Explicit constructions of pseudo-random generators and extractors

Thesis submitted for the degree
Doctor of Philosophy

by
Ronen Shaltiel

Submitted to the Senate of the Hebrew University in Jerusalem

October 2001

Explicit constructions of pseudo-random generators and extractors

Thesis submitted for the degree
Doctor of Philosophy

by
Ronen Shaltiel

Submitted to the Senate of the Hebrew University in Jerusalem

October 2001

This work was carried out under the supervision of

Professor Avi Wigderson

Acknowledgements

It is a great pleasure to thank the many people who helped me during my studies.

Above all, I thank my advisor Avi Wigderson. It was Avi who introduced me to complexity theory as an undergraduate and made me choose this research area. I am grateful to Avi for his patience, and for sharing with me so much of his knowledge. I wish I could remember everything he taught me. Working with Avi was a pleasure and an enriching experience. Yet, even more than his professional guidance, I am grateful to Avi for his personal advice and friendship. I feel that I wouldn't have completed my studies if it wasn't for Avi's constant encouragement.

I am also grateful to Omer Reingold. In addition to collaborating with me in some joint research, Omer has also been a very good friend. His patience and support helped me to get over some rough times I was experiencing. I particularly want to thank Omer for his cheerfulness and for being a good listener.

The Hebrew University has been great place to study. During my training as a computer scientist I had many excellent teachers. I particularly want to thank Michael Ben-Or, Nati Linial and Noam Nisan for introducing me to many interesting areas in the theory of computer science and for maintaining a great atmosphere in the theory group.

It was a pleasure to come to the "theory lab" and spend time with the theory students. I enjoyed the discussions we had on theory, basketball and life in general. I wish all the best to Dorit Aharonov, Roi Armoni, Ziv Bar-Yossef, Eli Ben-Sasson, Yonatan Bilu, Ido Bregman, Danny Gutfreund, Tamir Hazan, Shlomo Hoory, Boaz Leslau, Avner Magen, Ahuva Mu'alem, Dana Pe'er, Eyal Rozenman, Amir Shpilka and Amnon Ta-Shma.

During my stay at the Institute for Advanced Study, I enjoyed discussing research while eating cookies in the company of Misha Alekhnovich, Nicola Galessi, Valentine Kabanets, Satya Lokam, Dieter van-Melkebeek, Ran Raz, Alex Samorodnitsky, Venkatesh Srinivasan, Benny Sudakov and Salil Vadhan. I also want to thank all the participants of the "special complexity year".

I had the pleasure of collaborating with Russell Impagliazzo and with Chris Umans. These collaborations led to some of the results presented in this thesis.

A special thanks goes to Oded Goldreich who invited me to do my Postdoc with him at the Weizmann Institute.

Last but not least, I thank my wife Tami for her love and support. Thank you for putting up with my doubts and indecision. I realize it wasn't easy, and I love you very much.

Abstract

In this thesis we present explicit constructions of fundamental objects in the theory of computational randomness: pseudo-random generators and extractors. The motivation is to optimize the two main parameters in these objects which are the *seed length* and *output length*, and additionally to obtain simple and direct constructions. This thesis includes three different techniques:

- The first (based on joint work with Russell Impagliazzo and Avi Wigderson) builds on a new analysis of the Nisan-Wigderson generator [NW94]. This analysis exploits both successes and failures of the generator and enables us to use a shorter seed length. As a result we obtain the first construction of pseudo-random generators which uses the asymptotically optimal seed length for all "hardness levels", and the first extractor construction which uses the asymptotically optimal seed length for all "entropy levels". Previous constructions [IW97, STV99, Tre99] only achieved the asymptotically optimal seed length for the "high levels".

- The second (based on joint work with Omer Reingold and Avi Wigderson) introduces a new primitive called a *condenser*. We construct this object using (more efficient versions of) methods from [NZ96] and using ideas which resemble [SZ99, NTS99]. As a result (via repeated condensing) we obtain extractor constructions which optimize any of the two main parameters (seed length and output length) up to constant factors while paying only a *polylogarithmic* penalty in the other. Previous constructions [TS96, RRV99b, ISW00] payed a *polynomial* penalty when optimizing one of the parameters.

- The third (based on joint work with Chris Umans) builds on a new and simple method for constructing extractors given in [TSZS01]. We modify the construction and improve the proof technique to extend the construction to ”low entropy levels” and to significantly improve the parameters. As a result we obtain a simple and direct extractor construction which matches the parameters in some previous best constructions while stripping the complicated compositions and recursions which characterized previous work. Our improved construction and proof technique also allows us to use the method of [TSZS01] for constructing pseudo-random generators. As a result we obtain a simple and self contained pseudo-random generator which is not based on the Nisan-Wigderson construction. The parameters of this pseudo-random generator match those of [IW97, STV99] and as a result we obtain a new proof for the Impagliazzo-Wigderson theorem [IW97]. The same technique also yields an *optimal* hitting set generator. Both constructions produce generators for nondeterministic circuits when given a function with nondeterministic hardness. This improves and extends previous constructions by [AK97, KvM99, MV99].

Additional results contained in this thesis are a generic conversion of (regular) extractors into strong extractors, and a way to derandomize BPP given access to *many* ”candidate generators” where one of them is pseudo-random.

Contents

Chapter 1

Introduction

1.1 Organization of this thesis

The introduction chapter begins with some motivation and high level overview to pseudo-random generators and extractors (section 1.2). The following sections in the introduction give a technical introduction to the area. They give precise definitions of pseudo-random generators and extractors, and discuss some of the applications and connections between these two objects.

The three following chapters contain the main constructions of this thesis. They are based on journal versions to the relevant papers [ISW99, ISW00, RSW00, SU01], and are self contained given the formal definitions in the introduction chapter. Each one of these chapters starts with an "introduction section" which puts the presented construction in context and formally states the results. This section is followed by an "overview of the technique" section which attempts to give a detailed high level overview of the construction at a semi-intuitive, semi-technical level. I hope that this section will simplify the reader's task when reading the formal proof. This style of presentation causes quite a bit of repetition throughout this thesis. However, I feel that this may help the reader to directly access the parts he is specifically interested in.

1.2 Background

1.2.1 Probabilistic computation

A concept which revolutionized computer science in the few last decades is that of probabilistic computation. A machine which performs such a computation is allowed to use a string of independent coin tosses in addition to its usual input.

The machine is required to perform some task (which is a function of the input) and is allowed to err with small probability (over the choice of the random string). It is remarkable that adding the random input helps at all as it is independent of the input and thus, of the required output. Nevertheless, in some scenarios it was shown that probabilistic computation of some tasks can be done much more efficiently than any such deterministic computation. These scenarios include communication complexity, decision tree complexity [SW86], Byzantine agreement [BO83, Rab83] and many more. The reader is referred to [MR95] for a comprehensive treatment of probabilistic computation. In other scenarios (most notably in Cryptography) randomization allows concepts which cannot exist in a deterministic model. For example the concept of a zero knowledge proof [GMR82] makes no sense if the protocol is to be conducted by deterministic players.

In a very informal level, having access to random bits helps by providing the machine with some unpredictable information which was not fixed at the time it was designed. This is intuitively helpful in interactive scenarios. It also makes it tougher for an "opponent" to choose the worst input for the machine as the way the machine works on an a given input is not completely determined.

The usefulness of randomness leads us to consider randomness as one of the resources used by computation. It turns out that this resource is expensive and not easy to obtain. Quite a bit of research focused on trying to answer the following question.

The practical question: When we design probabilistic computations we assume that we have at our disposal a sequence of independent coin tosses. Can we physically achieve this premise?

1.2.2 Probabilistic polynomial time algorithms

A particular case of probabilistic computation is probabilistic algorithms. Here the machine is a polynomial time algorithm. It receives in addition to its input a polynomially long string of independent coin tosses. It is required to compute some boolean function of the input and is allowed to err with small probability. We differentiate between two notions of error which lead to different complexity classes:

- The class BPP contains languages which can be recognized by a probabilistic polynomial time algorithm with *two sided error*. That is, $L \in$ BPP if there is a poly-time algorithm $A(x, y)$ where $|y| = p(|x|)$ for some fixed polynomial p, and for every x, $\Pr_y[A(x, y) \neq 1_L(x)] \leq 1/3$. Here $1_L(x)$ denotes the characteristic function of L.

- The class RP contains languages which can be recognized by a probabilistic polynomial time algorithm with *one sided error*. That is, $L \in$ RP if there is a poly-time algorithm $A(x, y)$ where $|y| = p(|x|)$ for some fixed polynomial p, and for every x, if $x \in L$ then $\Pr_y[A(x, y) = 0] \leq 1/3$ and if $x \notin L$ then $\Pr_y[A(x, y) = 1] = 0$.

Note that a one sided error algorithm never errs when $x \notin L$. The choice of $1/3$ in the above definitions is not important and could be replaced by any function $e(|x|)$ such that $2^{-|x|^{O(1)}} \leq e(|x|) \leq 1/2 - |x|^{-O(1)}$.

At this moment there are functions for which we know a (polynomial time one sided error) probabilistic algorithm yet all known deterministic algorithms are extremely inefficient and take exponential time. For other functions we have probabilistic algorithms which are much easier to implement and allow parallelization whereas deterministic ones are complicated and sequential. (More details can be found for example in [MR95]). However, it may be that we still haven't found efficient and simple deterministic algorithms for these problems. This is a central question of complexity theory.

The philosophical question: What is the power of probabilistic algorithms? Can any probabilistic algorithm be efficiently simulated deterministically?

It will be best if the simulation takes polynomial time. However, we will also be interested in subexponential time simulations. It is trivial to get an exponential time simulation. This follows because it is possible to simulate a probabilistic algorithm which uses r random bits with slowdown 2^r by running the probabilistic algorithm on all 2^r strings of length r and taking the majority vote. This fact suggests a common strategy for "derandomizing" probabilistic algorithms which works in two phases: The first phase is to show that any probabilistic algorithm can be simulated using fewer random bits. The second phase is to simulate the algorithm of phase one deterministically with exponential slowdown. Note that if the first phase uses only logarithmically many random bits the entire simulation takes polynomial time.

1.2.3 Generating pseudo-randomness

We want to reduce the number of random bits used by probabilistic algorithms. The idea is to use a deterministic procedure, called a *pseudo-random generator*, to stretch few random bits (called the *seed*) into the appropriate length. The distribution produced should "look random" to all efficient algorithms. (We elaborate on the concept of pseudo-randomness in section 1.4. The reader is also referred to the excellent monograph [Gol98] devoted to this field and its varied connections to complexity theory, cryptography and learning theory). We can use generators to generate the random input of a probabilistic algorithm from very few truly random bits. It follows that the probabilistic algorithm behaves the same way when given the output of the generator as when given a random string and in particular recognizes the same language. Thus, with a pseudo-random generator we can simulate any probabilistic algorithm with fewer truly random bits. By running the generator over all seeds as described above we can simulate any probabilistic algorithm deterministically. Note that the total running time of this simulation is exponential in the seed length, it is thus crucial to reduce this parameter.

