Refining Randomness

Thesis submitted for the degree
Doctor of Philosophy

by

Amnon Ta-Shma

Submitted to the Senate of the Hebrew University
June 1996

This work was carried out under the supervision of

Professor Noam Nisan

To my beloved parents,
Dvora and Israel

Acknowledgments

I will start with my parents who have done so much to bring me up. I don't know how much I fit your plans, Mum and Dad, but that's me. I love you so very much. This thesis is mainly for you.

The thesis itself, however, was made possible because of you, Noam. I still remember the complexity course you gave while I was still in the army, and the excitement it arose in me. I already knew then that I wanted you to be my advisor, and I was never disappointed. The excitement vanished somewhere (www?), but it was fully compensated by your striking understanding of things. All I know - is yours.

And Noam - thanks for being so nice.

Studying complexity at the Hebrew University was a great pleasure (well, most of the time) and I want to thank Michael Ben-Or, Shafi Goldwasser, Dror Lapidot, Nati Linial, Muli Safra, Robert Szelepcsenyi, Avi Wigderson and David Zuckerman for the classes I took and the discussions we had. I want to specially thank Shafi for her course on Interactive Proofs, and Avi for his beautiful classes on complexity theory. They had a great influence on me. I also want to thank Oded Goldreich for carefully reading the thesis, and for his many excellent comments that significantly improved the thesis. Finally, I want to thank Oded Goldreich and Avi Wigderson for encouraging me. It really helped!

It is a great pleasure to thank Ilan Kremer, Shlomo Huri, Avner Magen, Dorit Aharonov and Roy Armoni for being friends and mates. Dorit - Good luck with the Qworld. Above all, I want to thank Roy for so many exciting discussions, that

helped clarify so many things. No one can deny we succeeded with Gal.

I want to conclude with my family. I will never forget the love and support I received from my brother and sisters even when we deeply disagreed. Nothing will divide us!

Many warm wishes to you Deanna and Harry. I appreciate your letting us go our way. Many times when I look at Gal I appreciate the sacrifice you have made. We love you very much. We wish our small world was smaller.

Finally, to the one who brought light into my lonely life. To the one with whom I share my life, happy or sad. Dear Paula and lovely Gal, my soul and blood - I love you.

Contents

Chapter 1

Introduction

1.1 Randomness Has Lots of Structure

When people say something behaves randomly, they usually mean they can not recognize any pattern in its behavior. Thus, gender and lottery are random. However, random things do have many patterns after all, and everyone knows that it is very rare to see a large family with only female children, or to see the same number winning two different lotteries.

In fact, it turns out that certain properties of random structures are very useful, and very hard to achieve deterministically. Take for example the following problem: A company wants to set up a telephone network between two groups of people. The network should be dynamic, i.e. no matter who currently uses the network, if some person A wants to speak to some person B, and A and B are not in the middle of some other conversation, then the network should supply the link. The company also wants the network to be small (i.e. with few wires) and shallow (i.e. with few interchanges between any two people speaking). A somewhat surprising fact is that it is very hard to explicitly build such a network. Even more surprising is that a random network (with the right degree and depth) will almost certainly be almost as good as the best network possible. Thus, the best way for the company to find a good network is to randomly choose one!

This brings to light an important phenomenon: Many explicit structures will almost always occur in a randomly chosen object. In fact, Erdos and many others after him, used this phenomenon to develop a method, "The probabilistic method", for proving the existence of combinatorial objects with certain properties. The method has been very successful, and the interested reader is referred to [AES92]. A similar very successful rule of thumb says that for many reasonable combinatorial problems, if the "natural" randomized construction does not have the required property, this property can not be achieved at all. In fact, this phenomenon is so widespread that it is commonplace to use it as the only test for the existence of non-explicit constructions. It is as if almost all structures with simple description have the property that if they exist, it is easy to randomly find an object with that structure [1] .

In light of all of the above, it must be clear that randomness does not mean disorder. There are rules behind our choices and even probabilistic rules are still rules [2]. Moreover, it seems that using these probabilistic rules, we can do many things more easily or more efficiently than what can be done using "conservative" deterministic rules. As a consequence two natural questions arise:

- Can we build probabilistic computers? Is randomness a feasible resource? and,

[1] It might be instructive to compare the strong belief in this rule of thumb to the strong disbelief that $NP \subseteq BPP$.

[2] Our world is a good example of that, as quantum theory states that things behave according to certain probabilistic rules and still quantum theory has a lot of structure and order.

- Do we really need randomness. Or, in other words, is there a way to simulate probabilistic algorithms deterministically?

In the next section we give an important example of a randomized algorithm. Then in section 1.3 we address the feasibility question, and in section 1.4 we discuss derandomization. Finally, in section 1.5 we present our new constructions and results.

1.2 An Example: Random Walks

It was already stated before that many problems have simple and efficient randomized algorithms solving them. As we know that random constructions easily achieve certain combinatorial properties, it might be suspected that the role of randomness in the solutions is to achieve an object with a certain required combinatorial property. However, it turns out that in many randomized algorithms you can not isolate any simple property that suffices to solve the problem. Thus, the solution benefits from the randomness in the deepest sense possible. Randomness is not just a tool to achieve certain constructions, randomness is the thing itself and it is too big to be described only by some simple properties. To demonstrate this, I want to give an important example that we will later study in Chapter 3. The problem is called the undirected s, t connectivity problem, but I prefer to present it here by a story:

Imagine that you wake up in a dark night at the middle of a deserted street of a strange city. In your desperate condition you decide to go to a police station and ask for help. The only problem is you have no knowledge where you are and where the police station is, and anyway you have no map of the city. Suppose you decide to wander around, at each intersection randomly choosing where you go next (you might even go back to the street where you came from). At this point the reader probably joins me in thinking "you poor thing, walking aimlessly in a strange city. What chance do you have to reach the police station?" So, let us face the question: what chance do you have?

At first glance, this random wandering looks like a pretty bad idea. In fact it might even be suspected that the way the city is built will cause you (with high probability) to stay within your current neighborhood and never go out of it. However, if you know the [AKL+79] theorem you know that no matter what the topology of the city is, with overwhelming probability it will not take too long until this random walk will take you to the police station. Thus wandering around randomly, is indeed a very good way to reach a destination! If you lack the knowledge - take a random action!

Notice how simple and elegant the random walk algorithm is. Notice also that the algorithm is local, taking into account only local data, and yet, it works for any city (and even cities in higher dimensions). Finally, though the theorem has a nice and simple proof, the proof does not reveal any combinatorial property that suffices to solve the problem. The random walk algorithm makes a direct use of randomness live and unabridged.

1.3 Is Randomness Feasible?

By now we should be quite convinced that randomness is a very nice tool. Let us first check whether randomness is a feasible resource. We first discuss the possible physical devices outputting "random" bits (section 1.3.1), and then we discuss whether these devices can be used to generate truly random bits (section 1.3.2).

1.3.1 Chaos, Quantum Mechanics and Crude Randomness

At first glance it seems we are surrounded with random phenomena, and so we can throw dice, for example, to get the required randomness. However, most things around us are not random but chaotic. Take dice for example. If you have knowledge of all relevant data prior to the dice throw, and if you have enough computational power, you can know the result in advance. And the same applies to lottery. Thus, chaotic systems are not truly random but instead only "look random" to a limited observer. Therefore, using a chaotic mechanism to output "seemingly random" bits might end up in bits that are strangely correlated, causing the generated distribution to be very far from uniform.

However, there is one significant exception to this. In a tremendous earthquake in the way we view our world, Quantum Theory introduced non-determinism into our world. Certain things can not be determined, not because we do not have the power or wits to analyze them, but because they behave non-deterministically. This startling understanding is so provocative that Einstein refused to accept it saying that "God does not play dice" (and here dice means random (not chaotic) behavior). Yet, it seems that Quantum Theory is correct, and accepting that, our world does have a source containing real randomness [3].

Indeed, circuits exploiting quantum mechanics can be built (e.g. using Zener diodes) and their output is, indeed, far from deterministic. Unfortunately, it turns out that the generated distribution is also very far from uniform. Thus, using

[3]As to God, I believe it must be very boring to run a deterministic world that can be fully predicted. What is the point in playing a game whose moves and results are known in advance. A probabilistic world sounds like a great improvement, and the quantum world with its simple and sophisticated mechanism seems to be an even better one. It might even be interesting to run and watch such a system!

such circuits we get a distribution that is truly unpredictable (i.e. it contains some randomness), yet is not uniform (i.e., there are many correlations among the different output bits). In fact, many times it is very hard to know what the generated distribution is and the only thing that can be said is that there is some randomness in it. In short, using quantum mechanics we can produce “crude” randomness. What we investigate next is whether we can transform this crude randomness into a nice uniform distribution.

1.3.2 Refining Crude Randomness

Let us have another look at the problem we have at hand. Assume our Zener diodes output three bits b_1, b_2, b_3, by uniformly choosing a string out of a set of four possible strings, and further assume that we do not know what these four strings are. In figure 1.3.2 we illustrate three such distributions. In Distribution A, b_1 is fixed on 0 and the two other bits are randomly chosen. In distribution B each pair of bits is random (i.e. all four possible combinations appear with equal probability) and each pair determines what is the remaining bit. In distribution C, no bit is uniform (every bit is “1” with only probability 1/4). The common property shared by all these distributions is that a string is uniformly chosen out of four possible strings, and thus in a very basic sense there are two bits of randomness in each of these distributions.

Can we extract even a single random bit, from such a distribution? And formally:

We look for a function $f : \{0,1\}^n \mapsto \{0,1\}$ s.t. *for any* distribution X on $\{0,1\}^n$ that is uniformly distributed over m strings, the distribution $f(X)$ is random.

000	001	000
001	010	001
010	100	010
011	111	100
Distribution A	Distribution B	Distribution C

Figure 1.1: Three distributions over 3 bits with 2-randomness

Taking $f(b_1, \ldots, b_n) = b_1$, i.e. f returns the first bit, is no good as X can fix the first bit (as happens in distribution A). Taking $f(b_1, \ldots, b_n) = b_1 \oplus b_2 \ldots \oplus b_n$ is also not good as this sum can also be a constant (as happens in distribution B). In fact, it is not hard to show that for any $f : \{0,1\}^n \mapsto \{0,1\}$ there is a distribution X that is uniform over a set of size $\frac{2^n}{2}$ s.t. $f(X)$ is fixed ([SV86]). That is, not a single bit can be extracted if the only thing we know is that the given distribution contains a large amount of randomness, without knowing explicitly what this distribution is.

Let us restate the core of our problem: whenever we choose a function $f : \{0,1\}^n \mapsto \{0,1\}$, there is a distribution X that is bad for f, and thus no f works well for any distribution X. We can think of it as a game against an adversary: we pick a function f, and our adversary picks the worst distribution X for this f.

So the reason we fail is that our malicious adversary makes his choice X after seeing our function f. But what happens if our function f uses some small amount of truly random bits y? Notice that in this case it is possible that the adversary will not be able to choose a distribution X that is bad for f, simply because the adversary does not know y and therefore does not have complete knowledge of f.

Indeed this amounts to a novel idea: keep some coins in your pocket and let the adversary make his choice without having complete knowledge. Such techniques have been found very successful in fooling adversaries in interactive proofs and in many other places. Can they be useful here? Let us be more formal. In our case the extracting function takes the following form:

> We look for a function $f : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}$ s.t. *for any* distribution X on $\{0,1\}^n$ that is uniformly distributed over $m < \frac{2^n}{2}$ strings, the distribution of $f(x,y)$ when x is chosen according to the distribution X, and y is chosen uniformly from $\{0,1\}^t$, is very close to uniform.

We call such a function an "extractor". In chapter 2 we will see that using few random bits (i.e. a very small t) we can extract almost all of the randomness present in the source X, no matter what X is. Thus, extractors extract almost all of the randomness (energy) from any source (fuel tank) by using few truly random bits (some little energy). Looking at it differently, extractors take crude randomness, and by investing a little amount of extra energy refine it to pure uniform randomness.

In particular, extractors can take the crude randomness created by Zener diodes and extract an almost uniform distribution from it - making randomness a feasible resource.

1.4 Derandomization

Now we turn into the second question: "Do we really need to use randomness?". I.e. is there a natural problem easily solvable by a randomized algorithm that has no efficient deterministic solution? Alternatively, can we find a simple criterion such that any probabilistic algorithm satisfying this criterion can be *derandomized*, yielding a matching deterministic algorithm?

In the next sections we first discuss derandomization of algorithms (section 1.4.1) and then derandomization of whole complexity classes (1.4.2).

1.4.1 Derandomizing algorithms

In the current state of the art, algorithms can be derandomized if they do not use randomness to its full extent, but instead only take advantage of certain properties that can be (more) easily achieved. Let us take, for example, an algorithm that uses a distribution over F^n that need only be pair-wise independent , i.e. any two elements are mutually independent (F is some field). One possible distribution is the uniform distribution over F^n, and its size (the number of possible values) is $|F|^n$. It turns out that we can build much smaller distributions (of size $|F|^2$) that have the same property. Thus, we can reduce our sample space from $|F|^n$ to $|F|^2$. Once our sample space has a smaller size, we can try all the elements in the sample space, and deterministically find out the result.

Two explicit constructions are extremely useful when derandomizing algorithms:

- k-wise independence
  - A small sample space that is k-wise independent (any k elements are mutually independent) [CG89].
  - An even smaller sample space that is almost k-wise independent (any k elements are *almost* mutually independent) [NN93, AGHP92].

- Expanders - explicit graphs with constant degree and strong expansion properties [Mar75, LPS86].

The interested reader is referred to [Wig94] for a survey of pair-wise independence, and [MR95] for a reading on probabilistic algorithms and derandomization.

We can also view extractors as explicit graphs with strong random properties. As such, they can serve as a derandomization tool. Indeed, during the last few years, many derandomization results were found using extractors (see [Nis96] for a list of applications, or section 2.5 for those applications improved by our new extractors). In particular, extractors provide the best deterministic amplification known today (see [Nis96]).

Thus, extractors are important not only because they allow us to use randomness in our real world computations, but also as a tool in studying the connection between randomized and deterministic computations.

1.4.2 Pseudo-Randomness

Wouldn't it be nice if we could derandomize all probabilistic algorithms belonging to a certain class C? I.e., show that if a problem is solvable by a probabilistic algorithm running in the class C, then it can also be solved in C without using random bits. Wouldn't it even be nicer if we could do that in a uniform way? I.e. if we could find a function outputting a distribution that looks random to all tests that can be done in the class C.

Definition 1.4.1 *$G : \{0,1\}^t \mapsto \{0,1\}^n$ is a pseudo random generator that fools the class C, if for every function $f : \{0,1\}^n \mapsto \{0,1\}$ computable in C,*

$$|Pr(f(u^n) = 1) - Pr(f(G(u^t) = 1)| \leq \frac{1}{6}$$

when u^n is chosen uniformly from $\{0,1\}^n$ and u^t is chosen uniformly from $\{0,1\}^t$.

It is clear that if we have a pseudo random generator $G : \{0,1\}^t \mapsto \{0,1\}^n$ that fools C, then we can replace the random string given to any probabilistic algorithm in C with the output of the generator G. Thus, any probabilistic algorithm taking time T can be simulated by a deterministic algorithm taking time $T \cdot 2^t$ (ignoring the time needed for the generator).

In a series of brilliant papers, a tight connection between pseudo-random generators for C and finding functions that are "hard" for C was made. On the intuitive level, if we have a "C–hard" function we can use it to generate bits that look random to any algorithm in C, and on the other hand if we have a pseudo-random generator for C then the function identifying all strings generated by the generator is hard for C.

Building on the pioneering work of Blum and Micali [BM82] and Yao [Yao82], Impagliazzo, Levin and Luby [Lev87, ILL89, HILL91] showed that pseudo-random generators that run in polynomial time (in the seed length) and fool polynomial-size circuits, exist, iff one-way functions exist (functions that are easy to compute but hard to invert). Nisan and Wigderson [NW88, BFNW93] studied the possible existence of pseudo-random generators that run in *exponential time* (in the seed length) and fool polynomial-size circuits, and showed that such functions exist iff there exists a function solvable in exponential time that is hard for any polynomial size circuit.

These results show that the problem of finding pseudo-random generators for small circuits is closely related to that of finding explicit functions that are hard for the class we want to derandomize. This later question has a long and very frustrating history. In fact, except for a single stunning success (for the class AC^0) almost no progress was made on this famous problem. Thus, the existence of pseudo-random generators has far reaching implications which currently seem to be beyond our reach.

However, space limited classes with a *read-once random tape* are a major exception to the above rule. The tests such a machine can perform to check whether a given string is truly random or not are limited not only by the class limitations (i.e. limited memory space), but also by a severe read-once limitation on accessing the tested string itself. The point is that since the machine can read each random bit only once, and since the machine's memory is limited, many different random strings will bring the machine to exactly the same configuration.

At this point it is worthwhile to mention that even such limited machines can use randomness in a highly non-trivial manner. In particular the random walk example given in section 1.2 can be solved by a machine using logarithmic space and a polynomially long read-once random tape. Thus, finding a pseudo-random generator for RL, Random Logspace, would yield the first deterministic Logspace solution to the undirected s, t connectivity problem, and many others.

Indeed, several pseudo-random generators for the class RL exist. These pseudo-random generators use the derandomization tools listed in the previous sections: [Nis92] uses hash functions, [INW94] use expanders and [NZ93, Arm, Zuc96] use extractors. Still, none of these constructions is optimal, and it is a major open problem to show that pseudo-random generators for RL exist in L (or even in P).

