sciences.

†Institute of computer science, Hebrew University, Jerusalem, Israel

‡Institute of computer science, Hebrew University, Jerusalem, Israel

Contents

1 Introduction

During the last two decades the use of randomization in the design of algorithms has become commonplace. There are many examples of randomized algorithms for various problems which are better than any known deterministic algorithm for the problem. The randomized algorithms may be faster, more space-efficient, use less communication, allow parallelization, or may be simply simpler than the deterministic counterparts. We refer the reader e.g. to [MR95] for a textbook on randomized algorithms.

Despite the wide spread use of randomized algorithms, in almost all cases it is not at all clear whether randomization is really necessary. As far as we know, it may be possible to convert *any* randomized algorithm to a deterministic one without paying any penalty in time, space, or other resources. In many cases, in fact, we do know how to convert a randomized algorithm to a deterministic one – "derandomizing" the algorithm. This notion of "derandomization" has also become quite common. By now, a standard technique for designing a deterministic algorithm is to first design a randomized one (which in many cases is easier to do) and then derandomize it.

This state of affairs in algorithmic design is parallel in the design of various combinatorial objects (such as graphs or hypergraphs with certain properties). The "probabilistic method" is many times used to non-constructively prove the existence of these sought after objects. Again, in many cases it is known how to "derandomize" these probabilistic proofs, and achieve an explicit construction. We refer the reader to [AS92a] for a survey of the probabilistic method.

Derandomization Techniques

It is possible to roughly categorize the techniques used for derandomization according to their generality. On one extreme are techniques which relate very strongly to the problem and algorithm at hand. These usually rely on a sophisticated understanding of the structure of the problem, and are not applicable to different ones. On the other extreme are completely general derandomization results – e.g. converting *any* polynomial time randomized algorithm to a deterministic one. This may be done with a sufficiently good pseudo-random generator, but with our current understanding of computational complexity such results *always* rely on unproven assumptions.

In the middle range of generality lie various techniques that apply to certain "types" of randomized algorithms. Algorithms that use their randomness in a "similar" way may be derandomized using similar techniques. Two main strategies for derandomization are commonly employed. The first is the construction of a "small sample space" for the algorithm. Instead of choosing a truly random string, we fix a small set of strings – the "small sample space" – and then take a string from the sample space instead of choosing it completely at random. This requires, of course, a proof that this sample space is good enough as a replacement for a truly random string. The second strategy is to adaptively "construct" a replacement for the random string – e.g. by gradually improving some conditional probability. In many cases both strategies are combined, and a replacement string is constructed in the small sample space.

It is probably fair to say that there are only two or three basic types of tools which are commonly used in the construction of small sample spaces for derandomizations:

1. Pairwise (and k-wise) independence and Hashing.
2. Small Bias Spaces.
3. Expanders.
4. Extractors and Dispersers.

We refer the reader, again, to [AS92a, MR95] for further information as well as for references.

Dispersers and Extractors

In this paper we deal with the fourth general type of tool: a family of graphs called *Dispersers* and *Extractors*. These graphs have certain strong "random-like" properties – hence they can be used in many cases where "random-like" properties are needed.

Think of a bipartite graph with $N = 2^n$ vertices on the left hand side, $M = 2^m$ vertices on the right hand side, and degree $D = 2^d$. Our functions will take a name of a vertex on the left-hand side and an index of an edge and will return the name of the vertex on the right hand side: $G : (n) \times (d) \rightarrow (m)$, where (l) denotes $\{0,1\}^l$.

DEFINITION 1.1 *A function $G : (n) \times (d) \rightarrow (m)$ is called a (k,ϵ)-disperser if for any $A \subseteq \{0,1\}^n$ of size at least $K = 2^k$, A has more than $(1-\epsilon)M$ distinct neighbors in $\{0,1\}^m$. G is a (k,ϵ)-extractor, if for any distribution X on $\{0,1\}^n$ with "k bits of randomness", the induced distribution $\Gamma(X)$ on $\{0,1\}^m$, obtained by taking $G(x,y)$ where x is chosen according to X and y is uniform, is ϵ-close to uniform.*

For the time being think of the term "X has k bits of randomness" as having the meaning that X is uniformly distributed over a set of size $K = 2^k$. We will later (Section 2) precisely define this notion, and the notion of being close to uniform.

It is easy to see that any (k,ϵ)–extractor is also a (k,ϵ)–disperser. For if there exists a set $A \subseteq \{0,1\}^n$ s.t. A misses many neighbors on the right hand side, then taking X to be the uniform distribution on A, we see that $\Gamma(X)$ is far away from uniform.

We can view extractors as taking an n bit string with some k randomness, investing d truly random bits and extracting m quasi-random bits. The quality of a disperser/extractor is therefore measured by the number of truly random bits d (we would like it to be small), the required amount of randomness in the somewhat random source k (again, we would like it to be small) and the number of quasi-random bits that are extracted m (we would like it to be as close as possible to $k+d$).

Survey of Previous Results

Dispersers were first defined (with somewhat different parameters) by Sipser [Sip88], while extractors were defined by Nisan and Zuckerman [NZ93]. The roots of the research on extractors lie mostly in the work on "somewhat random sources" done in the late 1980's, by Vazirani, Santha and Vazirani, Vazirani and Vazirani, Chor and Goldreich, and others [SV86, Vaz87a, Vaz86, Vaz87b, VV85, CG88]. The direct development of the constructions and applications of extractors and dispersers came first in papers written by Zuckerman in the early 1990's [Zuc90, Zuc91], and then in a sequence of papers by various authors [NZ93, WZ93, SZ94, SSZ95, Zuc93, Zuc].

The first explicit construction of extractors came in [NZ93], and relied on techniques developed in [Zuc90, Zuc91]. This construction had $d = polylog(n)$ for $k \geq n/polylog(n)$. An efficient extractor working for small k's, $k = \Theta(log(n))$, was obtained by [GW94, SZ94] using tiny families of hash functions [NN93, AGHP92]. This was used in [SZ94] to improve upon the [NZ93] extractor, and get it work for any $k > \sqrt{n}$. In [SSZ95] a disperser with $d = O(\log n)$ was obtained for any $k = n^{\Omega(1)}$. In [Zuc], $d = O(\log n)$ was obtained for $k = \Omega(n)$.

In the following table we list the currently known best explicitly constructible extractors and dispersers for various parameters.

required crude randomness	no. of truly random bits	number of output bits	reference
$k = \Omega(log(n))$	$d = k$	$m = (1 + \Omega(1)) \cdot k$	$\epsilon = 2^{-\Omega(k)}$ [GW94, SZ94]
$k = \Omega(n)$	$d = O(log(n) + log(\frac{1}{\epsilon}))$	$m = \Omega(k)$	[Zuc]
$k = \Omega(n^{1/2+\gamma})$	$d = O(log^2 n \cdot log(\frac{1}{\epsilon}))$	$m = n^\delta \ , \quad \delta \leq \gamma$	[SZ94]
$k = \Omega(n^\gamma)$	$d = O(logn)$	$m = n^\delta \ , \quad \delta < \gamma$	Disperser, $\epsilon = \frac{1}{2}$ [SSZ95]

New Results

We devise a new tool, a "merger", and show how to build good explicit mergers, and how mergers can be used to build better extractors. We then give two new explicit constructions. The first extractor we build works for *any* source whatever its min-entropy is, and extracts *all* the randomness in the given source.

Theorem 1 *For every constant $\gamma < 1$, $\epsilon \geq 2^{-n^\gamma}$, and every $k = k(n)$ there is an explicit (k, ϵ) extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (k)$.*

The extractor is the first to work for *any* source, no matter how much randomness it contains, and also the first to extract *all* of the randomness. Its only drawback is that the amount of truly random bits used is polynomial in what is optimal. The second extractor we present work only for some of the sources, extracts only some of the randomness, but uses a near-optimal number of truly random bits.

Theorem 2 *For every constant c and $\gamma > 0$ there is some constant $\delta > 0$ and an explicit $(n^\gamma, \frac{1}{n})$ extractor $E : (n) \times (O(log(n)log^{(c)}n)) \mapsto (\Omega(n^\delta))$, where $log^{(c)}n = \underbrace{loglog \ldots log}_{c}\ n$.*

required crude randomness	no. of truly random bits	number of output bits	
any k	$d = polylog(n) \cdot log(\frac{1}{\epsilon})$	$m = k$	$\epsilon \geq 2^{-\sqrt{n}}$
$k = \Omega(n^{\gamma})$	$d = O(log(n) \cdot \underbrace{loglog \ldots log}_{c}\ n)$	$m = \Omega(n^{\delta}), \delta < \gamma$	$\epsilon = \frac{1}{n}$ any constant c

Applications

There are many examples where extractors and dispersers are used for the purposes of derandomization – of algorithms or in explicit constructions. Our new constructions improve some of them. In the following we state the results we improve. Formal definitions and proofs are given in Section 6.

The first application is constructing explicit a–expanding graphs, obtained by plugging our first extractor into the [WZ93] construction:

Corollary 1.1 *For any N and $1 \leq a \leq N$ there is an explicitly constructible a–expanding graph with N vertices, and maximum degree $O(\frac{N}{a}2^{polyloglog(N)})$* [1] *.*

This corollary has applications on sorting [Pip87a, WZ93] and selecting [AKSS89, WZ93] in k rounds.

Corollary 1.2 *There are explicit algorithms for sorting in k rounds using $O(n^{1+\frac{1}{k}} \cdot 2^{polyloglog(n)})$ comparisons, and for selecting in k rounds using $O(n^{1+\frac{1}{2^k-1}} \cdot 2^{polyloglog(n)})$ comparisons.*

Corollary 1.3 *There are explicit algorithms to find all relations except $O(a \cdot nlog(n))$ among n elements, in one round and using $O(\frac{n^2}{a} \cdot 2^{polyloglog(n)})$ comparisons.*

Another corollary is for the construction of explicit small-depth superconcentrators. It is again obtained by plugging our first extractor into a previous construction by [WZ93]:

Corollary 1.4 *For every N there is an efficiently constructible depth 2 superconcentrator over N vertices with size $O(N \cdot 2^{polyloglog(N)})$.* [2]

Wigderson and Zuckerman [WZ93] prove that a direct corollary of this is:

Corollary 1.5 *For any N there is an explicitly constructible superconcentrator over N vertices, with linear size and $polyloglog(N)$ depth* [3]*.*

[1] The obvious lower bound is $\frac{N}{a}$. The previous upper bound [WZ93, SZ94] was $O(\frac{N}{a} \cdot 2^{log(N)^{1/2+o(1)}})$.

[2] This improves the previous upper bound of $O(N \cdot 2^{log(N)^{1/2+o(1)}})$ achieved using the [WZ93] technique and the [SZ94] extractor.

[3] This improves the previous upper bound of $O(log(N)^{1/2+o(1)})$.

We can also prove a deterministic version of the hardness of approximating the iterated log of MaxClique. See [Zuc93] for more details.

Finally, using our second extractor we get:

Corollary 1.6 *For any $\delta > 0$ and constant $k > 0$, BPP can be simulated in time $n^{O(\overbrace{loglog\ldots log}^{k} n)}$ using a weak random source X with minentropy at least n^{δ}* [4] *.*

Organization of the paper

In Section 2 we give formal definitions. We also provide the basics: go over some preliminaries and discuss simple lower and upper bounds. In Section 3 we give a survey of some of the previous constructions of extractors and dispersers, and explain some of the main ideas behind them. In Section 4 we present our new tool, the "merger", and construct our first explicit extractor, to be followed in Section 5 by our second explicit extractor. Finally, in Section 6, we survey what applications the existence of good dispersers/extractors have.

2 Basics

2.1 Preliminaries

Let us first define some notions which will be used throughout this paper.

Probability Distributions

We will constantly be discussing probability distributions, the distances between them, and the randomness hidden in them. In this subsection we give the basic definitions used to allow such a discussion.

A probability distribution X over a (finite) space Λ simply assigns to each $a \in \Lambda$ a positive real $X(a) > 0$, with the property that $\sum_{a \in \Lambda} X(a) = 1$. For a subset $S \subseteq \Lambda$ we denote $X(S) = \sum_{a \in S} X(a)$. The uniform distribution U on Λ is defined as $U(a) = 1/|\Lambda|$ for all $a \in \Lambda$.

When presenting our new constructions we would like to be precise and distinguish between distributions and random variables. We denote random variables by capital letters, and their corresponding distributions by a barred variable, e.g. $\overline{X}$ is the distribution corresponding to X. All conditional expressions, e.g. $(A|B)$, denote distributions. U_k denotes the uniform distribution over k bits. In all other sections we usually identify a random variable with its probability distribution (we make the distinction only where necessary), and use capitals $X, Z, \ldots$ to denote such random variables and probability distributions. We use small letters $a, x, z\ldots$ to denote elements in the probability space. Unless stated otherwise x is distributed according to X, z according to Z, etc.

We denote by $A \circ B$ the concatenation of two random variables A and B. A variable that appears twice (or more) in the same expression has the same value in all occurrences, i.e.

[4] Previous result [SZ94] was for $\delta > 1/2$, and required $n^{O(log(n))}$ time. Recently an optimal $poly(n)$ simulation was presented in [ACRT97] using different techniques.

$A \circ A$ denotes a random variable with values $a \circ a$. On the contrary, $A \times B$ denotes taking A and B independently, thus $A \times A$ denotes a random variable with values $a_1 \circ a_2$ where a_1 and a_2 are completely independent.

Finally, given a random variable $X = X_1 \circ \ldots \circ X_n$ we denote $X_i \circ \ldots \circ X_j$ by $X_{[i,j]}$, and the same applies for instances $x_{[i,j]}$.

Statistical Distance

DEFINITION 2.1 *Let X, Y be two distributions over the same space Λ. The* statistical distance *between them is given by:* $d(X,Y) = \frac{1}{2}|X - Y|_1 = \frac{1}{2}\Sigma_{a\in\Lambda}|X(a) - Y(a)| \; = \; max_{S\subseteq\Lambda}|X(S) - Y(S)|$.

It is easy to verify that the statistical distance is indeed a metric. We say X is ϵ-close to Y if $d(X,Y) \leq \epsilon$. We say X is ϵ quasi-random if it is ϵ-close to uniform.

Min-Entropy

We will need to measure the "amount of randomness" that a given distribution X has in it. The Shannon entropy of X certainly springs to mind, $H(X) \stackrel{\text{def}}{=} -\Sigma_a X(a)\log(X(a))$. However, this entropy is not that sensitive to "dominant" values. E.g., if we take

$$X(a) = \begin{cases} 1/2 & \text{if } a = 0^n \\ 1/2^n & \text{if } a \text{ does not start with a } 0 \\ 0 & \text{otherwise} \end{cases}$$

Then $H(X) \approx \frac{n}{2}$ and yet with probability one half X gets the same single value 0^n and therefore any extraction method will get the same result at least half of the times. Thus, we want an entropy measure that is very sensitive to values that have high probability.

DEFINITION 2.2 *The min-entropy of a distribution X is $H_\infty(X) = min_a(-log_2(X(a)))$.*

I.e., if $H_\infty(X) \geq k$ then for any x, $\Pr\ (X = x) \leq 2^{-k}$. It is easy to verify that $H_\infty(X)$ is always bounded from above by the Shannon entropy, $H_\infty(X) \leq H(X)$. Equality holds whenever X is uniform over a subset $S \subseteq \Lambda$, in which case $H_\infty(X) = H(X) = \log_2|S|$. It is useful to think of a distribution X with $H_\infty(X) = k$ as a generalization of being uniform over a set of size 2^k.

2.2 Extractors and Dispersers

Extractors and dispersers are very similar to each other, yet, in the literature, dispersers have been usually defined as graphs [Sip88, SSZ95], while extractors as functions [NZ93, SZ94, Zuc].

We will define both extractors and dispersers both as functions and as graphs, taking the view that it is the same combinatorial object, viewed in two different, useful, ways.

Graph Definitions

Extractors and dispersers are certain types of bipartite (multi-)graphs. Throughout this paper the left hand side of the graph will have $N = 2^n$ vertices and the right hand side of the graph $M = 2^m$ vertices. Vertices will be numbered by integers which we identify with their binary representation. Thus the left hand side of the graph is always $[N] = \{1...N\} = \{0,1\}^n$, and the right hand side is always $[M] = \{1...M\} = \{0,1\}^m$. The graphs will usually be highly imbalanced $n > m$. Furthermore, all vertices on the left-hand side will have the same degree, $D = 2^d$, which is usually very small $d << m$.

Given a graph $G = ([N],[M],E)$; for each vertex $a \in [N]$, we denote its neighbor set by $\Gamma(a) = \{z \in [M] | (a,z) \in E\}$. For a subset $A \subseteq [N]$ we denote its neighbor set by $\Gamma(A) = \bigcup_{a \in A} \Gamma(a)$. For a distribution X on $[N]$, we denote by $\Gamma(X)$ the probability distribution induced on $[M]$ by first choosing $x \in [N]$ according to X and then choosing uniformly a random neighbor $z \in \Gamma(x)$.

DEFINITION 2.3 *A (multi-)graph $G = ([N],[M],E)$ is a (k,ϵ)-disperser if for any $A \subseteq [N], |A| \geq K = 2^k$ we have that $|\Gamma(A)| \geq (1-\epsilon)M$.*

DEFINITION 2.4 *A (multi-)graph $G = ([N],[M],E)$ is a (k,ϵ)-extractor if for any distribution X on $[N]$, with $H_\infty(X) \geq k$, we have that $\Gamma(X)$ is ϵ-close to uniform on $[M]$.*

REMARK 2.1 *This definition is slightly different from the one in [NZ93, SZ94], who require that the random choice of the edge originating at x is is also almost independent from $\Gamma(X)$. The difference is minor, though, and we prefer this definition.*

The name "disperser" is natural from this point of view: the graph disperses any set of size K, to almost all the right hand side.

Function Definitions

The definitions below are equivalent to the previous definitions, but take the functional view. Recall that the left hand side of our graphs is $\{1...N\} = [N] = \{0,1\}^n$, the right hand side $\{1...M\} = [M] = \{0,1\}^m$. The left degree is $D = 2^d$, so the edges originating at each vertex on the left hand side will be labeled by $\{1...D\} = [D] = \{0,1\}^d$. For ease of notation we will denote $\{0,1\}^k$ by (k). Our functions will take a name of a vertex on the left-hand side and an index of an edge and will return the name of the vertex on the right hand side: $G : (n) \times (d) \to (m)$. For $A \subseteq \{0,1\}^n$ We will denote $\Gamma(A) = \{z = G(x,y) : x \in A,\ y \in \{0,1\}^d\}$. Similarly, for a distribution X on $\{0,1\}^n$, $\Gamma(X)$ will denote the distribution of $G(x,y)$ induced by choosing x according to X, and y uniformly in $\{0,1\}^d$.