At this time, it is not known how to explicitly construct pseudo-random generators. The main difficulty is that a consequence of such a construction is the existence of ”hard functions” which is the main open problem of complexity theory. The hardness versus randomness paradigm (introduced by [BM84, Yao82]) attempts to construct them assuming the existence of ”hard functions”. This direction resulted in a rich theory. One of the achievements of this line of research is a construction of a pseudo-random generator with logarithmic seed length given a sufficiently hard function [NW94, IW97].

1.2.4 Refining randomness

One possible answer to the ”practical question” raised above is that we can run probabilistic computation by using randomness from nature. Indeed, it is widely believed that there are distributions one can sample from in nature and obtain some randomness. (Most of these involve particle reactions). However, it is very unlikely that sampling from such distributions produces truly random bits (that is, independent coin tosses). A paradigm suggested by [SV86, Blu84] is to refine the imperfect randomness in nature into true randomness. The idea is to use procedures, called *extractors* (defined in [NZ96]), to extract (almost) true randomness from sources of imperfect randomness using a small number of additional truly random bits (called the *seed*). Though extractors were defined with the above motivation in mind, they turned out to have many varied uses in complexity, combinatorics and network design. We elaborate on extractors in section 1.5. The reader is also referred to the excellent survey papers [Nis96, NTS99]. It can be shown that the need to use a seed of additional randomness is inevitable. Still, using an extractor probabilistic computation can be carried out using the randomness in nature and an additional very small number of random bits. Thus, extractors lead to substantial savings in the costly resource of truly random bits.

In the particular case of probabilistic algorithms we saw how to simulate the effect of few truly random bits deterministically by enumerating all possibilities.

Using this procedure extractors enable us to simulate any probabilistic algorithm efficiently using only randomness from nature. Once again, the time of the simulation is exponential in the seed length and it is crucial to reduce this parameter.

To conclude this brief background, pseudo-random generators and extractors are fundamental tools in the attempt to "derandomize" probabilistic algorithms and a substantial effort has been made in the last decade to explicitly construct these objects with as small as possible seed length and as large as possible output length. There are also some striking connections between these two areas. Trevisan, [Tre99] observed that constructions of pseudo-random generators with certain properties yield extractors, and extractors turn out to be the key component in unconditional constructions of pseudo-random generators which fool algorithms with small memory, [Nis96, NZ96, INW94, RR99].

This thesis presents several methods of constructing pseudo-random generators and extractors with gradual improvements in the important parameters. The exact results are presented in sections 1.4.8,1.5.8, and summarized in tables 1.2,1.4. These results were obtained in collaboration with Russell Impagliazzo and Avi Wigderson [ISW99, ISW00], Omer Reingold and Avi Wigderson [RSW00] and Chris Umans [SU01].

1.3 Preliminaries

In the following sections we give precise definitions of pseudo-random generators and extractors. However, we do assume that the reader has some basic knowledge in complexity theory. Precise definitions of algorithms, probabilistic algorithms, boolean circuits and relevant complexity classes can be found in [Pap94]. We will use the following notation throughout the thesis. For a function f, $S(f)$ denotes the size of the smallest circuit which computes it. We use $f = \{f_l\}$ to denote a sequence of functions where f_l is on l bits. We use $x \in_D T$ to denote sampling x from the set T according to the distribution D. The uniform distribution on m bit strings is denoted by U_m. For a function f on T which is the domain of a distribution D, $f(D)$ denotes the distribution obtained by sampling $x \in_D T$ and evaluating $f(x)$. If D is the uniform distribution over the domain of f we omit D and use f or $f(\cdot)$ to also denote the distribution $f(D)$. The function $\log$ denotes the logarithm with base 2.

1.4 Pseudo-random generators

1.4.1 The concept of pseudo-randomness

One of the achievements of complexity theory is a fresh view on the notion of distance between probability distributions. In statistics a distribution is considered close to the uniform distribution if its probability function is close in some sense (the most common is the L1-norm distance) to the probability function of the uniform distribution. The notion of pseudo-randomness suggests that ”randomness is in the eyes of the beholder” and that a distribution is close to uniform *relative to some class of observers*, if no observer from that class can distinguish between samples form this distribution and from the uniform distribution. This motivates the following definition.

Definition 1.1 (pseudo-randomness). *Let A be a boolean predicate on m bit strings. We say that a distribution D on m bit strings* ϵ-fools *A if*

$$|\Pr_{x \in_{U_m} \{0,1\}^m}[A(x)] - \Pr_{x \in_D \{0,1\}^m}[A(x)]| < \epsilon$$

For a class $\mathcal{A}$ of such predicates, D is ϵ-pseudo-random *for $\mathcal{A}$ if D ϵ-fools each $A \in \mathcal{A}$.*

Naturally, the uniform distribution (as well as distributions which are close to it in the standard statistical sense) are pseudo-random. The potential of this definition depends on the possibility that distributions which are very different than the uniform distribution in the standard statistical sense are pseudo-random. The following lemma asserts that this potential is indeed fulfilled.

Lemma 1.2. *Let $\mathcal{A}$ be a collection of r predicates on $\{0,1\}^m$. There exists a distribution D which is ϵ-pseudo-random for $\mathcal{A}$ such that D is uniform on a (multi)-set of size $O(\log r/\epsilon^2)$.*

To the best of my knowledge this lemma first appeared (implicitly) in [Ade78]. The lemma says that if the recipient of the random bits is known in advance to belong to a small collection, the distribution of the above lemma can be used instead of the uniform distribution and the recipient cannot notice the difference and will behave the same way. Note that only $O(\log\log r + \log(1/\epsilon)$ bits are needed to sample from the pseudo-random distribution, and thus significant savings can be made if the collection is small. Unfortunately, the proof of lemma 1.2 is not constructive. It shows that with high probability a random multi-set of the promised size will do. This guarantees the existence of such a multi-set but it does not provide a way of explicitly constructing it. Substantial effort has been invested in recent years to try and explicitly construct pseudo-random distributions for interesting collections.

1.4.2 Pseudo-randomness for efficient computation

We are interested in simulating efficient probabilistic algorithms, and thus in our setting the recipient of the random bits is an efficient procedure. This motivates the following definition.

Definition 1.3 (Pseudo-randomness for small circuits). *Let $Size_m$ denote the collection of all circuits of size m. (Note that any such circuit takes at most m inputs). A distribution D is* ϵ-pseudo-random for size m *if it is ϵ-pseudo-random for $Size_m$.*

A distribution which is pseudo-random in the sense of the above definition can be used instead of truly random bits in efficient probabilistic algorithms. The next paragraph presents this argument.

Suppose you want to run a polynomial time probabilistic algorithm A on a given input x. This corresponds to running a polynomial time computation $A(x, y)$ for a uniformly chosen y. The computation of $A(x, y)$ can be simulated by a circuit $C(y) = A(x, y)$ of size $m = |x|^{O(1)}$. Thus, a pseudo-random distribution for size m suffices to simulate the behavior of A on x up to small error. It follows that a polynomial time probabilistic algorithm recognizes the same language when its "random input" is only pseudo-random.

Why does a non-uniform model of computation such as circuits come up when we are interested in uniform probabilistic algorithms? Isn't it sufficient for a pseudo-random distribution to fool polynomial time Turing machines? The answer to this question is that in the argument we just used we were really interested in simulating the algorithm A for every possible input x. Thus, if we look at $A(x, y)$ as a function of y we are dealing with a polynomial time machine which received some arbitrary polynomially long string. In the proof we used the observation that for every x, this computation can be simulated by a small circuit, and it seems that such a computation is not necessarily computable by polynomial time Turing machines[1]. The argument above also explains why we do not differentiate between the size of the circuit and the number of inputs it accepts. These are of the same magnitude in the simulation above.

[1]Suppose someone is interested in deterministically simulating a probabilistic polynomial time algorithm A on a sequence of inputs $\{x_n\}$ where this sequence cannot be generated in polynomial time (or is even uncomputable by Turing machines). The argument above does not guarantee that a pseudo-random generator for polynomial time Turing machines suffices. The argument does go through if we are only interested in x's that can be efficiently generated by uniform polynomial time machines. And thus, a pseudo-random distribution for uniform machines (note that we need to speak of a family of such distributions indexed by m for this to make sense) is sufficient to simulate probabilistic algorithms on inputs which can be efficiently generated by uniform machines. In fact, a somewhat similar concept was the one initially defined in the seminal papers [BM84, Yao82], and plays an important role in other works such as [IW98, Kab00].

We are interested in pseudo-random distributions which can be sampled from using few random bits. This motivates the following definition in which many pseudo-random bits are generated from few truly random bits.

Definition 1.4 (Pseudo-random generator). *A function*

$$G : \{0,1\}^d \to \{0,1\}^m$$

is an ϵ-pseudo-random generator *(PRG) if* $G(U_d)$ *is* ϵ*-pseudo-random for size* m*. An input of* G *is called* a seed *of* G*, and* d *is called the seed length of* G*.*

Note that lemma 1.2 guarantees the existence of a ϵ-PRG with seed length $O(\log(m/\epsilon))$. However, this generator is of little use to us as it is not known to be explicitly constructible[2]. The notion of an explicit generator is defined next. While the definition is a bit technical, it essentially says that G can be computed in time exponential in its seed length.

Definition 1.5 (Explicit pseudo-random generator). *Let* $m(d), \epsilon(d)$ *be functions such for every number* t *a* d *such that* $t \leq m(d) \leq t^{O(1)}$ *can be found in time polynomial in* t*. A family* $G = \{G_l\}$ *of functions*

$$G_d : \{0,1\}^d \to \{0,1\}^{m(d)}$$

is an explicit ϵ-pseudo-random generator *if it can be computed in time* $2^{O(d)}$ *and for every* d*,* G_d *is an* $\epsilon(d)$*-pseudo-random generator.*

Remark 1.6. *The definition of explicit pseudo-random generators we give here is different than the one used in the seminal papers on pseudo-random generators [BM84, Yao82] and follows the definition given by Nisan and Wigderson in [NW94]. The initial definition differs from the one given here in two respects. The first is that the generator is required to run in time polynomial in its seed length, and the second is that it is supposed to fool all circuits of size polynomial in its output length.*

Allowing generators to run in time exponential in their seed length was first suggested by Nisan and Wigderson in [NW94]. The idea is that if we plan to run the generator over all seeds (which is the case for derandomizing probabilistic algorithms) we may as well allow the generator to run in exponential time.