1.5 Our Work

In this section we state the new results we achieved. In section 1.5.1 we describe the new constructions of explicit extractors and some of their applications, and in section 1.5.2 we show that SL, the class of problems reducible to the undirected s, t connectivity problem, is closed under complement.

The results of section 1.5.1 were published in [TS96]. The results of section 1.5.2 are joint work with my advisor Noam Nisan, and have appeared in [NTS95, NT95].

1.5.1 New Explicit Extractors and Applications

This section is more technical. We will give formal definitions of (explicit) extractors. Then we will state the lower bound and size of best non-explicit construction. Finally we state what explicit extractors were known, and the new extractors we constructed, along with some new applications. In our construction we use a new general tool, called "a merger", which seems to be a useful tool for dealing with random sources. However, since it requires a lot of background, we defer its presentation to Chapter 2.

Formal definitions

First we define how we measure randomness. We say a distribution $\overline{D}$ contains m "randomness" if no string has probability greater than 2^{-m}. This measure is closely related to the Renyi entropy [Ren70], and was suggested by Chor and Goldreich [CG88] as the right measure for this problem.

Definition 1.5.1 *[Ren70, CG88] The min-entropy of a distribution* $\overline{D}$ *is* $H_\infty(\overline{D}) = min_x(-log(\overline{D}(x))$.

It is not hard to see that for any distribution $\overline{X}$, $H(\overline{X}) \geq H_\infty(\overline{X})$, where $H(\cdot)$ is the entropy function. Also, if $\overline{X}$ is uniform over some set $A \subseteq \{0,1\}^n$ then $H(\overline{X}) = H_\infty(\overline{X}) = log_2(|A|)$.

Next we define what an extractor is:

Definition 1.5.2 $E = \{E_n : \{0,1\}^n \times \{0,1\}^{t=t(n)} \mapsto \{0,1\}^{m'=m'(n)}\}$ *is an* $(m = m(n), \epsilon = \epsilon(n))$ *extractor if for every* $n \in N$, *and every distribution* $\overline{X}$ *on* $\{0,1\}^n$ *with* $H_\infty(\overline{X}) \geq m$, *the distribution of* $E_n(x, y)$ *when choosing* $x \in \overline{X}$ *and* y *randomly from* $\{0,1\}^{t(n)}$, *is* ϵ *close to uniform.*

Remark 1.5.1 *This definition is slightly different from the one in [NZ93, SZ94]. In [NZ93, SZ94]* E *is an extractor if* $\overline{Y} \circ E(\overline{X}, \overline{Y})$ *is close to uniform, while we only demand that* $E(\overline{X}, \overline{Y})$ *is close to uniform.*

Notation 1.5.1 *Instead of saying that* $E = \{E_n : \{0,1\}^n \times \{0,1\}^{t=t(n)} \mapsto \{0,1\}^{m'=m'(n)}\}$ *is an* $(m = m(n), \epsilon = \epsilon(n))$ *extractor, we will say that* $E : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}^{m'}$ *is an* (m, ϵ) *extractor.*

Lower bound and non-explicit constructions

Nisan and Zuckerman showed a lower bound on the number of truly random bits needed for an extractor to extract even one additional random bit. As mentioned before, Nisan and Zuckerman use a slightly different definition of extractors. However, slight adaptations to their proof yields the following lower bound:

Fact 1.5.1 *[NZ93] For any $m = m(n)$ and $\epsilon = \epsilon(n)$, any (m,ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}^{t+1}$, must have $t = \Omega(log(n-m) + log(\frac{1}{\epsilon}))$.*

This lower bound matches (up to a constant factor) the non-explicit construction:

Fact 1.5.2 *For every $m = m(m)$ and $\epsilon = \epsilon(n)$, there are (non-explicit) (m,ϵ) extractors $E : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}^m$, with $t = O(log(n) + log(\frac{1}{\epsilon}))$.*

Previous explicit extractors

We would like to have an *explicit* construction:

Definition 1.5.3 *We say E is an* explicit *(m,ϵ) –extractor $E : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}^{m'}$, if it is an extractor and for any $x \in \{0,1\}^n$ and $y \in \{0,1\}^t$, $E(x,y)$ can be computed in polynomial time in $n+t$.*

The following table summarizes the explicit extractors that were previously known:

m min-entropy	t truly random bits	m' extracted randomness	reference
$\Omega(n)$	$O(log^2 n \cdot log(\frac{1}{\epsilon}))$	$\Omega(m)$	[NZ93]
$\Omega(n)$	$O(log(n) + log(\frac{1}{\epsilon}))$	$\Omega(m)$	[Zuc96]
$\Omega(n^{1/2+\gamma})$	$O(log^2 n \cdot log(\frac{1}{\epsilon}))$	$n^\delta, \delta \leq \gamma$	[SZ94]
$\Omega(n^\gamma)$	$O(logn)$	$n^\delta, \delta < \gamma$	[SSZ95] Disperser[4]
any m	m	cm , constant $c > 1$	[SZ94, GW94]

[4]A disperser is a weak extractor. See definition 2.1.4 or [Nis96].

We see that all constructions require at least n^γ min-entropy for some constant $\gamma > 0$. Also, all constructions extract much less bits than the min-entropy that exists in the given source.

We devise two new explicit constructions that are summarized in the following table, and shortly discussed in the next two subsections:

m min-entropy	t truly random bits	m' extracted randomness	reference
any m	$polylog(n) \cdot log(\frac{1}{\epsilon})$	m	
n^γ	$O(log(n) \underbrace{loglog \dots log}_{k}\ n)$	n^δ	$\epsilon = \frac{1}{n}, \gamma > \delta > 0$ any constant k
any m	$O(logn + log(\frac{1}{\epsilon}))$	m	Lower bound

Extractor for any min-entropy!

We devise a new tool for building extractors, which we call "somewhere random mergers". We use this tool to achieve two new extractors. The first extractor we achieve is:

Theorem: *For every $\epsilon = \epsilon(n)$ and $m = m(n) \leq n$, there is an explicit (m, ϵ)–extractor $E : \{0,1\}^n \times \{0,1\}^{t=poly(log(n)} \mapsto \{0,1\}^{m'=m}$.*

That is, this extractor works for any min-entropy, *small or large*, and extracts *all* the randomness present in the given source. These properties turn out to be very important for some applications, most notably the following two corollaries:

Corollary: *(improving [WZ93]) For any N and $1 \leq a \leq N$ there is an explicitly constructible a–expanding graph with N vertices, and maximum degree*

$O(\frac{N}{a}2^{polyloglog(N)})$ [5].

Another important corollary, that solves a problem similar to the network problem presented in section 1.1, is:

Corollary: *(improving [WZ93]) For any N there is an explicitly constructible superconcentrator over N vertices, with linear size and $polyloglog(N)$ depth* [6].

See section 2.5 for more details on these and other applications.

Simulating random classes with sources having high min-entropy

Our second extractor is motivated by the problem of simulating BPP using only defective sources having high min-entropy.

In section 1.3.2 we discussed whether randomized algorithms are indeed feasible. We saw that crude randomness does exist in nature, and we looked for extractors to extract truly random bits from it. Let us formalize the problem. For every $n \in N$, we are given a source $\overline{X}$ over $\{0,1\}^n$ with high min-entropy $H_\infty(\overline{X})$. We want to simulate any algorithm in RP (probabilistic polynomial-time, one sided error) or BPP (two sided error) using the source $\overline{X}$ as our only source of randomness. We also want it to be a black-box simulation, i.e., it is done by calling the original algorithm (possibly several times) and replacing the required random strings with new strings we compute from $\overline{X}$.

Fact 1.5.3 *[CW89] Any polynomial time, black-box simulation of RP or BPP, must use a source $\overline{X}$ with $H_\infty(\overline{X}) \geq n^\gamma$ for some $\gamma > 0$.*

[5] See section 2.5 for the definition of a–expanding graphs. The obvious lower bound is $\frac{N}{a}$. The previous upper bound [WZ93, SZ94] was $O(\frac{N}{a} \cdot 2^{log(N)^{1/2+o(1)}})$.

[6] This improves the current upper bound of $O(log(N)^{1/2+o(1)})$ due to [WZ93, SZ94].

If we have an explicit (m, ϵ) extractor $E : \{0,1\}^n \times \{0,1\}^t \mapsto \{0,1\}^{m'}$, than by investing t truly random bits, we can extract from sources $\overline{X}$ with m min-entropy, m' almost truly random bits, which we can use as an input to the original randomized algorithm. It is true that we still need to invest t truly random bits, and we do not have a source outputting truly random bits. However, instead of using t truly random bits we can try all 2^t possibilities and decide according to the majority. Thus, by using the [SZ94] extractor (see the table above), we get:

Corollary 1.5.1 *[SZ94] For any $\gamma > 0$, BPP can be simulated using sources with $H_\infty(\overline{X}) \geq n^{1/2+\gamma}$, in $n^{O(log(n))}$ time.*

[SSZ95] showed that for RP, n^γ min-entropy suffices, and the simulation can be done in polynomial time:

Corollary 1.5.2 *[SSZ95] For any $\gamma > 0$, RP can be simulated in polynomial time, using sources with $H_\infty(\overline{X}) \geq n^\gamma$.*

The second extractor we build, works for sources with high min-entropy (n^γ for any constant $\gamma > 0$), and invests only slightly more than $O(log(n))$ truly random bits:

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

Corollary: *For any $\delta > 0$ and $k > 0$, BPP can be simulated in time $n^{O(log^{(k)}n)}$ using a weak random source $\overline{X}$ with min-entropy at least n^δ .*

1.5.2 SL and RL

We conclude with the s-t connectivity problem we presented in section 1.2.

A natural complexity measure is the amount of memory required to solve a problem. When modeling computations with Turing machines, this amounts to the space complexity - the size of the memory a Turing machine needs in order to solve the problem. Let us define L as the class of log-space languages, RL its one-sided analog and NL its non-deterministic analog. Next we give exact definitions of these classes:

Definition 1.5.4 *(L) A language $A \subseteq \{0,1\}^*$ is in L iff there is a deterministic Turing machine M s.t. :*

- *M can access a read-only tape whose content is the input x.*
- *M has a read/write tape of length $O(log(n))$.*

and $x \in A \leftrightarrow M(x)$ accepts.

That is, the machine has an input tape (which is read-only) of length n, and "working" tape of length only $O(log(n))$. Now, let us strengthen this class by allowing it to use randomness:

Definition 1.5.5 *(RL) A language $A \subseteq \{0,1\}^*$ is in RL, iff there is a deterministic Turing machine M s.t. :*

- *M can access a read-only tape whose content is the input x.*
- *M has a polynomially long read-once tape having a random content y.*

- *M has a (read/write) working tape of length $O(log(n))$.*

and

- $x \in A \rightarrow Pr_y$ $[M(x, y)$ *accepts*$]$ $\geq 1/2$.
- $x \notin A \rightarrow Pr_y$ $[M(x, y)$ *accepts*$]$ $= 0$.

There are several things to notice about this definition. First, notice that the machine can use a new random bit at any time it wishes. Second, notice that the machine can not remember much of these random bits, since the random tape is *read-once*, and the working space is limited. Thus, algorithms that utilize the random bits must do that in a "use and throw" way. An example for such an algorithm is the algorithm of section 1.2.

For completeness, we add the definition of the class *NL*, non-deterministic Logspace:

Definition 1.5.6 *(NL) A language $A \subseteq \{0, 1\}^*$ is in NL, iff there is a deterministic Turing machine M running in Logspace s.t. there is a polynomial $p(n)$ and for any $x \in \{0, 1\}^n$ there is a "witness" $y \in \{0, 1\}^{p(n)}$ such that $x \in A \leftrightarrow \exists_y$ $M(x, y)$ accepts.*

We can compare *RL* and *NL* in terms of the required size of the "witness set". We say y is a "witness" if $M(x, y)$ computes the right answer. Both in *RL* and *NL*, if x is not in the language then all y's are good witnesses. The situation is different when x does not belong to the language: in *NL* we require that there is at least one good witness, while in *RL* we require that at least half of the y's are good witnesses.

It is easy to see that $L \subseteq RL \subseteq NL$. It is also not hard to see that the directed s, t connectivity problem *STCON*, the problem whether two vertices s and t are connected in a given *directed* graph G, is complete for *NL*.

A special case of the connectivity problem for general graphs, is the connectivity problem for *undirected* graph, *USTCON*.

Definition 1.5.7 *(USTCON)*

Input *: an undirected graph $G = (V, E)$, and two vertices $s, t \in V$.*

Output *: whether s is connected to t in G.*

In a beautiful paper, Aleliunas, Karp, Lipton, Lovasz and Rackoff [AKL+79] showed that, with a high probability, a random walk over an undirected graph covers all the graph nodes in polynomial time, thus showing that *USTCON* can be solved in *RL* - a result we already mentioned in section 1.2.

So *STCON* is as hard as *NL*, while *USTCON* is not harder than *RL* which looks easier than *NL*. That's the time for a name for a new Class! Indeed Lewis and Papadimitriou [LP82] defined a class *SL*, Symmetric Logspace:

Definition 1.5.8 *(SL) [LP82] A language $A \subseteq \{0,1\}^*$ is in SL, iff there is a log-space reduction from A to $USTCON$.*

In fact Lewis and Papadimitriou showed that the following definitions to *SL* are equivalent:

1. Languages which can be reduced in *Logspace* via a many-one reduction to *USTCON*, the undirected st-connectivity problem.

2. Languages which can be recognized by symmetric nondeterministic Turing Machines that run within logarithmic space. See [LP82].

3. Languages that can be accepted by a uniform family of polynomial size contact schemes (also sometimes called switching networks.) See [Raz91].

In particular, the Aleliunas et al. result shows that $SL \subseteq RL$. Adding this to the former inclusions we get: $L \subseteq SL \subseteq RL \subseteq NL$. If we have to guess if these containments are tight what would be our first (or second) guess? I guess "NO". and as usually happens in complexity theory (and in life in general), pessimism rules until someone shows the contrary [7].

Thus, the proofs by Immerman and Szelepcseny [Imm88, Sze88] that NL is closed under complement, came as a great surprise to the scientific community. The same technique, inductive counting, was used by Borodin et al [BCD+89] to show that $SL \subseteq coRL$. However, this technique failed to solve the more general problem whether the class SL is closed under complement. As a consequence, an SL hierarchy was built [Rei82, BCD+89], and turned out to contain many interesting problems, such as 2-colorability [Rei82].

In a result co-authored with my advisor Noam Nisan, we develop a new technique and show that $SL = coSL$, collapsing, in particular, the SL hierarchy.

[7] But, in fact, what do we have to base our guess on? Do we have the slightest indication that $L \neq NL$? If we have any indications at all, they show that RL is very close to L, which turns the $L \neq NL$ question into $RL \neq NL$, which looks wide open.

Chapter 2

Explicit Extractors

In this chapter we present the currently known techniques for building explicit extractors. In section 2.2 we present the main ideas used in previous constructions, including the tiny hash function extractor (section 2.2.1), block-wise sources (section 2.2.2) and two techniques for converting arbitrary random sources to block-wise sources (section 2.2.3). In section 2.3 we present our new technique for building explicit extractors. We use it to build a new extractor working for any min-entropy and extracting all the randomness present in a random source. In section 2.4 we use these ideas to construct another new extractor, which uses less random bits. Finally, in section 2.5 we state some applications that were improved by our new constructions. We start the chapter with a short preliminary section (section 2.1), containing notation, definitions and some well known facts.

2.1 Preliminaries

2.1.1 Notation

We use standard notation for random variables and distributions. If $\overline{X}$ is a distribution, $x \in \overline{X}$ denotes picking x according to the distribution $\overline{X}$. If A is a random variable we denote by $\overline{A}$ the distribution A induces. If A and B are (possibly correlated) random variables then $(A \mid B = b)$ is the conditional distribution of A given that $B = b$. We denote by U_t the uniform distribution over $\{0,1\}^t$. For a random variable $X = X_1 \circ \ldots \circ X_n$ over $\{0,1\}^n$ we write $X_{[i,j]}$ as an abbreviation for the random variable $X_i \circ X_{i+1} \ldots \circ X_j$, and the same applies to instances $x_{[i,j]}$.

We define the variation distance between two distributions $\overline{X}$ and $\overline{Y}$ as:

$$d(\overline{X}, \overline{Y}) \stackrel{\text{def}}{=} \frac{1}{2}|\overline{X} - \overline{Y}|_1 \stackrel{\text{def}}{=} \frac{1}{2} \sum_a \mid \overline{X}(a) - \overline{Y}(a) \mid$$

We say $\overline{X}$ is ϵ-close to $\overline{Y}$ if $d(\overline{X}, \overline{Y}) \leq \epsilon$. We say two random variables A and B are ϵ–close, if $d(\overline{A}, \overline{B}) \leq \epsilon$. We say $\overline{X}$ is ϵ quasi-random, if it is ϵ-close to uniform.