DEFINITION 2.5 *A function $G : (n) \times (d) \to (m)$ is called a (k,ϵ)-disperser if for any $A \subseteq \{0,1\}^n, |A| \geq K = 2^k$, we have that $|\Gamma(A)| \geq (1-\epsilon)M$.*

DEFINITION 2.6 *A function $G : (n) \times (d) \to (m)$ is called a (k,ϵ)-extractor if for any distribution X on $\{0,1\}^n$, $H_\infty(X) \geq k$, we have that $\Gamma(X)$ is ϵ-close to uniform.*

The name "extractor" is natural from this point of view: the function extracts m-bits of randomness from the n-bits given to it, as long as these n bits have k min-entropy. In order to do this it uses d truly random bits as well.

Explicitness

We will be constructing explicit extractors and dispersers. Let us define what an explicit construction is. Let us recall that (in theoretical computer science) defining a combinatorial object as explicitly constructed only makes sense for a *family* of objects. In our case, we will be taking n as the parameter of the family, have $N = 2^n$, and view $M = 2^m, K = 2^k, D = 2^d, \epsilon$ as depending on it.

Definition 2.7 *A family of functions $G = \{G_n : (n) \times (d) \to (m)\}$ ($d = d(n), m = m(n)$), is called explicit if there is a deterministic Turing machine that for every $x \in \{0,1\}^n$ and $y \in \{0,1\}^d$, outputs $G_n(x, y)$ in polynomial time (in the length of the input x, y).*

2.3 Behavior of Parameters

In this section we discuss the basic relations between the different parameters of extractors and dispersers. Let us recall first the five different parameters we are interested in, and their intuitive meanings, both in the graph view and in the function view.

- $N = 2^n$. N is the number of vertices in the left hand side of G. n is the number of bits in the string x from which randomness should be extracted.
- $M = 2^m$. M is the number of vertices on the right hand side of G. m is the length of the near-random output z.
- $D = 2^d$. D is the left-degree of G. d is the number of the truly random bits we use.
- $K = 2^k$. K is the size of the set which should dispersed. k is the min-entropy of x.
- ϵ. The error parameter.

We view n as a parameter, m as being a second parameter anywhere in the range $1 \leq m \leq n$, and then we try to have k and ϵ as small as possible. Our main goal, for each choice of these parameters, is to get d to be as small as possible.

Non-Explicit Existence

Rather standard applications of the probabilistic method show that for all $n \geq m \geq 1, \epsilon > 0$, there exist extractors with $k = m$ and $D = O(n/\epsilon^2)$ and dispersers with $D = O(n/\epsilon)$ (I.e. in both cases $d = O(\log n + \log \epsilon^{-1})$). This calculation was done, for certain dispersers, in [Sip88]. For tight bounds see [RTS].

Lower Bounds

A trivial lower bound, for any n, m and $\epsilon < 1/2$, is $d \geq m - k - 1$. We will mostly consider the case where $k \geq m$ so this does not help us. In [NZ93] a lower bound of $d \geq \min(m, \Omega(\log(n-k) + \log \epsilon^{-1}))$ was proved for all $n, m, k \leq n-1, \epsilon < 1/2$. Better (and tight) lower bounds can be found in [RTS].

3 Survey of Previous Constructions

In this section we survey the main ingredients used in previous constructions and indicate how they are put together. We describe the constructions precisely, but provide only sketches of proofs for their validity.

All constructions for extractors and dispersers build, as a starting point, an extractor with very weak parameters, but which can be obtained easily. This is described in Section 3.1. In Section 3.2 we describe how extractors can be composed when given a source with some special properties - a "block-wise" source. In Section 3.3 we show how such a composition may be applied to arbitrary sources with high min-entropy. Finally, Section 3.4 sketches the different ways in which these elements can be combined to obtain some constructions of extractors and dispersers.

3.1 The Mother of All Extractors

A useful way to think about extractors is to consider the edges of the extractor as hashing the vertices on the left to vertices on the right. The requirement is that large enough sets are well hashed. Viewed this way we can construct an extractor from a family of hash functions as follows.

A Construction based on Hashing

Let H be a family of functions $h : [N] \to [L]$. The extractor defined by H is given by $G(x, h) = h(x) \circ h$. Thus $D = |H|$, and $M = DL$.

Now we define the property that we require from the family of hash functions in order for the graph to be an extractor.

DEFINITION 3.1 $H = \{h : [N] \mapsto [L]\}$ *is a family of hash functions with collision error* δ, *if for any* $x_1 \neq x_2 \in [N]$, $\Pr_{h\in H}[h(x_1) = h(x_2)] \leq (1 + \delta)/L$.

The following lemma is a variant of the of leftover hash lemma of Impagliazzo, Levin, and Luby [ILL89], stated in our terms:

Lemma 3.1 *[ILL89] Let* H *be a family of hash functions from* $[N]$ *to* $[L]$ *with collision error* δ. *Then, the extractor defined from* H *is a* (k, ϵ)*-extractor for* $K = 2^k = O(L/\delta)$ *and* $\epsilon = O(\sqrt{\delta})$.

Proof: (sketch) We need to show that for every distribution X on $[N]$ with $H_\infty(X) \geq k$, the distribution of $h \circ h(x)$ is ϵ-close to uniform (where x is chosen according to X and h uniformly in H). It turns out that the notion of collision probability is a convenient tool for this proof. The collision probability of a distribution X is defined to be $col(X) = \sum_a X(a)^2$. With this definition the following three steps, which imply the lemma, can be shown.

1. If $H_\infty(X) \geq k$ then $col(X) \leq 1/K$.

2. If $col(X)$ is small and H has a very small collision error, then the collision probability of $Z = (h \circ h(x))$ is very close to the collision probability of the uniform distribution (on $H \times [L]$).

3. If the collision probability of Z is very close to the collision probability of the uniform distribution (on the domain of Z) then Z is close to uniform.

$\square$

Universal Hashing

The first family of hash functions which can be used for extractors is the Carter-Wegman universal family of hash functions.

DEFINITION 3.2 *$H = \{h : [N] \to [L]\}$ is called a universal family of hash functions, if for any $x_1 \neq x_2 \in [N]$, and for any $w_1, w_2 \in [L]$, $\Pr_{h \in H} \ (\ h(x_1) = w_1 \wedge h(x_2) = w_2 \ ) \ = \ \frac{1}{L^2}$.*

It is clear that a universal family of hash functions, has, in particular, 0 collision error. For all ranges of $1 \leq l \leq n$, there exist constructions of universal families H of hash functions of size $|H| = poly(N = 2^n)$.

Using these families and the construction presented above, we get, for every $n \leq m \leq 2n$ extractors with high degree $d = O(n)$, and with $k = m - d + O(\log \epsilon^{-1})$ which is optimal for k (for this choice of n, m, d). This degree is much higher than we desire, and the output size is larger than what we are usually interested in (usually we want $m < n$).

We did get something useful though: the min-entropy on the left hand side is really extracted in full since the output randomness, m, is essentially the sum of the input min-entropy, k, and the extra randomness, d.

This construction is sufficient, as a starting point, for the composition of extractors described in Section 3.2. A better construction (which is easier to use) is implied by better families of hash functions, described next.

Tiny Families of Hash Functions

Universal families of hash functions have 0-collision error, and, in fact, 0 collision error does imply that their size $|H|$ must be large $|H| \geq N$. On the other hand, our purposes do not quite require 0-collision error but only small collision error. This will allow us to reduce the size of the family of hash functions, and hence, reduce the extractor degree. It turns out that "tiny families of hash functions" [SZ94, GW94] can be built using small-biased distributions.

Lemma 3.2 *[SZ94, GW94] For all $1 \leq L \leq N$ ($L = 2^l$, $N = 2^n$) and $\epsilon > 0$, there exist explicit families H of hash functions from $[N]$ to $[L]$ with ϵ collision error, and size $|H| = poly(n, \epsilon, L)$.*

This translates to,

Corollary 3.3 *For every $M \leq N$ ($M = 2^m$, $N = 2^n$) and $\epsilon > 0$, there exists an explicit (k, ϵ)-extractor with $D = poly(n, \epsilon, M)$ and $k = m - d + O(\log \epsilon^{-1})$.*

Notice that this extractor is almost optimal, except that D is also polynomial in M. Thus, for small M, i.e., for $M \leq poly(n)$ we get an optimal extractor.

A useful way to view this is that we manage to multiply the number of truly random bits from d to $m = (1+c)d$ for some constant $c > 0$, using a source with enough min-entropy to supply this increase.

Lemma 3.4 *[SZ94] There is some constant $c > 1$ s.t. for any $k = \Omega(log(n))$ there is an explicit $(2k, 2^{-k/5})$ extractor $A_k : (n) \times (k) \mapsto (ck)$. We denote this constant c by c_{tiny}.*

3.2 An Extractor For Block-Wise Sources

Another useful way to look at extractors is to consider them as multipliers of pure randomness (from d bits to m bits), as long as they are also given a source with enough min-entropy. They take a very short random string y, and output a long near-random string z. Of course, the extra randomness in z really comes from the source, X. Viewed this way, it is natural to take the output of one extractor and feed it into another extractor, compounding their "randomness multiplying" powers.

Composing Extractors

DEFINITION 3.3 *Let $G_1 : [N_1] \times [D_1] \to [M_1]$, and Let $G_2 : [N_2] \times [D_2] \to [M_2 = D_1]$ be extractors. Define $G_1 \circ G_2 : [N_1] \times [N_2] \times [D_2] \to [M_1]$ to be $G_1(x_1, G_2(x_2, y))$.*

Intuitively, it is clear that if X_1 has as much min-entropy in it as G_1 requires and X_2 as much min-entropy as G_2 requires then the output of $G_2 \circ G_1$ will indeed be close to random. A delicate issue to consider is the independence of the random variables X_1 and X_2: if they are indeed independent then this intuitive claim is true. If they are allowed to be correlated then the previous statement may not hold. The following key definition, first formulated in [CG88], turns out to suffice instead of total independence.

DEFINITION 3.4 *Let X_1, X_2 be (possibly correlated) random variables taking values, respectively, in $[N_1]$ and $[N_2]$. We say that (X_1, X_2) is a (k_1, k_2) block-wise source if*

1. *$H_\infty(X_1) \geq k_1$.*
2. *For every fixed value x_1 of X_1, $H_\infty(X_2 | X_1 = x_1) \geq k_2$. ("$X_2 | X_1 = x_1$" denotes the marginal distribution on X_2 conditioned on the event $X_1 = x_1$.)*

Lemma 3.5 *Let $G_1 : [N_1] \times [D_1] \to [M_1]$ be a (k_1, ϵ_1)-extractor, and Let $G_2 : [N_2] \times [D_2] \to [M_2 = D_1]$ be a (k_2, ϵ_2)-extractor. Let X_1, X_2 be a block-wise (k_1, k_2) source, and let y be chosen in random. Then the output of $G_1 \circ G_2(x_1, x_2, y)$ is $(\epsilon_1 + \epsilon_2)$-close to uniform.*

Proof: (sketch) Denote $w = G_2(x_2, y)$, and consider this random variable W. Fix any value x_1. Conditioned on $X_1 = x_1$, the distribution of W is ϵ_2 close to uniform (since G_2 is an extractor and by condition 2 of being a block-wise source). However this is true for every value x_1, thus the joint distribution of (X_1, W) is ϵ_2 close to the product distribution $X_1 \times U_{d_1}$. Thus the distribution of $G_1(x_1, w)$ is ϵ_2-close to the distribution of $G_1(x_1, w')$, where w' is chosen uniformly in $[D_1]$, *independently* from X_1. This last distribution is ϵ_1-close to uniform since G_1 is an extractor and by condition 1 of being a block-wise source. The lemma follows. □

Notice how we overcome the delicate issue of independence. Because this question is so central to our discussion we formalize this in:

Lemma 3.6 *[NZ93] Let X and Y be two correlated random variables. Let $\overline{B}$ be a distribution, and call an x "bad" if $(Y \mid X = x)$ is not ϵ close to $\overline{B}$. If $\Pr_{x \in \overline{X}}(x \text{ is bad}) \leq \eta$ then $\overline{X \circ Y}$ is $\epsilon + \eta$ close to $\overline{X} \times \overline{B}$.*

The idea of composing two extractors can of course be extended to allow composition of an arbitrary number of extractors. The truly random input, y_i, to each extractor is obtained from the output of the previous one, and they each get a separate "block" as the somewhat random input, x_i. We can also relax the requirement that for every prefix the next block has much min-entropy, to that for most prefixes the next block is close to a distribution with much min-entropy. Thus, the requirement that is needed for the output to be close to uniform is:

DEFINITION 3.5 *(extending [CG88]) Let $B_1, ..., B_t$ be correlated random variables taking values, respectively, in $[N_1], ..., [N_t]$. We say that $(B_1, ..., B_t)$ is a $(k_1, ..., k_t)$ block-wise source to within ϵ, if for each $1 \leq i \leq t$ and for all but an ϵ fraction of the sequences of values $x_1...x_{i-1}$, we have that $(B_i | B_1 = x_1, ..., B_{i-1} = x_{i-1})$ is ϵ close to a distribution with at least k_i min-entropy.*

Having as input a $(k_1, ..., k_t)$ block-wise source to within ϵ, we can combine t extractors $G_1...G_t$ (with $D_{i-1} = M_i$ for all $1 < i \leq t$, and G_i being a (k_i, ϵ_i)-extractor) to output m_1 near-random bits starting from d_t truly random ones. A useful way to view this is that $G_1 \circ \cdots \circ G_t$ multiplies the number of random bits by the product of the "randomness-multiplying" capabilities of each extractor. Of course, this multiplication requires a block-wise source, and not just any source with enough min-entropy.

Lemma 3.7 *[CG88, NZ93] Let $X = X_1 \circ X_2 \ldots \circ X_t$ be a $(k_1, \ldots, k_t)$ block-wise source to within ϵ where $k_t = \Omega(log(n))$ and $k_{i-1} = c_{tiny} k_i$. Then there is an explicit block extractor $BE(X, U)$, using k_t truly random bits and extracting $\Omega(\Sigma_{i=1}^{t} k_i)$ ϵ quasi-random bits with $O(2^{-\Omega(k_t)} + t\epsilon)$ error.*

3.3 Getting a Block-Wise Source

By composing, as described above, the basic extractors constructed in Section 3.1, we can directly extract essentially all the randomness from a block-wise source using only a very small number of truly random bits. Our problem, though, is to extract randomness from an arbitrary distribution with enough min-entropy. One possible solution is to first convert a general distribution X into a block-wise source $B_1...B_t$, and then proceed using this composition.

The first step in getting a block-wise source from a general distribution is to get the first block $B = B_1$. It turns out that once we can do this, in an appropriate way, the other blocks can be obtained in essentially the same way. What we need is for B to have enough min-entropy, even though it is much shorter than the original string (so that enough min-entropy is left for further blocks.)

In [NZ93] it is shown that choosing a subset of the bits of X in a pairwise independent way suffices for this. Other ways of sampling a subset of the bits of X behave similarly. We take a distribution S on strings of length l taken from $\{1...n\}$ such that for every $1 \le i < j \le n$: $\Pr_{s \in S}[i \in s \text{ and } j \in s] = \Pr_{s \in S}[i \in s] \cdot \Pr_{s \in S}[j \in s]$. This may be achieved in a sample space of size $O(n^2)$, by choosing the l elements pair-wise independently, thus the description of S requires only $O(\log n)$ bits. Given an n bit string $x_1...x_n$, and a subset $S \subseteq \{1...n\}$, denote x_S to be the string obtained by concatenating the bits x_i for all $i \in S$ (in the natural order).

Lemma 3.8 *[NZ93] For every distribution X and for for almost all choices of S, the distribution of X_S is close to some distribution W with $H_\infty(W) \ge \tilde{\Omega}(\frac{l}{n} H_\infty(X))$.*

Proof: (*intuition only*) Consider the entropy $H(X)$ of the source instead of the min-entropy. We associate with each bit i of X its conditional entropy $p_i = H(X_i | X_1 ... X_{i-1})$. A standard fact regarding conditional entropies implies that $\sum_i p_i = H(X)$. Since S is chosen in a pairwise independent way, w.h.p., $\sum_{i \in S} p_i$ is close to its expectation $l/n \cdot H(X)$. Again a standard calculation with conditional entropies will show that $H(X_S) \ge \sum_{i \in S} p_i$, which provides the lower bound required.

The above argument can indeed be, quite easily, made formal and does show that for almost all choices of S, $H(X_S) \ge \Omega(l/n \cdot H(X))$. The problem is that we require a lower bound on $H_\infty(X_S)$, which we can not obtain in a similar way since the "conditional min-entropies" of distributions do not behave as nicely as the conditional entropies.

The actual proof of this lemma proceeds by essentially carrying out the above argument separately for each possible value of x, and then combining all the x's back together. Unfortunately, this argument is quite cumbersome and delicate. $\square$

We summarize this as:

Lemma 3.9 *[NZ93, SZ94] Let X be a random source over $\{0,1\}^n$. For any $l > 0$ and $\delta > 0$, there is an explicit function $B(x, y)$ which gets $x \in X$ and a short random string y, and returns l bits, s.t.: If $H_\infty(\overline{X}) \ge \delta n$, then $\overline{B(X,U)}$ is $(\delta l)^{\Omega(-1)}$ close to a distribution $\overline{W}$ with $H_\infty(\overline{W}) \ge \Omega(\frac{\delta \cdot l}{log(\delta^{-1})}) \ge \Omega(\frac{\delta \cdot l}{log(n)})$.*

We can use the above lemma to convert any random source into a block-wise source. Let us start with a general random source X with k min-entropy. We start by extracting, as in the previous lemma, a block B_1 of length $l_1 << m$. Notice that for any value b_1 of B_1, $H_\infty(X \mid B_1 = b_1) \ge H_\infty(X) - \log \Pr[B_1 = b_1]$. We expect $\Pr[B_1 = b_1]$ to be about 2^{-l_1}. Values of b_1 which take significantly smaller probability can be safely ignored since they happen so rarely. This means that even conditioned on (almost) any value of B_1, X has sufficient min-entropy. Hence, choosing another subset of entries S_2 (independently from the first S), we can extract one more block B_2, which has high min-entropy even conditioned on the history b_1. Actually, as long as $l_1 + \ldots + l_{i-1} << m$, we can go on and extract another block B_i, which has high min-entropy even conditioned on the history.