The main difference between the two notions is that in the initial definition the circuit we are aiming to fool is large enough to run the generator whereas in

[2]Nevertheless, this generator is computable by a circuit of size $(m/\epsilon)^{O(1)}$. This circuit works by getting the set of all outputs of the generator as a non-uniform advice. This entails Adelman's theorem [Ade78] that BPP $\subseteq$ P/poly.

the second definition the generator runs in a time which is larger than the size of the circuit it attempts to fool. As we will see in the next section pseudo-random generators imply the existence of hard functions. However, the two definitions differ in the kind of hard functions they imply. The initial definition of [BM84, Yao82] implies the existence of one way functions. As explained in the next section the second definition implies a different kind of hardness which is believed to be weaker.

Using an explicit pseudo-random generator we can simulate any probabilistic algorithm deterministically by first generating the set of outputs of the generator, then running the algorithm on the given input using each of the generated outputs. The final answer can be computed by taking the majority vote of the answers of the algorithm[3].

Lemma 1.7. *If there exists an explicit $1/6$-pseudo-random generator for size m then any probabilistic algorithm which runs in time t (and therefore uses at most t random bits) can be simulated deterministically in time $2^{O(d)}t^{O(1)}$ where d is the seed length of for which $t \leq m(d) \leq t^{O(1)}$ as guaranteed in definition 1.5.*

It should be noted that the constant $1/6$ above can be replaced by any other constant smaller than $1/2$ and even by some functions of m which approach $1/2$. In the above lemma the quality of the simulation depends on the ratio between seed length and output length. Note that the existence of an explicit generator in which $m = 2^{\Omega(d)}$ implies that BPP = P. At this time, no unconditional construction of an explicit pseudo-random generators with $d < m$ is known.

1.4.3 Hardness versus randomness

The difficulty in constructing explicit pseudo-random generators is that the notion of explicit pseudo-randomness implies the existence of explicit hard functions. This follows because it is in particular hard for small circuits to recognize outputs of the generator. (Note that the generator has at most 2^d outputs out of all 2^m strings of length m). However, in time $2^{O(d)}$ one can generate all the outputs and therefore recognize them. Thus, the task of recognizing outputs of the generator is an explicit hard function. This is stated precisely in the following theorem. (Recall that $S(f)$ denotes the size of the smallest circuit which computes f).

[3]Note that here we are using the fact that the probabilistic algorithm has a boolean answer. This technique is applicable to some probabilistic algorithms with larger output. In general it is impossible to deterministically simulate probabilistic algorithms which compute functions with large outputs as producing a string y which have Kolmogorov complexity at least $|y| - 1$ can be performed by a probabilistic algorithm (which simply outputs a random y), yet no deterministic algorithm can compute it.

Theorem 1.8. *If there exists an explicit* $1/2$*-pseudo-random generator for size* m *then there exists a function* $f = \{f_l\}$ *such that* f *is computable in time* $2^{O(l)}$ *and* $S(f_{l+1}) > m(l)$.

Proof. (of theorem 1.8) Given an explicit $1/2$-pseudo-random generator which stretches d bits into $m(d)$ bits we define a function $f = \{f_l\}$, $f_l : \{0,1\}^l \to \{0,1\}$. Given an l bit input x, we define $f_l(x)$ to be 1 if and only if there exist some $z \in \{0,1\}^{m(l-1)-l}$ for which $x \circ z$ is an output of the generator G_{l-1}. It follows that $S(f_l) > m(l-1)$ since any circuit C computing f is not $1/2$-fooled by G. This is because C always answers 1 when given the output of the generator, and answers 1 with probability at most $1/2$ when given a uniformly chosen input since there are at most 2^{l-1} possible outputs of G_{l-1}. The function $f = \{f_l\}$ can be computed by constructing all the outputs of the generator on seeds of length $l-1$. This can be done in time $2^{O(l)}$. □

(It should be noted that a weaker notion of generators called "hitting set generators" to be defined in section 1.4.5 is sufficient to entail the conclusion). The work of [RR97] indicates that proving the existence of such hard functions requires the development of new proof techniques. At this time we can construct pseudo-random generators only assuming the existence of hard functions. Basing pseudo-randomness on hardness is often called the "hardness versus randomness" paradigm and was introduced by [BM84, Yao82]. (It should be noted that their definition of generators was somewhat different, see remark 1.6). Motivated by the above theorem we will allow ourselves to assume the following hardness assumption when constructing explicit pseudo-random generators.

Definition 1.9 (The $s(l)$**-hardness assumption).** *There exists a function* $f = \{f_l\}$ *which is computable in time* $2^{O(l)}$ *and for all l,* $S(f_l) > s(l)$.

The assumption is parameterized by the function $s(l)$ which measures how strong is the hardness assumption. Hardness versus randomness constructions give tradeoffs between the strength of the hardness assumption and the quality of the generator. By theorem 1.8 an explicit ϵ-pseudo-random generator which stretches d bits into $m(d)$ bits entails the $s(l)$-hardness assumption for $s(l+1) = m(l)$. A major question is whether the converse is also true. A long line of research [BFNW93, Imp95, IW97, STV99] focused in trying to prove that the $s(l)$-hardness assumption is sufficient to construct a pseudo-random generator which "stretches" d bits into $s(d)$ bits.

1.4.4 A note on optimality of generators

We use the terms "optimal generator" and "optimal conversion of hardness into pseudo-randomness" to denote a generator which uses a seed of length $d = O(l)$ and outputs $m = s^{\Omega(1)}$ bits using the $s(l)$-hardness assumption. In what sense is such a construction optimal?

Indeed, it is widely believed that there are pseudo-random generators which have seed length logarithmic in their output length and that $BPP = P$. In this case when given a function f with a "weak lower bound" (say $S(f_l) \geq 2^{\sqrt{l}}$), the generator construction may ignore the function f completely and run the best possible generator.

The optimality we have in mind is that of the *reduction* from hardness to pseudo-randomness. An optimal generator matches the rate of conversion of theorem 1.8 (which gives a reduction in the other direction). More formally, by theorem 1.8 any construction which converts hard functions into pseudo-random generators at a rate better than an "optimal one" also produces a harder function than the one initially supplied to it.

Another justification for this notion of optimality follows from Trevisan's method. It was shown by Trevisan, [Tre99] (see section 2.3.3) that any construction which converts hardness into a pseudo-random generator and has some additional relativization properties (stated precisely in definition 2.8 and shared by all known constructions) entails an extractor with the same seed length and output length. Thus, any such construction is limited by the lower bounds on extractors of [NZ96, RTS97] (which are presented in section 1.5.6), and in particular has $d(l) \geq l$ and $m(l) \leq s(l)$.

1.4.5 Hitting set generators

The definition of pseudo-random generator is suitable to derandomize BPP, that is probabilistic algorithms with two sided error. Many interesting probabilistic algorithms are in RP and have one sided error. A precise definition of RP appears in section 1.2.2. A weaker notion of pseudo-randomness suffices to simulate these algorithms. The one-sided version of a pseudo-random generator is called a hitting set generator.

Definition 1.10 (Hitting set generator for small circuits). *A function*

$$H : \{0,1\}^d \to \{0,1\}^m$$

is an ϵ-hitting-set generator *(HSG) if for every circuit C of size m such that*

$$\Pr_{x \in \{0,1\}^m}[C(x) = 1] \geq \epsilon$$

there exists a seed y such that $C(H(y)) = 1$.

The reader can verify that this definition is equivalent to the definition of pseudo-random generators if one omits the absolute value in definition 1.1. It follows that any ϵ-pseudo-random generator is also an ϵ-hitting set generator. One can define the notion of an *explicit hitting set generator* in a way equivalent to definition 1.5 and conclude a lemma analogous to 1.7 for one sided error algorithms. The reader may also verify that in theorem 1.8 it is actually enough to have a hitting set generator rather than a pseudo-random generator. In [ACR96] which was simplified in a sequence of works [ACRT97, BF99, GVW00] it was shown that hitting set generators suffice to derandomize two sided error algorithms. An even simpler proof of this is given in appendix B for hitting set generators with additional properties (as the ones constructed in this thesis).

1.4.6 Derandomizing AM and generators for nondeterministic circuits

Just as BPP is a randomized version of P, the class AM (defined in [Bab85, BM88]) is a randomized version of NP. An analogous question to the BPP versus P question is the AM versus NP question. Continuing the analogy, pseudo-random generators which fool small nondeterministic[4] circuits suffice to derandomize AM (see section 4.5 for precise definitions). In fact, hitting set generators suffice as the class AM coincides with its one-sided error analogue.

A sequence of works [AK97, KvM99, MV99] focused on derandomizing AM. The same argument used to prove theorem 1.8 shows pseudo-random generators which fool nondeterministic circuits entail a hard function for nondeterministic circuits. Following the analogy, it is natural to base such pseudo-random generators on a non-deterministic version of the $s(l)$-hardness assumption. While the pseudo-random generators constructed to fool (deterministic) circuits can be used to fool nondeterministic circuits (as was noticed by [AK97, KvM99]), they need to assume a stronger hardness assumption. They relied on functions which are hard not only to nondeterministic circuits but also to circuits with a SAT-oracle. The difference may be appreciated when considering the uniform analogue of these classes which are NP and P^{NP} respectively. In the uniform setting the first class is believed to be strictly contained in the second. An exception is the construction of [MV99] which is "specialized" for this setup and can be based on hardness for nondeterministic circuits. However, this construction only gives a hitting set generator and fails altogether if the hard function has hardness $s(l) \leq 2^{\sqrt{l}}$. A more

[4]For technical reason, it is actually necessary to fool co-nondeterministic circuits. The constructions of this paper fool both simultaneously, so we will keep referring to nondeterministic circuits.

elaborate discussion as well as precise definitions are given in section 4.5 where we present our results for this setting.

1.4.7 Milestones in constructing pseudo-random generators

As a very partial background we will survey important milestones in constructing pseudo-random generators using a hardness assumption.