We list some well known properties of the variation distance:

Fact 2.1.1 *Let $\overline{D_1}, \overline{D_2}$ be two distributions on Λ_1, and let $f : \Lambda_1 \mapsto \Lambda_2$ be any function, then $d(f(\overline{D_1}), f(\overline{D_2})) \leq d(\overline{D_1}, \overline{D_2})$. I.e., distance between distributions cannot be created out of nowhere.*

Fact 2.1.2 *Let A, B, C and D be any random variables, then $d(\overline{A}, \overline{B}) \leq d(\overline{A \circ C}, \overline{B \circ D})$.*

Fact 2.1.3 *Let A, B and C be any random variables, then $d(\overline{A \circ B}, \overline{A \circ C}) = \sum_{a \in \Lambda_A} Pr(A = a) \cdot d((B \mid A = a), (C \mid A = a))$.*

Finally, for integers t and T, (t) denotes $\{0,1\}^t$ while $[T]$ denotes $[1,\ldots,T]$.

2.1.2 Definitions

We restate the definitions given in section 1.5:

Definition 2.1.1 *[Ren70, CG88] The min-entropy of a distribution* $\overline{D}$ *is* $H_\infty(\overline{D}) = min_x(-log(\overline{D}(x)))$.

Definition 2.1.2 $E:(n)\times(t)\mapsto(m')$ *is an* (m,ϵ)*-extractor if for any distribution* $\overline{X}$ *on* $\{0,1\}^n$ *with* $H_\infty(\overline{X})\geq m$*, the distribution of* $E(x,y)$ *when choosing* $x\in\overline{X}$ *and* $y\in U_t$*, is* ϵ *close to* $U_{m'}$.

When, again, we remind the reader that (k) denotes $\{0,1\}^k$.

Remark 2.1.1 *This definition is different from the one in [NZ93, SZ94]. In [NZ93, SZ94]* E *is an extractor if* $\overline{Y}\circ E(\overline{X},\overline{Y})$ *is close to uniform, while we only demand that* $E(\overline{X},\overline{Y})$ *is close to uniform.*

Definition 2.1.3 *We say* $E:(n)\times(t)\mapsto(m')$ *is an explicit* (m,ϵ)*-extractor, if it is an extractor and for any* $x\in\{0,1\}^n$ *and* $y\in\{0,1\}^t$*,* $E(x,y)$ *can be computed in polynomial time in* $n+t$.

Now we give a graph interpretation of definition 2.1.2. We can view a function $E:(n)\times(t)\mapsto(m')$ as a regular bipartite graph $G=([N=2^n],[M'=2^m],E)$, with 2^n vertices at the left hand side, $2^{m'}$ vertices at the right hand side, and edges $(x,z)\in E$ iff there is some y such that $E(x,y)=z$ (i.e. the graph degree is 2^t). Clearly, if E is an (m,ϵ)-extractor then for any $X\subseteq[N]$ that is large enough

($|X| \geq 2^m$), and any $Y \subseteq [M']$, $\Gamma(X) = \{z \in [M'] \mid \exists x, y \; z = E(x,y)\}$ hits Y with about the right probability (i.e. about $\frac{|Y|}{|M'|}$).

The above definition is very strong and requires that we hit any large enough subset with about the right probability. A weaker requirement is that we just hit any large enough subset. This is captured by the definition of a "disperser", given by Sipser:

Definition 2.1.4 *[Sip88] A (multi-)graph $G = ([N], [M'], E)$ is a (M, ϵ)-disperser if for any $A \subseteq [N], |A| \geq M$, $|\Gamma(A)| \geq (1-\epsilon)M'$,*

It is clear from the above discussion that an explicit (m, ϵ) extractor $E : (n) \times (t) \mapsto (m')$ gives an explicit construction for a regular bipartite graph $G = ([N = 2^n], [M' = 2^{m'}], E)$ that is an $(M = 2^m, \epsilon)$ disperser, and has degree 2^t.

2.2 Previous Work

2.2.1 The Mother of All Extractors

Hashing is a very well known technique in computer science. Many times, in theory and practice, one needs to hash a small set residing in a huge domain to a much smaller domain, with as few collisions as possible. Notice that by definition extractors are good hash functions. The converse is stated in the leftover hash lemma [ILL89]:

Definition 2.2.1

- *Let $\overline{X}$ be a distribution. Define $Col(\overline{X}) \stackrel{\text{def}}{=} Prob_{x_1,x_2 \in \overline{X}}(x_1 = x_2)$.*
- *$H = \{h : [N] \mapsto [D]\}$ is a family of hash functions with collision error δ, if for any $x_1 \neq x_2 \in [N]$, $Prob_{h \in H}(h(x_1) = h(x_2)) \leq \frac{1}{|D|} \cdot (1 + \delta)$.*

The following lemma is a variant of the leftover hash lemma [ILL89]:

Lemma 2.2.1 *Let H be a family of hash functions from $[N]$ to $[D]$ with collision error δ. For any distribution $\overline{X}$ with $Col(\overline{X}) \leq \frac{\delta}{|D|}$, $(h, h(x))$ is quasi-random to within $\epsilon = \sqrt{\delta}$.*

Corollary 2.2.2 *If there exists an explicit family H of hash functions from $[N = 2^n]$ to $[D = 2^d]$, with ϵ^2 collision error, then there exists an explicit $(m = d + 2 \cdot log(\frac{1}{\epsilon}), \epsilon)$–extractor $E : (n) \times (t = log(|H|)) \mapsto (m' = t + d)$.*

Proof: If $H_\infty(\overline{X}) \geq m$ then $Col(\overline{X}) \leq 2^{-m} = \frac{1}{D} \cdot \epsilon^2$. Since H has ϵ^2 collision error, by lemma 2.2.1 $(h, h(x))$ is quasi-random to within ϵ. □

Let's look for small families of hash functions with small collision error. Let us start with families of hash function with *no* collision error.

Definition 2.2.2 *[CW89] $H = \{h : [N] \mapsto [D]\}$ is called a universal family of hash functions, if for any $x_1 \neq x_2 \in [N]$, and for any $y_1, y_2 \in [D]$, $Prob_{h \in H}$ ($h(x_1) = y_1 \wedge h(x_2) = y_2$) $= \frac{1}{|D|^2}$.*

It is clear that a universal family of hash functions has in particular 0 collision error. Also, it is not hard to see that there exists a universal family of hash functions $|H|$ of size $poly(|N|, |D|) = poly(|N|)$. Using this family we need to invest n truly random bits. Can we do any better?

In [CGH$^+$85] it was shown that if we want pairwise independence, and allow no error, we *cannot* do much better. However, remember that we don't really need 0 collision error, and we can afford some small collision error. In other words, we only need *almost* pairwise independence. Amazingly, Naor and Naor [NN93] showed that in this case we can do much better.

Definition 2.2.3 *[NN93] A set S of n-bit vectors is "d wise ρ biased", if for any d indices I (i.e. $I \subseteq \{1, \ldots, n\}$, $|I| = d$), and for any d values $b_1, \ldots, b_d \in \{0, 1\}$:*

$$| \; [Prob_{x \in S} \; (\wedge_{i \in I} x_i = b_i) \;] \; - \; 2^{-|I|} | \; \leq \; \rho$$

Theorem: *[NN93] (see also [AGHP92]) For every integer q, $d \leq q$ and every $\rho > 0$, there is an explicit set S of q-bits vectors that is d-wise, ρ biased, and of cardinality $O(\; (d \cdot log(q) \cdot \frac{1}{\rho})^2 \;)$.*

Srinivasan and Zuckerman used this in a very simple way to show:

Lemma 2.2.3 *([SZ94] and independently [GW94]) There exists an explicit family H of hash functions from $[N]$ to $[D]$, with ϵ collision error, and $poly(log(|N|), \frac{1}{\epsilon}, |D|)$ size.*

Proof: Any function $h : [N] \mapsto [D]$ can be represented by writing all its values on $[N]$. This representation takes $q = |N| \cdot log(|D|)$ bits. Take S to be a set of q-bit vectors that is $d = 2log(|D|)$ wise $\rho = (\frac{\epsilon}{|D|})^2$-biased, and of cardinality $O(\ (d \cdot log(q) \cdot \frac{1}{\rho})^2\ ) = poly(log(|N|), \frac{1}{\epsilon}, |D|)$. Let H be the set of hash functions corresponding to the elements in S. For any $x_1 \neq x_2 \in [N]$:

$$|Prob_{h\in H}\ [h(x_1) = h(x_2)] - \frac{1}{|D|}| \leq \sum_{b\in D} |Prob_{h\in H}(h(x_1) = b \wedge h(x_2) = b) - \frac{1}{|D|^2}| \leq |D|\rho$$

Therefore,

$$Prob_{h\in H}(h(x_1) = h(x_2)) \leq \frac{1}{|D|} + |D|\rho \leq \frac{1}{|D|} + |D|\frac{\epsilon^2}{|D|^2} \leq \frac{1}{|D|}(1 + \epsilon^2)$$

$\square$

Thus, combining this with lemma 2.2.1 we get:

Lemma 2.2.4 *([SZ94] and independently [GW94]) There is some constant $c > 1$ s.t. for any $m = \Omega(log(n))$ there is an explicit $(m, \epsilon = 2^{-m/5})$ extractor $A_m : (n) \times (t = m) \mapsto (m' = cm)$.*

DEFINITION 2.2.1 *Denote the constant c in lemma 2.2.4 by c_{tiny}.*

Notice that this extractor is optimal up to a constant factor when $m = \Theta(log(n))$, I.e., it operates best on the hardest random sources!

2.2.2 Extracting Randomness From Block-wise Sources

The extractor presented in the previous section invests m truly random bits to extract $\Omega(m)$ additional random bits. This is fine for sources with $m = \Theta(log(n))$ min-entropy, but is far too much when m is much larger.

Next we present a novel idea initiated by Zuckerman: using the randomness in the defective source to further extract more and more randomness.

Imagine that you sit in your parked car and you want to start the engine. There is a lot of fuel and energy down there, if only you could use it. So, what do you do? You use a battery that starts the engine which then supplies the energy not only to keep the engine running, but also to get moving. As the process goes on, the fuel that is burnt supplies a larger and larger amount of energy, yet the whole process was triggered by the feeble powers of a battery!

Unfortunately, this nice idea does not work directly for arbitrary random sources. In fact, that is the reason for most of our work. However, for a special class of random sources, which we call block-wise sources, it does work in a very elegant way.

Let us demonstrate the idea on two blocks:

Definition 2.2.4 *A random variable $X = X_1 \circ X_2$ is a $(2,(m_1,m_2),\epsilon)$ block-wise source, if*

- *$\overline{X_1}$ is ϵ close to some $\overline{W}$ with $H_\infty(\overline{W}) \geq m_1$*
- *Call x_1 "good" if $(X_2 \mid X_1 = x_1)$ is ϵ close to some $\overline{W}$ with $H_\infty(\overline{W}) \geq m_2$. Then, $Prob_{x_1 \in X_1}(x_1$is* not *"good"$) \leq \epsilon$.*

  Informally, for most x_1, $H_\infty(X_2 \mid X_1 = x_1) \geq m_2$.

Now, we present the algorithm extracting randomness from such sources, when $m_1 = c_{tiny} m_2$:

ALGORITHM 2.2.1 *Let A_{m_1}, A_{m_2} be the extractors from lemma 2.2.4. Define the extractor E to be $E(x_1 \circ x_2, y) \stackrel{\text{def}}{=} A_{m_1}(x_1\ ,\ A_{m_2}(x_2, y))$.*

We start with m_2 truly random bits. We expect the second block to contain m_2 min-entropy, and therefore by using the extractor A_{m_2} we expect to be left with $c_{tiny} \cdot m_2 = m_1$ quasi-random bits. Then we use the randomness just extracted to further extract randomness from X_1.

Lemma 2.2.5 *[NZ93, SZ94] E is an extractor.*

Proof:

For the proof we need the following basic lemma:

Lemma 2.2.6 *[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 $Prob_{x \in \overline{X}}(x \text{ is bad}) \leq \eta$ then $\overline{X \circ Y}$ is $\epsilon + \eta$ close to $\overline{X} \times \overline{B}$.*

Now, since X is a block-wise source, for most prefixes x_1, $H_\infty(X_2 \mid X_1 = x_1) \geq m_2$. Therefore, for most prefixes x_1, the distribution of $(A_{m_2}(x_2, y) \mid X_1 = x_1)$ is ϵ_1 close to uniform. Thus, it must be the case that $\overline{X_1 \times A_{m_2}(x_2, y)}$ is ϵ_1 close to the distribution $\overline{X_1} \times U$.

Hence, applying A_{m_1} on x_1 with the random string $A_{m_2}(x_2, y)$ is only ϵ_1 far from applying A_{m_1} on x_1 with a truly random string. Therefore, $A_{m_1}(x_1, A_{m_2}(x_2, y))$ is $\epsilon_1 + \epsilon_2$ close to uniform. $\square$

Obviously, if we have more blocks we can continue the process.

Definition 2.2.5 *Let* $X = X_1 \circ X_2 \ldots \circ X_k$. *We say* $x_{[1,i-1]}$ *is a "good" prefix if* $(X_i \mid X_{[1,i-1]} = x_{[1,i-1]})$ *is* ϵ *close to some distribution* $\overline{W}$ *with* $H_\infty(\overline{W}) \geq m_i$. *We say* X *is a* $(k, (m_1, \ldots, m_k), \epsilon)$ *block-wise source, if for any* $1 \leq i \leq k$, $Prob_{x_{[1,i-1]}}(x_{[1,i-1]}$ *is not good*$) \leq \epsilon$. *If* $m_1 = \ldots = m_k = m$ *we say* X *is a* (k, m, ϵ) *block-wise source.*

Lemma 2.2.7 *[SZ94] Let* $X = X_1 \circ X_2 \ldots \circ X_k$ *be a* $(k, (m_1, \ldots, m_k), \epsilon)$ *block-wise source, where* $m_k = \Omega(log(n))$ *and* $m_{i-1} = c_{tiny} m_i$. *Then there is an explicit (block-wise) extractor* $BE(X, U)$, *using* m_k *truly random bits and extracting* $\Omega(\Sigma_{i=1}^k m_i)$ *quasi-random bits with* $O(2^{-\Omega(m_k)} + k\epsilon)$ *error.*

Finally, we give a short discussion of the historical development of these results. Santha and Vazirani [SV86] considered a source over n bits with the property that each bit, even conditioned on the history, has "enough" randomness in it. This was generalized by Chor and Goldreich [CG88] to a source with l blocks, each containing, even conditioned on the history, enough randomness. This is almost the same as definition 2.2.5, with the following changes:

- We allow the blocks to have different lengths.
- We allow the blocks to have different amount of randomness, m_i.
- Instead of requiring that each block has m_i randomness, we only require that each block is *close* to a distribution with that amount of randomness.

However, all these changes are minor and almost every technique that works for "Chor-Goldreich" sources, will also work for block-wise sources.

Chor and Goldreich presented an extractor for (their) block-wise sources, that used one random string y to extract randomness from all the different blocks. This technique was later improved by Nisan and Zuckerman [NZ93], where the *extracted* randomness was used to further extract more randomness, as is done in algorithm 2.2.1. Finally, Srinivasan and Zuckerman plugged the improved basic extractor of lemma 2.2.4 into algorithm 2.2.1 to get an almost optimal block-wise extractor.

2.2.3 Converting an Arbitrary Random Source to a Block-wise Source

In the previous section we saw how to extract randomness from block-wise sources. Now we check whether given a source, we can partition it into blocks that form a block-wise source. Suppose $\overline{X}$ is a distribution over $\{0,1\}^n$ and that $H_\infty(\overline{X}) \geq m = m_1+m_2+s$. Is there a partition of x into two blocks $X = X_1 \circ X_2$ s.t. $H_\infty(\overline{X_1}) > m_1$, and for most strings x_1, $H_\infty(X_2 \mid X_1 = x_1) > m_2$?

Unfortunately, the answer is no. Take for example the following distribution $\overline{X}$ which chooses a string uniformly from the set

$$\{\ 0 \circ \{0,1\}^m \circ 0^{n-m-1}\ ,\ 1 \circ 0^{n-m-1} \circ \{0,1\}^m\ \}$$

and assume $n >> m$. Assume there is a good splitting point at location i. Then, to guarantee that $H_\infty(\overline{X_1}) \geq m_1$, it must be the case that $i \geq n - m_2 - s$. But then for half of the strings x_1, $H_\infty(X_2 \mid X_1 = x_1) = 0$.

This example illustrates both the difficulty and the way to overcome it. Although there is no one good splitting point, *each string* has a good splitting point. In the next sections we explain this, and show two methods using this idea to convert an arbitrary source to a block-wise source.

Block Extraction

In [NZ93] Nisan and Zuckerman showed how to get a block-wise source from a general random source. Let us say a bit is "surprising" if we expected it to be different. For example, if given the history we have probability of 0.6 to see "1" and of 0.4 to see "0", then when we see a "0" we are surprised. The main idea is that if X has high min-entropy, then for *most strings* x, there are many bits in x that surprised us (conditioned on their prefix). Of course, we do not know which bits are surprising and which are not, but by choosing bits pairwise independently we can, with high probability, get a block with many "surprising" bits, and this block, with high probability, has high min-entropy.