Thus we get:

Lemma 3.10 *[NZ93] Let X be a distribution on $[N]$ with $H_\infty(X) \geq k$. Then for any choice of $l_1 ... l_t$ where $\sum l_i < k/2$, the above procedure extracts a source $B = B_1 \circ \ldots \circ B_t$ which is close to a $(k_1, ..., k_t)$ block-wise source with $k_i = \tilde{\Omega}(kl_i/n)$.*

Notice that this lemma gives non-trivial parameters as long as $k \geq \Omega(\sqrt{n})$. When $k \leq \sqrt{n}$ then l_1 has to be at least $\sqrt{n}$ to make the expected amount of min-entropy in the first block, k_1, at least 1, and then it may happen that given the first block there is no randomness left in the source X (since $k < l_1$). Next we shortly mention a different idea which works even for smaller values of k.

An Alternative Method

This alternative method for effectively getting a block-wise source was introduced in [SSZ95] and will be further extended in Section 4. The intuition can again be best obtained by considering the entropy $H(X)$ instead of the min-entropy.

Let us again be given a source $X = (X_1 ... X_n)$ from which we aim to get a block-wise $(k_1 ... k_t)$ source. If we denote $e_i = H(X_1 ... X_i)$, then we have for all i, $0 \leq e_i - e_{i-1} \leq 1$. We also have $e_0 = 0$ and $e_n \geq k$. Thus, for any $k_1 \leq k$, there always exists an index i_1 such that $k_1 \leq H(X_1 ... X_{i_1}) \leq k_1 + 1$ – this is our first block, B_1. Now, the conditional entropy of X conditioned on B_1 is $H(X_{i_1+1} ... X_n | X_1 ... X_{i_1}) = H(X) - H(X_1 ... X_{i_1}) \geq k - k_1$. We can thus continue and find i_2 such that $k_2 \leq H(X_{i_1+1} ... X_{i_2} | X_1 ... X_{i_1}) \leq k_2 + 1$. This process can be continued as long as $\sum_j k_j < k$, and gives our block-wise source.

As before, this indeed gives blocks which have high entropy conditioned on the previous ones, but cannot directly give a real block-wise source, which requires high min-entropy. Again, the solution is to mimic the above idea for each input value $x_1 ... x_n$ separately – which requires non-trivial details that we give in Section 4.

An important aspect omitted is how these $i_1, \ldots, i_t$ are explicitly found. This indeed cannot be done without knowing the source X. However, it turns out that for dispersers we can simply try all the choices, and combine them together (i.e. take the union of edges that each choice implies). As long as t is relatively small, the total number of choices is manageable. Two more tricks can be used. In [SSZ95] a "universal" set of choices for $i_1, \ldots, i_t$ with smaller (for the interesting cases – polynomial) size is presented, thus achieving good dispersers (for certain ranges of parameters). In Section 4 we present a method by which these many choices for $i_1, \ldots, i_t$ can be "merged", thus achieving near-optimal constructions of extractors.

3.4 Some Constructions of Extractors

In Section 3.3 we saw how to convert any random source with high min-entropy into a block-wise source; in Section 3.2 we saw how to extract randomness from such sources, using a composition of basic extractors; and in Section 3.1 we saw how to build these basic extractors. Let us see several ways by which these ingredients can be put together in order to get good extractors.

We first discuss the basic strategy used in [Zuc90, Zuc91, NZ93, SZ94] and present it essentially in the form found in [SZ94].

Choose $t = O(\log n)$ blocks each of size $O(k/\log n)$ as described in Section 3.3. This gives a (near) block-wise source. We now compose t extractors each which multiplies the number of random bits by a constant factor – as presented in Section 3.1. We start with $O(\log n)$ bits, which after going through the composed t extractors get multiplied all the way to the min-entropy present in the final block, $\tilde{O}(k/n \cdot k/\log n)$ – which is our output ($m = \tilde{\Omega}(k^2/n)$).

The total number d of truly random bits used by this construction is the sum of the following: (A) $O(\log n)$ bits for the random input of the first extractor; (B) $O(\log n)$ bits for each of the t blocks – used to choose the pairwise independent set used to get the block. All together this gives $d = O(\log^2 n)$. As remarked in Section 3.3. this can work as long as $k = H_\infty(X) \geq \tilde{\Omega}(\sqrt{n})$ (which also reflects in the bound $m = \tilde{\Omega}(k^2/n)$).

Thus, we get:

Lemma 3.11 *[SZ94]* [5] *Let $k(n) \geq n^{1/2+\gamma}$ for some constant $\gamma > 0$, then for any ϵ there is an explicit $(k(n), \epsilon)$ extractor $E : (n) \times (O(log^2 n \cdot log(\frac{1}{\epsilon}))) \mapsto (\frac{k^2(n)}{n})$.*

For $k = H_\infty(X) = \Omega(n)$, we can choose the sizes of the t blocks more carefully, making sure that the final block has length $\Omega(n)$, and thus we can extract $m = \Omega(n)$ bits.

Reducing d

We now show a simple idea from [SZ94], showing that by composing two of the extractors just designed we can reduce d even further.

Let us first choose a block of length $k/2$ and a second block of length $n' = 2^{O(\sqrt{\log n})}$. Choose an extractor for the second block as described above – taking into account that now the length is n' (instead of n). Thus, using $O(\log^2 n') = O(\log n)$ bits we can extract many more random bits. A simple calculation will show that the number of bits extracted is more than the $O(\log^2 n)$ bits required for extraction of randomness from the first block. The total number of bits required is twice $O(\log n)$ bits for getting the two blocks, and $O(\log n)$ bits for random input to the second block.

The error parameter ϵ of this construction depends on the smaller block and is $1/poly(n')$, which is larger than $1/poly(n)$. By carefully controlling the composition of many extractors of differing lengths, [SZ94, Zuc] show that $d = O(\log n + \log \epsilon^{-1})$ suffices for any ϵ as long as $k = \Omega(n)$ (and even slightly less).

A different way of decreasing d is given in the construction of our second extractor in Section 5.

Increasing m

A simple repetition can increase the value of m. After using d_1 bits to extract m_1 bits from a source X with $H_\infty(X) \geq k$, we can use d_2 fresh random bits to extract m_2 more bits from X, as long as we use an extractor that can extract random bits from sources with minentropy $k - m_1$. These m_2 bits are independent from the first m_1 bits since, even conditioned on

[5] The parameters here are simplified. The real parameters appearing in [SZ94] are somewhat better.

any particular value of the first m_1 bits, X (almost always) still has minentropy of at least $k - m_1$, and thus the extractor produces random bits. This idea is implicit in [WZ93].

The above idea uses $d_1 + d_2$ bits to extract $m_1 + m_2$ bits and can of course be repeated until $k - \sum m_i$ is too small. Repeating the previous construction $O(1)$ times allows extracting $m = k(1-\eta)$ random bits using only $d = O(\log n)$ truly random bits for any constant $\eta > 0$ and $k = \Omega(n)$ [Zuc].

Decreasing k

This is done using the "alternative method" sketched in Section 3.3, and will be presented in Section 4.

4 The First Construction: An Extractor for Any Min-entropy!

In this section we present our first new extractor. The extractor works for any min-entropy, and extracts almost all of the min-entropy from the given source. We start with some preliminaries, and we continue with an informal construction followed by rigorous constructions and proofs.

4.1 An Informal Construction

Recall the "alternative method" for getting a block-wise source, presented in Section 3.3. The argument there shows that for any source $X = X_1 \ldots X_n$ and any $k < H(X)$, there is a splitting point $1 < i < n$ s.t. $k < H(X_1 \ldots X_i) < k+1$ and $H(X_{[i+1,n]} \mid X_{[1,i]}) = H(X) - H(X_{[1,i]})$.

Unfortunately, such a splitting point does not exist when we consider min-entropy. This can be demonstrated by considering the uniform distribution over the set $\{\ 0 \circ \{0,1\}^k \circ 0^{n-k-1}\ ,\ 1 \circ 0^{n-k-1} \circ \{0,1\}^k\ \}$, where $k < n/2$. To get a splitting point with $H_\infty(X_{[1,i]}) \geq 1$ we have to take $i > n-k$, but then $H_\infty(X_{[i+1,n]} \mid X_{[1,i]})$ is 0.

Instead, for any source X with enough min-entropy, and most strings $x \in X$, there is some splitting point $1 \leq i \leq n$ that splits x into $x_1 \circ x_2$ s.t. both $\Pr(X_1 = x_1)$ and $\Pr(X_2 = x_2 \mid X_1 = x_1)$ are small. E.g., in the distribution given above strings of the form $0 \circ \{0,1\}^k \circ 0^{n-k-1}$ have their splitting point i in the range $[1,k]$ while strings of the form $1 \circ 0^{n-k-1} \circ \{0,1\}^k$ splits in the range $[n-k, n]$. Thus, instead of having one global splitting point each string has its own "good" splitting point.

Lemma 4.1 *Let $\overline{X}$ be a distribution over $\{0,1\}^n$ with $H_\infty(\overline{X}) \geq k_1 + k_2 + s$. Call an $x \in \overline{X}$ "good", if there is some i (dependent on x) s.t.*

- $\Pr(X_{[1,i]} = x_{[1,i]}) \leq 2^{-k_1}$ *and*
- $\Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) \leq 2^{-k_2}$

Then $\Pr_{x \in \overline{X}}$ *(x is* not *good)* $\leq 2^{-s}$.

Proof: Let $x \in \overline{X}$. Let i be the first location splitting x into two blocks $x_{[1,i]} \circ x_{[i+1,n]}$ s.t.

$$\Pr(X_{[1,i]} = x_{[1,i]}) \leq 2^{-k_1} \tag{1}$$

Since i is the first such location,

$$\Pr(X_{[1,i-1]} = x_{[1,i-1]}) \geq 2^{-k_1} \tag{2}$$

Since $H_\infty(\overline{X}) \geq k_1 + k_2 + s$

$$\Pr(X_{[1,n]} = x_{[1,n]}) \leq 2^{-(k_1+k_2+s)} \tag{3}$$

Putting this together we get:

$$\begin{aligned}
\Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) &= \frac{\Pr(X_{[1,n]} = x_{[1,n]})}{\Pr(X_{[1,i]} = x_{[1,i]})} \\
&= \frac{\Pr(X_{[1,n]} = x_{[1,n]})}{\Pr(X_{[1,i-1]} = x_{[1,i-1]}) \cdot \Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]})} \\
&\leq \frac{2^{-(k_1+k_2+s)}}{2^{-k_1} \cdot \Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]})}
\end{aligned}$$

Hence, for all strings $x \in X$ s.t. $\Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]}) \geq 2^{-s}$, it holds that $\Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) \leq 2^{-k_2}$, and x is good. In particular $\Pr(x \text{ is not good}) \leq 2^{-s}$. $\square$

A crucial point is that there are only n possible splitting points. If we want to split $x_{[1,n]}$ into t blocks, then we are interested in a good *splitting set*, i.e. a set $\{i_1, \ldots, i_{t-1}\}$ of splitting points that splits $x_{[1,n]}$ into t blocks each having enough min-entropy even given the history. Using the above argument we see that most strings (all but $t \cdot 2^{-s}$) have a good splitting set. Also, clearly, there are at most n^t possible splitting sets. Therefore, we can split the universe $\{0,1\}^n$ to $n^t + 1$ classes, each class containing strings that are good for one particular splitting set, and one for all strings that do not have a good splitting set.

Suppose we are only given inputs x that belong to a specific class S. Then, by definition, X is a block-wise source with the partition S. Therefore, by the results of Section 3, we can extract randomness from it.

Of course, given x we do not know what is the right partition for it. However, since there are so few classes (n^t), we can try all of them. Let us denote by Z_i the output of the block-wise extractor over X assuming the i'th possible partition S_i. Trying all n^t possible partitions gives us n^t outputs $Z_1, \ldots, Z_{n^t}$. Intuitively, one of the n^t output strings is random.

Let us define more precisely the type of source we get. We have $b = n^t$ distributions $Z_1, \ldots, Z_b$, and we know there is some selector function $Y = Y(x)$ (that assigns each good string to a class with a right splitting set), s.t. $(Z_i \mid Y = i) \approx U$. So let us define:

Definition 4.1 *$Z = Z_1 \circ \ldots \circ Z_b$ is a b-block (k, ϵ, η) somewhere random source, if each Z_i is a random variable over $\{0,1\}^k$, and there is a random variable Y over $[0..b]$ s.t.:*

- *For any* $i \in [1..b]$: $d((Z_i|Y = i), U_k) \leq \epsilon$.
- $\Pr(Y = 0) \leq \eta$.

We also say that Y *is a* (k, ϵ, η) *selector for* Z.

A b–block (k, ϵ, η)–somewhere random source Z can be viewed intuitively as a source composed of b strings of length k with a selector function that for all but an η fraction of the inputs finds a block that is ϵ quasi-random. The following lemma (proved in Appendix A) states that such a source is $\epsilon + \eta$ close to a "pure" $(k, 0, 0)$ somewhere random source, and that it contains k min-entropy.

Lemma 4.2 *(1) Any* (k, ϵ, η) *somewhere random source* Z *is* $\epsilon + \eta$ *close to a* $(k, 0, 0)$–*somewhere random source* Z'. *(2) For any* $(k, 0, 0)$ *somewhere random source* Z, $H_\infty(\overline{Z}) \geq k$.

Notice the nice and simple structure somewhere random sources have. We will see (Section 4.5) that it is much easier to extract randomness from such sources. Let us call an extractor working only on somewhere random sources a "somewhere random merger":

DEFINITION 4.2 $M : (k)^b \times (d) \mapsto (m)$ *is an* ϵ*–somewhere random merger, if for any* b*–block* $(k, 0, 0)$ *somewhere random source* Z, *the distribution of* $E(z, y)$ *when choosing* $z \in \overline{Z}$ *and* $y \in U_d$, *is* ϵ *close to* U_m.

Note that by Lemma 4.2 it is indeed enough to define somewhere random mergers only for pure $(k, 0, 0)$ somewhere random sources.

Next we define:

DEFINITION 4.3 *We say* $M = \{M_n\}$ *is an explicit* ϵ *–somewhere random merger, if there is a Turing machine that given an input* z, y *to* M_n *computes* $M_n(z, y)$ *in polynomial time (in the length of the inputs* z, y*).*

We will see that it is not hard to build efficient somewhere random mergers. Building on that, our extractor does the following:

1. Try all $b = n^t$ partitions of x into $t = \Theta(log(n))$ blocks.
2. For each partition set S_i, consider X as a block-wise source with the partition S_i and use the techniques of Section 3 to extract the randomness from it. Call the output Z_i.
3. $Z = Z_1 \circ \ldots \circ Z_b$ form a somewhere random source. Use a merger to merge the randomness in Z into a single almost uniform distribution.

In the coming sections we rigorously develop the above ideas. The formal presentation differs from the informal ideas above in two ways: first, the formal construction is done in polynomial time as opposed to time $n^{O(log(n))}$ above. Second, in the formal description we will give full formal proofs, and thus we will have to specify all the details needed to implement it. To ease the reading, we advise the reader to keep this intuitive and informal construction in mind.

4.2 Composing Two Extractors

In the previous section we suggested an extractor that works by trying all partitions into $t = \Theta(log(n))$ blocks, and then merging all the results. In this section we build an extractor that works by trying all partitions into *two* blocks (i.e. $t = 2$). It turns out that once we build such an extractor, we can build an extractor that works for any t.

ALGORITHM 4.1 *Suppose $E_1 : (n) \times (d_1) \mapsto (m_1 = d_2)$ is an (k_1, ζ_1)–extractor, $E_2 : (n) \times (d_2) \mapsto (m_2)$ is an (k_2, ζ_2)–extractor, and $M : (m_2)^n \times (\mu_1) \mapsto (m)$ is a ζ_3–somewhere random merger. Define the function $E_2 \overset{M}{\odot} E_1 : (n) \times (d_1 + \mu_1) \mapsto (m)$ as follows:*

Given $a \in \{0,1\}^n$, $r_1 \in \{0,1\}^{d_1}$, $r_2 \in \{0,1\}^{\mu_1}$:

1. *Let $q_i = E_1(a_{[i,n]}, r_1)$ and $z_i = E_2(a_{[1,i-1]}, q_i)$, for $i = 1, \ldots, n$.*
2. *Let $E_2 \ominus E_1 \;=\; z_1 \circ \ldots \circ z_n$, and $E_2 \overset{M}{\odot} E_1 \;=\; M(E_2 \ominus E_1, r_2)$.*

Theorem 3 *Suppose E_1, E_2 and M are as above. Then for every safety parameter $s > 0$, $E_1 \overset{M}{\odot} E_2$ is an $(k_1 + k_2 + s, \zeta_1 + \zeta_2 + \zeta_3 + 8n2^{-s/3})$–extractor.*

Proof: Obviously, it is enough to show that $E_1 \ominus E_2$ is an $(m_2, \zeta_1 + \zeta_2, 8n2^{-s/3})$–somewhere random source. To prove this, assume $H_\infty(\overline{X}) \geq k_1 + k_2 + s$. Denote by Q_i and Z_i the random variables with values q_i and z_i respectively. Also, let $\epsilon_3 = 2^{-s/3}$, $\epsilon_2 = 2\epsilon_3$, and $\epsilon_1 = 2\epsilon_2$.

We define a selector for $Z \;=\; Z_1 \circ \ldots \circ Z_n \;=\; E_1 \ominus E_2$ in two phases: first we define a function f which is intuitively what the selector function should be: for each string x split it at the last place s.t. the remaining block is still "random" enough (notice the similarity to Lemma 4.1).

DEFINITION 4.4 *Define $f(w)$ to be the last i s.t $\Pr(X_{[i,n]} = w_{[i,n]} \mid X_{[1,i-1]} = w_{[1,i-1]}) \leq (\epsilon_2 - \epsilon_3) \cdot 2^{-k_1}$.*

Some of the splitting points are rare (e.g. if only few strings split at some location i), and therefore may cause strange behavior (see the proof of Lemma 4.1 for an example). Next, we identify these rare (and bad) cases:

DEFINITION 4.5 *Define w to be "bad" if $f(w) = i$ and:*

1. $\Pr_{x \in X}(f(x) = i) \leq \epsilon_1$*, or,*
2. $\Pr_{x \in X}(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \epsilon_2$*, or,*
3. $\Pr_{x \in X}(X_i = w_i \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \epsilon_3$

We denote by B the set of all bad w. We denote by B_i ($i = 1, 2, 3$) the set of all w satisfying condition (i).

We get rid of the bad cases to get our selector function:

DEFINITION 4.6 *Let Y be the random variable obtained by taking the input a and letting $Y = Y(a)$, where:*

$$Y(w) = \begin{cases} 0 & w \textit{ is bad} \\ f(w) & \textit{otherwise} \end{cases}$$

It holds that $\Pr(w \textit{ is bad}) \leq n(\epsilon_1 + \epsilon_2 + \epsilon_3) \leq 8n \cdot 2^{-s/3}$ (the proof is easy, see Appendix C). We complete the proof by showing that $(Z_i \mid Y = i)$ is $\zeta_1 + \zeta_2$–close to uniform.