As noted earlier in remark 1.6 this paradigm appeared in the works of [BM84, Yao82] to construct a slightly different notion of pseudo-random generators which implies the existence of one way functions. Nisan and Wigderson [NW94] noted that for the purpose of derandomizing probabilistic algorithms one can relax the definition of a pseudo-random generator to the one given here and allow it to run in time slightly larger than the adversary it attempts to fool. This notion of pseudo-random generators is not known to imply one way functions. Nisan and Wigderson also gave a construction of a pseudo-random generator (based on Nisan's generator for constant depth circuits [Nis91]). This construction assumes the existence of functions which are computable in exponential time and yet every small circuit computes them correctly on a fraction of inputs which is not significantly larger than one half. This seminal work initiated a long line of research [BFNW93, Imp95, IW97, STV99] which is summarized in table 1.1. These works used the NW-generator as is and focused on trying to achieve the (average case) hardness required by the NW-generator using worst-case hardness. The major milestone along this direction was achieved in [IW97] (and later significantly simplified in [STV99]). Impagliazzo and Wigderson provided a pseudo-random generator strong enough to entail P $=$ BPP based on the $2^{\Omega(l)}$-hardness assumption. (see definition 1.9). The state of affairs following [STV99] is that worst case hardness could be transformed into average case hardness without paying any (noticeable) quantitive penalty. However, the NW-generator fails to transform (average case) hardness into pseudo-randomness at a rate comparable to theorem 1.8.

1.4.8 Results

This thesis suggests two different techniques to overcome the above problem. Both lead to new constructions of pseudo-random generators. Exact results are presented in table 1.2, see table 1.1 for comparison with previous results.

- In chapter 2 (which is based on joint work with Russell Impagliazzo and Avi Wigderson [ISW99, ISW00]) a new and more sophisticated way of using the NW-generator is suggested. The main idea is to run the NW-generator with "the wrong" parameters and then argue that upon a failure to produce

Table 1.1: Milestones in pseudo-random generators constructions:

Reference	seed size	circuit size	type
[BFNW93]	$d = O(l^4 \log^3 s)$	$m = s^{\Omega(1)}$	PRG
[Imp95]	$d = O(l^2 \log s)$	$m = s^{\Omega(1)}$	PRG
[IW97]*	$d = O(\frac{l^4}{\log^3 s})$	$m = s^{\Omega(1)}$	PRG
[STV99]	$d = O(\frac{l^2}{\log s})$	$m = s^{\Omega(1)}$	PRG
optimal†	$d = O(l)$	$m = s^{\Omega(1)}$	PRG

All results assume the $s(l)$-hardness assumption.

*Impagliazzo and Wigderson state their result for $s(l) = 2^{\Omega(l)}$ and their result gives $d = O(l)$, (which implies $BPP = P$) for such an assumption.

†See section 1.4.4 for the exact notion of optimality.

Table 1.2: New pseudo-random generators constructions:

Reference	seed size	circuit size	type
Theorem 2.1	$d = O(l)$	$m = s^{\Omega(\frac{1}{\log\log(l/\log s)})}$	PRG
Theorem 4.5	$d = O(\frac{l^2}{\log s})$	$m = s^{\Omega(1)}$	PRG
Theorem 4.6	$d = O(l)$	$m = s^{\Omega(1)}$	HSG
optimal*	$d = O(l)$	$m = s^{\Omega(1)}$	PRG

All results assume the $s(l)$-hardness assumption.

*See section 1.4.4 for the exact notion of optimality.

a pseudo-random generator we obtain a harder function. Using this argument recursively (as well as some other tricks) we obtain the first pseudo-random generator which uses the asymptotically optimal seed length (that is one which is *linear* in the input length of the hard function f), regardless of its hardness. Unfortunately, the recursive construction causes the output length to be slightly smaller than expected and we can only fool circuits of size $m(l) = s(l)^{\Omega(1/\log\log l)}$. See theorem 2.1 for precise formulation.

- In chapter 4 (which is based on joint work with Chris Umans [SU01]) we build on a recent new method for constructing extractors presented in [TSZS01]. We use Trevisan's argument "the other way" to use this method to construct generators. An essential component in our construction is improving the proof technique of [TSZS01] to make it efficient enough to yield a small circuit for the underlying hard function in case the generator fails. Using this technique we are able to construct a totally new pseudo-random generator which is *not* based on the Nisan and Wigderson construction and matches the parameters of [STV99] (see theorem 4.5). We are also able to construct a hitting set generator which has both the asymptotically optimal seed length and output length simultaneously (see theorem 4.6). This concludes the efforts to give a conversion of hardness into pseudo-randomness at a rate that matches the conversion in the other direction stated in theorem 1.8.

  The above construction also produces PRG's and HSG's which fool nondeterministic circuits based on nondeterministic hardness (see theorems 4.7,4.8). This improves previous works by [KvM99] (which needs to assume hardness for circuits with SAT-oracle, an assumption stronger than nondeterministic hardness) and [MV99] (which produces only HSG's and works only when the function is hard for very large nondeterministic circuits).

In appendix B (which is based on joint work with Russell Impagliazzo and Avi Wigderson [ISW99]) we show how to use a collection of "candidate functions" where one of them is guaranteed to be a pseudo-random generator to derandomize BPP. The scenario of having many "candidate generators" appears in chapter 2,4. This provides an alternative solution to the ones suggested in these chapters.

1.5 Extractors

1.5.1 Imperfect random sources

In section 1.2 we motivated the task of extracting randomness from distributions which "contain some randomness". We will call such distributions imperfect random sources. It is instructive to see some examples before giving precise definitions:

- **A bit fixing source:** A distribution X on $\{0,1\}^n$ on which $n-k$ bits are fixed and the remaining k bits are uniform and independent of each other. Such a distribution has an "embedded" copy of the uniform distribution on k bits.
- **A subset source:** A distribution X on $\{0,1\}^n$ which is uniform on some subset $S \subseteq \{0,1\}^n$ of size 2^k. This is a generalization of the previous item and intuitively "contains" k bits of randomness.
- **A Santha-Vazirani source:** A distribution X on $\{0,1\}^n$ with the property that for every $1 \le i \le n$ and $b_1, \cdots, b_{i-1} \in \{0,1\}$

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

  In words every bit is slightly unpredictable given previous bits.

The reader may verify that in all these distributions the probability of every string $x \in \{0,1\}^n$ does not exceed 2^{-k}. In other words none of these distributions contains "heavy elements". We cannot hope to extract k random bits from distributions which have "heavy elements" (elements with probability lager than 2^{-k}). This is because a "heavy element" in the input implies a "heavy element" in the output. However the uniform distribution on k bit strings does not have such a "heavy element". In this sense not having "heavy elements" is a necessary condition for sources from which we can extract randomness[5]. Zuckerman suggested to use this criteria to measure the amount of randomness in distributions from which we want to extract randomness.

Definition 1.11 (min-entropy). *The* min-entropy *of a distribution X (denoted by $H_\infty(X)$) is defined by*

$$H_\infty(X) = min_x(-\log \Pr[X = x])$$

*We say that a distribution X is a k-*source *(or informally that "it contains k bits of randomness" if $H_\infty(X) \ge k$ or equivalently if $\Pr[X = x] \le 2^{-k}$ for all x.*

[5]In the formal definition of extractors we will only require that the output distribution is statistically close to uniform. A similar argument still applies. The total weight of "heavy elements" in such a distribution which is statistically close to uniform is small and therefore the initial distribution cannot put a lot of weight on "heavy elements", and is thus statistically close to not having "heavy elements".

As its name suggests min-entropy is somewhat similar to Shannon's entropy function. min-entropy measures the amount of randomness a distribution contains on the worst case (whereas Shannon's entropy measures the amount of randomness a distribution contains on average and replaces "minimum" in definition 1.11 with an "expectation"). We are interested in the worst case notion because we only get to sample from the distribution once.

1.5.2 Definition of extractors

Our goal is to construct a *single* function which extracts randomness from *every* distribution which contain sufficient randomness. This is clearly impossible. (Once the "extractor function" is fixed, the uniform distribution on strings on which the output of the function is fixed to some value demonstrates the failure of the extractor).[6] We would have to settle for less and allow extractors to use a short seed of truly random bits as a secondary input. This adds "unpredictability" to the extractor function as the value of the seed is not fixed. Using a seed of additional randomness makes "economic sense" as long as the number of random bits extracted exceeds the number of random bits spent. It turns out that even after this relaxation we cannot extract truly random bits and will have to settle for distributions which are close to uniform in a statistical sense.

Definition 1.12 (statistical distance). *Two distribution P, Q over the same domain T are* ϵ-close *if the $L1$-distance between them is bounded by 2ϵ, namely:*

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

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

The latter definition explains the usefulness of this notion. The two distributions are practically indistinguishable. (In the language of the previous section a distribution which is ϵ-close to uniform is ϵ-pseudo-random for all predicates). We are now ready to formally define extractors.

Definition 1.13 (extractor). *A* (k, ϵ)-extractor *is a function*

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

Such that for every k-source X the distribution $Ext(X, U_d)$ is ϵ-close to U_m.

[6] Santha and Vazirani [SV86] showed that it is also possible to design a "Santha Vazirani source" (see examples above) which demonstrates the failure of any fixed "extractor function". This is not at all trivial and shows that it is impossible to use a fixed function to extract randomness even from a "nicely structured" source such as a Santha-Vazirani source.

A non-constructive argument shows the existence of an *optimal* extractor which matches the known lower bounds by [RTS97] (to be surveyed in section 1.5.6). However, as in the case of pseudo-random generator this extractor is of little use to us as we do not know how to explicitly construct it.

Definition 1.14 (Explicit extractor). *Let $k(n), \epsilon(n), d(n), m(n)$ be integer functions. A family $Ext = \{Ext_n\}$ of functions*

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

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

Using an explicit extractor we can simulate any probabilistic algorithm given a sample from an imperfect random source. This is done by first generating the set of outputs of the extractor on the given source element using all possible seeds, then running the algorithm on the given input using each of the generated outputs. The final answer can be computed by taking the majority vote of the answers of the algorithm. We conclude that:

Lemma 1.15. *If there exists an explicit $(k, 1/6)$-extractor G then any probabilistic algorithm which runs in time t (and therefore uses at most t random bits) can be simulated in time $2^{O(d(n))}n^{O(1)}$ given a single sample from a k-source. Where n is such that $m(n) \geq t$.*

The tradeoff in lemma 1.15 resembles the one in lemma 1.7. We elaborate more on the connection in section 1.6.1.

1.5.3 Extractors as graphs

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

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

This behavior resembles that of expander graphs. An important difference is that extractors are *unbalanced* bipartite graphs (by that we mean that the two sides differ in size). Another analogy to expander graphs is that the above property entails that the set S ”sees” a $1-\epsilon$ fraction of the nodes in the right hand side. This can be interpreted as ”relative expansion” as the volume the set takes in the space it lives in is expanded from 2^{k-n} to $1-\epsilon$.[7] Viewed this way, the effort spent to construct explicit extractors with small seed length is a continuation of the effort spent to construct explicit expander graphs with small degree. In contrast to expander graphs, ”extractor graphs” cannot have constant degree. (This follows from lower bounds on seed length of extractors to be surveyed in the following sections).