Next, we state the Nisan and Zuckerman lemma somewhat more formally. We do not prove the lemma, and the interested reader is referred to the original paper [NZ93]. Let X be a random source over $\{0,1\}^n$. Nisan and Zuckerman construct a function $B_l(x, y)$ which gets $x \in X$ and a short random string y, and returns l bits, s.t.:

Lemma 2.2.8 *[NZ93, SZ94] If $H_\infty(\overline{X}) \geq \delta n$, then $\overline{B(X,U)}$ is $(\delta l)^{\Omega(-1)}$ close to a distribution $\overline{W}$ with $H_\infty(\overline{W}) \geq \Omega(\frac{\delta \cdot l}{log(\delta^{-1})}) \geq \Omega(\frac{\delta \cdot l}{log(n)})$.*

Now we use the above lemma to convert any random source into a block-wise source. Let us start with a general random source $\overline{X}$ with m min-entropy. We can extract, pairwise independently, a block B_1 of length $l << m$ (and therefore with high probability it has $\Omega(l\frac{m}{n \cdot log(n)})$ min-entropy). For most values b_1 of B_1, $H_\infty(X \mid B_1 = b_1) \geq H_\infty(\overline{X}) - O(|B_1|) = m - O(l)$, hence we can extract one more block B_2, which has high min-entropy even conditioned on the history b_1. Actually, as long as $|B_1| + \ldots + |B_k| << m$, we can extract another block, which also has high min-entropy even conditioned on the history. Certainly we can do that $log(n)$ times.

Thus we get:

Lemma 2.2.9 *[NZ93, SZ94] Let $\overline{X}$ be a distribution on $\{0,1\}^n$ with $H_\infty(\overline{X}) \geq n^{1/2+2\gamma}$ for some $\gamma > 0$. Define $b_i = B_l(x, y_i)$ where $l = n^{1/2}$ and $1 \leq i \leq k =$*

$O(log(n))$. Then, $B = B_1 \circ \ldots \circ B_k$ is an $(k, n^\gamma, n^{-\Omega(1)})$ block-wise source.

Notice also the inherent limitation of this method. If we start with less than $n^{1/2}$ min-entropy, we need the first block length to be at least $n^{1/2}$, or else we do not expect even a single random bit. But then it may happen that the first block "stole" all the randomness present in X, and so the second block, given the history, has no randomness at all. Thus, this method seems to work only for sources having at least $n^{1/2}$ min-entropy. In section 2.4 we will use new tools to strengthen this method.

The SSZ dispersers

Srinivasan, Saks and Zhou [SSZ95] showed that randomness can be extracted, in a weak sense, from random sources having n^γ min-entropy, for any constant $\gamma > 0$. Thus, the SSZ method breaks the $n^{1/2}$ bound imposed by the block extraction method. Here we are not interested in the result itself, but rather in the method. Next, we are going to present a simplified version of the SSZ method, in a rather informal way.

Following an idea in [NZ93], Srinivasan, Saks and Zhou look at specific strings. They show that for most strings x, there is a good partition of x to log(n) blocks, s.t. for all i the distribution of X_i given the history $x_{[1,i-1]}$ contains a lot of randomness. Let us say that x "likes" the partition π, if π is good for x. Then, Srinivasan, Saks and Zhou show that there exists a family of partitions, whose size k is polynomial in n, s.t. most strings like some partition from the family.

This, they claim, can be used as follows: partition the universe $\{0,1\}^n$ to $k+1$ classes, each class containing only strings that like the i'th partition (and one class containing all strings that like no partition). All the small classes can be ignored, since they are small. All the large classes are block-wise sources, when we add the condition that we look only at strings that belong to them. We already know how to deal with block-wise sources. Thus, when we look at the blocks extracted from each class, with high probability one of the blocks is uniform, given the right conditioning. Saks, Srinivasan and Zhou use this property to achieve some weak randomness.

In the following sections we are going to study and develop these ideas. By this, we will achieve new and more general results, and we will also be able to put the Srinivasan, Saks and Zhou result in a new context, shedding new light on the method.

2.2.4 Summary

We saw the fundamental "hash" extractor, using m truly random bits to extract some additional $\Omega(m)$ random bits out of m min-entropy. We saw the basic idea of using extracted randomness to further extract more randomness, and saw this work on a special class of random random sources which we called block-wise sources. Finally, we saw two methods to convert an arbitrary source to a block-wise source.

This results in the following extractors:

Lemma 2.2.10 *[SZ94]* [1] *Let $m(n) \geq n^{1/2+\gamma}$ for some constant $\gamma > 0$. For any ϵ there is an explicit $(m(n), \epsilon)$ extractor $E : (n) \times (O(log^2 n \cdot log(\frac{1}{\epsilon}))) \mapsto (\frac{m^2(n)}{n})$.*

Lemma 2.2.11 *[Zuc96] Let $m(n) = \Theta(n)$. For any ϵ there is an explicit $(m(n), \epsilon)$ extractor $E : (n) \times (O(log(n) + log(\frac{1}{\epsilon}))) \mapsto (\Omega(n))$.*

Lemma 2.2.12 *[SSZ95] Let $m(n) \geq n^{\gamma}$ for some constant $\gamma > 0$, then there is some constant $\delta > 0$ and an explicit $(n, m(n), O(log(n)), n^{\delta}, \frac{1}{4})-$ disperser.* [2]

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

[2] We did not define what a disperser is. The reader is referred to [Nis96] for a survey.

2.3 An Extractor For Any Min-Entropy!

2.3.1 An Informal Description

First we notice that for any source X and most strings $x \in \overline{X}$, 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.

Lemma 2.3.1 *Let $\overline{X}$ be a distribution over $\{0,1\}^n$ with $H_\infty(\overline{X}) \geq m_1 + m_2 + s$. Call an $x \in \overline{X}$ "good", if there is some i (dependent on x) s.t.*

- $Pr(X_{[1,i]} = x_{[1,i]}) \leq 2^{-m_1}$ *and*
- $Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) \leq 2^{-m_2}$

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.

$$Pr(X_{[1,i]} = x_{[1,i]}) \quad \leq \quad 2^{-m_1} \tag{2.1}$$

Since i is the first such location,

$$Pr(X_{[1,i-1]} = x_{[1,i-1]}) \quad \geq \quad 2^{-m_1} \tag{2.2}$$

Since $H_\infty(\overline{X}) \geq m_1 + m_2 + s$

$$Pr(X_{[1,n]} = x_{[1,n]}) \quad \leq \quad 2^{-(m_1+m_2+s)} \tag{2.3}$$

Putting this together we get:

$$\begin{aligned}
Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) &= \frac{Pr(X_{[1,n]} = x_{[1,n]})}{Pr(X_{[1,i]} = x_{[1,i]})} \\
&= \frac{Pr(X_{[1,n]} = x_{[1,n]})}{Pr(X_{[1,i-1]} = x_{[1,i-1]}) \cdot Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]})} \\
&\leq \frac{2^{-(m_1+m_2+s)}}{2^{-m_1} \cdot Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]})}
\end{aligned}$$

Hence, for all strings $x \in \overline{X}$ s.t. $Pr(X_i = x_i \mid X_{[1,i-1]} = x_{[1,i-1]}) \geq 2^{-s}$, it holds that $Pr(X_{[i+1,n]} = x_{[i+1,n]} \mid X_{[1,i]} = x_{[1,i]}) \leq 2^{-m_2}$, and x is good. In particular $Pr(x \text{ is not good}) \leq 2^{-s}$. $\square$

The crucial point is that there are only n possible splitting points. If we want to split $x_{[1,n]}$ into k blocks, there are only n^k sets of splitting points, and most strings (all but $k \cdot 2^{-s}$) have a good splitting set. Therefore, we can split the universe $\{0,1\}^n$ to $n^k + 1$ 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 that belong to a specific class S. Then, what we actually see is an input from a block-wise distribution, with the known partition S. Therefore, we know how to extract randomness from it. It is true that given a string x, we have no idea what is the right class (or partition set) for it, but since there are so few classes, we can try all of them. This gives us n^k output strings, one of which is random.

Let us define more precisely the type of source we achieve. We have $d = n^k$ distributions $X_1, \ldots, X_d$, 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. $(X_i \mid Y = i) \approx U$. So let us define:

Definition 2.3.1 *$X = X_1 \circ \ldots \circ X_d$ is a d–block (m, ϵ, η) somewhere random source, if each X_i is a random variable over $\{0,1\}^m$, and there is a random variable $Y = Y(X)$ over $[0..d]$ s.t.:*

- *For any $i \in [1..d]$: $d((X_i | Y = i), U_m) \leq \epsilon$.*
- *$Prob(Y = 0) \leq \eta$.*

We also say that Y is an (m, ϵ, η) selector for X.

The following lemma (proved in appendix A.1) shows that any d–block (m, ϵ, η) somewhere random source, is close to a source with m min-entropy.

Lemma 2.3.2 *(1) Any (m, ϵ, η) somewhere random source X is $\epsilon + \eta$ close to an $(m, 0, 0)$–somewhere random source X'. (2) For any $(m, 0, 0)$ somewhere random source X, $H_\infty(\overline{X}) \geq m$.*

Thus, any extractor that extracts randomness from sources having m min-entropy, also extracts randomness from d–block (m, ϵ, η) somewhere random source. However, d–block (m, ϵ, η) somewhere random source, have an additional structure, and we will see (section 2.3.5) that this nice and simple structure makes it much easier to extract randomness from such sources. Let us call an extractor working only on somewhere random sources, a somewhere random merger:

Definition 2.3.2 *$M : (m)^d \times (t) \mapsto (m')$ is an epsilon–somewhere random merger, if for any d–block $(m, 0, 0)$ somewhere random source X, the distribution of $M(x, y)$ when choosing $x \in \overline{X}$ and $y \in U_t$, is ϵ close to $U_{m'}$.*

Definition 2.3.3 *We say* $M = \{M_n\} : (m)^d \times (t) \mapsto (m')$ *is an explicit* ϵ *–somewhere random merger, if there is a Turing machine that given* $x \in \{0,1\}^{dm}$ *and* $y \in \{0,1\}^t$ *outputs* $M_n(x, y)$ *in polynomial time in* $dm + t$*.*

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 n^k partitions of x into $k = \Theta(log(n))$ blocks.
2. For each partition set i, extract the randomness as from a block-wise source, to get a random string B_i.
3. The distributions $B_1, \ldots, B_k$ form a somewhere random source. Use a merger to merge the randomness in the somewhere random sourceinto 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 $n^{O(log(n))}$ in the scheme above. Second, in the formal description we will give full formal proofs, and thus we will have to specify all the details and hard work needed to implement the ideas above. To ease the reading, we advise the reader to keep this intuitive and informal construction in mind.

2.3.2 Composing Two Extractors

We already know how to extract randomness from sources X that can be "broken" into blocks $X_1 \circ X_2$, s.t. X_1 and $(X_2 \mid X_1 = x_1)$ contain a lot of randomness. We would like to use this for extracting randomness from arbitrary sources. We have already seen that even though not all random sources have such a splitting point, most strings do have such a splitting point. The algorithm we suggest tries all possible n splitting points, and then merges the n results.

To be more precise, given an input string x:

1. split x into two consecutive strings $x_1 \circ x_2$, s.t. the splitting point is good for x
2. use E_1 to extract randomness from x_2
3. use E_2 with the *extracted randomness* to further extract randomness from x_1

Obviously, given a string x, we do not know what is the right splitting point, so we try *all* $|x| = n$ possible ones. This gives us a somewhere random source with n blocks, that can be merged into a single quasi-random string by a good somewhere random merger.

Algorithm 2.3.1 *Suppose $E_1 : (n) \times (t_1) \mapsto (t_2)$ is an (m_1, ζ_1)–extractor, $E_2 : (n) \times (t_2) \mapsto (t_3)$ is an (m_2, ζ_2)–extractor, and $M : (t_3)^n \times (\mu_1) \mapsto (o_1)$ is a ζ_3–somewhere random merger. Define the function $E_2 \overset{M}{\odot} E_1$ as follows: Given $a \in \{0,1\}^n$, choose r_1 uniformly from $\{0,1\}^{t_1}$, and choose r_2 uniformly from $\{0,1\}^{\mu_1}$.*

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 1 *Suppose* $E_1 : (n) \times (t_1) \mapsto (t_2)$ *is an* (m_1, ζ_1)*–extractor,* $E_2 : (n) \times (t_2) \mapsto (t_3)$ *is an* (m_2, ζ_2)*–extractor, and* $M : (t_3)^n \times (\mu_1) \mapsto (o_1)$ *is a* ζ_3*–somewhere random merger. Then for every safety parameter* $s > 0$, $E_1 \overset{M}{\odot} E_2 : (n) \times (t_1 + \mu_1) \mapsto (o_1)$ *is an* $(m_1 + m_2 + s, \zeta_1 + \zeta_2 + \zeta_3 + 8n2^{-s/3})$*–extractor.*

Proof: To prove this, assume $H_\infty(\overline{X}) \geq m_1 + m_2 + s$.

We will show that $E_1 \ominus E_2$ is an $(t_3, \zeta_1 + \zeta_2, 8n2^{-s/3})$–somewhere random source. Thus, by lemma 2.3.2 $E_1 \ominus E_2$ is $\zeta_1 + \zeta_2, 8n2^{-s/3})$–close to a $(t_3, 0, 0)$–somewhere random source. Since M is a merger, by definition 2.3.2 we get that $E(X, U) = M(E_1 \ominus E_2)$ is quasi-random as required.

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 almost the selector but has few "bad" values, then we correct f to obtain the selector Y.

Definition 2.3.1 *Define* $f(w)$ *to be the last* i *s.t* $Prob(X_{[i,n]} = w_{[i,n]} \mid X_{[1,i-1]} = w_{[1,i-1]}) \leq (\epsilon_2 - \epsilon_3) \cdot 2^{-m_1}$.

Definition 2.3.2 *Define* w *to be "bad" if* $f(w) = i$ *and:*

1. $Prob_{x \in X}(f(x) = i) \leq \epsilon_1$, *or,*

2. $Prob_{x \in X}(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \epsilon_2$, *or,*

3. $Prob_{x \in X}(X_i = w_i \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \epsilon_3$

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).

DEFINITION 2.3.3 *Let Y be the random variable obtained by taking the input a and letting $Y = Y(a)$, where:*

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

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

Claim 2.3.1 *If $Prob(Y = i \mid X_{[1,i-1]} = w_{[1,i-1]}) > 0$ then $H_\infty(X_{[i,n]} \mid Y = i \text{ and } X_{[1,i-1]} = w_{[1,i-1]}) \geq m_1$*

Therefore, for any such $w_{[1,i-1]}$, $(Q_i \mid Y = i \text{ and } X_{[1,i-1]} = w_{[1,i-1]})$ is ζ_1–close to random (since E_1 is an extractor). Hence by lemma 2.2.6, the distribution $(X_{[1,i-1]} \mid Y = i) \times (Q_i \mid Y = i \text{ and } X_{[1,i-1]} = w_{[1,i-1]})$ is ζ_1–close to the distribution $(X_{[1,i-1]} \mid Y = i) \times U$. But,

Claim 2.3.2 *$H_\infty(X_{[1,i-1]} \mid Y = i) \geq m_2$.*

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

□

Now we prove claims 2.3.1 and 2.3.2:

Proof: [of claim 2.3.1]

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

$$Prob(X_{[i,n]} = w_{[i,n]} | X_{[1,i-1]} = w_{[1,i-1]}, Y(x) = i) \quad \leq$$

$$\frac{Prob(\ X_{[i,n]}=w_{[i,n]} \ | \ X_{[1,i-1]}=w_{[1,i-1]} \ )}{Prob(Y(x)=i \ | \ X_{[1,i-1]}=w_{[1,i-1]} \ )} \quad \leq$$

$$\frac{(\epsilon_2-\epsilon_3)\cdot 2^{-m_1}}{Prob(Y(x)=i \ | \ X_{[1,i-1]}=w_{[1,i-1]} \ )} \quad \leq$$

$$\frac{(\epsilon_2-\epsilon_3)\cdot 2^{-m_1}}{\epsilon_2-\epsilon_3} = 2^{-m_1}$$

The first line is true since $Prob(A \mid B) \leq \frac{Prob(A)}{Prob(B)}$, the second line since $f(w) = i$, and the third follows from Claim A.3.1. $\square$

Proof: [of claim 2.3.2]

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

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

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

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

However,

- $Prob(X_{[i+1,n]} = w_{[i+1,n]} | X_{[1,i]}) \geq (\epsilon_2 - \epsilon_3) 2^{-m_1}$
- $Prob(X_i = w_i \mid X_{[1,i-1]} = w_{[1,i-1]}) \geq \epsilon_3$
- $Prob(X_{[1,n]} = w_{[1,n]}) \leq 2^{-(m_1+m_2+s)}$

The first line is true because $f(w) = i$, the second because $w \notin B_3$, and the third because $H_\infty(X) \geq m_1 + m_2 + s$. Thus,

$$Prob(X_{[1,i-1]} = w_{[1,i-1]}) \quad \leq \quad \frac{2^{-m_2-s}}{\epsilon_3 \ \cdot (\epsilon_2 - \epsilon_3)} \tag{2.4}$$

Therefore,

$$\begin{aligned} & Prob(X_{[1,i-1]} = w_{[1,i-1]} \mid Y(x) = i) && \leq \\ & \frac{Prob(\ X_{[1,i-1]} = w_{[1,i-1]})}{Prob(Y(x)=i)} && \leq \\ & \frac{2^{-m_2-s}}{\epsilon_3 \cdot (\epsilon_2-\epsilon_3) \cdot Prob(Y(x)=i)} && \leq \\ & \frac{2^{-m_2-s}}{\epsilon_3 \ \cdot (\epsilon_2-\epsilon_3) \cdot (\epsilon_1-\epsilon_2-\epsilon_3)} \ = \ 2^{-m_2} \end{aligned}$$

The first line is true because $Prob(A \mid B) \leq \frac{Prob(A)}{Prob(B)}$. The second follows from Eq. (2.4). The third follows from Claim (A.3.2). $\square$

2.3.3 Composing Many Extractors

Now we define composition of many extractors by:

Definition 2.3.4 *Suppose* $E_i : (n) \times (t_i) \mapsto (t_{i+1} + s_{i+1})$ *is an* (m_i, ζ_i)*–extractor, for* $i = 1, \ldots, k$, $s_i \geq 0$ *and* $s_2 = 0$. *Suppose* $M_i : (t_{i+2} + s_{i+2})^n \times (\mu_i) \mapsto (t_{i+2})$ *is a* $\bar{\zeta}_i$ *–somewhere random merger, for any* $i = 1 \ldots k-1$. *We define the function* $E_k \overset{M_{k-1}}{\odot} E_{k-1} \overset{M_{k-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1$ *by induction to equal* $E_k \overset{M_{k-1}}{\odot} (E_{k-1} \overset{M_{k-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1)$.