Claim 4.1 *If* $\Pr(Y = i \mid X_{[1,i-1]} = w_{[1,i-1]}) > 0$ *then* $H_\infty(X_{[i,n]} \mid Y = i \textit{ and } X_{[1,i-1]} = w_{[1,i-1]}) \geq k_1$

Therefore, for any such $w_{[1,i-1]}$, $(Q_i \mid Y = i \textit{ and } X_{[1,i-1]} = w_{[1,i-1]})$ is ζ_1–close to random (since E_1 is an extractor). Hence by Lemma 3.6, the distribution $(X_{[1,i-1]} \mid Y = i) \circ (Q_i \mid Y = i \textit{ and } X_{[1,i-1]} = w_{[1,i-1]})$ is ζ_1–close to the distribution $(X_{[1,i-1]} \mid Y = i) \times U$. But,

Claim 4.2 $H_\infty(X_{[1,i-1]} \mid Y = i) \geq k_2$.

Therefore, using the extractor E_2 we get that $(Z_i \mid Y = i)$ is $\zeta_1 + \zeta_2$–close to uniform.

$\square$

Now we prove Claims 4.1 and 4.2:

Proof: [of Claim 4.1]

For any w s.t. $Y(w) = i$:

$$\begin{aligned}
\Pr(X_{[i,n]} = w_{[i,n]} | X_{[1,i-1]} = w_{[1,i-1]}, Y(x) = i) &\leq \quad (\text{Since } \Pr(A \mid B) \leq \tfrac{\Pr(A)}{\Pr(B)}) \\
\tfrac{\Pr(\ X_{[i,n]} = w_{[i,n]} \ \mid \ X_{[1,i-1]} = w_{[1,i-1]}\ )}{\Pr(Y(x) = i \ \mid \ X_{[1,i-1]} = w_{[1,i-1]}\ )} &\leq \quad (\text{Since } f(w) = i) \\
\tfrac{(\epsilon_2 - \epsilon_3) \cdot 2^{-k_1}}{\Pr(Y(x) = i \ \mid \ X_{[1,i-1]} = w_{[1,i-1]}\ )} &\leq \quad (\text{Claim C.1}) \\
\tfrac{(\epsilon_2 - \epsilon_3) \cdot 2^{-k_1}}{\epsilon_2 - \epsilon_3} &= \quad 2^{-k_1}
\end{aligned}$$

$\square$

Proof: [of Claim 4.2]

Take any $w_{[1,i-1]}$ that can be extended to some w with $Y(w) = i$.

$$\Pr(X_{[1,i-1]} = w_{[1,i-1]}) \;=$$

$$\frac{\Pr(X_{[1,n]} = w_{[1,n]})}{\Pr(X_{[i,n]} = w_{[i,n]} \mid X_{[1,i-1]} = w_{[1,i-1]})} \;=$$

$$\frac{\Pr(X_{[1,n]} = w_{[1,n]})}{\Pr(X_i = w_i | X_{[1,i-1]}) \Pr(X_{[i+1,n]} = w_{[i+1,n]} | X_{[1,i]})}$$

However,

- $\Pr(X_{[i+1,n]} = w_{[i+1,n]} | X_{[1,i]}) \geq (\epsilon_2 - \epsilon_3) 2^{-k_1}$ (Because $f(w) = i$)
- $\Pr(X_i = w_i \mid X_{[1,i-1]} = w_{[1,i-1]}) \geq \epsilon_3$ (Since $w \notin B_3$)
- $\Pr(X_{[1,n]} = w_{[1,n]}) \leq 2^{-(k_1+k_2+s)}$ (Since $H_\infty(X) \geq k_1 + k_2 + s$)

Thus,

$$\Pr(X_{[1,i-1]} = w_{[1,i-1]}) \;\leq\; \frac{2^{-k_2-s}}{\epsilon_3 \cdot (\epsilon_2 - \epsilon_3)} \tag{4}$$

Therefore,

$$\begin{array}{lll}
\Pr(X_{[1,i-1]} = w_{[1,i-1]} \mid Y(x) = i) & \leq & (\text{Since } \Pr(A \mid B) \leq \frac{\Pr(A)}{\Pr(B)}) \\
\frac{\Pr(X_{[1,i-1]} = w_{[1,i-1]})}{\Pr(Y(x)=i)} & \leq & (\text{Eq. (4)}) \\
\frac{2^{-k_2-s}}{\epsilon_3 \cdot (\epsilon_2 - \epsilon_3) \cdot \Pr(Y(x)=i)} & \leq & (\text{Claim (C.2)}) \\
\frac{2^{-k_2-s}}{\epsilon_3 \cdot (\epsilon_2 - \epsilon_3) \cdot (\epsilon_1 - \epsilon_2 - \epsilon_3)} & = & 2^{-k_2}
\end{array}$$

□

4.3 Composing Many Extractors

Now we build an extractor that works by trying all input partitions into t blocks. We do that by defining composition of many extractors.

Definition 4.7 *Suppose $E_i : (n) \times (d_i) \mapsto (d_{i+1} + s_{i+1})$ is an (k_i, ζ_i)–extractor, for $i = 1, \ldots, t$, $s_i \geq 0$ and $s_2 = 0$. Suppose $M_i : (d_{i+2} + s_{i+2})^n \times (\mu_i) \mapsto (d_{i+2})$ is an $\bar{\zeta}_i$ –somewhere random merger, for any $i = 1 \ldots t-1$. We define the function $E = E_t \overset{M_{t-1}}{\odot} E_{t-1} \overset{M_{t-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1$ by induction to equal $E_t \overset{M_{t-1}}{\odot} (E_{t-1} \overset{M_{t-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1)$.*

Theorem 4 *Suppose E_i, M_i, E are as above, then for any safety parameter $s > 0$, $E : (n) \times (d_1 + \mu_1 + \ldots + \mu_{t-1}) \mapsto (d_{t+1})$ is an $(n, \Sigma_{i=1}^{t} k_i + (t-1)s, \Sigma_{i=1}^{t} \zeta_i + \Sigma_{i=1}^{t-1} \bar{\zeta}_i + (t-1)n 2^{-s/3+3})$ -extractor. If E_i, M_i are explicit, then so is E.*

Proof:

Correctness :

By induction on t. For $t = 2$ this follows from Theorem 3. For larger t's this is a straight forward combination of the induction hypothesis and Theorem 3.

Running time :

We compute $E_t \overset{M_{t-1}}{\odot} E_{t-1} \overset{M_{t-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1$ using a dynamic programming procedure:

1. Input: $x \in \overline{X}$, $y \in \{0,1\}^{d_1}$ and $y_j \in \{0,1\}^{\mu_j}$, for $j = 1, \ldots, t-1$.
2. We compute the matrix M where

$$M[j,i] = (E_j \overset{M_{j-1}}{\odot} E_{j-1} \overset{M_{j-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1)(x_{[i,n]}, y \circ y_1 \circ \ldots \circ y_{j-1})$$

for $1 \leq i \leq n$ and $1 \leq j \leq t$.

The entries of the first row of M, $M[1,i]$ can be filled by evaluating $E_1(x_{[i,n]}, y)$. Suppose we know how to fill the j'th row of M. We show how to fill the $j+1$'th row.

- Denote $q_l = M[j,l]$ for $l = i, \ldots, n$, and let $z_l = E_{j+1}(x_{[i,l-1]}, q_l)$.
- Set $M[j+1,i] = M_j(z_i \circ \ldots z_n, y_j)$.

By the definition of composition $M[j,i]$ has the correct value, and clearly, the computation takes polynomial time in n.

$\square$

REMARK 4.1 *Notice that using left associativity (rather than right associativity) we could use some of the quasi-randomness we get, for doing the merges. Thus, it may appear that left associativity is more efficient in terms of the number of truly random bits used. However, we know how to implement right associativity composition in polynomial time (using a dynamic programming procedure) and we do not know of such an algorithm for left associativity composition.*

4.4 Good Mergers Imply Good Extractors

Let us continue following the informal construction given in Section 4.1. We want to try all input partitions into $t = \Theta(log(n))$ blocks, and for each partition we want to use the block-wise extractor of Lemma 3.7. In our new terminology this amounts to taking $E = A_k \overset{M}{\odot} \ldots A_{b^2\bar{k}} \overset{M}{\odot} A_{b\bar{k}} \overset{M}{\odot} A_{\bar{k}}$, where A_m is the extractor of Lemma 3.4 and b is some constant, $1 < b < c_{tiny}$. This extractor extracts $\Omega(k)$ bits from sources having k min-entropy. The only missing component in this construction is the existence of explicit good somewhere random mergers. However assuming the existence of good somewhere random mergers, we get good extractors.

Lemma 4.3 *Suppose for any $\bar{k} \leq k \leq \bar{\bar{k}}$ there is an explicit ϵ somewhere random merger $M_k : (k)^n \times (d) \mapsto (\Delta \cdot k)$, where Δ is a constant s.t. $\frac{1}{c_{tiny}} < \Delta < 1$. Then, for any $\bar{k} \leq k \leq \bar{\bar{k}}$ there is an explicit $(k, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (O(\bar{k} \cdot log(\frac{1}{\epsilon}) + log(n) \cdot d)) \mapsto (\Omega(k))$ extractor.*

Proof: Let $b = c_{tiny} \cdot \Delta$. Clearly b is a constant, and $1 < b < c_{tiny}$. Define $k_i = b^i \cdot \bar{k} \cdot log(\frac{1}{\epsilon})$, and let t be the first integer s.t. $\Sigma_{i=1}^{t} 2k_i \leq \frac{k}{2}$.

Define $E = E_t \overset{M_{t-1}}{\odot} E_{t-1} \ldots \overset{M_1}{\odot} E_1$, where:

- $E_i : (n) \times (k_i) \mapsto (c_{tiny} k_i)$ is the $(k_i, 2^{-k_i/5})$–extractor A_{k_i} from Lemma 3.4
- $M_i : (c_{tiny} \cdot k_{i+1})^n \times (d) \mapsto (k_{i+2} = b \cdot k_{i+1} = \Delta \cdot c_{tiny} \cdot k_{i+1})$ is an ϵ–somewhere random merger given in the hypothesis of the lemma.

Now we use Theorem 4 with $d_i = k_i$ and $s_i = (c_{tiny} - b)k_{i-1}$. To see that the above extractors and mergers can be indeed composed, notice that $d_i + s_i = bk_{i-1} + (c_{tiny} - b)k_{i-1} = c_{tiny} \cdot k_{i-1}$. Therefore, by Theorem 4, E is an extractor.

Now choose the safety parameter s to be $s = \frac{k}{2t}$, and let us check the parameters we get are as stated. Note that $t = O(log(n))$ and that $\Sigma_{i=1}^{t} k_i + (t-1)s < k$. Also note that $k_{t+1} = \Omega(\Sigma_{i=1}^{t} k_i) = \Omega(k)$. It is easy to check that the error is as stated. Finally, since A_i, M_i are explicit, so is E. □

Just to demonstrate the above, assume for every m there is an explicit ϵ somewhere random merger $M : (m)^n \times (d) \mapsto (\Delta \cdot m)$. Then notice that by Lemma 4.3 we get an extractor for any min-entropy that extracts almost all of the min-entropy. Formally, assuming the above, we get for every m an explicit (m, ϵ) extractor $E : (n) \times (d \cdot polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(m))$. Thus, our next objective is designing good somewhere random mergers.

4.5 Explicit Mergers

We start with a somewhere random merger for a 2-block somewhere random source. By definition a (k, ϵ) extractor $E : (2k) \times (d) \mapsto (m)$ extracts randomness from any source X over $\{0,1\}^{2k}$ with $H_\infty(\overline{X}) \geq k$. In particular, by Lemma 4.2, it extracts randomness from any 2-block $(k, 0, 0)$ somewhere random source.

Corollary 4.4 *Any (k, ϵ) extractor $E : (2k) \times (d) \mapsto (m)$ is also an ϵ somewhere random merger $E : (k)^2 \times (d) \mapsto (m)$.*

A b–block Somewhere Random Merger

Now we show how to use 2-block mergers to build b-block mergers. Given a b–block somewhere random source, we merge the blocks in pairs in a tree wise fashion, resulting in a single block. We show that after each level of merges we still have a somewhere random source, and thus the resulting single block is necessarily quasi-random.

ALGORITHM 4.2 *Let $M : (k)^2 \times (d(k)) \mapsto (k - m(k))$ be a merger. We build a merger $M_l : (k)^{2^l} \times (l \cdot d(k)) \mapsto (k - l \cdot m(k))$, by induction on l:*

Input *: $x^l = x^l_1 \circ \ldots x^l_{2^l}$, where each $x^l_i \in \{0,1\}^k$.*

Output *: Let $d = d_1 \circ \ldots d_l$, where d_j is chosen uniformly from $\{0,1\}^{d(k)}$. If $l = 0$ output x^l, otherwise:*

1. *Let $x^{l-1}_i = M(x^l_{2i-1} \circ x^l_{2i} \ , \ d_l)$, for $i = 1, \ldots, 2^{l-1}$.*
2. *Let the output be $M_{l-1}(x^{l-1}_1 \circ \ldots x^{l-1}_{2^{l-1}} \ , \ d_1 \circ \ldots d_{l-1})$.*

Theorem 5 *Assume for every k there is an explicit $\epsilon(k)$ somewhere random merger $M : (k)^2 \times (d(k)) \mapsto (k - m(k))$, for some monotone functions t, k and ϵ^{-1}. Let $M_l : (k)^{2^l} \times (l \cdot d(k)) \mapsto (k - l \cdot m(k))$ be as above. Then M_l is an $l \cdot \epsilon(m - l \cdot k(m)))$ somewhere random merger.*

Proof: For $j = l, \ldots, 0$ denote by Z^j the random variable whose value is $x^j = x^j_1 \circ \ldots x^j_{2^j}$, where the input x is chosen according to $\overline{X}$, and d is uniform. Notice that $\overline{Z^l}$ is the distribution $\overline{X}$, and $\overline{Z^0}$ is the distribution of the output.

The theorem clearly follows the following claim:

Claim 4.3 *Denote $k_j = k - (l-j)m(k)$. If X is a $(k, 0, 0)$ somewhere random sourcewith the selector function Y, then for any $1 \le i \le 2^j$, $d(\ (Z^j_i \mid Y \in [2^{l-j}(i-1)+1, 2^{l-j}i])\ ,\ U_{k_j}\ ) \le (l-j) \cdot \epsilon(k)$*

$\square$

Proof:

The proof is by downward induction on j. The basis $j = l$ simply says that for any i , $d((X_i \mid Y = i)\ ,\ U_k\ ) = 0$, which is exactly the hypothesis. Suppose it is true for j, we prove it for $j - 1$. By the induction hypothesis:

- $d(\ (Z^j_{2i-1} \mid Y \in [2^{l-j}(2i-2)+1, 2^{l-j}(2i-1)])\ ,\ U_{k_j}\ ) \le (l-j)\epsilon(k_j)$
- $d(\ (Z^j_{2i} \mid Y \in [2^{l-j}(2i-1)+1, 2^{l-j}2i])\ ,\ U_{k_j}\ ) \le (l-j) \cdot \epsilon(k_j)$

In Appendix B we prove:

Lemma 4.5 *Let A, B and Y be any random variables. Suppose that $d((A \mid Y \in S_1), U_k) \leq \epsilon$ and $d((B \mid Y \in S_2), U_k) \leq \epsilon$ for some disjoint sets S_1 and S_2. Then $(A \circ B \mid Y \in S_1 \cup S_2)$ is ϵ–close to some $\overline{X}$ with $H_\infty(\overline{X}) \geq k$.*

Therefore:

- $(Z^j_{2i-1} \circ Z^j_{2i} \mid Y \in [2^{l-j+1}(i-1)+1, 2^{l-j+1}i])$ is $(l-j) \cdot \epsilon(k_j)$ close to some $\overline{W}$ with $H_\infty(\overline{W}) \geq k_j$.
- Since $Z^{j-1}_i = M(Z^j_{2i-1} \circ Z^j_{2i}\ ,\ d_j)$, it follows that $(Z^{j-1}_i \mid Y \in [2^{l-j+1}(i-1)+1, 2^{l-j+1}i])$ is $(l-j) \cdot \epsilon(m)$ close to $M(x, d_j)$ where $x \in \overline{X}$ and $H_\infty(\overline{X}) \geq k_j$. Therefore, it is $(l-j) \cdot \epsilon(k_j) + \epsilon(k_j)$ close to random, as required.

□

Remark 4.2 *Notice that we use the same random string d_j for all merges occurring in the j'th layer, and that this is possible because in a somewhere random source we do not care about dependencies between different blocks. Also notice that the error is additive in the depth of the tree of merges (i.e. in l), rather than in the size of the tree (2^l).*

4.6 Putting It Together

Recall that by Lemma 4.3 having mergers $M : (k)^n \times (d) \mapsto (\Delta k)$, where Δ is some constant, implies the existence of good extractors. Theorem 5 states that it is enough to find good 2-block somewhere random mergers that do not lose much of the min-entropy. We also saw (Corollary 4.4) that it is enough to find a (k, ϵ) extractor $E : (2k) \times (d) \mapsto (m)$ with m very close to k. To be more specific, we need $m \geq k - O(\frac{k}{log(n)})$. By [NZ93, SZ94, Zuc] we know to find such extractors with $m = \Omega(k)$. Using a simple idea due to Wigderson and Zuckerman [WZ93] we can get m much closer to k.

More Bits Using The Same Extractor

Suppose we have an extractor E that extracts randomness from any source having at least k min-entropy. How much randomness can we extract from sources having K min-entropy when $K >> k$?