1.5.4 Strong extractors

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

Definition 1.16 (Strong extractor). *A* (k,ϵ)-strong extractor *is a function*

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

Such that for every k-source X the distribution $Ext(X,U_d) \circ U_d$ is ϵ-close to U_{m+d}.

A useful property of strong extractors is that for every source almost all seeds extract randomness from the source. It is interesting to note that in the seminal paper on extractors the strong version was defined and the weaker version first appeared in [TS96]. This thesis shows how to explicitly transform any extractor into a strong extractor.

1.5.5 Applications of extractors

While simulating probabilistic algorithms given access to an imperfect random source was the main motivation behind extractors, they turn out to have many other applications. These include unconditional constructions of pseudo-random generators for probabilistic algorithms with small memory [NZ96], randomness efficient oblivious sampling and ”deterministic amplification” of the success probability of probabilistic algorithms [Zuc97], hardness of approximation results for CLIQUE

[7] In fact, this property is weaker than the one above, and defines a different object called a ”disperser”. Some of the applications require only this property.

[Zuc96a] some Σ_2^P problems [Uma99] and approximating the VC-dimension [MU01], a different proof that BPP $\subseteq$ PH [GZ97], explicit constructions of depth 2 superconcentrators, nonblocking networks and certain expander graphs with expansion properties stronger than can be obtained using the eigenvalue method, protocols for sorting and selecting in rounds [WZ99] and leader election [Zuc97, RZ98], and error correcting codes with strong list decoding properties [TSZ01]. The reader is encouraged to look at the excellent (though slightly outdated) survey papers [Nis96, NTS99]. It should be noted that all these applications require explicit constructions of extractors, and that in many of them improved constructions immediately translates to an in improvement in the application.

1.5.6 About the parameters

The definition of extractors involves five parameters. It is customary to think of n, k and ϵ as given and try to optimize the seed length and output length as a function of them. We will refer to the ratio m/k, as the *fraction of randomness extracted by the extractor* and to the ratio k/n as the *entropy rate of the source*.

In [RTS97] it was shown that the seed length d has to be at least $\log(n/k) + 2\log(1/\epsilon) - O(1)$ as long as $\epsilon < 1/2$ and the extractor is non-trivial. (Slightly weaker bounds were previously given in [NZ96]). [RTS97] also showed that extractors have to suffer some "entropy loss" and cannot extract all the $k + d$ bits of randomness present in their input. More precisely, there is an upper bound of $k + d - 2\log(1/\epsilon) + O(1)$ on the output length of an extractor as long as $\epsilon < 1/2$. A non-constructive argument shows the existence of an "optimal extractor" which matches both lower bounds on seed length and output length simultaneously (except for the exact constant "hidden" in the $O(1)$ notation) and achieves this for all "entropy-levels" (by that we mean for all k). While explicit constructions of extractors still do not match this behavior, the "quest for the optimal extractor" gradually produces better results. Some of these constructions are summarized in the next section.

1.5.7 Milestones in constructing extractors

As a very partial background we will survey important milestones in extractor constructions. These results also appear in table 1.3.

The attempt to produce random bits from distributions which are not random goes back to von Neumann, [vN51] who considered a source of independent biassed coins with unknown bias. The paradigm of extracting randomness from imperfect random sources was initiated by [Blu84, SV86]. Following this seminal work, a variety of types of sources were considered until the concept of extractors emerged from the works of Zuckerman [Zuc90, Zuc96b] and was explicitly defined in [NZ96].[8] The first construction of extractors appeared implicitly in [ILL89] a couple of years before extractors were defined. This construction was based on hash functions and had an enormous seed length. In fact, one has to spend more truly random bits than are present in the source, and the only reason this makes "economic sense" is because this "investment" is returned with a "dividend" of random bits extracted from the source. In their seminal paper [NZ96] Nisan and Zuckerman used this initial extractor to construct extractors with much shorter seed length. This was achieved by "composing" the initial extractor with itself in various ways. Subsequent work [SZ99, SSZ98, TS96, Zuc97, TS98, NTS99] used these extractors and more composition methods to significantly improve the parameters. (Some of these constructions produce weaker objects called "dispersers" which are the "one-sided" analogue of extractors). Important milestones achieved by this line of work is the construction of an extractor which works for all entropy levels and spends poly $\log(n)$ random bits to extract all the randomness out of the source [TS96], and an extractor which spends the "asymptotically optimal" seed length $O(\log(n/\epsilon))$ to extract any constant fraction of the randomness in the source as long as the source contains $k = \Omega(n)$ random bits, [Zuc97]. As these constructions used compositions and iterations of previous constructions (which in turn used compositions and iterations of previous constructions) the resulting constructions are explicit but somewhat complicated.

A major breakthrough was made by Trevisan in [Tre99]. He observed that methods used to construct pseudo-random generators from hard functions actually produce extractors. He went on to construct a direct and simple extractor using the pseudo-random generator of [NW94, IW97]. This extractor improved previous constructions (and in particular it achieves the asymptotically optimal seed length for constant error when $k = n^{\Omega(1)}$) but suffers from the limitations of the Nisan-Wigderson generator on which we elaborate in section 2.2. In [RRV99b] the NW-generator was modified to give an extractor which extracts all the randomness from both the source and seed (except for the unavoidable entropy loss). However, This construction does not achieve the asymptomatically optimal seed length. Two more constructions by [TSUZ01, TSZS01] are surveyed in the next section, as they appeared after some of the results reported in this thesis.

There are also some ways to trade between the different parameters.

- **Increasing the output length:** Intuitively, if the extractor did not extract all the randomness from the source then conditioned on the extractor's output the source still contains some randomness. This can be extracted using

[8] Actually, the object defined there is slightly different and often referred to as a "strong extractor". See section 1.5.4 for more details.

another extractor. (It is important to note that this extractor should have a smaller min-entropy threshold.) The penalty is that one needs to use more random bits to run the second extractor. This idea first appeared in [WZ99]. This transformation enables us to transform an extractor with $m = \Omega(k)$ into one with $m = (1 - \delta)k$ (for any constant δ) paying a penalty of multiplying the seed length by a constant or alternatively into one with $m = k$ multiplying the seed length by $O(\log k)$. A better analysis of this method was given in [RRV99b]. It was used there to show how to extract *all* the randomness from both the source and seed (except for the unavoidable entropy loss of $2\log(1/\epsilon) + O(1)$), at the cost of increasing the seed length.

- **Reducing the error:** In [RRV99a] it was shown how to explicitly convert an extractor with error $1/m$ into one with arbitrary small error, while increasing the seed length from d to $d' = O(d + \log(1/\epsilon))$ and reducing the output length from m to $\Omega(m) - O(\log 1/\epsilon)$. This construction can also start with constant error, however in that case the seed length is also multiplied by an additional factor of poly $\log\log(n)$. The importance of this construction is that as long as its losses are bearable the error parameter can be safely fixed and ignored.

1.5.8 Results

This thesis uses three different strategies to constructing extractors. Exact results are summarized in table 1.4, see table 1.3 for comparison with some previous constructions.

- In chapter 2 (which is based on joint work with Russell Impagliazzo and Avi Wigderson [ISW99, ISW00]) a new and more sophisticated way of using the NW-generator is suggested, which exploits both successes and failures of the generator. This results in a new pseudo-random generator which does achieve the asymptotically optimal seed length. Using Trevisan's transformation (which we observe to be more efficient if hardness is measured by Kolmogorov complexity instead of circuit complexity) we obtain the first extractor which achieves the asymptotically optimal seed length for all entropy levels (see theorem 2.2). This method actually ”favors” the lower entropy levels and for such we extract a much larger fraction of the randomness in the source (see theorem 2.3). (This was further exploited in [TSUZ01] which improves this construction to get an extractor which extracts all the randomness using the asymptotically optimal seed length for low entropy levels.)

Table 1.3: Milestones in extractors constructions:

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

All the results are stated for constant error ϵ.

$\delta > 0$ and i are arbitrary constants and the constants in $O(\cdot), \Omega(\cdot)$ may depend on them.

*Here, $\log^{(i)}$ is the i times iterated log function.

†This result followed the results of chapter 2 and improves theorem 2.3 presented there.

‡This paper also achieves other tradeoffs of parameters.

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

Table 1.4: New extractors constructions:

Reference	min-entropy threshold	seed length	output length
Theorem 2.2	any k	$d = O(\log n)$	$m = k^{1-\delta}$
Theorem 2.3	$k \leq 2^{\sqrt{\log n}}$	$d = O(\log n)$	$m = \Omega(\frac{k}{\log \log \log n})$
Theorem 3.1	any k	$d = O(\log n \cdot (\log \log n)^2)$	$m = \Omega(k)$
Theorem 3.3	any k	$d = O(\log n)$	$m = \Omega(k/\log n)$
Theorem 4.1	$k \geq \log^{O(1/\delta)} n$	$d = O(\log n)$	$m = k^{1-\delta}$
Theorem 4.2	any k	$d = (1+\alpha) \log n$	$m = k^{\Omega(\alpha)}$
Theorem 4.4	any k	$d = (1+\alpha) \log n$	$m = \frac{k}{\log^{O(1/\alpha)} n}$
optimal*	any k	$d = \log n + O(1)$	$m = k + d - O(1)$

All the results are stated for constant error ϵ.

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

$\alpha < 1$ is an arbitrary vanishing function.

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

- In chapter 3 (which is based on joint work with Omer Reingold and Avi Wigderson [RSW00]) we improve the ”composition techniques” of [NZ96, SZ99, NTS99] to give a new extractor construction. We obtain an extractor which optimizes any of one of the two main parameters (seed length and output length) up to a constant factor while paying only a *poly-logarithmic* penalty in the other (see theorems 3.1,3.1). (Previous best constructions, [TS96, RRV99b, ISW00] payed a *polynomial* penalty). The main new concept in this work is that of a *condenser*. This is an object somewhat similar to an extractor but whereas an extractor is required to output the uniform distribution, a condenser is only required to output a distribution which is more ”condensed” than the initial one. (Condensers were previously defined in another context in [RR99]).

- In chapter 4 (which is based on joint work with Chris Umans [SU01]) we build on a recent new method for constructing extractors presented in [TSZS01]. This method produces simple and direct extractors and is very suitable for achieving the optimal seed length (with the correct leading constant). However, the construction of [TSZS01] works only for ”high” entropy levels and extracts a small fraction of the randomness in the sources. By modifying the construction and improving the proof technique we obtain a simple and direct extractor construction which works for all entropy levels (see theorems 4.1,4.2,4.3,4.4). This matches the parameters of chapter 2 and some of those in 3 using much simpler constructions.