Theorem 2 *Suppose* E_i, M_i *are as above, then for any safety parameter* $s > 0$, $E = E_k \overset{M_{k-1}}{\odot} E_{k-1} \overset{M_{k-2}}{\odot} \ldots E_2 \overset{M_1}{\odot} E_1$, $E : (n) \times (t_1 + \Sigma_{i=1}^{k-1} \mu_i) \mapsto (t_{k+1})$ *is an* $(\Sigma_{i=1}^{k} m_i + (k-1)s, \Sigma_{i=1}^{k} \zeta_i + \Sigma_{i=1}^{k-1} \bar{\zeta}_i + (k-1)n2^{-s/3+3})$ *–extractor. If* E_i, M_i *are explicit, then so is* E.

Proof:

Correctness :

By induction on k. For $k = 2$ this follows from theorem 1. For larger k's this is a straight forward combination of the induction hypothesis and Theorem 1.

Running time :

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

1. Given $x \in \overline{X}$, choose y uniformly from $\{0,1\}^{t_1}$ and y_j uniformly from $\{0,1\}^{\mu_j}$, for $j = 1, \ldots, k$.

2. Next, we compute the matrix M where

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

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

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

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

$\square$

Remark 2.3.1 *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.*

2.3.4 Assuming Explicit Somewhere Random Mergers

Assume for every $m \geq \bar{m}$ we have a good somewhere random merger M. Then we can let $E = A_m \overset{M}{\odot} \ldots A_{b^2\bar{m}} \overset{M}{\odot} A_{b\bar{m}} \overset{M}{\odot} A_{\bar{m}}$, where A_i is the extractor of lemma 2.2.4 and b is some constant, $1 < b < c_{tiny}$, to get an extractor that extracts $\Omega(m)$ bits from sources having m min-entropy. Thus, good somewhere random mergers imply good extractors.

Lemma 2.3.3 *Suppose for any $\bar{m} \leq m \leq \bar{\bar{m}}$ there is an explicit ϵ somewhere random merger$M_m : (m)^n \times (t) \mapsto (\Delta \cdot m)$, where Δ is a constant and $\frac{1}{c_{tiny}} < \Delta < 1$. Then, for any $\bar{m} \leq m \leq \bar{\bar{m}}$ there is an explicit $(m, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (O(\bar{m} \cdot log(\frac{1}{\epsilon}) + log(n) \cdot t)) \mapsto (\Omega(m))$.*

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

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

- $E_i : (n) \times (m_i) \mapsto (c_{tiny} \cdot m_i)$ is the $(m_i, 2^{-m_i/5})$–extractor A_{m_i} from lemma 2.2.4
- $M_i : (c_{tiny} \cdot m_{i+1})^n \times (t) \mapsto (b \cdot m_{i+1})$ is the $\epsilon)$–somewhere random merger given in the hypothesis of the lemma.

Now we use Theorem 2 with $t_i = m_i$ and $s_i = (c_{tiny} - b)m_{i-1}$ (and therefore $t_i + s_i = c_{tiny} \cdot m_{i-1}$), and we also take $s = \frac{m}{2l}$. By Theorem 2, $E : (n) \times (t_1 + \Sigma_{i=1}^{l-1} t) \mapsto (t_{l+1})$ is an $(\Sigma_{i=1}^{l} m_i + (l-1)s, \Sigma_{i=1}^{l} 2^{-m_0/5} + \Sigma_{i=1}^{l-1} \epsilon + (l-1)n2^{-s/3+3})$ –extractor. Since $l = O(log(n))$ and $\epsilon \geq 2^{-s/3}$ (otherwise the result is trivial), E is an extractor as required. Since A_i, M_i are explicit, so is E. $\square$

Just to demonstrate the above, assume for every m there is an explicit $M_m : (m)^n \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Delta \cdot m)$ $poly(n) \cdot \epsilon$ somewhere random merger. Then notice that by lemma 2.3.3, this implies an explicit $(m, poly(n) \cdot \epsilon)$–extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(m))$, for any m.

2.3.5 Explicit Somewhere Random Mergers

In this section we construct explicit somewhere random mergers. We observe that a 2–block merger can be obtained from the previously designed extractors of [NZ93, SZ94]. Once such a merger is obtained, any number of blocks can be merged in a binary-tree fashion.

A 2-block somewhere random merger

A d–block (m, ϵ, η)–somewhere random source X, can be viewed intuitively as a source composed of d strings of length m, with a selector function that, for all but an η fraction of the inputs, can find a block that is ϵ quasi-random. Indeed, by lemma 2.3.2 we know that $\overline{X}$ is $\epsilon + \eta$ close to a distribution with m min-entropy.

Thus, any (m, ϵ)–extractor $E : (2m) \times (t) \mapsto (m')$, extracts randomness from any source X with $H_\infty(\overline{X}) \geq m$, and in particular it extracts randomness from any $(m, 0, 0)$ somewhere random source. Therefore, by definition, any such E : $(m)^2 \times (t) \mapsto (m')$ is an ϵ–somewhere random merger.

Corollary 2.3.4 *Any (m, ϵ)–extractor $E : (2m) \times (t) \mapsto (m')$ (which can also be viewed as $E : (m)^2 \times (t) \mapsto (m')$) is an ϵ–somewhere random merger.*

A d–block somewhere random merger

Given a d–block somewhere random source, we merge the blocks in pairs in a tree like 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 2.3.2 *Assume we can build an* $\epsilon(m)$ *somewhere random merger* $E : (m)^2 \times (t(m)) \mapsto (m - k(m))$. *We build* $M_l : (m)^{2^l} \times (l \cdot t(m)) \mapsto (m - l \cdot k(m))$, *by induction on* l:

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

Output : *Let* $t = t_1 \circ \ldots t_l$, *where* t_j *is chosen uniformly from* $\{0,1\}^{t(m)}$. *If* $l = 0$ *output* x^l, *otherwise:*

1. *Let* $x^{l-1}_i = E(x^l_{2i-1} \circ x^l_{2i} \ , \ t_l)$, *for* $i = 1, \ldots, 2^{l-1}$.
2. *Let the output be* $M_{l-1}(x^{l-1}_1 \circ \ldots x^{l-1}_{2^{l-1}} \ , \ t_1 \circ \ldots t_{l-1})$.

Theorem 3 *Assume for every* m, $E_m : (m)^2 \times (t(m)) \mapsto (m - k(m))$ *is an explicit* $\epsilon(m)$ *somewhere random merger, for some monotone functions* t, k *and* ϵ^{-1}. *Then* $M_l : (m)^{2^l} \times (\Sigma^l_{j=1} t(m)) \mapsto (m - l \cdot k(m))$ *is an explicit* $l \cdot \epsilon(m - l \cdot k(m))$ *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 t is uniform. Notice that $\overline{Z^l}$ is the distribution $\overline{X}$, and $\overline{Z^0}$ is the distribution of the output.

The theorem follows immediately from the following claim:

Claim 2.3.3 *Denote* $m_j = m - (l-j)k(m)$. *If* X *is an* $(m_l, 0, 0)$ *somewhere random source, then for any* $1 \leq i \leq 2^j$, $d(\ (Z^j_i \mid Y \in [2^{l-j}(i-1)+1, 2^{l-j}i]) \ , \ U_{m_j} \ ) \leq (l-j) \cdot \epsilon(m)$

$\square$

Proof:

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

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

In appendix A we prove:

Lemma 2.3.5 *Let A, B and Y be any random variables. Suppose that $d((A \mid Y \in S_1), U_m) \leq \epsilon$ and $d((B \mid Y \in S_2), U_m) \leq \epsilon$ 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 $H_\infty(\overline{X}) \geq m$.*

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 $(l-j) \cdot \epsilon(m_j)$ close to some $\overline{W}$ with $H_\infty(\overline{W}) \geq m_j$.
- Since $Z^{j-1}_i \ = \ E(Z^j_{2i-1} \circ Z^j_{2i} \ , \ t_j)$, 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 $E_{m_j}(x, t_k)$ where $x \in \overline{X}$ and $H_\infty(\overline{X}) \geq m_j$. Therefore, it is $(l-j) \cdot \epsilon(m_j) + \epsilon(m_j)$ close to random, as required.

□

Remark 2.3.2 *Notice that we use the same random string t_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).*

2.3.6 Putting It Together

What we need is good somewhere random mergers for any m. We are going to achieve this by building good 2-block somewhere random mergers and using Theorem 3. Thus we need extractors working on sources with very high min-entropy, and losing only a very small fraction of the min-entropy present in the random source. Fortunately, a simple idea due to Wigderson and Zuckerman [WZ93] suffices.

More of The Same

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

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 $M - m$ output bits. The idea is that as long as $|E(x, r_1) \circ \ldots \circ E(x, r_k)|$ is less then $M - m$, with high probability $(X \mid E(x, r_1) \circ \ldots \circ E(x, r_k))$ still contains m 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 A.4:

Lemma 2.3.6 *Suppose that for some m there is an explicit (m, ϵ)–extractor E_m : $(n) \times (t) \mapsto (m')$. Then, for any $M \geq m$, and any safety parameter $s > 0$, there is an explicit $(M, k(\epsilon + 2^{-s}))$–extractor $E : (n) \times (kt) \mapsto (min\{km', M - m - s\})$.*

Lemma 2.3.7 *Suppose that for any $m \geq \bar{m}$ there is an explicit $(m, \epsilon(n))$–extractor $E_m : (n) \times (t(n)) \mapsto (\frac{m}{f(n)})$. Then, for any m, there is an explicit $(m, log(n)(\epsilon + 2^{-t(n)}))$–extractor $E : (n) \times (O(f(n)log(n)t(n))) \mapsto (m - \bar{m})$.*

Corollary 2.3.8 *Suppose $\bar{m} = \bar{m}(n)$ is a function s.t. for every $m \geq \bar{m}(n)$ there is an explicit ϵ somewhere random merger $M : (m)^n \times (t) \mapsto (\Delta \cdot m)$, where $\frac{1}{c_{tiny}} < \Delta < 1$. Then for any m there is an explicit $(m, poly(n) \cdot \epsilon)$– $E : (n) \times (O(\bar{m} \cdot log(n) \cdot log(\frac{1}{\epsilon}) + log^2(n) \cdot t)) \mapsto (m)$.*

Proof: The lemma follows from lemma 2.3.3 using lemma 2.3.7. $\square$

Mergers That Do Not Lose Much.

The [SZ94] extractor of lemma 2.2.10 works for any source with $H_\infty(\overline{X}) \geq n^{1/2+\gamma}$. Thus, using lemma 2.3.6 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 3 we get a good n–merger. Thus:

Lemma 2.3.9 *Let $b > 1$ be a constant and suppose $f = f(m) = f(m(n))$ is a function s.t. $f(m) \leq \sqrt[3]{m}$ and for every $m \geq m_0(n) \quad : \quad f(m) \geq b \cdot log(n)$. Then for every $m \geq m_0$ there is an explicit $log(n) \cdot poly(m) \cdot \epsilon$ somewhere random mergerM $: (m)^n \times (log(n) \cdot polylog(m) \cdot f^2(m) \cdot log(\frac{1}{\epsilon})) \mapsto (m - \frac{m}{b})$.*

Proof:

- By lemma 2.2.10 there is an explicit $(\frac{m}{f(m)}, \epsilon)$ extractor $E_1 : (m) \times (O(log^2 m \cdot log(\frac{1}{\epsilon}))) \mapsto (\frac{m}{f^2(m)})$. extractor.
- By lemma 2.3.6 there is an explicit $(m, poly(m) \cdot \epsilon)$ extractor $E_2 : (2m) \times (O(f^2(m) \cdot log^2 m \cdot log(\frac{1}{\epsilon}))) \mapsto (m - \frac{m}{f(m)})$.

- By Theorem 3 there is an explicit $log(n) \cdot poly(m) \cdot \epsilon$ somewhere random merger$M_1 : (m)^n \times (O(log(n) \cdot polylog(m) \cdot f^2(m) \cdot log(\frac{1}{\epsilon})) \mapsto (m - log(n) \cdot \frac{m}{f(m)})$. Since $\frac{m}{f(m)} \leq \frac{m}{b \cdot log(n)}$ for any $m \geq m_0$, we have that $log(n) \cdot \frac{m}{f(m)} \leq \frac{m}{b}$.

□

Corollary 2.3.10 *For every $m \geq 2^{\sqrt{log(n)}}$, there is an ϵ somewhere random merger $M = M_m : (m)^n \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(m))$.*

Proof: Take $f(m) = log^d m$ for some constant $d > 2$. For any constant b, $m \geq 2^{\sqrt{log(n)}}$ and n large enough, $log^d m \geq b \cdot log(n)$, and the corollary follows lemma 2.3.9. □

Notice that Theorem 3 and corollary 2.3.10 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 2.2.10 does not apply.

Extractors That Work For High Min-Entropy

Corollary 2.3.10 asserts the existence of good mergers for $m \geq 2^{\sqrt{log(n)}}$, and therefore plugging this into corollary 2.3.8 we get:

Corollary 2.3.11 *For every m there is an $(m, poly(n) \cdot \epsilon)$ extractor $B_m : (n) \times (O(2^{\sqrt{log(n)}} \cdot polylog(n) \cdot log(\frac{1}{\epsilon}))) \mapsto (m)$.*

The extractor B in corollary 2.3.11 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 $O(2^{\sqrt{log(n)}} \cdot$

$polylog(n) \cdot log(\frac{1}{\epsilon}))$ 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 2.3.11, E_{sz} the extractor from lemma 2.2.10 and M is the merger from corollary 2.3.10, then $E = B \overset{M}{\odot} E_{sz}$ extracts $\Omega(m)$ bits from sources having $m \geq n^{2/3}$ min-entropy, using only $polylog(n)$ truly random bits! That is, we get the following lemma:

Lemma 2.3.12 *Let $\epsilon \geq 2^{-n^{\gamma}}$ for some constant $\gamma < 1$. There is some constant $\beta < 1$ s.t. for every $m \geq n^{\beta}$ there is an explicit $(m, poly(n) \cdot \epsilon)$ extractor $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (\Omega(m))$.*

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

- $E_{sz} : (n) \times (O(log^2 n \cdot log(\frac{1}{\epsilon}))) \mapsto (n^{2\beta-1})$ is the (n^{β}, ϵ) –extractor of lemma 2.2.10.
- B_m is the extractor from corollary 2.3.11.
- M is the merger from corollary 2.3.10.

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

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 2.3.7 and Theorem 3 we have good somewhere random mergers, for every m. Thus by corollary 2.3.8 we have good extractors for every m.

Theorem 4 *For every constant* $\gamma < 1$, $\epsilon \geq 2^{-n^\gamma}$, *and every* $m = m(n)$ *there is an explicit* (m, ϵ)*–extractor* $E : (n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (m)$.

Proof:

- By lemma 2.3.7, lemma 2.3.12 implies an explicit $(n, poly(n) \cdot \epsilon)$ extractor $E_1 : (2n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (n - n^\beta)$.
- There is some constant d (that depends only on γ) s.t. for every $log^d n \leq m \leq n$, $log(n) \cdot m^\beta \leq \frac{m}{\bar{c}}$, 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 3, for every m there is an explicit $poly(n) \cdot \epsilon$ somewhere random merger $M : (m)^n \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (m - \frac{m}{\bar{c}})$.
- By corollary2.3.8, this implies an explicit $(m, poly(n) \cdot \epsilon)$–extractor E_m : $(n) \times (polylog(n) \cdot log(\frac{1}{\epsilon})) \mapsto (m)$, for any m. Plugging $\epsilon' = \frac{\epsilon}{poly(n)}$, gives the theorem.

$\square$

2.4 An Extractor Using Less Truly Random Bits

In this section we build our second extractor.

Theorem 5 *For every constant k and $\gamma > 0$ there is some constant $\delta > 0$ and an $(n^\gamma, \frac{1}{n})$ extractor $D : (n) \times (O(log(n)log^{(k)}n)) \mapsto (\Omega(n^\delta))$, where $log^{(k)}n = \underbrace{loglog \ldots log}_{k}\ n$.*

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 $n^{1/2}$ min-entropy to achieve extractors for any n^γ min-entropy.