The following algorithm is implicit in [WZ93]: use the same extractor E many times over the same string x, each time with a fresh truly random string r_i, until you get $K - k$ output bits. The idea is that as long as $|E(x, r_1) \circ \ldots \circ E(x, r_t)|$ is less then $K - k$, with high probability $(X \mid E(x, r_1) \circ \ldots \circ E(x, r_t))$ still contains k min-entropy, and therefore we can use the extractor E to further extract randomness from it. Thus, we have the following two lemmas, that are proven in detail in Appendix D:

Lemma 4.6 *Suppose that for some k there is an explicit (k, ϵ) extractor $E_k : (n) \times (d) \mapsto (m)$. Then, for any $K \geq k$, and any safety parameter $s > 0$, there is an explicit $(K, t(\epsilon + 2^{-s}))$ extractor $E : (n) \times (td) \mapsto min\{tm, K - k - s\}$*

Lemma 4.7 *Suppose that for any $k \geq \bar{k}$ there is an explicit $(k, \epsilon(n))$ extractor $E_k : (n) \times (d(n)) \times (\frac{k}{f(n)})$. Then, for any k, there is an explicit $(k, f(n)log(n)(\epsilon + 2^{-d(n)}))$ extractor $E : (n) \times (O(f(n)log(n)d(n))) \mapsto (k - \bar{k})$.*

As a side corollary we can strengthen Lemma 4.7, to get an extractor that extracts the whole entropy of the source rather than just a constant fraction of it. This is done by applying Lemma 4.7, or more specifically, by using the same extractor $O(log(n))$ times. Thus, combining Lemma 4.3 and Lemma 4.7 we get:

Corollary 4.8 *Suppose $\bar{k} = \bar{k}(n)$ is a function s.t. for every $k \geq \bar{k}(n)$ there is an explicit ϵ somewhere random merger $M : (k)^n \times (d) \mapsto (\Delta \cdot k)$, where $\frac{1}{c_{tiny}} < \Delta < 1$. Then for any k there is an explicit $(k, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (O(\bar{k} \cdot log(n) \cdot log(\frac{1}{\epsilon}) + log^2(n) \cdot d)) \mapsto (k)$.*

Mergers That Do Not Lose Much Min-entropy

The [SZ94] extractor of Lemma 3.11 works for any source with $H_\infty(\overline{X}) \geq n^{1/2+\gamma}$. Thus, using Lemma 4.6 by repeatedly using the [SZ94] extractor, we can extract at least $\frac{n}{2} - n^{1/2+\gamma}$ quasi-random bits from a source having $H_\infty(\overline{X}) \geq \frac{n}{2}$. Thus, we have a 2–merger that does not lose much randomness in the merging process. Applying Theorem 5 we get a good n–merger. Thus:

Lemma 4.9 *Let $b > 1$ be a constant and suppose $f = f(k) = f(k(n))$ is a function s.t. $f(k) \leq \sqrt[3]{k}$ and for every $k \geq k_0(n)$ it holds that $f(k) \geq b \cdot log(n)$. Then for every $k \geq k_0$ there is an explicit $log(n) \cdot poly(k) \cdot \epsilon$ somewhere random merger $E : (n) \times (log(n) \cdot polylog(k) \cdot f^2(k) \cdot log(\frac{1}{\epsilon})) \mapsto k - \frac{k}{b}$.*

Proof:

- By Lemma 3.11 there is an explicit $(\frac{k}{f(k)}, \epsilon)$ extractor $E : (k) \times (O(log^2 k \cdot log(\frac{1}{\epsilon}))) \mapsto (\frac{k}{f^2(k)})$.
- By Lemma 4.6 there is an explicit $(k, poly(k) \cdot \epsilon)$ extractor $E : (2k) \times (O(f^2(k) \cdot log^2 k \cdot log(\frac{1}{\epsilon})) \mapsto (k - \frac{k}{f(k)})$.
- By Theorem 5 there is an explicit $log(n) \cdot poly(k) \cdot \epsilon$ somewhere random merger $M : (k)^n \times (O(log(n) \cdot polylog(k) \cdot f^2(k) \cdot log(\frac{1}{\epsilon})) \mapsto (k - log(n) \cdot \frac{k}{f(k)})$. Since $\frac{k}{f(k)} \leq \frac{k}{b \cdot log(n)}$ for any $k \geq k_0$, we have that $log(n) \cdot \frac{k}{f(k)} \leq \frac{k}{b}$.

$\square$

Corollary 4.10 *For every $k \geq 2^{\sqrt{log(n)}}$, there is a $polylog(n) \cdot \epsilon$ somewhere random merger $M_k : (k)^n \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(k))$*

Proof: Take $f(k) = log^c k$ for some constant $c > 2$. For any constant b, $k \geq 2^{\sqrt{log(n)}}$ and n large enough, $log^c m \geq b \cdot log(n)$, and the corollary follows Lemma 4.9. $\Box$

Notice that Theorem 5 and Corollary 4.10 take advantage of the simple structure of somewhere random sources, giving us an explicit somewhere random merger that works even for sources with very small min-entropy to which the [SZ94] extractor of Lemma 3.11 does not apply.

Extractors That Work For High Min-entropy

Corollary 4.10 asserts the existence of good mergers for $k \geq 2^{\sqrt{log(n)}}$, and therefore plugging this into Corollary 4.8 we get:

Corollary 4.11 *For every k there is a $(k, poly(n) \cdot \epsilon)$ extractor* $B_k : (k) \times (O(2^{\sqrt{log(n)}} \cdot polylog(n) \cdot log(\frac{1}{\epsilon}))) \mapsto (k)$.

The extractor B in Corollary 4.11 uses $O(2^{\sqrt{log(n)}} \cdot polylog(n) \cdot log(\frac{1}{\epsilon}))$ truly random bits to extract all the randomness in the given source. Although $2^{\sqrt{log(n)}}$ is quite a large amount of truly random bits, we can use the [SZ94] extractor to extract $n^{1/3}$ bits from $n^{2/3}$ min-entropy, and then use these $n^{1/3} >> O(2^{\sqrt{log(n)}} \cdot polylog(n) \cdot log(\frac{1}{\epsilon}))$ bits to further extract all the remaining min-entropy. More precisely, if B is the extractor in Corollary 4.11, E_{sz} the extractor from Lemma 3.11 and M is the merger from Corollary 4.10, then $E = B \overset{M}{\odot} E_{sz}$ extracts $\Omega(k)$ bits from sources having $k \geq n^{2/3}$ min-entropy, using only $polylog(n)$ truly random bits! That is, we get the following lemma:

Lemma 4.12 *Let $\epsilon \geq 2^{-n^{\gamma}}$ for some constant $\gamma < 1$. There is some constant $\beta < 1$ s.t. for every $k \geq n^{\beta}$ there is an explicit $(k, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(k))$.*

Proof: Choose $\delta = \frac{1-\gamma}{2}$ and $\beta = 1 - \frac{\delta}{2}$. Let the extractor E be $E = B_k \overset{M}{\odot} E_{sz}$ where

- E_{sz} is the (n^{β}, ϵ) extractor $E : (n) \times (O(log^2 n \cdot log(\frac{1}{\epsilon}))) \mapsto (n^{2\beta-1})$ of Lemma 3.11.
- B_k is the extractor from Corollary 4.11.
- M is the merger from Corollary 4.10.

Since $n^{2\beta-1} = n^{\delta} \cdot n^{\gamma} = \Omega(2^{\sqrt{log(n)}} \cdot log(\frac{1}{\epsilon}))$, $E = B_k \overset{M}{\odot} E_{sz}$ is well-defined. By Theorem 3, for every k, E is an explicit $(k + n^{\beta} + n^{\gamma}, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(k))$. In particular if $H_{\infty}(X) = \Omega(n^{\beta})$ we extract $\Omega(H_{\infty}(X))$ as required. $\Box$

The Final Result

Now that we know how to extract all the randomness from sources having $\Omega(n^{\beta})$ min-entropy with only $polylog(n)$ truly random bits, by Lemmas 4.7 and Theorem 5 we have good somewhere random mergers, for *every* k. Thus by Corollary 4.8 we have good extractors for every k.

Theorem 6 *For every constant $\gamma < 1$, $\epsilon \geq 2^{-n^\gamma}$, and every $k = k(n)$ there is an explicit (k, ϵ) extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (k)$.*

Proof:

- By Lemma 4.7, Lemma 4.12 implies an explicit $(n, poly(n) \cdot \epsilon)$ extractor $E : (2n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (n - n^\beta)$.
- There is some constant c (that depends only on γ) s.t. for every $log^c n \leq k \leq n$, $log(n) \cdot k^\beta \leq \frac{k}{\bar{c}}$, where $\bar{c}$ is some constant s.t. $\frac{1}{c_{tiny}} < 1 - \frac{1}{\bar{c}} < 1$ (e.g. $\bar{c} = \frac{2c_{tiny}}{c_{tiny}-1}$). Therefore by Theorem 5, for every k there is an explicit $poly(n) \cdot \epsilon$ somewhere random merger $M : (k)^n \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (m - \frac{m}{\bar{c}})$.
- By Corollary 4.8, this implies an explicit $(k, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (k)$ for any k. Plugging $\epsilon' = \frac{\epsilon}{poly(n)}$, gives the theorem.

$\square$

5 The Second Construction: An Extractor Using Less Truly Random Bits

In this section we build an extractor that uses less truly random bits. The extractor works only for sources having at least $n^{\Omega(1)}$ min-entropy, and extracts only some small fraction of the min-entropy present in the original source. Yet, this extractor improves upon the previous construction of [SZ94] in two ways: first it uses much less truly random bits (almost linear), and second, it works for sources having less than $n^{1/2}$ min-entropy.

Theorem 7 *For every constant c and $\gamma > 0$ there is some constant $\delta > 0$ and an $(n^\gamma, \frac{1}{n})$ extractor $E : (n) \times (O(log(n) log^{(c)} n)) \mapsto (\Omega(n^\delta))$, where $log^{(c)} n = \underbrace{loglog \ldots log}_{c}\ n$.*

The extractor uses two main building blocks: The first shows how to reduce the number of truly random bits needed for sources having $n^{1/2}$ min-entropy. The second shows how to use extractors for $n^{1/d}$ min-entropy to achieve extractors for $n^{1/d+1}$ min-entropy.

Lemma 5.1 *Let $f(n)$ be an arbitrary function. Assume $\forall \gamma > 0$, $\exists \delta > 0$ s.t. there is a $(k = n^\gamma, n^{-\Omega(1)})$ extractor $E : (n) \times (log(n) \cdot f(n)) \mapsto (m = \Omega(n^\delta))$. Then $\forall \gamma' > 0$, $\exists \delta' > 0$ s.t. there is a $(k = n^{\frac{1}{2}+\gamma'}, \epsilon = n^{-\Omega(1)})$ extractor $E' : (n) \times (O(log(n) \cdot log(f(n)))) \mapsto (\Omega(n^{\delta'}))$.*

Lemma 5.2 *Assume*

- *$\forall \gamma' > 0$, $\exists \delta' > 0$ s.t. there exists a $(k = n^{\gamma'}, n^{-\Omega(1)})$ extractor $E : (n) \times (O(log(n) \cdot 2^{f(n)})) \mapsto (\Omega(n^{\delta'}))$.*

- $\forall \gamma'' > 0,\ \exists \delta'' > 0$ *s.t. there exists a* $(k = n^{\frac{1}{d}+\gamma''}, n^{-\Omega(1)})$ *extractor* $F : (n) \times (O(log(n) \cdot f(n))) \mapsto (\Omega(n^{\delta''}))$.

Then $\forall \gamma > 0, \exists \delta > 0$ *s.t. there exists a* $(k = n^{\frac{1}{d+1}+\gamma}, n^{-\Omega(1)})$ *extractor* $E' : (n) \times (O(log(n) \cdot f(n))) \mapsto (\Omega(n^{\delta}))$.

Using these two lemmas we can prove Theorem 7:

Proof: [of Theorem 7]

We prove the equivalent claim:

Claim: For every constant c and $\gamma > 0$ there is some constant $\delta > 0$ and an $(n^{\gamma}, \frac{1}{n})$ extractor $E : (n) \times (O(log(n) \cdot (log^{(c)} n)^{a})) \mapsto (\Omega(n^{\delta}))$, where a is some fixed constant.

Proof:

By induction on c. For $c = 1$ this follows from Theorem 6.

Assume for c. Denote $f(n) = log^{(c+1)} n$. The induction hypothesis states that $\forall \gamma > 0,\ \exists \delta > 0$ s.t. there is an $(k = n^{\gamma}, n^{-\Omega(1)})$ extractor $E : (n) \times (log(n) \cdot 2^{O(f(n))}) \mapsto (\Omega(n^{\delta}))$.

By Lemma 5.1, $\forall \gamma' > 0,\ \exists \delta' > 0$ s.t. there is an $(n^{1/2+\gamma'}, n^{-\Omega(1)})$ extractor $E' : (n) \times (O(log(n) \cdot f(n))) \mapsto (\Omega(n^{\delta'}))$.

But now all the requirements of Lemma 5.2 are met, and therefore, using Lemma 5.2 repeatedly $\frac{1}{\gamma}$ times, we get the desired extractor. □ □

5.1 A Better Extractor For Sources Having $n^{1/2+\gamma}$ Min-entropy

In this section we prove Lemma 5.1. We show that combining the extractor of Theorem 6 with the [NZ93] block extractor, we can extract randomness from sources having $n^{\frac{1}{2}+\gamma}$ min-entropy using less random bits. The idea behind the construction is the following: since the given source X has $H_{\infty}(\overline{X}) \geq n^{\frac{1}{2}+\gamma'}$, we can use the [NZ93] block extraction to extract $t = O(log(f(n)))$ blocks that together form a block-wise source with each block containing some $n^{\Omega(1)}$ min-entropy. Then, by investing $O(log(n))$ bits, we can extract some $log(n) \cdot 2^{\Omega(t)} = log(n) \cdot f(n)$ random bits. Finally, we can use these bits in the extractor given by the hypothesis of the lemma, to extract $n^{\Omega(1)}$ quasi-random bits.

Proof: [of Lemma 5.1]

Consider the following algorithm:

ALGORITHM 5.1 *Fix* $t = O(log(f(n)))$, $l = n^{1/2}$. *Choose* $y_1, \ldots, y_t \in \{0,1\}^{O(log(n))}$, *and* $y \in \{0,1\}^{O(log(n))}$. *Given* $x \in X$:

1. *Extract* t *blocks* $b_1 = B(x, y_1), \ldots, b_t = B(x, y_t)$, *where* B *is the block extraction operator of Lemma 3.9.*

2. *compute* $z = BE(b_2 \ldots b_t, y)$, *where* BE *is the function extracting randomness from block-wise sources, of Lemma 3.7.*

3. Finally, let the output be $E(b_1, z)$, where E is the extractor given in the hypothesis.

To prove correctness, notice that,

Claim: Fix $y_1, \ldots, y_t$ arbitrarily. The probability that there exists an $1 \leq i \leq t$, s.t. $H_\infty(X \mid B_1 = b_1, \ldots, B_i = b_i) \leq n^{1/2+\gamma/2}$ is less than ϵ.

Proof: The total number of bits in $b_1 \ldots b_i$ is at most $t \cdot l << n^{1/2} \cdot log(n)$. Denote

$$Bad = \{b_1 \circ \ldots \circ b_i \mid \Pr(b_1 \ldots b_i) \leq 2^{-n^{1/2+\gamma/2}}\}$$

Then, $\Pr(Bad) \leq |Bad| \cdot 2^{-n^{1/2+\gamma/2}} << n^{-\Omega(1)}$. Also, for any $b_1 \ldots b_i \notin Bad$, and any x:

$$\begin{aligned} \Pr(X = x \mid B_1 = b_1, \ldots, B_i = b_i) &\leq \frac{\Pr(X = x)}{\Pr(B_1 = b_1, \ldots, B_i = b_i)} \\ &\leq \frac{2^{-n^{1/2+\gamma}}}{2^{-n^{1/2+\gamma/2}}} \\ &<< 2^{-n^{1/2+\gamma/2}} \end{aligned}$$

and therefore for most prefixes, $H_\infty(X \mid B_1 = b_1, \ldots, B_i = b_i) \geq n^{1/2+\gamma/2}$ as required. □

Therefore, using the block extractor of Lemma 3.9 we get:

Claim: $B = B_1 \circ \ldots \circ B_t$ is an $(n^{\gamma/3}, \ldots, n^{\gamma/3})$ block-wise source to within $n^{-\Omega(1)}$.

Proof: For almost every prefix $b_1, \ldots, b_{i-1}$, $H_\infty(X \mid B_1 = b_1, \ldots, B_i = b_i) \geq n^{1/2+\gamma/2}$. Therefore, by Lemma 3.9, for almost every prefix $b_1, \ldots, b_{i-1}$, $(B_i \mid B_1 = b_1, \ldots, B_{i-1} = b_{i-1})$ is close to a distribution with at least $n^{\gamma/3}$ min-entropy. □

Therefore,

Claim: $\overline{B_1 \times Z}$ is $n^{-\Omega(1)}$ close to $\overline{B_1} \times U$

Proof: For most prefixes b_1, $(B_2 \times \ldots \times B_t \mid B_1 = b_1)$ is a block-wise source. Therefore, by Lemma 3.7, $(Z \mid B_1 = b_1)$ is close to uniform. Hence, the claim follows by Lemma 3.6. □

Notice that $\overline{Z}$ is distributed over $log(n) \cdot 2^{\Omega(d)} = O(log(n) \cdot f(n))$ bits, therefore applying the extractor E, we get $\Omega(n^{\delta'})$ quasi random bits. □

5.2 An Extractor For n^γ Min-entropy.

Here we prove Lemma 5.2. We show how given an extractor for sources with $n^{\frac{1}{d}+\gamma}$ min-entropy, we can build an extractor for sources with $n^{\frac{1}{d+1}+\gamma}$ min-entropy. The idea is to extract t blocks. If all the blocks took their "fair share" of randomness, then they form a block-wise source, and we can treat them as before. If they do not form a block-wise source, then it must be the case that one of the blocks "stole all the randomness" present in the source. But then this block is much more condensed, and we can extract randomness from it.

Algorithm 5.2 *Fix* $t = O(f(n))$ *and* $l = n^{1-\frac{1}{d+1}}$.
Choose $r_1, \ldots, r_t \in \{0,1\}^{O(log(n))}$, $r', r''' \in \{0,1\}^{O(log(n) \cdot f(n))}$ *and* $r'' \in \{0,1\}^{O(log(n)}$.

Compute:

- $b_i = B(x, r_i)$ *for* $i = 1, .., t$*, where* B *is the block extraction operator of Lemma 3.9.*
- $b'_i = F(b_i, r')$ *for* $i = 1, .., t-1$*, where* F *is given in the hypothesis of the lemma.*
- $b'_t = E(b_1, BE(b_2, \ldots, b_t, r''))$*, where* BE *is the function extracting randomness from block-wise sources, of Lemma 3.7.*

Let the output be $F(b'_1 \circ \ldots \circ b'_t, r''')$.