In appendix A (which is based on joint work with Omer Reingold and Avi Wigderson [RSW00]) we show how to convert regular extractors into strong extractors. This conversion changes the seed length and output length only slightly more then they need to be changed (see theorem A.2).

1.6 Connections between pseudo-random generators and extractors

Although pseudo-random generators and extractors both serve as tools to eliminate or reduce the usage of randomness in probabilistic algorithms they seem to belong to two very different realms. Pseudo-random generators belong to the ”computational realm”. Their output is not at all random by standard statistical standards. (For example the Shannon entropy of the output distribution is bounded by the seed length and is much smaller than the length of the output). In fact, the whole concept of ”generating randomness deterministically” is ridiculous from the point of view of information theory. Extractors on the other hand do belong to the ”information theoretic realm" and their output is close to random in the information theoretic sense. In light of this, the connections to be immediately presented between these objects are very surprising.

1.6.1 Trevisan's discovery

In [Tre99], Trevisan observed that certain hardness versus randomness tradeoffs actually give extractor constructions. In this section we give an informal presentation of Trevisan's argument. A precise statement of the Trevisan's result [Tre99] appears in section 2.3.3, see definitions 2.8,2.9 and theorem 2.10 there.

Hardness versus randomness tradeoffs such as those based on the generator of Nisan and Wigderson [NW94] (see section 1.4.3) convert a function which is hard for small circuits into a pseudo-random distribution for small circuits. It was observed in by Kilvans and van-Melkebeek [KvM99] that the proof of Nisan and Wigderson relativizes. More precisely, what Nisan and Wigderson really prove is that given a predicate A on m bits that is not fooled by the NW-generator it is possible to construct a small circuit (with A-gates) that computes the function on which the generator is based. Thus, if A is "small" in some sense (for example can be computed by small circuits) the resulting circuit will be small in the same sense.

Trevisan's insight was that this argument which seems specifically tailored to the "computational realm" can actually be used in the "information theoretic realm". Suppose we want to argue that the output distribution is close to uniform in the statistical sense, that is that it fools *all* predicates A on m bits. We do not have any bound on the complexity of A, this seems to prevent us from placing any bound on the complexity of the function computed by the circuit constructed using A in the proof of Nisan and Wigderson. Indeed, as this circuit uses A, the function it computes may be arbitrarily complex. A key observation of Trevisan in [Tre99] is that this function is "simple" in some sense: It is computable by a *small* circuit with A-gates. For every fixed A there are only few such functions. Thus, if we sample an element from an imperfect random source and interpret it as the truth table of a function with high probability it will not be computable by a small circuit with A gates, and thus the NW-generator will fool A. As this is true for any predicate A, any Nisan-Wigderson style hardness versus randomness construction is an extractor!

In fact, when viewing the Nisan-Wigderson generator in an information theoretic perspective, it is beneficial to replace the notion of hardness from computational to information theoretic and use Kolmogorov complexity instead of circuit complexity. Indeed, the argument above only used small circuits as small descriptions and ignored the computational features of circuits.

This observation plays an important role in the construction of chapter 2.[9] There we design a new Nisan-Wigderson style generator in which there is a significant difference between the circuit size and the description size of the circuit constructed from the predicate A. Using the above observation allows us to significantly improve the parameters of the extractor constructed there.

1.6.2 A unified theory: using the same parameters

Following Trevisan's discovery it makes sense to use the same choice of letters in extractors and pseudo-random generators. The scenario of constructing pseudo-random generators given a hard function is now thought of as analogous to the scenario of extracting randomness from an imperfect random source using the following "dictionary":

- The *length of the truth table of the hard function* is analogous to the *length of the source*.
- The *complexity bound on the hard function* is analogous to the *amount of hardness in the random source*.
- The *seed length, output length and error* play the same role in both constructs.

To complete the analogy it is sometimes preferable to arrange the parameters in hardness versus randomness tradeoffs in the same way as in extractors terminology. Thus, the hard function is thought of as taking $l = \log n$ bits as input, its complexity is measured as a function of n (rather than l), and the seed length and output length are expressed as functions of n and k (compare to definition 1.4).

1.6.3 Pseudo-random generators for space bounded probabilistic algorithms

Another striking connection between pseudo-random generators and extractors is that extractors are a key component in unconditional constructions of pseudo-random generators for space bounded probabilistic algorithms [Nis92, NZ96, INW94, RR99]. In fact, in the seminal paper of Nisan and Zuckerman [NZ96] the main application of extractors was such a construction.

The concept of space bounded machines allow considering algorithms which use less memory than the length of their input. This is done by differentiating between different types of memory. The input is stored on a "read only" tape, while

[9] See definition 2.9 and theorem 2.10 for exact details.

computations are made on a different (and shorter) tape on which the machine can both read and write. Only the size of the latter tape is counted as the space of the machine. For precise definitions see for example [Pap94]. A probabilistic space bounded algorithm is also allowed to toss coins. Probabilistic algorithms which use Markov chains often lead to space bounded probabilistic algorithms. An example is a probabilistic algorithm which runs in *logarithmic* space and solves the *undirected connectivity* problem (in which one is given an undirected graph and two vertices and should determine if there is a path connecting the vertices).

A pseudo-random generator for space bounded probabilistic algorithms is required to generate a long sequence of ”pseudo-random” bits from a short seed of truly random bits, so that when the ”pseudo-random” sequence is given to a bounded space machine as random coin tosses, it behaves the same way as it does when given uniform coin tosses.

To outline the connection between the extractors and pseudo-random generators for space bounded machines we outline the idea of the construction of [NZ96]. Let s be the bound on the space of the algorithm. We will now outline how to generate $3s$ pseudo-random bits from roughly $2s$ truly random bits.

The generator starts by outputting $2s$ random bits. After reading these the algorithm may remembers at most s bits of information about these bits. This means that from its point of view the initial $2s$ bits are a random source which contains s random bits. The generator can now spend few random bits as seed to extract those s bits from the source and these can given to the algorithm as they are random from its point of view.

This process can be repeated after the algorithm ”consumes” the s bits as at that point the algorithm still remembers at most s bits of information about the initial $2s$ bits. By using this process recursively Nisan and Zuckerman [NZ96] constructed a pseudo-random generator which runs in space $O(s)$ and stretches $O(s)$ bits into $s^{O(1)}$ pseudo-random bits which fool algorithms which have space s. An important consequence is that any Markov chain with 2^s states can be simulated for $s^{O(1)}$ steps using only $O(s)$ random bits. We stress again that this result is unconditional!

Chapter 2

A recursive analysis of the NW-generator

2.1 Introduction

The Nisan-Wigderson pseudo-random generator [NW94] was constructed to derandomize probabilistic algorithms under the assumption that hard functions exist. They showed that every difficult problem (in $E = dtime(2^{O(n)})$) could be used to construct a pseudo-random generator. The quality of this NW-generator (i.e. its seed length) was shown to relate to the difficulty of the given function. Their work has been quantitatively improved and qualitatively extended ([BFNW93, Imp95, IW97, IW98, STV99, KvM99, CNS99]), but their construction remains central to work in this area. While the best "hardness vs. randomness" trade-off to be expected from such a construction should yield a seed whose size is `linear` in the input size of the given hard function, this was not achieved yet, and the best construction so far [STV99] has seeds that can be nearly `quadratic`.

Trevisan ([Tre99]) recently showed that any "Nisan Wigderson style" generator (converting arbitrary hard functions into pseudo-random distributions) could be used to construct an extractor. Moreover, the "seeds" in both constructions are of the same length under this translation. [Tre99] (and then [RRV99b]) proceeded to use the NW-generator to give the best and simplest constructions of extractors known. As above, even the best can sometimes require nearly quadratic seed length in the "optimal" value (proved existentially by a counting argument matching the known lower bound). Thus, improving the NW-generator will impact both directions.

We give the first construction of a pseudo-random generator with `linear` seed length that uses arbitrary hardness assumption (namely almost any superpolynomial circuit lower bound). Such generators were previously known only assuming exponential hardness [IW97, STV99].

The key is a new analysis of the NW-generator [NW94]. We show that it fails to be pseudo random only if a much harder function can be efficiently constructed from the given hard function. This leads to a new recursive generator, which may be viewed as a reduction from the general case of arbitrary hardness to the solved case of exponential hardness.

We also give the first construction of an extractor which uses asymptotically optimal seed length, for random sources of arbitrary min-entropy. Thus, our construction is the first to use the optimal seed length for sub-polynomial entropy levels. It builds on the fundamental connection between extractors and pseudo-random generators discovered by Trevisan [Tre99], combined with the construction above. Using Kolmogorov Complexity rather than circuit size in the analysis of the extractor enables us to increase the number of bits extracted and extract a polynomial fraction of the initial randomness.

2.1.1 Results

Below we state our main theorems formalizing the above. They will be repeated in the technical sections, with the precise definitions. An important note regarding both is that while we achieve optimal seed length, the output length in both is suboptimal, and improving it constitutes an interesting open problem.

Theorem 2.1 (`Optimal seed generators`). *For every (hardness) function $k(l)$ with $l < k(l) < 2^l$, there is a function $m(l) = k(l)^{\Omega(1/\log\log(l/\log k))}$ so that the following holds: Assume there is a function $f = \{f_l\}$ that is computable in time $2^{O(l)}$, so that for all l, f_l cannot be computed by circuits of size $k(l)$. Then there exists a pseudo-random generator $G : \{0,1\}^{O(l)} \to \{0,1\}^{m(l)}$ that is computable in time $2^{O(l)}$ and fools all $m(l)$ size circuits.*

This theorem is stated more precisely in theorem 2.7 and its proof is given in section 2.5. See tables 1.1 and 1.2 for comparison with other results.

Theorem 2.2 (`Optimal seed extractors`). *For every n, k and ϵ there exists an explicit (k, ϵ)-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{\epsilon})} \to \{0,1\}^m$, where $m = k^{1-\delta}$ for arbitrary $\delta > 0$.*

This theorem follows from theorem 2.12 (in which $\epsilon = 1/m$) and the error reduction technique of [RRV99a]. Details are given in section 2.3.4. See tables 1.3 and 1.4 for comparison with some other results.

An interesting feature of our construction is that it favors sources with low entropy (whereas all previous constructions work better when the entropy is large).

An example is the following theorem in which we get a large improvement in the number of extracted bits if the entropy is small.

Theorem 2.3 (`Extractors for low min-entropy`). *For every n, k and ϵ there exists an explicit (k, ϵ)-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{\epsilon} + \log^2 k)} \to \{0,1\}^m$, for $m = \Omega(\frac{k}{\log\log(\log n/\log k)})$.*

Using the method of [WZ99] the number of bits extracted can be enlarged at the expense of enlarging the seed length. Using this we can improve the fraction of bits extracted to a constant fraction paying only a $O(\log\log(\log n/\log k))$ penalty in the seed length. See corollary 2.14 for details.