Lemma 2.4.1 *Let $f(n)$ be an arbitrary function. Assume $\forall \gamma > 0$, $\exists \delta > 0$ s.t. there is an $(m = n^\gamma, n^{-\Omega(1)})$ extractor $E : (n) \times (t = log(n) \cdot f(n)) \mapsto (m' = \Omega(n^\delta))$. Then $\forall \gamma' > 0$, $\exists \delta' > 0$ s.t. there is an $(n^{\frac{1}{2}+\gamma'}, \epsilon = n^{-\Omega(1)})$ extractor $F : (n) \times (t = O(log(n) \cdot log(f(n)))) \mapsto (m' = \Omega(n^{\delta'}))$.*

Lemma 2.4.2 *Assume*

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

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

Using these two lemmas we can prove Theorem 5:

Proof: [of Thm 5]

We prove the equivalent claim:

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

By induction on k. For $k = 1$ this follows from Theorem 4.

Assume for k. Denote $f_{k+1}(n) = log^{(k+1)}n$. The induction hypothesis says that $\forall \gamma > 0$, $\exists \delta > 0$ s.t. there is an $(m = n^\gamma, n^{-\Omega(1)})$ extractor $E : (n) \times (t = log(n) \cdot 2^{O(f_{k+1}(n))}) \mapsto (m' = \Omega(n^\delta))$. extractor.

By lemma 2.4.1, $\forall \gamma' > 0$, $\exists \delta' > 0$ s.t. there is an $(n^{1/2+\gamma'}, \epsilon = n^{-\Omega(1)})$ extractor $F : (n) \times (t = O(log(n) \cdot f_{k+1}(n))) \mapsto (m' = \Omega(n^{\delta'}))$.

All the requirements of lemma 2.4.2 are met, and therefore, using lemma 2.4.2 repeatedly a constant number of times, we get the desired extractor. $\square$

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

In this section we prove lemma 2.4.1. We show that combining the extractor of Theorem 4 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 $H_\infty(X) \geq n^{\frac{1}{2}+\gamma'}$, we can use the [NZ93] block extraction to extract $d = O(log(f(n)))$ 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 $log(n) \cdot 2^{\Omega(d)} = log(n) \cdot f(n)$ 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 2.4.1]

Consider the following algorithm:

ALGORITHM 2.4.1 *Fix* $d = O(log(f(n)))$, $l = n^{1/2}$.
Choose $y_1, \ldots, y_d \in \{0,1\}^{O(log(n))}$, *and* $y \in \{0,1\}^{log(n)}$.
Given $x \in X$:

1. *Extract* d *blocks* $b_1 = BC(x, y_1), \ldots, b_d = BC(x, y_d)$, *where* BC *is the block extraction operator of lemma 2.2.8.*
2. *compute* $z = BE(b_2 \ldots b_d, y)$, *where* BE *is the function extracting randomness from block-wise sources, from lemma 2.2.7.*
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 $y_1, \ldots, y_d$ arbitrarily. The probability that there exists an $1 \leq i \leq d$, 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 $d \cdot l << n^{1/2} \cdot log(n)$. 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:

$$Pr(X = x \mid B_1 = b_1, \ldots, B_i = b_i) \quad \leq \quad \frac{Pr(X = x)}{Pr(B_1 = b_1, \ldots, B_i = b_i)}$$

$$
\begin{aligned}
&\leq \frac{2^{-n^{1/2+\gamma}}}{2^{-n^{1/2+\gamma/2}}} \\
&<< 2^{-n^{1/2+\gamma/2}}
\end{aligned}
$$

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. $\square$

Therefore, using the block extractor of lemma 2.2.8 we get:

Claim: $B = B_1 \circ \ldots \circ B_d$ is a $(d, n^{\gamma/3}, n^{-\Omega(1)})$ block-wise source.

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 2.2.8, 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. $\square$

Finally,

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

Proof: For most prefixes b_1, $(B_2 \times \ldots \times B_d \mid B_1 = b_1)$ is a block-wise source. Therefore, by lemma 2.2.7, $(Z \mid B_1 = b_1)$ is close to uniform. Hence, the claim follows by lemma 2.2.6. $\square$

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. $\square$

2.4.2 An Extractor For n^γ Min-entropy.

Here we prove lemma 2.4.2. We show how given an extractor for sources with $n^{\frac{1}{k}+\gamma}$ min-entropy, we can build an extractor for sources with $n^{\frac{1}{k+1}+\gamma}$ min-entropy. The idea is to extract d 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 2.4.2 *Fix* $d = O(f(n))$ *and* $l = n^{1-\frac{1}{k+1}}$.
Choose $r_1, \ldots, r_d$ *uniformly from* $\{0,1\}^{O(log(n))}$, r', r''' *uniformly from* $\{0,1\}^{O(log(n)\cdot f(n))}$ *and* r'' *uniformly from* $\{0,1\}^{O(log(n)}$.

Compute:

- $b_i = BC(x, r_i)$ *for* $i = 1, .., d$, *where* BC *is the block extraction operator of lemma 2.2.8.*
- $b'_i = F(b_i, r')$ *for* $i = 1, .., d-1$, *where* F *is given in the hypothesis of the lemma.*
- $b'_d = E(b_1, BE(b_2, \ldots, b_d, r''))$.

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

Proof: [of lemma 2.4.2]:

Let us denote $B = (B_1, \ldots, B_d)$ and $B' = (B'_1, \ldots, B'_d)$ (where B_i (B'_i) is the random variable with the value b_i (b'_i)). We will soon prove that:

Claim 2.4.1 B' *is a* $(d, m = n^{\Omega(1)}, n^{-\Omega(1)})$ *somewhere random source.*

Hence, by lemma 2.3.2, B' is $n^{-\Omega(1)}$-close to some B'' that is distributed over $N = md$ variables and has $H_\infty(B'') \geq m$. Thus, B'' is very "condensed", i.e.

$H_\infty(B'') \geq m >> N^{2/3}$. Thus, by the hypothesis, $F(B', r''')$ is $n^{-\Omega(1)}$-close to the uniform distribution. □

Proof: [of claim 2.4.1]

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 $f(b) = d$.

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

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

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

Definition 2.4.1 *b is bad in either of the following two cases:*

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

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

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 2.4.3 $Pr(Y = 0) \leq d\epsilon + \Sigma_i \epsilon_i$

Second, we will show that

Claim 2.4.2 *For any* $1 \leq i \leq d-1$: $\quad H_\infty(B_i \mid Y = i) \geq n^{\frac{1}{k+1} + \frac{\gamma}{2}}$.

Therefore, $H_\infty(B_i | Y = i) \geq n^{\frac{k}{k+1}\frac{1}{k} + \frac{k}{k+1}\frac{k+1}{k}\frac{\gamma}{2}} = l^{\frac{1}{k} + \frac{k+1}{k}\frac{\gamma}{2}}$, 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 2.4.3 $(B \; | Y = d)$ *is a* $(d, n^{\gamma/2}, n^{-\Omega(1)})$ *block-wise source.*

Therefore, with the conditioning that $Y = d$, $\overline{B_1 \circ BE(B_1 \circ \ldots \circ B_d, r'')}$ is $n^{-\Omega(1)}$ close to $\overline{B_1} \times U_{log(n) \cdot 2^{\Omega(d)}}$. Hence, using the extractor E, $(B'_d \mid Y = d)$ is $n^{-\Omega(1)}$ close to uniform.

Putting it together, B' is a $(d, n^{\Omega(1)}, n^{-\Omega(1)})$ 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 A.5:

Claim 2.4.4 *For any* $0 < i < d$:

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 $Y(b) = d$:

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

We prove claim 2.4.3 in appendix A.5. Let us now prove claim 2.4.2:

Proof: [of claim 2.4.2]

Let $1 \leq i \leq d-1$. 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.

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

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

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

Therefore:

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

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

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

Also, by claim 2.4.4, $Prob(Y = i \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon$.

Therefore,

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

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

Finally, let us prove claim 2.4.3:

Proof: [of claim 2.4.3]

Fix an $i \in [1..d]$. We need to show that for any prefix $b_{[1,i-1]}$, $(B_i \mid Y = d\ and\ B_{[1,i-1]} = b_{[1,i-1]})$ 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 $Y(b_0) = d$. Since $Y(b_0) = d$, no block so far "stole" too much entropy, i.e., if we denote $\overline{Z} = (X \mid B_{[1,i-1]} = b_{[1,i-1]}\ and\ Y = d)$, then:

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

i.e.

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

Also, by claim 2.4.4, $Prob(Y = d \mid B_{[1,i-1]} = b_{[1,i-1]}) \geq \epsilon$.

Therefore,

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

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

$\square$

2.5 Applications

Extractors have many applications in computer science (see [Nis96] for a survey). Here we list only those applications that benefited from our new constructions. Most of the results are achieved by plugging in our new extractor instead of the previous ones.

2.5.1 a-Expanding Graphs

Definition 2.5.1 *[Pip87] An undirected graph is a–expanding if any two disjoint sets of vertices of size at least a are joined by an edge.*

The obvious lower bound on the degree of an a–expanding graph is $\frac{N-a}{a}$. The previous upper bound was $O(\frac{N}{a} \cdot 2^{log(n)^{1/2+o(1)}})$[WZ93, SZ94]. In [WZ93], Wigderson and Zuckerman suggest a simple construction of a–expanding graphs which they improve using a recursive construction. Using the extractor we developed in section 2.3 we can use their simple construction and get:

Corollary 2.5.1 *(following [WZ93]) For every N and $1 \leq a \leq N$, there is an efficiently constructible a–expanding graph with N vertices, and maximum degree $O(\frac{N}{a} 2^{polyloglog(N)})$*

For completeness, we give the proof:

Proof: (based on [WZ93]) Let V be a set with $N = 2^n$ vertices. We use an $(k, \frac{1}{6})$–extractor $F : (n) \times (t) \mapsto (k)$, with $t = poly(log(n))$, and denote $K = 2^k$,$T = 2^t$.

- First build a bipartite graph $G' = (V', W', E')$ where $V' = V = \{0,1\}^n$, $W' = \{0,1\}^k$ and $(x, y) \in E' \iff \exists r \in \{0,1\}^t \; s.t. \; y = F(x, r)$.

Notice that in G' any two subsets of V' of size $K = 2^k$ have a common neighbor in W'.

- Denote $BAD = \{w \in W \mid deg(w) > 6d_{avg}\}$ where $d_{avg} = \frac{N \cdot T}{K}$ is the average degree of vertices in W. It is clear that $|BAD| \leq \frac{1}{6}|K|$.

- From G' build $G = (V, E)$ as follows: $V = V'$ and $(v_1, v_2) \in E$ iff v_1, v_2 have a common neighbor in $W \setminus BAD$.

Call an $X \subseteq V$ "big" if $|X| \geq K$. Since F is a disperser, for any big X, $|\Gamma(X)| \geq \frac{3}{4} \cdot K$. Since $|BAD| \leq \frac{1}{6} \cdot K$ we have that $|\Gamma(X) \cap (W \setminus Bad)| \geq (\frac{3}{4} - \frac{1}{6}) \cdot K > \frac{1}{2} \cdot K$. It follows that for every two big sets X and Y, $\Gamma(X) \cap \Gamma(Y) \cap (W \setminus Bad) \neq \emptyset$. Therefore, there is an edge going from X to Y in G, and therefore G is K–expanding. The maximal degree of a vertex in G is at most $T \cdot 6d_{avg} = O(\frac{N}{K} \cdot T^2)$.

$\square$

[Pip87, WZ93] showed that constructing good explicit a–expanding graphs has applications to other problems. Plugging in our new extractor we get:

Corollary 2.5.2 *(following [Pip87], see [WZ93] lemma 5) 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 2.5.3 *(following [AKSS89], see [WZ93] lemma 6) 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.*

In both cases our new construction replaces the term $2^{log^\delta n}$ with the term $2^{polyloglog(n)}$.

2.5.2 Superconcentrators of Small Depth

Definition 2.5.2 *$G = ((A, C, B), E)$ is a superconcentrator if G is a layered graph with input vertices A, output vertices B, and for any sets $X \subseteq A$,$Y \subseteq B$ of size k, there are at least k vertex-disjoint paths from X to Y.*

Much research was done on finding small explicit superconcentrators of small depth (see [WZ93] for references). Again, using our new extractor with the ideas used in previous constructions we manage to reduce the size of a depth 2 superconcentrator from $O(N \cdot 2^{log(N)^{1/2+o(1)}})$ to $O(N \cdot 2^{polyloglog(N)})$.

Lemma 2.5.4 *(following [WZ93]) For every N there is an efficiently constructible depth 2 superconcentrator over N vertices with size $O(N \cdot 2^{polyloglog(N)})$.*

Proof: (following the simple idea in [WZ93])

We are going to use the following lemma:

Lemma 2.5.5 *[Mes84] $G = ((A, C, B), E)$ is a superconcentrator of depth 2 iff for any $1 \leq k \leq n$ and any sets $X \subseteq A$, $Y \subseteq B$ of size k, $|\Gamma(X) \cap \Gamma(Y)| \geq k$.*

We are going to use an $(m, \frac{1}{3})$ extractor $E_m : (n) \times (t) \mapsto (m + 3)$, with $t = polylog(n, log(\frac{1}{\epsilon}))$. We build the superconcentrator as follows:

Input and output layers: The input and output layers A and C are of sizes $N = 2^n$. We identify each input/output vertex with a string in $\{0, 1\}^n$.

The middle layer: We let the middle layer B be the union of n disjoint sets $B_1, \ldots, B_n$, $|B_m| = 4 \cdot 2^{m+1}$. Again, we describe each vertex in B_m as a string in $\{0, 1\}^{m+3}$.

Edges going from A to B: For every $x \in A = \{0,1\}^n$, $1 \leq m \leq n$ and $r \in \{0,1\}^t$ we add an edge going from x to $E_m(x,r) \in \{0,1\}^{m+3} = B_m$.

Edges going from C to B: These are the mirror images of the edges going from A to B.

Claim 2.5.1 *For any $X \subseteq A$ of size $2^m \leq k \leq 2^{m+1}$, $|\Gamma(X) \cap B_m| \geq \frac{2}{3}|B_m|$.*

Proof: Consider the uniform distribution $\overline{D}$ over X. Clearly, $H_\infty(\overline{D}) \geq m$. Hence, $d(E(\overline{D}, U_t), U_{m+3}) \leq \frac{1}{3}$. However, $|\Gamma(X) \cap B_m| < \frac{2}{3}|B_m|$ implies that $d(E(\overline{D}, U_t), U_{m+3}) > \frac{1}{3}$ - a contradiction. □

Therefore, for any $X \subseteq A$ and $Y \subseteq C$ of size $2^m \leq k \leq 2^{m+1}$, $|\Gamma(X) \cap \Gamma(Y) \cap B_m| \geq \frac{1}{3}|B_m| \geq k$. Hence by lemma 2.5.5, our graph is a superconcentrator. □

[WZ93] showed how to convert a small depth 2 superconcentrator, to a linear-size superconcentrator with small depth. Plugging in the above result into their lemma, we achieve a linear-size superconcentrator of $polyloglog(N)$ depth. Notice that the best previous linear construction had $O(log(n)^{1/2+o(1)})$ depth, so we achieved an exponential improvement.

Corollary 2.5.6 *(Following [WZ93], lemma 10) For every N there is an explicitly constructible superconcentrator over N vertices, with linear size and $polyloglog(N)$ depth.*

2.5.3 Deterministic Amplification

Our goal now is to convert a BPP algorithm that uses n random bits and has $\frac{1}{2} - \frac{1}{n}$ error, into one that errs with probability at most 2^{-k}. We want to achieve this using as few random bits as possible.

This problem, known as the "deterministic amplification" problem, was extensively studied by [KPS85, CG89, IZ89, CW89] and many others. Using expanders, this can be done using only $n + O(k)$ random bits [AKS87, IZ89, CW89]. Sipser [Sip88] noted that the existence of explicit extractors imply stronger amplification.

Theorem: *Assume there is an* $(m = n, \epsilon = \frac{1}{n})$ *extractor* $E : (n + k) \times (t) \mapsto (n)$*. If* L *is accepted by a* $BPTime(f(n))$ *algorithm using* n *random bits and having* $\frac{1}{2} - \frac{1}{n}$ *error, then* L *is also accepted by a* $BPTime(f(n) \cdot 2^t)$ *algorithm using* $n + k$ *random bits and having* $\frac{1}{2^k}$ *error.*

Proof:

New Algorithm : Choose randomly $x \in A = \{0,1\}^{n+k}$. Denote

$$\Gamma(\{x\}) = \{z \in \{0,1\}^n \mid \exists_{y \in \{0,1\}^t} \;\; z = E(x,y)\}$$

For any $z \in \Gamma(\{x\})$ run M with z as the random string, and decide according to the majority of the results.

Correctness : Denote $W \subseteq \{0,1\}^n$ the set of witnesses leading to the wrong decision. Call an $x \in A$ "bad", if most of its neighbors lie in W. We arrive at the wrong result iff we choose some bad x, however since E is an extractor, $Pr(x \text{ is bad}) \leq \frac{2^n}{2^{n+k}} = 2^{-k}$.

$\square$

Notice that the extractor of section 2.3 gives such a strong amplification but in quasi-polynomial time.

Corollary 2.5.7 *if* L *is accepted by a* BPP *algorithm using* n *random bits and having* $\frac{1}{2} - \frac{1}{n}$ *error, then* L *is also accepted by a* $\widetilde{BPP}$ *algorithm using* $n + k$ *random bits and having* $\frac{1}{2^k}$ *error.*

2.5.4 The Hardness of Approximating The Iterated Log of Max Clique.

Zuckerman [Zuc93] uses extractors to show the hardness of approximating any iterated log of MAX-Clique. In his constructions Zuckerman uses a non-explicit $(r, 1/2)$ extractor $E : (R) \times (t = log(R+2)) \mapsto (r)$ [3] that can be found by choosing a random bipartite graph with the right degree. Notice how close t is to the lower bound. If we could explicitly find such extractors, we could replace the random classes in Zuckerman's result with deterministic classes, and in particular this would have shown that approximating log(MAX-Clique) to within some constant factor is NP–hard. Unfortunately, we have explicit constructions only for $t = polylog(R)$.