Proof: [of Lemma 5.2]:

Let us denote $B = (B_1, \ldots, B_t)$ and $B' = (B'_1, \ldots, B'_t)$. We will soon prove that:

Claim 5.1 B' *is a* $(t, k = n^{\Omega(1)}, n^{-\Omega(1)})$ *somewhere random source.*

Hence, by Lemma 4.2, B' is $n^{-\Omega(1)}$-close to some $\overline{B''}$ that is distributed over $N = kt$ variables and has $H_\infty(\overline{B''}) \geq k$. Thus, $\overline{B''}$ is very "condensed", i.e. $H_\infty(\overline{B''}) \geq k >> N^{2/3}$. Thus, by the hypothesis, $F(B', r''')$ is $n^{-\Omega(1)}$-close to the uniform distribution. □

Proof: [of Claim 5.1]

First we define the selector function. Given $b \in B$ we look for a prefix i s.t. the min-entropy of $(X \mid B_{[1,i]} = b_{[1,i]})$ has dropped significantly. If such an i exists, and assume i_0 is the first such i, then it means that the i_0 block "stole" a lot of randomness, and we let $f(b) = i_0$. If no such i exists, we expect B to be a block-wise source, and we let $f(b) = t$.

Now we have to quantify what "dropped significantly" means. Initially, $H_\infty(\overline{X}) \geq n^{\frac{1}{d+1}+\gamma}$. Let us denote $\mu_0 = n^{\frac{1}{d+1}+\gamma}$. If no block stole randomness, at the end we expect $H_\infty(X \mid B_{[1,t]} = b_{[1,t]})$ to be at least $\frac{\mu_0}{2}$. So let us define, for $i = 1, \ldots, t-1$, $\mu_i = \mu_0 - \frac{i}{2t}\mu_0$, and for $b \in B$ let the selector function $f(b)$ be:

$$f(b) = \begin{cases} i & \text{if } H_\infty(X \mid B_{[1,i]} = b_{[1,i]}) < \mu_i, \\ & \text{and this first happens at } i \\ t & \text{otherwise} \end{cases}$$

Now we "fix" the selector function to avoid some rare bad cases:

Definition 5.1 *b is bad in either of the following two cases:*

- $f(b) = i \in [1..t-1]$ *and* $\Pr(f = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \epsilon$.
- $f(b) = t$ *and there is some* $1 \leq i \leq t$ *s.t.* $\Pr(f = t \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \epsilon_i$.

where $\epsilon_t = \epsilon = n^{-\Theta(1)}$ *and* $\epsilon_{i-1} = 4\epsilon_i$*, for* $i = 2, \ldots, t$.

Define

$$Y(b) = \begin{cases} 0 & \text{b is bad} \\ f(b) & \text{otherwise} \end{cases}$$

We are going to prove several lemmas. First,

Lemma 5.3 $\Pr(Y = 0) \le t\epsilon + \Sigma_i \epsilon_i$

Second, we will show that

Claim 5.2 *For any* $1 \le i \le t-1$*:* $\quad H_\infty(B_i \mid Y = i) \ge n^{\frac{1}{d+1}+\frac{\gamma}{2}}$.

Therefore, $H_\infty(B_i|Y = i) \ge n^{\frac{d}{d+1}\frac{1}{d}+\frac{d}{d+1}\frac{d+1}{d}\frac{\gamma}{2}} = l^{\frac{1}{d}+\frac{d+1}{d}\frac{\gamma}{2}}$, and by assumption, from such sources F extracts randomness. Hence, $(B'_i \mid Y = i)$ is $n^{-\Omega(1)}$-close to uniform.

Finally, we will show that:

Claim 5.3 *$(B\ |Y = t)$ is an $n^{\gamma/2}$ block-wise source to within $n^{-\Omega(1)}$.*

Therefore, with the conditioning that $Y = t$, $\overline{B_1 \circ BE(B_1 \circ \ldots \circ B_t, r'')}$ is $n^{-\Omega(1)}$ close to $\overline{B_1} \times U_{log(n)\cdot 2^{\Omega(t)}}$. Hence, using the extractor E, $(B'_t \mid Y = t)$ is $n^{-\Omega(1)}$ close to uniform.

Putting it together, B' is a $(t, n^{\Omega(1)}, n^{-\Omega(1)})$ somewhere random source. $\square$

So now we have to prove the above three claims. The proof does not involve any new idea, and is a straight-forward check that indeed all the necessary things hold.

We first need a technical claim which we prove in Appendix E:

Claim 5.4 *For any* $0 < i < t$*:*

1. *For any $b_{[1,i-1]}$ that can be extended to some b with $Y(b) = i$:*

$$\Pr(Y = i \mid B_{[1,i-1]} = b_{[1,i-1]}) = \Pr(f = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \ge \epsilon$$

2. *For any $b_{[1,i-1]}$ that can be extended to some b with $Y(b) = t$:*

$$\Pr(Y = t \mid B_{[1,i-1]} = b_{[1,i-1]}) \ge \epsilon_{i-1} - \Sigma_{j=i}^{t}\ \epsilon_j \ge \epsilon$$

We prove Claim 5.3 in Appendix E. Let us now prove Claim 5.2:

Proof: [of Claim 5.2]

Let $1 \le i \le t-1$. Fix any prefix $b_{[1..i-1]}$ that can be extended to some b_0 with $Y(b_0) = i$. Take any b with that prefix and $Y(b) = i$.

Since $Y(b) = i$,

$$H_\infty(X \mid B_{[1,i]} = b_{[1,i]}) < \mu_i$$

Therefore there is some x_0 s.t.

$$\Pr(X = x_0 \mid B_{[1,i]} = b_{[1,i]}) \geq 2^{-\mu_i}$$

Since $Y(b) > i-1$,

$$H_\infty(X \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \mu_{i-1}$$

Therefore:

$$\begin{aligned}
2^{-\mu_{i-1}} &\geq \Pr(X = x_0 \mid B_{[1,i-1]} = b_{[1,i-1]}) \\
&\geq \Pr(X = x_0 \mid B_i = b_i \textit{ and } B_{[1,i-1]} = b_{[1,i-1]}) \cdot \Pr(B_i = b_i \mid B_{[1,i-1]} = b_{[1,i-1]}) \\
&\geq 2^{-\mu_i} \cdot \Pr(B_i = b_i \mid B_{[1,i-1]} = b_{[1,i-1]})
\end{aligned}$$

Therefore for any b with the prefix $b_{[1,i-1]}$ and $Y(b) = i$,

$$\Pr(B_i = b_i \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq 2^{\mu_i - \mu_{i-1}} = 2^{-\frac{\mu_0}{2t}}$$

Also, by Claim 5.4, $\Pr(Y = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon$.

Therefore,

$\Pr(B_i = b_i \mid B_{[1,i-1]} = b_{[1,i-1]} \textit{ and } Y = i) \leq \frac{\Pr(B_i = b_i \mid B_{[1,i-1]} = b_{[1,i-1]})}{\Pr(Y=i \mid B_{[1,i-1]} = b_{[1,i-1]})} \leq \frac{1}{\epsilon} \cdot 2^{-\frac{\mu_0}{2t}}$.

Since this holds for any prefix $b_{[1,i-1]}$, $H_\infty(B_i \mid Y = i) \geq \frac{\mu_0}{2t} - O(log(n)) \geq n^{\frac{1}{d+1} + \frac{\gamma}{2}}$ as required. □

Finally, let us prove Claim 5.3:

Proof: [of Claim 5.3]

Fix an $i \in [1..t]$. We need to show that for any prefix $b_{[1,i-1]}$, $(B_i \mid Y = t \textit{ and } B_{[1,i-1]} = b_{[1,i-1]})$ is $n^{-\Omega(1)}$–close to a distribution $\overline{W}$ with $H_\infty(\overline{W}) \geq n^{\gamma/2}$.

Fix any $b_{[1,i-1]}$ that can be extended to some b_0 with $Y(b_0) = t$. Since $Y(b_0) = t$, no block so far "stole" too much entropy, i.e., if we denote $\overline{Z} = (X \mid B_{[1,i-1]} = b_{[1,i-1]} \textit{ and } Y = t)$, then:

$$H_\infty(X \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \mu_{i-1}$$

i.e.

$$\text{for any } x, \quad \Pr(X = x \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq 2^{-\mu_{i-1}}$$

Also, by Claim 5.4, $\Pr(Y = d \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon$.

Therefore,

$$\Pr(X = x \mid B_{[1,i-1]} = b_{[1,i-1]} \; and \; Y = t) \leq \frac{\Pr(X = x \mid B_{[1,i-1]} = b_{[1,i-1]})}{\Pr(Y = t \mid B_{[1,i-1]} = b_{[1,i-1]})} \leq \frac{1}{\epsilon} \cdot 2^{-\mu_{i-1}}$$

Hence, $H_\infty(\overline{Z}) \geq \mu_{i-1} - O(log(n)) = \Omega(\mu_{i-1})$, which formally states that after the first $i-1$ blocks of b there is still a lot of min-entropy in X.

By Lemma 3.9, $B(Z, r_i) = (B_i \mid B_{[1,i-1]} = b_{[1,i-1]} \; and \; Y = t)$ is $O(l^{-\Omega(1)}) = n^{-\Omega(1)}$ close to a distribution $\overline{W}$, with $H_\infty(\overline{W}) = \frac{l}{n} \cdot \Omega(\frac{\mu_{i-1}}{log(n)}) \geq \Omega(\frac{n^{1-\frac{1}{d+1}} \cdot n^{\frac{1}{d+1}+\gamma}}{n \; polylog(n)}) \geq \Omega(n^{\gamma/2})$.

□

6 A Survey of The Applications

In this section we survey the main applications of extractors and dispersers. For each application we mention the best result achieved, and whether our new constructions improve the previous best result.

We will mostly use the graph view of extractors and dispersers, and will continue working with the parameters used so far, $N = 2^n$, $M = 2^m$, $K = 2^k$, $D = 2^d$, and ϵ. In many cases the parameters of the extractor or disperser used will be derived directly from the application in mind. In these cases we will "cleverly" define the application already with the appropriate names for parameters. All theorems thus implicitly start with "let G be any extractor with parameters N, M, K, D, ϵ, then...".

6.1 Simulating BPP Using Defective Random Sources

While randomized algorithms seem useful, the following question must be addressed before they can actually be used: How do computers get random bits? Currently some pseudo-random generators are used, usually without any proof that it "works". In many cases, these pseudo-random bits are not good enough as a replacement for the truly random bits needed by the algorithm.

An alternative solution is to rely on some physical source which produces some "real" randomness. A popular example of such sources are Zener Diodes, which (can be made to) produce quantum mechanical noise – thus are random since their output should be impossible to predict physically.

The problem is that although these sources contain a substantial amount of randomness, they do not output truly unbiased and independent random bits – thus they cannot be used directly in randomized algorithms.

A natural idea is to, deterministically, convert this source into truly random bits. For certain types of sources this can indeed be done. E.g. [Blu86] shows how it can be done if the source is a (known) Markov chain. For more general sources it can be shown that this cannot be done [SV86]. Instead, we may use the somewhat random source to indirectly simulate a given randomized algorithm.

The kind of simulation we have in mind is a black-box simulation: one that does not require us to understand the randomized algorithm in question. let us define this "universal" type of simulation explicitly for randomized algorithms which have one-sided error (RP-type algorithms.)

A black box simulation of RP

We want to run a randomized algorithm which has one-sided error and requires m random bits; we are given a random source which has distribution X on $[N]$, from which we can get a single element $x \in [N]$. From x we generate, deterministically, D different strings $z_1...z_D \in [M]$, $(M = 2^m,)$ and then run the original algorithm on each z_i. We accept if the original algorithm accepted for at least one of the z_i's. The simulation is polynomial if D is polynomial (in n.)

DEFINITION 6.1 *The above procedure is a black box simulation of RP using source X with error α if for every set $W \subseteq [M]$, $|W| \geq M/2$,* $\Pr_x[\exists\, i: z_i \in W] \geq 1 - \alpha$.

If the original RP algorithm accepted then for at least half the possible $z \in [M]$ it accepts – call this set W. In this case the simulation, using X, accepts with probability at least $1 - \alpha$. If, on the other hand, the original algorithm rejected, then for no $z \in [M]$ does it accept (since the error is one-sided), and thus the simulation will reject. The analogous procedure for simulating BPP, does not accept whenever one of the z_i's causes the original algorithm to accept but rather accepts according to some other condition which depends on the answers obtained for all z_i's – usually their majority vote.

Our usual goal would be to obtain this simulation without knowing the exact distribution of the somewhat random source X. Our simulation should only rely on the fact that X has some given property, which is the model of "somewhat randomness". Clearly, physicists should try to produce physical source where this property is as strong as possible, while computer scientists should aim to rely on a property which is as week as possible.

Several models of "somewhat random" sources were considered in the literature. Santha and Vazirani [SV86], Vazirani [Vaz87a, Vaz86, Vaz87b] and Vazirani and Vazirani [VV85] studied a class of sources, called "slightly random sources", and showed that BPP can be simulated given such a source. Chor and Goldreich [CG88] generalized this model, and showed that BPP can be simulated even using the more general source. Many authors studied other restricted classes of random sources (e.g. [CW89, CGH+85, LLS89]). Finally, Zuckerman [Zuc90] suggested a general model generalizing all the previous models. His model was simply all random sources having high min-entropy. Later [Zuc91] he showed that BPP can be simulated given such a source (with enough min-entropy).

In a very basic sense, Zuckerman's model is the most general source we can think off.

Lemma 6.1 *If a black box RP simulation requiring m bits can be obtained, with error α, using source X, then X is $O(\alpha)$-close to some distribution X' with $H_\infty(X') \geq m - \log D - 1$.*

Proof: (sketch) Consider the set S of all the elements x which get a probability higher than $2^m/(2D)$. This total probability cannot be greater than α since otherwise we can have W be the set of all elements z which are never produced by an element of S, a set which has size $|W| \geq M - |S| \cdot D \geq M/2$. The distribution X' is obtained by zeroing the probability of elements in S and correcting the total probability to 1. $\square$

Using extractors a simulation good for all sources with enough min-entropy is easily obtained.

Theorem: *RP (and BPP) with m bits can be simulated in time D using any random source X with $H_\infty(X) \geq k$. (The error is $exp(k - H_\infty(X))$.)*

Proof: (for RP) Get x from the distribution, and set $z_i = G(x, i)$ for $i = 1...D$, where G is a disperser with $\epsilon < 1/2$. Let us calculate the probability that this simulation fails, i.e. that all $z_i \notin W$. Let us look at the set S of these "bad" x's – it cannot be larger than K, since the neighbor set of any set of size K is larger than $M/2$ and thus intersects W. The total probability assigned by X to this set of "bad" x's can be at most $|S| \cdot 2^{-H_\infty(X)}$. $\square$

A similar argument for BPP uses an extractor.

With current constructions, we get a polynomial time (in n) simulation (i.e. $D = poly(n)$) for RP as long as $H_\infty(X) \geq n^\delta$ for any $\delta > 0$ [SSZ95]; a polynomial time simulation for BPP as long as $H_\infty(X) = \Omega(n)$ (or even slightly less) [Zuc93]; and using our new results presented in Section 5, a slightly quasi-polynomial ($n^{log^{(c)} n}$ for any constant c) time simulation for BPP for any $H_\infty(X)$:

Corollary 6.2 *For any $\delta > 0$ and constant $k > 0$, BPP can be simulated in time $n^{O(\overbrace{loglog \ldots log}^{k} n)}$ using a weak random source X with minentropy at least n^δ* [6].

6.2 Deterministic Amplification

It turns out that extractors and dispersers can be used to decrease the probability of error of randomized algorithms in several settings. We describe some of these settings in this subsection.

6.2.1 Basic Amplification

Our goal now is to convert a BPP (or RP) algorithm that uses m random bits and has error, say, $1/4$, into one that errs with probability at most 2^{-t}. A trivial solution is to run the original algorithm $O(t)$ times using independent random bits, and take the majority vote – this requires $O(tm)$ random bits. We want to achieve this using as few random bits as possible.

This problem, known as the "deterministic amplification" problem, was extensively studied [CG89, IZ89, CW89]. Using expanders, this can be done using only $n + O(t)$ random bits

[6] Previous result [SZ94] was for $\delta > 1/2$, and required $n^{O(log(n))}$ time.

[AKS87, IZ89, CW89]. Sipser [Sip88] defined dispersers as a tool which implies stronger RP amplification. Using extractors, we obtain BPP amplification.

Theorem: *If L is accepted by a BPP algorithm using m random bits and error $1/4$, then L is also accepted by a BPP algorithm using n random bits and having error K/N.*

Proof: Use an extractor G with $\epsilon < 1/4$.

The Algorithm: Choose randomly $x \in [N]$. For any $z \in \Gamma(x)$ run the machine accepting L with z as the random string, and decide according to the majority of the results.

Correctness: Denote $W \subseteq [M]$ the set of witnesses z leading to the wrong answer by the machine accepting L. Thus $|W| \leq M/4$. Let $B \subseteq [N]$ be the set of "bad" x's – those with most of their neighbors in W. The simulation arrives at the wrong result iff a "bad" x is chosen. Since G is an extractor, $|B| < K$, since otherwise a uniform distribution on B would, on one hand have min-entropy k, but on the other hand, $\Gamma(B)$ is far from uniform since it gives weight of at least $1/2$ to a set W of size at most $1/4$. □

It can easily be seen that if we want to amplify an RP algorithm we can use dispersers instead of extractors.

Using current constructions we can use only $n = (1+\alpha)(m+t)$ bits to get error 2^{-t}, for any fixed $\alpha > 0$ [Zuc93, Zuc]. Using the construction of Section 4, we can use only $m+t$ random bits, but the running time becomes quasi-polynomial.

6.2.2 Oblivious Sampling

The above simulation may be generalized to give what is called an oblivious sampler. These are sampling procedures that give a good estimate for the expected value of a real valued function on some finite domain.

DEFINITION 6.2 *[BR94] An oblivious (δ, ϵ)-sampler is a deterministic function that for each $x \in [N]$ produces a set $\Gamma(x) = \{z_1 ... z_D\} \subseteq [M]$ such that for every function $f : [M] \to [0,1]$, (where $[0,1]$ is the real interval between 0 and 1),*

$$\Pr_x[|\frac{\sum_{i\in[D]} f(z_i)}{D} - \frac{\sum_{z\in[M]} f(z)}{M}| \geq \epsilon] \leq \delta$$

For given ϵ, δ, and m, our wish is to have D be polynomial and n be as small as possible. Oblivious samplers were constructed in [BR94] who used them for interactive proof systems. The best known results to date, which use extractors, appear in [Zuc].

Theorem: *There exist explicitly constructible (δ, ϵ) oblivious samplers where $\delta = 2K/N$.*

Using the best constructions of extractors [Zuc], we get oblivious samplers with $n = (1+\alpha)(m + \log \delta^{-1})$ and $D = poly(m, \epsilon^{-1}, \log \delta^{-1})$ (for any $\alpha > 0$).