2.1.2 Organization of the chapter

In section 2.2 we give an informal overview of the technique used in this chapter. In section 2.3, we present the necessary background definitions and results. In section 2.4, we present our main construction, and prove that it "transforms hardness into pseudo-randomness". In section 2.5, we show how to deduce our results from the main construction.

2.2 Overview of the technique

Our goal is to transform a function $f : \{0,1\}^l \to \{0,1\}$ which cannot be computed by circuits of size k into a pseudo random generator with optimal seed length:

$$G : \{0,1\}^{O(l)} \to \{0,1\}^m$$

that fools circuits of size m for large as possible m. (Our final result has $m = k^{\Omega(1/\log\log(l/\log k))}$ which falls slightly short of the desired $m = k^{\Omega(1)}$). Our starting point is the NW-generator[1] which uses a larger seed under the same assumption:

$$NW : \{0,1\}^{O(\frac{l^2}{\log k})} \to \{0,1\}^m$$

(here $m = k^{\Omega(1)}$). The proof of Nisan and Wigderson shows how to construct a circuit C which computes the function f given a circuit A which is not fooled by their generator. The circuit C is constructed by "attaching" a circuit which

[1] Actually the NW-generator starts from a slightly stronger assumption, that is that the function is hard on average for circuits of size k. The result stated follows from the work of [STV99] who show how to transform a function in $E = dtime(2^{O(n)}$ that is hard on the worst case into a function in E that is hard on average while increasing the input length by only a constant. Their construction is also used in this paper.

computes some function f' to the circuit A. Such a function f' (which depends on f and A) is shown to exist and given as a non-uniform advice to the circuit C. In order to guarantee that C is "small", Nisan and Wigderson have to make sure that f' has a small circuit. They achieve this by setting the parameters so that f' is a function over very few bits. However, it turns out that the size of the input of f' *depends* on the seed length of the generator. If f is not assumed to be extremely hard, Nisan and Wigderson have to increase the seed length to guarantee that f' has a small enough circuit to contradict the hardness assumption.

The main idea of this chapter is to use the NW-generator with the "wrong" parameters. We set the seed length to the optimal $O(l)$ independent of k, that is regardless of how hard is the hard function f. While this is not guaranteed to produce a pseudo-random generator, if the argument of Nisan and Wigderson fails, it must be because f' does not have a small circuit. The parameters are chosen in a way that guarantees that the input length to f' is at most half of that of f. Thus, (since we measure the complexity of functions relative to the length of their input), f' is actually harder than f!

This idea is used recursively on f' and in each step either we construct a pseudo random generator with optimal seed length or we obtain a harder function. Eventually, we obtain a function which is extremely hard and can use the standard analysis to construct our generator. Thus, our construction can be viewed as a reduction from the general case of arbitrary hardness to the solved case of exponential hardness.

The main complication when coming to perform the above argument is that the identity of the function f' is determined by a non-uniform argument – it only shows the existence of such a function but does not tell us how to find it. More precisely, the argument shows that we can index $2^{O(l)}$ functions, one of which is the required one.

Considering all these potential functions at all levels of recursion, we can picture it as a tree. The root is labelled by the initial function f. The descendants of a node labelled by a function g are the ($2^{O(l)}$) functions which may arise as from the g in the NW-argument. The leaves have functions on $O(\log k)$ inputs.

One can construct a generator from all functions labelling nodes of this tree. This approach produces *many* (roughly $2^{O(l)}$) candidate generators where one of them is guaranteed to be pseudo-random. To complete the construction we show how to "combine" all the candidate into a single pseudo-random generator[2] Two natural ideas come to mind.

- In every node of the above tree replace the functions $f_1, \cdots, f_{2^{O(l)}}$ by a

[2]Observe that a hitting-set generator, and thus also a disperser, is easy to get already at this point

single function with two arguments: $\bar{f}(i,x) = f_i(x)$. Intuitively, if one of the candidate functions is hard so is the "concatenated" function.

- Combine all the candidate generators by running them with independently chosen seeds and take the exclusive-or of the outputs. Intuitively, if one of the candidates is pseudo-random, so is the xor-generator.

Both ideas fail in our setting due to the huge number of candidates. The first idea will cause the input size of $\bar{f}$ to be significantly *larger* than that of any of the f_i's as the length of i is larger than l. We will no longer have that $\bar{f}$ is over fewer bits than f and will not be able to deduce that $\bar{f}$ is harder than its ancestor. The second idea blows the seed length by a factor of the number of candidate generators.

However, both ideas are essential components of our construction. In order to be able to use them we need to reduce the degree of the tree from $2^{O(l)}$ to poly(k), where k is the hardness of the original function f. Assume for the moment that this can be magically done. Once achieved, the first idea is applicable since the length of the added input is now at most $O(\log k)$ (which will not drastically increase the input length of functions in the tree). In other words, a tree of such a small degree (poly(k) can be collapsed into a path whose length is the depth of the tree.

Since this path is of small length $O(\log\log(l/\log k))$, we can now apply the second idea and xor these few candidate generators. We avoid even the small loss to be suffered because of their number by arranging the seed lengths of different levels to go down geometrically.

The missing important step is cutting down the degree of the tree from $2^{O(l)}$ to k^2. This requires a slightly finer inspection of the argument of Nisan and Wigderson. We show that when we fail to produce a pseudo-random generator, not only one of the descendants of the function is hard, but rather a non-negligible fraction of them, (more precisely a $1/k$-fraction). Thus a sample of k^2 descendants will almost surely "hit" a hard one if it exists. Of course, trivial sampling costs more random bits than we can afford, but this can be done just as well with pairwise independence, increasing the seed length of the generator only by another $O(l)$ bits which we can easily tolerate.

There is however some price to performing this recursion. The circuit lower bound we can guarantee for f' is slightly smaller than we can guarantee for f. (If f requires circuits of size k, we can only guarantee that f' requires circuits of size $k/m^{O(1)}$). This loss is accumulated over the $\log\log(l/\log k)$ levels and causes us to have output length $m = k^{\Theta(1/\log\log(l/\log k))}$ rather than $k^{\Omega(1)}$.

This difficulty can be avoided when applying Trevisan's technique for constructing extractors. In such a case the hardness measure can be changed from circuit size to description size (which we define formally using Kolmogorov Complexity). This is beneficial because the procedure which constructs a circuit for

f given a circuit for f' can be *described* using much fewer bits than the size of the final circuit. Using this measure the hardness guarantee we get on f' is much larger, ($k - O(m)$ instead of $k/m^{O(1)}$), and as a result, we can increase m and extract any polynomial fraction of the initial entropy and at the same time simplify the argument.

2.3 Definitions and Ingredients

2.3.1 Complexity measures for functions and strings

We identify between functions $f : \{0,1\}^l \rightarrow \{0,1\}$ and strings in $\{0,1\}^{n=2^l}$ in the obvious way setting $f_i = f(i)$. This identification is helpful since we use two complexity measures: circuit complexity (which is defined for functions and is the measure we use for constructing pseudo-random generators) and Kolmogorov complexity (which is defined for strings and is the measure we use for constructing extractors). For both measures, we will argue that our construction transforms a "hard" function/string into a "pseudo-random" distribution. The exact meaning of this statement is that if a test A is not fooled by the "pseudo-random" distribution then A can be used as an oracle to compute/describe the initial function/string. Thus, we define circuit complexity and Kolmogorov complexity relative to a predicate A.

Definition 2.4. *Let A be a predicate on m bit inputs.*

- *Define $S_A(f)$ to be the size of the smallest circuit that computes f and is allowed to use A-gates (in addition to the standard boolean gates). Thus, $S(f)$ denotes the circuit complexity of f. We use the same convention in the next definitions.*

- *For $0 < \delta < 1/2$, let $S_{A,\delta}(f)$ be the minimum of $S_A(f')$ over all strings f' with hamming distance at most $\delta|f|$ from f.*

The last item in the above definition measures circuit complexity of functions on random inputs. If $S_{A,\delta}(f)$ is large, then f is not only hard to compute, but also hard to compute on average. We proceed and define these concepts for Kolmogorov complexity.

Definition 2.5. *Let A be a predicate on m bit inputs.*

- *Define $K_A(f)$, the Kolmogorov complexity of f given A, as the length (in bits) of the smallest description of f which is allowed to use a description of A for free, i.e., that of an oracle Turing Machine which, using oracle A, outputs f.*

- *For $0 < \delta < 1/2$, let $K_{A,\delta}(f)$ be the minimum of $K_A(f')$ over all strings f' with hamming distance at most $\delta|f|$ from f.*

2.3.2 Hardness versus Randomness

The only known constructions of pseudo-random generators use unproven assumptions. The most common assumption used is the existence of "hard functions". Specifically one assumes that there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$, yet every circuit of size $k(l)$ (where k is an integer function which measures the hardness of f), cannot compute f. Using this assumption a large number of papers [BFNW93, Imp95, IW97, IW98, STV99] construct pseudo-random generators.

All the results stated above use the Nisan Wigderson generator exactly as stated in [NW94], and improvements are gained by a pre-processing stage of hardness amplification. We will elaborate on this stage in section 2.3.7. The best previous construction of pseudo-random generators is by [STV99].

Theorem 2.6. *[STV99] If there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, $S(f_l) > k(l)$, then there exists functions $d(l) = O(\frac{l^2}{\log k(l)})$, $m(l) = k(l)^{\Omega(1)}$ and a $(1/m(l), m(l))$-pseudo random generator $G : \{0,1\}^{d(l)} \to \{0,1\}^{m(l)}$ which is computable in time $2^{O(d(l))}$.*

In this chapter we get a pseudo-random generator with seed length $O(l)$. A small gap from the best expected generator is left since we are only able to fool slightly smaller circuits. (The following theorem rephrases theorem 2.1 with a more formal notation).

Theorem 2.7. *If there exists a function $f = \{f_l\}$ which is computable in time $2^{O(l)}$ and for all l, $S(f_l) > k(l)$, then there exists functions $d(l) = O(l)$, $m(l) = k(l)^{\Omega(\frac{1}{\log\log(l/\log k)})}$ and a $(1/m(l), m(l))$-pseudo random generator $G : \{0,1\}^{d(l)} \to \{0,1\}^{m(l)}$ which is computable in time $2^{O(d(l))}$.*

2.3.3 Trevisan's method

Trevisan, [Tre99] discovered that any conversion of hard functions into pseudo-random generators with certain "relativization properties" also gives extractors. The following definition summarizes these properties.