Zuckerman achieves the hardness result by amplifying the [ALM+92] PCP proof system for NP, and using the $FGLSS$ reduction from SAT to $MAX - Clique$. In the following we write down the PCP amplification we achieve using the extractor of section 2.3, and the hardness result we get. We do not describe what a PCP proof system is, and how the [FGL+91] reduction works. The interested read is referred to [FGL+91, BGLR93, Zuc93].

Theorem: *[ALM+92, AS92]* $NP \subseteq PCP(r = O(log(n)), m = O(1), a = O(1), \epsilon = \frac{1}{2})$.

The amplification process we use is exactly as the one for amplifying an RP algorithm with a good extractor. We get:

Lemma 2.5.8 *[Zuc91] If there is an $(r, \frac{1}{2})$–extractor $F : (r+l) \times (t) \mapsto (r)$, then $PCP(r, m, a, \frac{1}{2}) \subseteq PCP(r+l, 2^t m, a, 2^{-l})$.*

[3]Actually, it is enough to use a disperser, and extractors can be replaced with dispersers throughout all of this subsection. However, since we did not define what a disperser is, we use extractors.

Thus:

Corollary 2.5.9 *For any $l \geq 0$, $NP \subseteq PCP(r = O(log(n)) + l, m = 2^{polylog(l,r)}, a = O(1), \epsilon = 2^{-l})$.*

Now we are going to Plug in this result into Zuckerman's construction.

DEFINITION 2.5.1 *We denote $\underbrace{loglog \ldots log}_{k} n$ by $log^{(k)}(n)$. For an integer e we define $P_{e,k}(n)$ by:*

$$P_{e,k}(n) = \left.2^{2^{\cdot^{\cdot^{elog^{(k)}n}}}}\right\} k\ 2's$$

Notice that $P_{e,1} = n^e$ is polynomial, $P_{e,2} = 2^{log^e n}$ is quasi-polynomial, and in general $P_{e,k}(n)$ is more than quasi-polynomial, but only "quasi more".

DEFINITION 2.5.2 *We denote the size of the largest clique in G by $\omega = \omega(G)$.*

Corollary 2.5.10 *(following [Zuc93]) Let $k \geq 3$ be a constant. If for any constant b approximating $log^{(k)}\omega$ to within a factor of b is in $\bigcup_e DTime(P_{e,k}(n))$, then $\bigcup_e NTime(P_{e,k}(n)) = \bigcup_e DTime(P_{e,k}(n))$.*

The full proof of this corollary appears in appendix A.6.

2.5.5 Simulating BPP Using Weak Random Sources

In section 1.5.1 we saw that any polynomial time, black-box simulation of RP or BPP, must use a source $\overline{X}$ with $H_\infty(\overline{X}) \geq n^\gamma$ for some $\gamma > 0$. We mentioned that Srinivasan, Saks and Zhou [SSZ95] showed a polynomial time, black-box simulation of RP using any random source X having $H_\infty(\overline{X}) \geq n^\gamma$, and Srinivasan and Zuckerman [SZ94] showed an $n^{O(log(n))}$ time, black-box simulation of BPP using any random source X having $H_\infty(\overline{X}) \geq n^\gamma$, for $\gamma > 1/2$.

These constructions use the following simple lemma

Lemma 2.5.11 *If for any $\delta > 0$ there is some $\eta > 0$ and an $(n^\delta, \frac{1}{6})$ extractor $E : (n) \times (t) \mapsto (n^\eta)$, then for any $\delta > 0$, BPP can be simulated in time poly$(n, 2^t)$ using any source X with $H_\infty(\overline{X}) \geq n^\delta$.*

Proof:

Black-Box Simulation :

1. Ask for an $n^{\frac{1}{\eta}}$-bit string x from the random source X.
2. For all $y \in \{0,1\}^t$, let $z = E(x, y)$, and find the original algorithm's answer when z is its random string.
3. Answer according to the majority of the answers.

Correctness :

Since $H_\infty(X) \geq n^\delta$, if we uniformly choose y from $\{0,1\}^t$, then the distribution Z (over n bits) is quasi-uniform, and the black box simulation answers don't differ much than those of the original algorithm running with *truly random bits*. Finally, instead of choosing y uniformly, we check all possible values of y, and answer according to the majority.

$\square$

Plugging in the extractor of section 2.4 into the above lemma we get an almost polynomial time, black box simulation of BPP:

Corollary 2.5.12 *For any* $\delta > 0$ *and* $k > 0$, *BPP can be simulated in time* $n^{O(log^{(k)} n)}$ *using a weak random source* X *with min entropy at least* n^{δ}, *where* $log^{(k)} n = \underbrace{loglog \ldots log}_{k}\ n$.

Chapter 3

Non-deterministic Symmetric LogSpace

In the next section we show our proof that $SL = coSL$. We develop and use a new technique for showing closure under complement. The results of this chapter are joint work with my advisor Noam Nisan, and were published in [NTS95, NT95].

3.1 An Informal Solution

We want to find a many-one *LogSpace* reduction from the undirected s,t *non-connectivity* problem, to *USTCON*, the undirected s,t connectivity problem. I.e, given (G,s,t) we want to build (in *LogSpace*) another undirected problem (G',s',t') s.t. s is *not* connected to t in G iff s' is connected to t' in G'.

For the time being let us consider an easier problem: given (G,s,t) we want to build $(G_1,s_1,t_1),\ldots,(G_m,s_m,t_m)$ s.t. there is some *monotone* function

$f : \{0,1\}^m \mapsto \{0,1\}$ with the property that s is *not* connected to t in G iff $f(USTCON(G_1, s_1, t_1), \ldots, USTCON(G_1, s_m, t_m)) = 1$.

Notice, that if we do not require that f is monotone the problem is trivial - just let $G_1 = G,\ s_1 = s,\ t_1 = t$ and let $f : \{0,1\} \mapsto \{0,1\}$ be the negation function. The whole essence of the problem is to express the negation operator in a "monotone" way, or, more precisely, to confine all the non-monotone operations to the *LogSpace* construction of the graphs G_i.

So, let us consider the above problem. One thing we can do, is to choose all vertices that have a bigger neighbor (which can clearly be done by a function of the required form). This, in a sense, isolates one vertex from each connected component, and thus upper bounds the number of connected components of the graph. This is also very much the same like taking the transitive closure of the graph, which can be easily done if we can solve *USTCON*.

The other direction is the crux of the construction: We "count" the size of any spanning forest of G. The counting is done using the simple (and well known) observation that an edge $e = (i,j)$ *does not* belong to the lexicographically first spanning forest iff i is connected to j in the graph containing only the edges that appear (in the input) before e. This gives a lower bound to the number of connected components using the easy property that the number of connected components of G plus the number of edges in a spanning forest of G is exactly the number of vertices of G.

The whole essence of the algorithm is, therefore, that we can express the "non-monotone" property of the size of the spanning forest, by a non-monotone reduction to a monotone connectivity problem.

Finally, to finish the construction

we need to translate $f(USTCON(G_1, s_1, t_1), \ldots, USTCON(G_1, s_m, t_m))$ to a single undirected s, t connectivity problem. To do that we show that in our case, not only f is monotone but also has a small (polynomial in the input length) representation as a monotone formulae. I.e., f is composed of (not too many) "AND" and "OR" operations. By showing how to take care of these two basic operations, we show that, indeed, we can translate $f(USTCON(G_1, s_1, t_1), \ldots, USTCON(G_1, s_m, t_m))$ to a single undirected s, t connectivity problem, and the proof is completed.

3.2 SL=coSL

We design a many-one reduction from co*USTCON* to *USTCON*. We start by developing, in subsection 3.2.1, simple tools for combining reductions. In particular these tools will allow us to use the AKS sorting networks in order to "count". At this point, the main ingredient of the reduction will be the calculation of the number of connected components in a graph. An upper bound to this number is easily obtained using transitive closure, while the main idea of the proof is to obtain a lower bound by computing a spanning forest of the graph, which is done in subsection 3.2.2. In subsection 3.2.3 everything is put together.

3.2.1 Projections to $USTCON$.

We will use only the simplest kind of reductions, i.e. *LogSpace* uniform projection reductions [SV85]. Moreover, we will only be interested in reductions to *USTCON*. In this subsection we define this kind of reduction and we show some of its basic properties.

NOTATION 3.2.1 *Given* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *denote by* $f_n : \{0,1\}^n \mapsto \{0,1\}^*$ *the restriction of* f *to inputs of length* n. *Denote by* $f_{n,k}$ *the* k*'th bit function of* f_n, *i.e. if* $f_n : \{0,1\}^n \mapsto \{0,1\}^{k(n)}$ *then* $f_n = (f_{n,1}, \ldots, f_{n,k(n)})$.

NOTATION 3.2.2 *We represent an* n*-node undirected graph* G *using* $\binom{n}{2}$ *variables* $\vec{x} = \{x_{i,j}\}_{1 \le i < j \le n}$ *s.t.* $x_{i,j}$ *is* 1 *iff* $(i,j) \in E(G)$. *If* $f(\vec{x})$ *operates on graphs , we will write* $f(G)$ *meaning that the input to* f *is a binary vector of length* $\binom{n}{2}$ *representing* G.

We say that $f : \{0,1\}^* \mapsto \{0,1\}^*$ reduces to $USTCON(m)$ if we can (uniformly and in *LogSpace*) label the edges of a graph of size m with $\{0, 1, x_i, \neg x_i\}_{1 \le i \le n}$, s.t. $f_{n,k}(\bar{x}) = 1 \iff$ there is a path from 1 to m in the corresponding graph. Formally,

Definition 3.2.1 *We say that* $f : \{0,1\}^* \mapsto \{0,1\}^*$ *reduces to* $USTCON(m)$, $m = m(n)$, *if there is a uniform family of Space*$(log(n))$ *functions* $\{\sigma_{n,k}\}$ *s.t. for all* n *and* k:

- $\sigma_{n,k}$ *is a projection, i.e.:* $\sigma_{n,k}$ *is a mapping from* $\{i,j\}_{1 \le i < j \le m}$ *to* $\{0, 1, x_i, \neg x_i\}_{1 \le i \le n}$
- *Given* $\vec{x}$ *define* $G_{\vec{x},k}$ *to be the graph* $G_{\vec{x},k} = (\{1, \ldots, m\}, E)$ *where* $E = \{(i,j) \mid \sigma_{n,k}(i,j) = 1$ *or* $\sigma_{n,k}(i,j) = x_i$ *and* $x_i = 1$ *or* $\sigma_{n,k}(i,j) = \neg x_i$ *and* $x_i = 0\}$.
- $f_{n,k}(\vec{x}) = 1 \iff$ *there is a path from* 1 *to* m *in* $G_{\vec{x},k}$.

If σ *is restricted to the set* $\{0, 1, x_i\}_{1 \le i \le n}$ *we say that* f monotonically *reduces to* $USTCON(m)$.

Lemma 3.2.1 *If f has uniform monotone formulae of size $s(n)$ then f is monotonically reducible to USTCON$(O(s(n)))$.*

Proof: Given a formula ϕ recursively build (G, s, t) as follows:

- If $\phi = x_i$ then build a graph with two vertices s and t, and one edge between them labeled with x_i.
- If $\phi = \phi_1 \wedge \phi_2$, and (G_i, s_i, t_i) the graphs for ϕ_i, $i = 1, 2$, then identify s_2 with t_1 and define $s = s_1, t = t_2$.
- If $\phi = \phi_1 \vee \phi_2$, and (G_i, s_i, t_i) the graphs for ϕ_i, $i = 1, 2$, then identify s_1 with t_1 and s_2 with t_2 and define $s = s_1 = t_1$ and $t = s_2 = t_2$.

$\square$

DEFINITION 3.2.1 *Sort : $\{0,1\}^n \mapsto \{0,1\}^n$ is the boolean sorting function, i.e. it moves all the zeroes to the beginning of the string.*

Using the AKS sorting networks [AKS83], which belong to NC^1 , we get:

Corollary 3.2.2 *Sort is monotonically reducible to USTCON(poly).*

Lemma 3.2.3 *If f monotonically reduces to USTCON(m_1) and g reduces to USTCON(m_2) then $f \circ g$ reduces to USTCON$(m_1^2 \cdot m_2)$, where $\circ$ is the standard function composition operator.*

Proof: The function f monotonically reduces to a graph with m_1 vertices, where each edge is labeled with one of $\{0, 1, x_i\}$. In the composition $f \circ g$, each x_i is replaced by $x_i = g_i(\vec{y})$ which can be reduced to a connectivity problem of size m_2. Replace each edge labeled x_i with its corresponding connectivity problem. There can be m_1^2 edges, each replaced by a graph with m_2 vertices, hence the new graph has $m_1^2 \cdot m_2$ vertices. $\square$

3.2.2 Finding a Spanning Forest.

In this section we show how to build a spanning forest using *USTCON*. This basic idea was already noticed by Reif and independently by Cook [Rei82].

Given a graph G index the edges from 1 to m. We can view the indices as weights for the edges, and as no two edges have the same weight, we know that there is a unique minimal spanning forest F. In our case, where the edges are indexed, this minimal forest is the lexicographically first spanning forest.

It is well known that the greedy algorithm finds a minimal spanning forest. Let us recall how the greedy algorithm works in our case. The algorithm builds a spanning forest F which is initially empty $F = \emptyset$. Then the algorithm checks the edges one by one according to their order, and for each edge e, if e does not close a cycle in F then e is added to the forest, i.e. $F = F \cup \{e\}$.

At first glance the algorithm looks sequential, however, claim 3.2.2 shows that the greedy algorithm is actually highly parallel. Moreover, all we need to check that an edge does not participate in the forest, is one st connectivity problem over an easily obtainable graph.

DEFINITION 3.2.2 *For an undirected graph G, denote by $LFF(G)$ the lexicographically first spanning forest of G. Let*

$SF(G) \mapsto \{0,1\}^{\binom{n}{2}}$ *be:*

$$SF_{i,j}(G) = \begin{cases} 0 & (i,j) \in LFF(G) \\ 1 & otherwise \end{cases}$$

Lemma 3.2.4 *SF reduces to USTCON(poly)*

Proof: Let F be the lexicographically first spanning forest of G. For $e \in E$ define G_e to be the subgraph of G containing only the edges $\{e' \in E \mid index(e') < index(e)\}$.

Claim: $e = (i,j) \in F \iff e \in E$ *and* i is not connected to j in G_e.

Proof: Let $e = (i,j) \in E$. Denote by F_e the forest which the greedy algorithm built when it was checking e. So $e \in F \iff e$ does not close a cycle in F_e.

($\Longrightarrow$) $e \in F$ and therefore e does not close a cycle in F_e, but then e does not close a cycle in the transitive closure of F_e, and in particular e does not close a cycle in G_e.

($\Longleftarrow$) e does not close a cycle in G_e therefore e does not close a cycle in F_e and $e \in F$. $\square$

Therefore $SF_{i,j}(G) = \neg x_{i,j} \vee i$ is connected to j in $G_{(i,j)}$.

Since $\neg x_{i,j}$ can be viewed as the connectivity problem over the graph with two vertices and one edge labeled $\neg x_{i,j}$, it follows from lemmas 3.2.1 and 3.2.3 that SF reduces to $USTCON$. Notice, however, that the reduction is not monotone.

$\square$

3.2.3 Putting It Together.

First, we want to construct a function that takes one representative from each connected component. We define $LI_i(G)$ to be 0 iff the vertex i has the largest index in its connected component.

DEFINITION 3.2.3 $LI(G) \mapsto \{0,1\}^n$

$$LI_i(G) = \begin{cases} 0 & \textit{i has the largest index} \\ & \textit{in its connected component} \\ 1 & \textit{otherwise} \end{cases}$$

Lemma 3.2.5 *LI reduces to USTCON(poly)*

Proof:

$LI_i(G) = \bigvee_{j=i+1}^{n} \quad (i \text{ is connected to } j \text{ in } G)$.

So LI is a simple monotone formula over connectivity problems, and by lemmas 3.2.1 and 3.2.3, LI reduces to $USTCON$. This is, actually, a monotone reduction.

□

Using the spanning forest and the LI function we can compute the number of connected components of G exactly, i.e.: given G we can compute a function NCC_i which is 1 iff there are exactly i connected components in G.

DEFINITION 3.2.4 $NCC(G) \mapsto \{0,1\}^n$

$$NCC_i(G) = \begin{cases} 1 & \textit{there are exactly } i \\ & \textit{connected components} \\ & \textit{in } G \\ 0 & \textit{otherwise} \end{cases}$$

Lemma 3.2.6 *NCC reduces to USTCON(poly)*

Proof:

Let F be a spanning forest of G. It is easy to see that if G has k connected components then $|F| = n - k$.

Define:

$$f(G) = Sort \circ LI(G)$$
$$g(G) = Sort \circ SF(G).$$

Then:

$$f_i(G) = 1 \implies k < i$$
$$g_i(G) = 1 \implies n - k < i \implies k > n - i.$$

and thus: $NCC_i(G) = f_{i+1}(G) \wedge g_{n-i+1}(G)$

Therefore applying lemmas 3.2.1, 3.2.2, 3.2.3, 3.2.4, 3.2.5 proves the lemma.