Proof: Take an extractor G and use the $\Gamma(x)$ from the extractor. Denote the expected value of f by $e = (\sum_{z\in[M]} f(z))/M$. The proof follows by considering the set $S_<$ of x's such that $(\sum_{z\in\Gamma(x)} f(z))/D < e - \epsilon$, and separately $S_>$, the set of x's such that $(\sum_{z\in\Gamma(x)} f(z))/D > e + \epsilon$. These sets must each be of size less than K, since otherwise the uniform distribution on one of them has high enough min-entropy and thus gives near (to within ϵ) uniform distribution on the z's. This cannot be true since an ϵ-close to uniform distribution on the z's gives an ϵ-close estimate of e, in contradiction to each element in $S_<$ (resp, $S_>$) erring on e by at least ϵ. $\square$

6.2.3 Approximating Clique

Zuckerman [Zuc93] showed how deterministic amplification obtained by extractors implies that

Theorem: *[Zuc93] Approximating $log(Clique(G))$ to within an constant factor is $\tilde{NP}$–hard.*

In fact Zuckerman showed that, for any constant j, approximating the j'th iterated log of clique to within any constant is $\tilde{NP}$–hard. For the precise statement see [Zuc93, SZ94]. Our new construction can derandomize Zuckerman's result.

The proof itself builds on the known hardness results for approximating clique [AS92b, ALM+92, FGL+91], and is best understood when viewing it as deterministic amplification for PCP systems.

6.2.4 Time vs. Space

Sipser [Sip88] defined dispersers in order to obtain the following theorem (which became a theorem only with the recent constructions of [SSZ95]). Again, the dispersers are needed for deterministic amplification. The [SSZ95] dispersers are good enough, and our new constructions do not improve this result.

Theorem: *If $P \neq RP$ then there is some $0 < \alpha$ s.t. for any (nice) function $t = t(n)$, $Time(t) \subseteq ioSpace(t^{1-\alpha})$, where $ioSpace(s)$ is the class of languages solvable by algorithms that for infinitely many inputs use at most space s.*

This is an unexpected connection between the question of the power of randomness and the (seemingly unrelated) question of time vs. space.

6.3 Explicit Graphs With Random Properties

In this section we present some explicit constructions of certain kinds of graphs. In these cases, the "random-like" properties of extractors and dispersers suffice as a replacement for using random graphs, and thus convert a non-constructive proof to a construction.

6.3.1 Super Concentrators

DEFINITION 6.3 *[Pip77] Let $H = (V, E)$ be a directed graph with a specified subset $I \subset V$ of nodes called input nodes, and a disjoint subset, $O \subseteq V$, called output nodes. Assume $|I| = |O| = N$. H is called a super-concentrator if for any sets $W \subseteq I, Z \subseteq O$ of size K each, there are at least K vertex-disjoint paths from W to Z.*

The parameters of interest are, first, the *size* which is the number of edges in H, and second, the *depth* which is the length of the longest directed path in it.

Gabber and Galil [GG81] constructed, using expanders, the first explicit linear size super concentrators. The graph they construct has $O(log(N))$ depth. For depth 2, non-constructive proofs show that super concentrators of depth 2 and size $O(N \cdot log^2(N))$ exists [Pip82]. Meshulam [Mes84] showed an explicit depth 2, super concentrator of size $O(N^{1+1/2})$. [WZ93] give a construction based on extractors.

Theorem: *(following [WZ93]) There is an explicit super concentrator of depth 2 and size $N \cdot O(\sum_{k=1}^{n} D_k 2^k / M_k)$, where D_k, M_k are the parameters for dispersers with $K = 2^k$.*

The work done in Section 4 imply depth 2 super concentrators of size $N \cdot 2^{polyloglog(N)}$.

Proof:

Meshulam [Mes84] showed that H is a super concentrator of depth 2, iff for any $1 \leq K' \leq N$ and any two sets $W \subseteq I, Z \subseteq O$ of size K' each, there are at least K' common neighbors. We will build a depth 2 graph with this property.

Build the graph as follows:

- I and O each have $N = 2^n$ vertices.
- The other vertices are partitioned into disjoint sets C_k for $1 \leq k \leq n$. $|C_k| = 4K$.
- For each k, we partition C_k into $4K/M_k$ equal-sized sets each of size M_k and put a disperser between I and each set, as well as between O and each set. (We take a disperser with $\epsilon = 1/4$.)

Now take any two sets $W \subseteq I$ and $Z \subseteq O$ of size K' each. Let $K = 2^k$ be the largest power of 2 which is less or equal to K'. Consider only edges connecting W and Z to C_k. In each one of the $4K/M_k$ different subsets of C_k, we have that $\Gamma(W), \Gamma(Z) \geq 3M_k/4$ (since we put a disperser there). It follows that in each of these subsets, $|\Gamma(W) \cap \Gamma(Z)| \geq M_k/2$. Taking all $4K/M_k$ subsets together, the intersection size is at least $2K \geq K'$. $\square$

Using the extractor of Section 4:

Corollary 6.3 *For every N there is an efficiently constructible depth 2 superconcentrator over N vertices with size $O(N \cdot 2^{polyloglog(N)})$.*

Wigderson and Zuckerman showed that such a construction would also imply an explicit linear size super concentrator of small depth. With the results of Section 4, of depth $polyloglog(N)$.

Corollary 6.4 *For any N there is an explicitly constructible superconcentrator over N vertices, with linear size and .*

6.3.2 Highly Expanding Graphs

We now consider expanders with very strong expansion properties.

DEFINITION 6.4 *A graph H on N vertices is called a K-expander if any two sets W, Z of vertices, $|W| = |Z| = K$, have a common neighbor.*

Since any two sets of size K have a common neighbor, a set of size K must have at least $N - K$ neighbors and therefore the degree of the graph is at least $(N - K)/K$. We would like to explicitly build K-expanding graphs with degree as close as possible to N/K. The eigenvalue methods for constructing expanders give such expanders for $K \geq \sqrt{N}$, but do not give anything for smaller values of K. [WZ93] show how dispersers can be used to construct expanders with small values of K.

Theorem: *(following [WZ93]) There are constructible K-expanding graph with N vertices, and maximum degree $O(ND^2/M)$.*

Using the extractors of Section 4 we get maximum degree of $\frac{N}{K} \cdot exp(polyloglogN)$ for any value of K. Let us remark that the construction takes time polynomial in N. It does not allow computing whether an edge exists between two given vertices in time polynomial in n.

Proof: We use a disperser G with $\epsilon = 1/5$.

Let $B \subseteq [M]$ be the set of z's of degree greater than $2DN/M$. Notice that $|B| \leq M/2$ since the total number of edges in the disperser is DN. Now we connect a vertex $x_1 \in [N]$ to a vertex $x_2 \in [N]$ if they share a neighbor not in B, i.e. if there exist $y_1, y_2 \in [D]$ such that $G(x_1, y_1) = G(x_2, y_2) \notin B$.

Now for every set $W \subseteq [N], |W| \geq K$ we have, in the disperser, $\Gamma(W) \geq 4M/5$, and thus $|\Gamma(W) \cap \Gamma(Z)| \geq 3M/5 \geq |B|$. Thus, in the expander we have just built, W and Z will share a neighbor.

The number of edges is bounded from above $D \cdot 2DN/M$.

$\square$

Using the extractor of Section 4:

Corollary 6.5 *For any N and $1 \leq a \leq N$ there is an explicitly constructible a–expanding graph with N vertices, and maximum degree $O(\frac{N}{a} 2^{polyloglog(N)})$.*

Pippenger [Pip87b] showed that good explicit highly expanding graphs, yield good algorithms for "sorting in rounds" and for "selecting in rounds".

6.4 Pseudo-random Generators

Extractors can also be used to construct pseudo-random generators which fool certain classes of algorithms. The generator can use any good extractor for high min-entropies, and our new constructions do not improve its operation. The following theorem of [NZ93] improves on previous results of [AKS87].

Theorem: *[NZ93] There exists a pseudo-random generator which converts $O(S)$ truly random bits into poly(S) bits which look random to any algorithm which runs in space S. The generator runs in $O(S)$ space and poly(S) time.*

Proof: (sketch) We will build a generator that, for any t, converts $n + td$ bits into tm bits which look random (to within ϵ) to space $S = n - k - O(\log \epsilon^{-1})$ algorithms. Current extractors G with $k, m = \Omega(n)$ imply, for all $t \geq S$, conversion of $O(t)$ bits to $\Omega(t \cdot S/\log S)$ bits. To get any *poly*(S) factor gain in the number of bits simply iterate (i.e. compose the pseudo-random generators).

The Pseudo-random Generator

Input: x, $y_1 ... y_t$.

Output: $G(x, y_1), ..., G(x, y_t)$.

for each $i = 0...t$, consider the probability distribution A_i on the internal state of the algorithm after im random bits $z_1...z_i$ have been read by the algorithm versus the distribution A_i' in the case that pseudo-random bits $G(x, y_1)...G(x, y_i)$ have been read. The proof that this generator indeed looks random to space S algorithms proceeds by induction on i of the following claim:

Claim: The distribution A_i is close to the distribution A_i'.

Proof: Fix a typical internal state a of the algorithm. We will show that, conditioned on a being the state reached after $(i-1)m$ bits were read, A_i and A_i' are close. The claim will follow by averaging over all a's and noticing that the probabilities of getting to a, in A_{i-1} vs. A_{i-1}', are close by the induction hypothesis.

Let us see what is the conditional distribution on X conditioned upon a being the state reached after $(i-1)m$ bits were read. The space bound S on the algorithm implies that there are at most 2^S different possible values of a, thus we would expect that the probability of getting to a is about 2^{-S} (otherwise a can be ignored). Looking at the conditional probabilities we have that $H_\infty(X|a) \geq n - S \geq k$. Therefore, the distribution of $G(x, y_i)$ is nearly uniform under this conditioning, since G is an extractor. The distributions A_i and A_i', under this conditioning, are completely determined by z and $G(x, y_i)$, resp., and thus are close to each other.

$\square$

Acknowledgments We would like to thank David Zuckerman and Avi Wigderson for many helpful discussions. We thank Oded Goldreich for many helpful comments.

References

[ACRT97] A. E. Andreev, A. E.F. Clementi, J. D.P. Rolim, and L. Trevisan. Weak random sources, hitting sets, and bpp simulations. Technical report, Electronic Colloquium on Computational Complexity, 1997.

[AGHP92] Alon, Goldreich, Hastad, and Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3, 1992.

[AKS87] Ajtai, Komlos, and Szemeredi. Deterministic simulation in LOGSPACE. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[AKSS89] M. Ajtai, J. Komlos, W. Steiger, and E. Szemeredi. Almost sorting in one round. In *Advances in Computer Research*, volume 5, pages 117–125, 1989.

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 14–23, 1992.

[AS92a] N. Alon and J. H. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992.

[AS92b] S. Arora and S. Safra. Probabilistic checking of proofs; a new characterization of NP. In *Proceedings of the 33rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 2–13, 1992.

[Blu86] M. Blum. Independent unbiased coin flips from a correlated biased source: a finite markov chain. *Combinatorica*, 6(2):97–108, 1986.

[BR94] Bellare and Rompel. Randomness-efficient oblivious sampling. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

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

[CG89] Chor and Goldreich. On the power of two-point based sampling. *Journal of Complexity*, 5, 1989.

[CGH+85] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky. The bit extraction problem and t-resilient functions. In *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, pages 396–407, 1985.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification, and weak random sources. In *Proceedings of the 30th Annual IEEE Symposium on the Foundations of Computer Science*, pages 14–19, 1989.

[FGL+91] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 2–12, 1991.

[GG81] Gabber and Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22, 1981.

[GW94] O. Goldreich and A. Wigderson. Tiny families of functions with random properties: A quality-size trade-off for hashing. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM, pages 574–583, 1994.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, ACM, pages 12–24, 1989.

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

[LLS89] Lichtenstein, Linial, and Saks. Some extremal problems arising from discrete control processes. *Combinatorica*, 9, 1989.

[Mes84] R. Meshulam. A geometric construction of a superconcentrator of depth 2. *Theoretical Computer Science*, 32:215–219, 1984.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Nis96] N. Nisan. Refining randomness: Why and how. In *Annual Conference on Structure in Complexity Theory*, 1996.

[NN93] Naor and Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22, 1993.

[NZ93] N. Nisan and D. Zuckerman. More deterministic simulation in logspace. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM, pages 235–244, 1993.

[Pip77] Pippenger. Superconcentrators. *SICOMP: SIAM Journal on Computing*, 1977.

[Pip82] Pippenger. Superconcentrators of depth 2. *JCSS: Journal of Computer and System Sciences*, 24, 1982.

[Pip87a] N. Pippenger. Sorting and selecting in rounds. *SIAM Journal on Computing*, 16:1032–1038, 1987.

[Pip87b] N. Pippenger. Sorting and selecting in rounds. *SIAM Journal on Computing*, 16:1032–1038, 1987.

[RTS] J. Radhakrishnan and A. Ta-Shma. Tight bounds for depth-two superconcentrators. To appear in FOCS 1997.

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

[SSZ95] M. Saks, A. Srinivasan, and S. Zhou. Explicit dispersers with polylog degree. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM, 1995.

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

[SZ94] A. Srinivasan and D. Zuckerman. Computing with very weak random sources. In *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, 1994.

[Ta-96] Ta-Shma. On extracting randomness from weak random sources. In *ACM Symposium on Theory of Computing (STOC)*, 1996.

[Vaz86] U. Vazirani. *Randomness, Adversaries and Computation.* PhD thesis, University of California, Berkeley, 1986.

[Vaz87a] U. Vazirani. Efficiency considerations in using semi-random sources. In *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, ACM, pages 160–168, 1987.

[Vaz87b] U. Vazirani. Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources. *Combinatorica*, 7(4):375–392, 1987.

[VV85] U. Vazirani and V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science*, IEEE, pages 417–428, 1985.

[WZ93] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM, pages 245–251, 1993.

[Zuc] D. Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. Private Communication.

[Zuc90] D. Zuckerman. General weak random sources. In *Proceedings of the 31st Annual IEEE Symposium on the Foundations of Computer Science*, pages 534–543, 1990.

[Zuc91] D. Zuckerman. Simulating BPP using a general weak random source. In *Proceedings of the 32nd Annual IEEE Symposium on the Foundations of Computer Science*, pages 79–89, 1991.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the 8th Structures in Complexity Theory*, IEEE, pages 305–312, 1993.

A A Somewhere Random Source Has Large Min-entropy

Lemma A.1 *If $X = X_1 \circ \ldots \circ X_b$ is a (k, ϵ, η) somewhere random source, then X is η–close to a $(k, \epsilon, 0)$ somewhere random source X'.*

Proof: [of Lemma A.1]

Let Y be a (k, ϵ, η) selector for X. Denote $p = \Pr(Y = 0) \le \eta$. Define the distribution $\overline{D}$ by:

$$\overline{D}(i, x) \;=\; \begin{cases} 0 & \text{If } i = 0 \\ \frac{\Pr(\ (Y,X)=(i,x)\ )}{1-p} & \text{otherwise} \end{cases}$$

It is easy to see that $\overline{D}$ is a distribution. Define the random variable $Y' \circ X'$ as the result of choosing (i, x) uniformly from $\overline{D}$, i.e. $\overline{Y' \circ X'} = \overline{D}$. It is clear that $d(\overline{X}, \overline{X'}) \le d(\overline{Y \circ X}, \overline{Y' \circ X'}) = p \le \eta$.

Now we want to show that Y' is a $(k, \epsilon, 0)$ selector for X'. It is clear that $\Pr(Y' = 0) = 0$. It is not hard to see that for any $i > 0$ we have: $\Pr(X' = x \mid Y' = i) \;=\; \Pr(X = x \mid Y = i)$. Therefore, since we know that $(X_i | Y = i)$ is ϵ–close to U_k, we also know that $(X_i' | Y' = i)$ is ϵ close to U_k, thus completing the proof.

□

Lemma A.2 *Let $X = X_1 \circ \ldots \circ X_b$ be a $(k, \epsilon, 0)$–somewhere random source, then X is ϵ close to a $(k, 0, 0)$–somewhere random source Z.*

Proof: [of Lemma A.2]

Let Y be a $(k, \epsilon, 0)$ selector for X. Fix some $i \in [1..b]$. We know that $d((X_i \mid Y = i), U_k) \le \epsilon$. Define a distribution $Z^{(i)}$ by:

$$Z^{(i)}(x) = \begin{cases} \frac{1}{2^k} \cdot \Pr(X = x \mid X_i = x_i \textit{ and } Y = i) & \text{if } \Pr(X_i = x_i \textit{ and } Y = i) > 0 \\ \frac{1}{2^k} \cdot 1 & \text{if } \Pr(X_i = x_i \textit{ and } Y = i) = 0 \\ & \text{and for every } j \neq i:\ x_j = 0^k \\ 0 & \text{otherwise} \end{cases}$$

It is easy to check that $Z^{(i)}$ is indeed a distribution, and that $Z_i^{(i)} = U_k$. Define $Y \circ Z$ to be the random variable obtained by choosing i according to Y, then choosing z according to $Z^{(i)}$, i.e., for all $i > 0$, $(Z \mid Y = i) = Z^{(i)}$. Also, denote $X^{(i)} = (X \mid Y = i)$. Then:

$$\Pr(Z_i = z_i \mid Y = i) \;=\; Z_i^{(i)}(z_i) \;=\; 2^{-k}$$

We will soon prove that:

Claim A.1 $d(X^{(i)}, Z^{(i)}) \le \epsilon$.

Thus:

$$d(\overline{X}, \overline{Z}) \leq d(\overline{Y \circ X}, \overline{Y \circ Z}) \ =$$

$$\Sigma_{i>0} \ \Pr(Y = i) \ \cdot \ d((X \mid Y = i), (Z \mid Y = i)) \ =$$

$$\Sigma_{i>0} \ \Pr(Y = i) \ \cdot \ d(X^{(i)}, Z^{(i)}) \ \leq \epsilon$$

Hence Z satisfies the requirements of the lemma.

□

Proof: [of Claim A.1]

We need to show that for any $A \subseteq \Lambda_X$, $|X^{(i)}(A) - Z^{(i)}(A)| \leq \epsilon$. It is sufficient to show this for the set A containing all $x \in \Lambda_X$ s.t. $X^{(i)}(x) > Z^{(i)}(x)$. This can be easily seen, using the fact that for any $x \in A$: $\Pr(Z = x \mid Z_i = a_i \ and \ Y = i) = \frac{\Pr(Z=x \mid Y=i)}{\Pr(Z_i=a_i \mid Y=i)} = \Pr(X = x \mid X_i = a_i \ and \ Y = i)$.