□

Finally we can reduce the non-connectivity problem to the connectivity problem, thus proving that $SL = coSL$.

Lemma 3.2.7 *coUSTCON reduces to USTCON(poly)*

Proof:

Given (G, s, t) define G^+ to be the graph $G \cup \{(s, t)\}$.

Denote by $\#CC(H)$ the number of connected components in the undirected graph H.

$$s \text{ is not connected to } t \text{ in } G \iff$$

$$\# \, CC(G^+) = \# \, CC(G) - 1 \iff$$

$$\bigvee_{i=2,\ldots,n} \quad NCC_i(G) \wedge NCC_{i-1}(G^+).$$

Therefore applying lemmas 3.2.1, 3.2.3, 3.2.6 proves the lemma. □

3.3 Extensions

Denote by $L^{<SL>}$ the class of languages accepted by *Logspace* oracle Turing machines with an oracle from SL. An oracle Turing machine has a work tape and a write-only query tape (with unlimited length) which is initialized after every query. We get:

Corollary 3.3.1 $L^{<SL>} = SL$.

Proof:

Let *Lang* be a language in $L^{<SL>}$ computed by an oracle Turing machine M running in $L^{<SL>}$, and fix an input $\vec{x}$ to M.

We build the "configuration" graph $G(V,E)$ of M, by:

- Let V contain all possible configurations.
- $(v,w) \in E$ with the label "q is (not) s–t connected", if starting from configuration v the next query is q, and after the oracle answers that "q is (not) connected" the machine moves to configuration w.

Notice that we can ignore the direction of the edges, as backward edges do not help us. The reason is that from any vertex v, there is only one forward edge leaving v that can be traversed (i.e. whose label matches the oracle's answer). Therefore if we reach v using a "backward edge" $w \mapsto v$, then the only forward edge leaving v that can be traversed is $v \mapsto w$.

Now we can replace query edges labeled "q is connected" with the s–t connectivity problem q, and edges labeled "q is not connected" with the s–t connectivity problem obtained using our theorem that $SL = coSL$, resulting in one, not too big, s–t connectivity problem. It is also clear that this can be done in *LogSpace*, completing the proof.

$\square$

As the symmetric Logspace hierarchy defined in [Rei82] is known to be within $L^{<SL>}$, this hierarchy collapses to SL.

As can easily be seen, the above argument holds for any undirected graph with undirected query edges, which is exactly the definition of $SL^{<SL>}$ given by [BPS92]. Thus, $SL^{<SL>} = SL$, and by induction the SL hierarchy defined in [BPS92] collapses to SL.

Bibliography

[AES92] N. Alon, P. Erdős, and J. H. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992. This book describes the Probabilistic Method as developed by Paul Erdős and its applications in Discrete Mathematics and Theoretical Computer Science.

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

[AKL+79] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal sequences and the complexity of maze problems. In *Proceedings of the 20th Annual IEEE Symposium on the Foundations of Computer Science*, 1979.

[AKS83] M. Ajtai, J. Komlos, and E. Szemeredi. An $O(n \log n)$ sorting network. In *Proc. 15th ACM Symposium on Theory of Computing (STOC)*, pages 1–9, 1983.

[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.

[Arm] R. Armony. Private Communication.

[AS92] 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.

[BCD+89] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BFNW93] Babai, Fortnow, Nisan, and Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3, 1993.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, ACM, pages 294–304, 1993.

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982.

[BPS92] Y. Ben-Asher, D. Peleg, and A. Schuster. The complexity of reconfiguring networks models. In *Proc. of the Israel Symposium on the Theory of Computing and Systems*, May 1992. To appear Information and Computation.

[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.

[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.

[HILL91] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Construction of a pseudo-random generator from any one-way function. Technical Report TR-91-068, International Computer Science Institute, Berkeley, CA, December 1991.

[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.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17, 1988.

[INW94] Impagliazzo, Nisan, and Wigderson. Pseudorandomness for network algorithms. In *ACM Symposium on Theory of Computing (STOC)*, 1994.

[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.

[KPS85] R. Karp, N. Pippernger, and M. Sipser. A time randomness tradeoff. In *AMS Conference on Probabilistic Computational Complexity*, 1985.

[Lev87] Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7, 1987.

[LP82] Lewis and Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19, 1982.

[LPS86] Lubotzky, Phillips, and Sarnak. Explicit expanders and the ramanujan conjectures. In *ACM Symposium on Theory of Computing (STOC)*, 1986.

[Mar75] G. A. Margulis. Explicit construction of concentrators. *Problems of Information Transmission*, 1975.

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

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[Nis92] N. Nisan. RL $\subseteq$ SC. In *Proc. 24th ACM Symposium on Theory of Computing (STOC)*, pages 619–623, 1992.

[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.

[NT95] N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. In *Chicago Journal of Theoretical Computer Science*, 1995.

[NTS95] Noam Nisan and Amnon Ta-Shma. Symmetric Logspace is closed under complement. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 140–146, Las Vegas, Nevada, 29 May–1 June 1995.

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. In *Proc. 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 2–11, 1988.

[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.

[Pip87] N. Pippenger. Sorting and selecting in rounds. *SIAM Journal on Computing*, 16:1032–1038, 1987.

[Raz91] A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Proceedings of the 8th FCT, Lecture Notes in Computer Science,* 529, pages 47–60, New York/Berlin, 1991. Springer-Verlag.

[Rei82] J. H. Reif. Symmetric complementation. In *Proc. 14th ACM Symposium on Theory of Computing (STOC)*, pages 201–214, 1982.

[Ren70] A. Renyi. *Probability Theory.* North-Holland,Amsterdam, 1970.

[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.

[SV85] Skyum and Valiant. A complexity theory based on boolean algebra. *Journal of the ACM*, 1985.

[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.

[Sze88] Szelepcsenyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26, 1988.

[TS96] Amnon Ta-Shma. On extracting randomness from weak random sources (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 276–285, Philadelphia, Pennsylvania, 22–24 May 1996.

[Wig94] Wigderson. The amazing power of pairwise independence. In *ACM Symposium on Theory of Computing (STOC)*, 1994.

[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.

[Yao82] A.C. Yao. Theory and application of trapdoor functions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1982.

[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.

[Zuc96] D. Zuckerman. Randomness-optimal sampling, extractors, and constructive leader election. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing*, ACM, 1996.

Appendix A

Explicit Extractors

A.1 A Somewhere Random Source Has Large Min-Entropy

Lemma A.1.1 *If $X = X_1 \circ \ldots \circ X_d$ is an (m, ϵ, η) somewhere random source, then X is η–close to an $(m, \epsilon, 0)$ somewhere random source X'.*

Proof: [of lemma A.1.1]

Let Y be an (m, ϵ, η) selector for X. Denote $p = Prob(Y = 0) \leq \eta$. Define the distribution $\overline{D}$ by:

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

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

$d(\overline{X},\overline{X'}) \leq d(\overline{Y \circ X},\overline{Y' \circ X'}) = p \leq \eta$.

Now we want to show that Y' is an $(m,\epsilon,0)$ selector for X'. It is clear that $Prob(Y'=0)=0$. It is not hard to see that for any $i>0$ we have: $Prob(X' = x \mid Y'=i) \;=\; Prob(X=x \mid Y=i)$.

Therefore, since we know that $(X_i|Y=i)$ is ϵ–close to U_m, we also know that $(X'_i|Y'=i)$ is ϵ close to U_m, thus completing the proof.

$\square$

Lemma A.1.2 *Let $X = X_1 \circ \ldots \circ X_d$ be an $(m,\epsilon,0)$–somewhere random source, then X is ϵ close to an $(m,0,0)$–somewhere random source Z.*

Proof: [of lemma A.1.2]

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

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

It is easy to check that $Z^{(i)}$ is indeed a distribution, and that $Z^{(i)}_i = U_m$. 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:

$$Prob(Z_i = z_i \mid Y=i) \;=\; Z^{(i)}_i(z_i) \;=\; 2^{-m}$$

We will soon prove that:

Claim A.1.1 $d(X^{(i)}, Z^{(i)}) \leq \epsilon$.

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.1.1]

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 $x \in A$: $Pr(Z = x \mid Z_i = a_i \text{ and } Y = i) = \frac{Pr(Z=x \mid Y=i)}{Pr(Z_i=a_i \mid Y=i)} = Pr(X = x \mid X_i = a_i \text{ and } Y = i)$.

□

Lemma A.1.3 *Let $X = X_1 \circ \ldots \circ X_d$ be an $(m, 0, 0)$ somewhere random source, then $H_\infty(\overline{X}) \geq m$.*

Proof: Suppose Y is an $(m, 0, 0)$ selector for X.

$$Prob(X = x) =$$

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

$$\Sigma_{i\in[1..d]}\ Prob(Y = i) \cdot 2^{-m} = 2^{-m}$$

$\square$

Combining lemmas A.1.1, A.1.2 and lemma A.1.3 we get lemma 2.3.2.

A.2 A Lemma For d–Block Mergers

We prove lemma 2.3.5:

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 A.2.1 $(A' \mid Z' = 1) = (A \mid Y \in S_1)$ *and* $(B' \mid Z' = 2) = (B \mid Y \in S_2)$.

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

Therefore by lemma 2.3.2, $\overline{A' \circ B'}$ is ϵ–close to some $\overline{X}$ with $H_\infty(\overline{X}) \geq m$. 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 $H_\infty(\overline{X}) \geq m$, thus completing the proof. □

A.3 Lemmas For Composing Two Extractors

In this section we prove some easy technical lemmas used in section 2.3.2.

Claim A.3.1 *For any i and any $w_{[1,i-1]}$, if $Prob_{x \in X}(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) > 0$, then $Prob_{x \in X}(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) \geq \epsilon_2 - \epsilon_3$.*

Proof:

Since $w_{[1,i-1]}$ can be extended to some w with $Y(w) = i \neq 0$, by definition 2.3.2:

$$Prob(f(x) = i) \geq \epsilon_1 \text{ , } and$$

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

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,

$$Prob(Y(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) =$$

$$Prob(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) - Prob(f(x) = i \text{ } and \text{ } x \in B \mid x_{[1,i-1]} = w_{[1,i-1]}) =$$

$$Prob(f(x) = i \mid x_{[1,i-1]} = w_{[1,i-1]}) - Prob(f(x) = i \text{ } and \text{ } x \in B_3 \mid x_{[1,i-1]} = w_{[1,i-1]}) \geq$$

$$\epsilon_2 - \epsilon_3$$

The last inequality uses claim A.3.3.

$\square$

Claim A.3.2 *For any i, if $Prob_{x \in X}(Y(x) = i) > 0$, then $Prob_{x \in X}(Y(x) = i) \geq \epsilon_1 - \epsilon_2 - \epsilon_3$.*

Proof:

Since there is some w' s.t. $Y(w') = i \neq 0$, by definition 2.3.2:

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

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

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

The last inequality uses claim A.3.3.

$\square$

Claim A.3.3

1. *For any i: $Prob(f(x) = i \; and \; x \in B_2) \leq \epsilon_2$*

2. *For any i and $w_{[1,i-1]}$: $Prob(f(x) = i \text{ and } x \in B_3 \mid x_{[1,i-1]} = w_{[1,i-1]}) \leq \epsilon_3$*

3. *For any i: $Prob(f(x) = i \text{ and } x \in B_3) \leq \epsilon_3$*

4. *$Prob(x \in B_i) \leq n\epsilon_i$, for $i = 1, 2, 3$.*

Proof:

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

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

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

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

$\square$

A.4 More Bits Using The Same Extractor

In this section we prove lemmas 2.3.6 and 2.3.7.

Proof: [Of lemma 2.3.6]

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 A.4.1 *We say that $a_{[1,i]}$ is "s-tiny" if $Prob(A_{[1,i]} = a_{[1,i]}) \leq 2^{-l_i - s}$*

Claim: For any $1 \leq i \leq k$, $Prob(a_{[1,i]}\ is\ \text{s}-tiny) \leq 2^{-s}$.

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}. $\Box$

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

Proof: For any x,

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

$\Box$

Claim: If $l_{i-1} \leq M - m - s$, 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 $l_i \leq M - m - s$, then for any prefix $a_{[1,i]}$ that is not s–tiny, $H_\infty(X \mid A_{[1,i]} = a_{[1,i]}) \geq M - l_i - s \geq m$. 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 2.2.6, $\overline{A_{[1,i+1]}}$ 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. $\Box$

Therefore, if we take k s.t. $l_k \leq M - m - s$, we invest kt random bits, and we get km' bits that are $k(2^{-s} + \epsilon)$ quasi-random, as required.

$\Box$

Proof: [of lemma 2.3.7]

Define $E(x, r_1 \circ \ldots \circ r_k) = E_{m_1}(x, r_1) \circ \ldots \circ E_{m_k}(x, r_k)$, where $s = t(n)$, $l_0 = 0$, $m_i = m - l_{i-1} - s$, and $l_i = l_{i-1} + \frac{m_i}{f(n)}$. Denote by A_i the random variable $E_{m_i}(X, R_i)$, and let $A_{[1,i]} = A_1 \circ \ldots A_i$. Intuitively, $l_i = |A_{[1,i]}|$, and m_i is the amount of min-entropy left in $(X \mid A_{[1,i]} = a_{[1,i]})$ with the safety parameter $s = t(n)$.

Claim: If $m_i \geq \bar{m}$ 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, $H_\infty(X \mid A_{[1,i]} = a_{[1,i]}) \geq m - l_i - s = m_{i+1} \geq \bar{m}$. 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 2.2.6, $\overline{A_{[1,i+1]}}$ 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. $\Box$

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

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

$\square$

A.5 Lemmas For The Second Extractor

In this section we prove some easy technical lemmas used in section 2.4. Let us start with the proof of claim 2.4.4:

Proof: [of claim 2.4.4]

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) :

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

The last inequality is from claim A.5.1.

$\square$

Now we state our last lemma, from which claim 2.4.3 also easily follows. First we give a definition:

DEFINITION A.5.1 *For b s.t. $f(b) = d$ and $Y(b) = 0$ define $YF(b)$ to be the first $i \in [1, d]$ s.t. $Prob(f = d \mid B_{[1,i-1]} = b_{[1,i-1]}) \leq \epsilon_i$, i.e., $YF(b)$ indicates the reason why b is bad.*

Claim A.5.1

1. *For any $1 \leq i \leq d-1$ 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 $Y(b) = d$:*

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

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

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

Proof: [of claim A.5.1]

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

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

Now,

$$Pr_b(f = d \ \wedge \ YF = j \mid B_{[1,i-1]} = b_{[1,i-1]}) =$$
$$\sum_{b_{[i,j-1]}} Pr(B_{[i,j-1]} = b_{[i,j-1]} | B_{[1,i-1]} = b_{[1,i-1]}) \cdot Pr_b(f = d \wedge YF = j \mid B_{[1,j-1]} = b_{[1,j-1]}) \leq$$
$$\sum_{b_{[i,j]}} Pr(B_{[i,j-1]} = b_{[i,j-1]} \mid B_{[1,i-1]} = b_{[1,i-1]}) \cdot \epsilon_j \leq \epsilon_j$$

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

Therefore,

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

The last inequality is by (2).

□

A.6 The Hardness of Approximating The Iterated Log of Max Clique.

The proof is based on a result by [AS92, ALM+92] characterizing NP as the set all languages having a "small" PCP proof system, and a result by [FGL+91]showing how to translate this into hardness of approximating MAX-Clique.

First, we describe the [FGL+91] result concerning the hardness of approximating MAX-Clique:

Fact A.6.1 *[FGL+91] Given a language $L \in PCP(r, m, a, \epsilon)$ and some input $x \in \{0,1\}^n$, we can easily build a graph with 2^{r+ma} vertices s.t.:*

$$\omega(G) = \begin{cases} 2^r & \text{if } x \in L \\ \epsilon \cdot 2^r & \text{if } x \notin L \end{cases}$$

Now we prove corollary 2.5.10:

Proof: [of corollary 2.5.10]

Given a language $L \in NP$ (actually, we should start with $L \in Time(P_{e,k}(n))$, but for simplicity we prove for $L \in NP$) and an input $x \in \{0,1\}^n$:

Take l s.t. $log^{(k-1)}l = 2 \cdot log^{(k)}n$. Use fact A.6.1 to translate the PCP system of lemma 2.5.8 into a graph G. We know that:

$$\omega(G) = \begin{cases} 2^r & \text{if } x \in L \\ \epsilon 2^r = poly(n) & \text{if } x \notin L \end{cases}$$

and that the size of G is $|G| = 2^{O(m)}$.

Notice that:

$$log^{(k)}(\omega(G)) = \begin{cases} log^{(k)}(2^r) \geq log^{(k)}(2^l) = log^{(k-1)}(l) = 2log^{(k)}(n) & \text{if } x \in L \\ log^{(k)}(poly(n)) \leq log^{(k)}(n) + O(1) & \text{if } x \notin L \end{cases}$$

Thus, if we can approximate $log^{(k)}(\omega(G))$ to within $\frac{1}{2}$, we can solve L in $DTime(2^{O(m)})$. However, $2^{O(m)} = 2^{2^{polylog(l,r)}} = 2^{2^{2^{cloglog(l)}}} \leq Pc, k(n))$ for some constant c. $\square$