□

Lemma A.3 *Let $X = X_1 \circ \ldots \circ X_b$ be a $(k, 0, 0)$ somewhere random source, then $H_\infty(\overline{X}) \geq k$.*

Proof: Suppose Y is a $(k, 0, 0)$ selector for X.

$$\Pr(X = x) \ =$$

$$\Sigma_{i \in [1..b]} \ \Pr(Y = i) \ \cdot \ \Pr(X_i = x_i \mid Y = i) \leq$$

$$\Sigma_{i \in [1..b]} \ \Pr(Y = i) \ \cdot \ 2^{-k} = 2^{-k}$$

□

Combining Lemmas A.1, A.2 and Lemma A.3 we get Lemma 4.2.

B A Lemma For B–Block Mergers

We prove Lemma 4.5:

Lemma B.1 (Lemma 4.5) *:*

Let A, B and Y be any random variables. Suppose that $d((A \mid Y \in S_1), U_k) \leq \epsilon$ and $d((B \mid Y \in S_2), U_k) \leq \epsilon$ for some disjoint sets S_1 and S_2. Then $(A \circ B \mid Y \in S_1 \cup S_2)$ is ϵ–close to some $\overline{X}$ with $H_\infty(\overline{X}) \geq k$.

Proof: We define random variables $Y', A' \circ B'$ as follows:

- Choose $Y' = i \in S_1 \cup S_2$ with: $\Pr(Y' = i) = \Pr(Y = i \mid Y \in S_1 \cup S_2)$.
- Choose $a' \circ b' \in (A \circ B \mid Y = i)$.

It is easy to prove that:

Claim: $\Pr(A' = a' \mid Y' = i) = \Pr(A = a' \mid Y = i)$ and $\Pr(B' = b' \mid Y' = i) = \Pr(B = b' \mid Y = i)$.

Define

$$Z' = \begin{cases} 1 & \text{If } Y' \in S_1 \\ 2 & \text{Otherwise, i.e. } Y' \in S_2 \end{cases}$$

It is not hard to see that:

Claim B.1 *$(A' \mid Z' = 1) = (A \mid Y \in S_1)$ and $(B' \mid Z' = 2) = (B \mid Y \in S_2)$.*

Hence, Z' is a $(k, \epsilon, 0)$ selector for $A' \circ B'$.

Therefore by Lemma 4.2, $\overline{A' \circ B'}$ is ϵ–close to some $\overline{X}$ with $H_\infty(\overline{X}) \geq k$. However, it is not hard to see that:

Claim: $\overline{A' \circ B'} = (A \circ B \mid Y \in S_1 \cup S_2)$.

Thus, $(A \circ B \mid Y \in S_1 \cup S_2) = \overline{A' \circ B'}$ is ϵ–close to some $\overline{X}$ with $H_\infty(\overline{X}) \geq k$, thus completing the proof. $\square$

C Lemmas For Composing Two Extractors

In this section we prove some easy technical lemmas used in Section 4.2.

Claim C.1 *For any i and any $w_{[1,i-1]}$, if $\Pr_{x\in X}(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) > 0$, then $\Pr_{x\in X}(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) \geq \epsilon_2 - \epsilon_3$.*

Proof:

Since $w_{[1,i-1]}$ can be extended to some w with $Y(w) = i \neq 0$, by Definition 4.5:

$$\Pr(f(x) = i) \geq \epsilon_1 \text{ , } and$$
$$\Pr(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) \geq \epsilon_2$$

However, this implies that for *any* extension w' of $w_{[1,i-1]}$ with $f(w') = i$, it holds that $w' \notin B_1 \cup B_2$. Hence,

$$\Pr(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) =$$
$$\Pr(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) - \Pr(f(x) = i \text{ } and \text{ } x \in B \mid x_{[1,i-1]} = w_{[1,i-1]}) =$$
$$\Pr(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) - \Pr(f(x) = i \text{ } and \text{ } x \in B_3 \mid x_{[1,i-1]} = w_{[1,i-1]}) \geq$$
$$\epsilon_2 - \epsilon_3$$

The last inequality uses Claim C.3.

□

Claim C.2 *For any i, if $\Pr_{x\in X}(Y(x)=i)>0$, then $\Pr_{x\in X}(Y(x)=i)\geq \epsilon_1-\epsilon_2-\epsilon_3$.*

Proof:

Since there is some w' s.t. $Y(w')=i\neq 0$, by Definition 4.5:

$$\Pr(f(x)=i)\geq \epsilon_1$$

This implies that for *any* w' with $f(w')=i$, we know that $w'\notin B_1$. Hence,

$$\begin{aligned}
&\Pr(Y(x)=i) \;=\\
&\Pr(f(x)=i) \;-\Pr(f(x)=i \textit{ and } x\in B)\;\geq\\
&\Pr(f(x)=i) \;-\Pr(f(x)=i \textit{ and } x\in B_2)-\Pr(f(x)=i \textit{ and } x\in B_3)\;\geq\\
&\epsilon_1-\epsilon_2-\epsilon_3
\end{aligned}$$

The last inequality uses Claim C.3.

□

Claim C.3

1. *For any i:* $\Pr(f(x)=i \textit{ and } x\in B_2)\leq\epsilon_2$
2. *For any i and $w_{[1,i-1]}$:* $\Pr(f(x)=i \textit{ and } x\in B_3 \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_3$
3. *For any i:* $\Pr(f(x)=i \textit{ and } x\in B_3)\leq\epsilon_3$
4. $\Pr(x\in B_i)\leq n\epsilon_i$, *for* $i=1,2,3$.

Proof:

1) If for some $w_{[1,i-1]}$ $\Pr(f(x)=i \textit{ and } x\in B_2 \mid x_{[1,i-1]}=w_{[1,i-1]})>0$ then there is an extension w of $w_{[1,i-1]}$ s.t.: $f(w)=i \textit{ and } w\in B_2$, and therefore, $\Pr(f(x)=i \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_2$. Thus, for all $w_{[1,i-1]}$, $\Pr(f(x)=i \textit{ and } x\in B_2 \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_2$. Therefore, $\Pr(f(x)=i \textit{ and } x\in B_2) = \Sigma_{w_{[1,i-1]}} \Pr(x_{[1,i-1]}=w_{[1,i-1]})\cdot\Pr(f(x)=i \textit{ and } x\in B_2 \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\Sigma_{w_{[1,i-1]}} \Pr(x_{[1,i-1]}=w_{[1,i-1]})\cdot\epsilon_2\leq\epsilon_2$.

2) If for some $w_{[1,i-1]}$ $\Pr(f(x)=i \textit{ and } x\in B_3 \mid x_{[1,i-1]}=w_{[1,i-1]})>0$ then there is an extension w of $w_{[1,i-1]}$ s.t.: $f(w)=i \textit{ and } w\in B_3$, and therefore, $\Pr(x_i=w_i \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_3$. In particular, $\Pr(x\in B_3 \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\Pr(x_i=w_i \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_3$. Thus, for all $w_{[1,i-1]}$, $\Pr(f(x)=i \textit{ and } x\in B_3 \mid x_{[1,i-1]}=w_{[1,i-1]})\leq\epsilon_3$.

3) $\Pr(f(x) = i \textit{ and } x \in B_3) \leq \Sigma_{w_{[1,i-1]}} \Pr(x_{[1,i-1]} = w_{[1,i-1]}) \cdot \Pr(f(x) = i \textit{ and } x \in B_3 \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \Sigma_{w_{[1,i-1]}} \Pr(x_{[1,i-1]} = w_{[1,i-1]}) \cdot \epsilon_3 \leq \epsilon_3$.

4) The case $i = 2$ follows (1) since, $\Pr(x \in B_2) \leq \Sigma_{i=1}^{n} \Pr(x \in B_2 \textit{ and } f(x) = i) \leq n\epsilon_2$. Similarly for $i = 3$. As for $i = 1$: if there is an x with $f(x) = i \textit{ and } x \in B_1$, then $\Pr(f(x) = i) \leq \epsilon_1$. Thus, $\Pr(x \in B_1 \textit{ and } f(x) = i) \leq \epsilon_1$, and $\Pr(x \in B_1) \leq \Sigma_{i=1}^{n} \Pr(x \in B_1 \textit{ and } f(x) = i) \leq n\epsilon_1$.

□

D More Bits Using The Same Extractor

In this section we prove Lemmas 4.6 and 4.7.

Lemma D.1 (Lemma 4.6) *:*

Suppose that for some k there is an explicit (k, ϵ) extractor $E_k : (n) \times (d) \mapsto (m)$. Then, for any $K \geq k$, and any safety parameter $s > 0$, there is an explicit $(K, t(\epsilon + 2^{-s}))$ extractor $E : (n) \times (td) \mapsto min\{tm, K - k - s\}$

Proof:

Denote by A_i the random variable with value $E(X, R_i)$. Denote by $A_{[1,i]} = A_1 \circ \ldots \circ A_i$ the random variable whose value is $E(X, R_1) \circ \ldots \circ E(X, R_i)$, and let $l_i = |A_{[1,i]}|$.

Definition D.1 *We say that $a_{[1,i]}$ is "s-tiny" if* $\Pr(A_{[1,i]} = a_{[1,i]}) \leq 2^{-l_i - s}$

Claim: For any $1 \leq i \leq t$, $\Pr(a_{[1,i]} \textit{ is } \mathrm{s}{-}tiny) \leq 2^{-s}$.

Proof: $A_{[1,i]}$ can have at most 2^{l_i} possible values, and each tiny value has probability at most $2^{-l_i - s}$. □

Claim: For any prefix $a_{[1,i]}$ that is not s–tiny, $H_\infty(X \mid A_{[1,i]} = a_{[1,i]}) \geq K - l_i - s$

Proof: For any x,

$$\Pr(X = x \mid A_{[1,i]} = a_{[1,i]}) \leq \frac{\Pr(X = x)}{\Pr(A_{[1,i]} = a_{[1,i]})} \leq \frac{2^{-K}}{2^{-l_i - s}} = 2^{-K + l_i + s}$$

□

Claim: If $l_{i-1} \leq K - k - s$, then $\overline{A_{[1,i]}}$ is $i(2^{-s} + \epsilon)$ quasi-random.

Proof: By induction on i. For $i = 1$ this follows from the properties of E. Assume for i, and let us prove for $i + 1$.

Since $l_i \leq K - k - s$, then for any prefix $a_{[1,i]}$ that is not s–tiny, $H_\infty(X \mid A_{[1,i]} = a_{[1,i]}) \geq K - l_i - s \geq k$. Therefore, for any non-tiny prefix $a_{[1,i]}$, $\overline{(A_{i+1} \mid A_{[1,i]} = a_{[1,i]})}$ is ϵ quasi-random. Therefore by Lemma 3.6, $\overline{A_{[1,i+1]}}$ is $2^{-s} + \epsilon$ close to the distribution $\overline{A_{[1,i]}} \times U$, and by induction $\overline{A_{[1,i+1]}}$ is $(i + 1)(2^{-s} + \epsilon)$ quasi-random. □

Therefore, if we take t s.t. $l_t \leq K - k - s$, we invest td random bits, and we get tm bits that are $t(2^{-s} + \epsilon)$ quasi-random, as required.

$\square$

Lemma D.2 (Lemma 4.7) *:*

Suppose that for any $k \geq \bar{k}$ there is an explicit $(k, \epsilon(n))$ extractor $E_k : (n) \times (d(n)) \times (\frac{k}{f(n)})$. Then, for any k, there is an explicit $(k, f(n)log(n)(\epsilon + 2^{-d(n)}))$ extractor $E : (n) \times (O(f(n)log(n)d(n))) \mapsto (k - \bar{k})$.

Proof:

Define $E(x, r_1 \circ \ldots \circ r_t) = E_{k_1}(x, r_1) \circ \ldots \circ E_{k_t}(x, r_t)$, where $s = d(n)$, $l_0 = 0$, $k_i = k - l_{i-1} - s$, and $l_i = l_{i-1} + \frac{k_i}{f(n)}$. Denote by A_i the random variable $E_{k_i}(X, R_i)$, and let $A_{[1,i]} = A_1 \circ \ldots A_i$. Intuitively, $l_i = |A_{[1,i]}|$, and k_i is the amount of min-entropy left in $(X \mid A_{[1,i]} = a_{[1,i]})$ with the safety parameter $s = d(n)$.

Claim: If $k_i \geq \bar{k}$ then $\overline{A_{[1,i]}}$ is $i(2^{-s} + \epsilon)$ quasi-random.

Proof: By induction on i. For $i = 1$ this follows from the properties of E. Assume for i, and let us prove for $i + 1$.

For any prefix $a_{[1,i]}$ that is not s-tiny, $H_\infty(X \mid A_{[1,i]} = a_{[1,i]}) \geq k - l_i - s = k_{i+1} \geq \bar{k}$. Therefore, for any non-tiny prefix $a_{[1,i]}$, $(A_{i+1} \mid A_{[1,i]} = a_{[1,i]})$ is ϵ quasi-random. Therefore by Lemma 3.6, $\overline{A_{[1,i+1]}}$ is $2^{-s} + \epsilon$ close to the distribution $\overline{A_{[1,i]}} \times U$, and by induction $\overline{A_{[1,i+1]}}$ is $(i + 1)(2^{-s} + \epsilon)$ quasi-random. $\square$

How big do we need t to be? Let us denote $q_i = k - l_i$, i.e., q_i is the number of bits still missing. Notice that $q_i = k - l_i = k - (l_{i-1} + \frac{k_i}{f(n)}) = q_{i-1} - \frac{k_i}{f(n)} = q_{i-1} - \frac{q_{i-1} - d(n)}{f(n)}$. Therefore, if $\frac{q_{i-1}}{2} \geq d(n)$, then $q_i \leq (1 - \frac{1}{2f(n)})q_{i-1}$. Thus, after $O(f(n)log(n))$ steps, either $q_{i-1} \leq 2d(n)$, or else $k_i \leq \bar{k}$. In the first case, $q_{i-1} \leq 2d(n)$, and we can fill all the $2d(n)$ missing bits with a truly random string. In the second case, $k_i \leq \bar{k}$, i.e., $q_{i-1} \leq \bar{k} + s$, so if we add $s = d(n)$ truly random bits, there are only $\bar{k}$ missing bits as required.

Therefore it is sufficient to take $t = O(f(n)log(n))$, and let the final extractor be $E(x, r) \circ y$, where y is of length $2d(n)$ and is truly random.

$\square$

E Lemmas For The Second Extractor

In this section we prove some easy technical lemmas used in Section 5. Let us start with the proof of Claim 5.4:

Claim E.1 (Claim 5.4) *:*

For any $0 < i < t$:

1. *For any $b_{[1,i-1]}$ that can be extended to some b with $Y(b) = i$:*

$$\Pr(Y = i \mid B_{[1,i-1]} = b_{[1,i-1]}) = \Pr(f = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon$$

2. *For any $b_{[1,i-1]}$ that can be extended to some b with $Y(b) = t$:*

$$\Pr(Y = t \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon_{i-1} - \Sigma_{j=i}^{t} \epsilon_j \geq \epsilon$$

Proof:

proof of (1) :

Since $b_{[1,i-1]}$ can be extended to some b with $Y(b) = i$, any extension b' of $b_{[1,i-1]}$ with $f(b') = i$ is not bad. Therefore,

$$\Pr(Y = i \mid B_{[1,i-1]} = b_{[1,i-1]}) = \Pr(f = i \mid B_{[1,i-1]} = b_{[1,i-1]})$$

Also, since b is not bad:

$$\Pr(f = i \mid B_{[1,i-1]} = b_{[1,i-1]}) > \epsilon$$

and this completes the proof of (1).

proof of (2) :

$$\begin{aligned} \Pr(Y = t \mid B_{[1,i-1]} = b_{[1,i-1]}) &\geq \Pr(f = t \mid B_{[1,i-1]} = b_{[1,i-1]}) - \\ &\quad \Pr(f = t \text{ and } Y = 0 \mid B_{[1,i-1]} = b_{[1,i-1]}) \\ &\geq \epsilon_{i-1} - \Sigma_{j=i}^{d} \epsilon_j \end{aligned}$$

The last inequality is from Claim E.2.

□

Now we state our last lemma, from which Claim 5.3 also easily follows. First we give a definition:

DEFINITION E.1 *For b s.t. $f(b) = t$ and $Y(b) = 0$ define $YF(b)$ to be the first $i \in [1, t]$ s.t. $\Pr(f = t \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \epsilon_i$, i.e., $YF(b)$ indicates the reason why b is bad.*

Claim E.2

1. *For any $1 \leq i \leq t - 1$ and any $b_{[1,i-1]}$:*

$$\Pr_b(f = i \wedge Y = 0 \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \epsilon$$

2. *For any* $b_{[1,i-1]}$ *that can be extended to* b *with* $Y(b) = t$*:*

$$\Pr_b(f = t \ \wedge \ YF = j \mid B_{[1,i-1]} = b_{[1,i-1]}) \le \epsilon_j$$

3. *For any* $b_{[1,i-1]}$ *that can be extended to some* b *with* $Y(b) = t$*:*

$$\Pr(f = t \ \mathit{and} \ Y = 0 \mid B_{[1,i-1]} = b_{[1,i-1]}) \le \Sigma_{j=i}^{t} \ \epsilon_j$$

Proof: [of Claim E.2]

proof of (1) given $b_{[1,i-1]}$, $f = i \ \wedge \ Y = 0$ implies that $\Pr_b(f = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \le \epsilon$, which proves what we require.

proof of (2) First of all it is clear that $\Pr_b(f = t \ \wedge \ YF = j \mid B_{[1,i-1]} = b_{[1,j-1]}) \le \epsilon_j$.

Now,

$$\begin{aligned}
&\Pr_b(f = t \ \wedge \ YF = j \mid B_{[1,i-1]} = b_{[1,i-1]}) &= \\
&\textstyle\sum_{b_{[i,j-1]}} \Pr(B_{[i,j-1]} = b_{[i,j-1]} | B_{[1,i-1]} = b_{[1,i-1]}) \cdot \Pr_b(f = t \wedge YF = j \mid B_{[1,j-1]} = b_{[1,j-1]}) &\le \\
&\textstyle\sum_{b_{[i,j]}} \Pr(B_{[i,j-1]} = b_{[i,j-1]} \mid B_{[1,i-1]} = b_{[1,i-1]}) \cdot \epsilon_j \le \epsilon_j
\end{aligned}$$

proof of (3) Since $b_{[1,i-1]}$ can be extended to some b with $Y(b) = t$, it must hold that $YF(b) \ge i$.

Therefore,

$$\begin{aligned}
\Pr(f = t \ \mathit{and} \ Y = 0 \mid B_{[1,i-1]} = b_{[1,i-1]}) &\le \\
\Sigma_{j=i}^{t} \ \Pr(f = t \ \mathit{and} \ YF = j \mid B_{[1,i-1]} = b_{[1,i-1]}) &\le \Sigma_{j=i}^{t} \ \epsilon_j
\end{aligned}$$

The last inequality is by (2).

$\square$