Definition 2.8. *A (k, ϵ)-pseudo-random generator scheme is a function $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property: For any $f \in \{0,1\}^n$ and predicate A on m bits, if $G^f(\cdot) = G(f, \cdot)$ does not ϵ-fool A then $S_A(f) \leq k$.*

Note that given such a scheme any hard function can be used to construct a pseudo-random generator. Trevisan observed that all known hardness versus randomness tradeoffs actually construct pseudo-random generator schemes. Our construction is no exception. He also proved that any such scheme is an extractor. It turns out that for this connection it is more natural to replace circuit complexity by Kolmogorov complexity.

Definition 2.9. *A (k, ϵ)-extractor scheme is a function $G : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ with the following property: For any $f \in \{0,1\}^n$ and predicate A on m bits, if G^f does not ϵ-fool A then $K_A(f) \leq k$.*

Any (k, ϵ)-pseudo random generator scheme is also a $(O(k \log k), \epsilon)$-extractor scheme. (Simply because a circuit of size k can be described using $O(k \log k)$ bits). However, the converse is not necessarily true. The following theorem (by [Tre99]) asserts that extractor schemes are extractors.

Theorem 2.10. *[Tre99] Any (k, ϵ)-extractor scheme is a $(k+\log(1/\epsilon), 2\epsilon)$-extractor.*

Proof. Let Ext be a (k, ϵ)-extractor scheme, let A be any test, and let P be a distribution with min-entropy at least $k + \log(1/\epsilon)$. The bias of A on $Ext(P, U_d)$ is the expectation for $h \in_P \{0,1\}^n$ of the bias of Ext^h. For all but 2^k h's, the latter bias has absolute value less than ϵ. The at most 2^k exceptions have total probability at most ϵ. Therefore, the total bias is at most 2ϵ. □

There is also a partial converse:

Lemma 2.11. *Any explicit (k, ϵ)-extractor is a $(k + O(1), \epsilon)$-extractor scheme.*

Proof. Let Ext be a (k, ϵ)-extractor. Let H_A be the set of all h so that Ext^h does not ϵ-fool A. H_A is constructible using A as an oracle, so $K_A(h) \leq \log |H_A| + O(1)$ for any $h \in H_A$. Without loss of generality, assume that, for half the elements h in H_A, A is ϵ more likely for an output of Ext^h than for a random element. Then the same is the case if h is chosen uniformly from this subset of H_A, and x is chosen uniformly, and we compute $A(Ext(h, x))$. Thus, by the definition of extractors, this distribution on A has min-entropy less than k , i.e., $|H_A|/2 < 2^k$ or $\log |H_A| < k + 1$. □

Following Trevisan, we use theorem 2.10 to reduce the problem of constructing extractors into proving hardness versus randomness tradeoffs. We use lemma 2.11 to place known explicit extractors at the base of our recursion.

2.3.4 Error reduction in extractors

Our technique yields extractors with rather large error ($\epsilon = 1/m$). In [RRV99a] it was shown how to transform an extractor with large error into one that works for arbitrary small error ϵ. When one starts with error $\epsilon = 1/m$, this transformation increases the seed length only by an inevitable and asymptotically optimal additive factor of $O(\log \frac{1}{\epsilon})$. The transformation of [RRV99a] slightly hurts the output length. The extractor one gets only extracts a constant fraction of the randomness extracted by the initial extractor. This does not matter in our case since we lose a constant fraction of the randomness anyway. (The reader is referred to [RRV99a] for exact formulation). In light of this, we will "shoot" for error $1/m$. Our main construction produces the following explicit extractors, Performing the above transformation gives theorems 2.2,2.3.

Theorem 2.12. *For every n, k there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^m$ (where $m = k^{1-\delta}$ for arbitrary $\delta > 0$).*

Theorem 2.13. *For every n, k there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n + \log^2 k)} \to \{0,1\}^m$, for $m = \Omega(\frac{k}{\log\log(\log n/\log k)})$.*

Note that this extractor uses optimal seed length for $k < 2^{\sqrt{\log n}}$. In [WZ99] it was shown how to transform an extractor which extract $m = k/r$ bits into one which extracts $m = (1-\delta)k$, (for arbitrary constant δ) while multiplying d by $O(r)$. Using this with theorem 2.13 gives:

Corollary 2.14. *For every n and $k < 2^{\sqrt{\log n}}$ there exists an explicit $(k, 1/m)$-extractor $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n \cdot \log\log\log n)} \to \{0,1\}^m$ (where $m = (1-\delta)k$ for arbitrary $\delta > 0$).*

2.3.5 The Nisan-Wigderson generator

Almost all schemes of converting hardness into pseudo-randomness, as well as the above extractor constructions, use the NW-generator from [NW94]. Their construction converts a "hard" Boolean function f on l bit inputs, into a pseudo-random generator taking an input seed of size $d > l$ to an output of length $m >> d$. To use the construction for derandomization, one needs to specify the hard function f, and a family Δ of subsets of $\{1, \cdots, d\}$ such that each pair of sets has small intersection. Such families are called "designs", and the intersection sizes determine the quality of the pseudo-random generator.

Definition 2.15. *A family of sets $\Delta = (S_1, \cdots, S_m \subseteq [d])$ is called a (l, u)-design if*

- *For all i, $|S_i| = l$.*
- *For all $i \neq j$, $|S_i \cap S_j| \leq u$.*

Definition 2.16. *(The NW-generator) Let $l < d < m$ be integers, and let $n = 2^l$. Let Δ be a (l, u)-design. Define a function $NW^\Delta : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ in the following way: Given $f \in \{0,1\}^n$ and $x \in \{0,1\}^d$, we view f as a function over $l = \log n$ bits (by having $f(v) = f_v$). Let $x|_S$ denote the $|S|$ bit string obtained by restricting x to the indices in S. Define:*

$$NW^\Delta(f, x) = f(x|_{S_1}) \circ f(x|_{S_2}) .. \circ f(x|_{S_m})$$

For a fixed f, let $NW^{f,\Delta}$ be the function from $\{0,1\}^d \to \{0,1\}^m$ defined by $NW^{f,\Delta}(x) = NW^\Delta(f, x)$.

Given Δ, f and x as inputs, $NW^\Delta(f, x)$ can be computed in polynomial time. $NW^{f,\Delta}(x)$ can be computed, given Δ, in time polynomial in m, with an oracle for f. [NW94] also show how to construct good designs Δ.

Theorem 2.17. *[NW94] There exist constants c_1, c_2 such that for every l, d, m, such that $l < d < m$ and $d > c_2 \log m$ there exists a (l, u)-design $\Delta = (S_1, \cdots, S_m \subseteq [d])$, with $u = max(\frac{c_1 l^2}{d}, \log m)$, Furthermore, this design can be constructed in time $poly(2^d)$.*

Remark 2.18. *It can be shown that the designs constructed by Nisan and Wigderson are optimal up to constants for the parameters we are interested in. Thus, it is impossible to improve the seed length of the NW-generator by constructing more efficient designs. More precisely, shooting for the optimal seed length $d = O(l)$, it has to be the case that $u = \Omega(l)$. This easily follows from the inclusion-exclusion formula, see [ISW99] for details.*

In [RRV99b] it was observed that a weaker combinatorial property (which they call "weak designs") is sufficient for performing the analysis of Nisan and Wigderson. Weak designs are better than designs when u is very small. However, the lower bound of $u = \Omega(l)$ when $d = O(l)$ applies also to weak designs. Our construction can be carried out with weak designs but this will not improve the parameters of our pseudo-random generators/extractors.

The proof that the NW construction is a good pseudo-random generator involves looking at certain restrictions of f.

Definition 2.19. *Given an (l, u)-design $\Delta = (S_1, \cdots, S_m \subseteq [d])$ and a function $f : \{0,1\}^l \to \{0,1\}$, define a collection of functions*

$$\mathcal{R}_f^\Delta = \{f_{i,j,\beta}^\Delta | \beta \in \{0,1\}^d, 1 \leq i < j \leq m\}$$

as follows:

$$f^{\Delta}_{i,j,\beta} : \{0,1\}^u \to \{0,1\}$$

is the function defined by: On input $z \in \{0,1\}^u$, construct a string $s \in \{0,1\}^d$ by first assigning, for each $p \notin S_i \cap S_j$, $s_p = \beta_p$, and then filling the remaining (at most) u bits according to (the first bits of) z.

$$f^{\Delta}_{i,j,\beta}(z) = NW^{f,\Delta}(s)_j$$

For $1 \leq i \leq m$, let $f^{\Delta}_{i,\beta}(j,z) = f^{\Delta}_{i,j,\beta}(z)$. We refer to these functions as the "restrictions" of f. Note that the input size of each such restriction is $u + \log m$.

Note that given Δ and a function f (encoded as a $n = 2^l$ bit truth table) and $\beta \in \{0,1\}^d$ the truth table of $f^{\Delta}_{i,\beta}$ can be computed in polynomial time. These functions are over $u + \log m$ bits, which trivially entails that $S(f^{\Delta}_{i,\beta})$ and $K(f^{\Delta}_{i,\beta})$ are bounded by $m2^u$.

The following lemma is implicit in [NW94][3]:

Lemma 2.20. *There is a polynomial-time oracle Turing Machine $M^{g,A}$ with the following property. Assume $NW^{f,\Delta}$ does not ϵ-fool a test A. Choose uniformly a β, an i, an m bit string α, and an l bit string x. Then if M is run on (x, α, β) using oracles $g = f^{\Delta}_{i,\beta}$ and A*

$$|Prob[M^{g,A}(x,\alpha,\beta) = f(x)] - 1/2| = \Omega(\epsilon/m)$$

The conclusion of lemma 2.20 is that when given oracle access to a test A which is not fooled by the NW-generator we can use g to compute f (or the negation of f) correctly on a $1/2 + \Omega(\epsilon/m)$ fraction of the inputs. For completeness we give a proof of lemma 2.20. We need the following theorem by Yao.

[3][NW94] prove that $NW^{f,\Delta}$ is ϵ-pseudo-random for all tests computable in size $S_{1/2-\Omega(\epsilon/m)}(f)/(m2^u)$. This means that the existence of functions which are "hard to approximate" implies the existence of a pseudo-random generator.

With the above terminology, the original argument of [NW94] can be presented this way: [NW94] uses designs with very small u (which in turn forces d to be relatively large). This makes the circuit complexity of all the $f^{\Delta}_{i,\beta}$'s relatively small. Lemma 2.20 shows that any circuit A of size m which is not fooled by $NW^{f,\Delta}$ can be combined with the circuits for the restricted functions to construct a circuit of size $poly(m)2^u$ which approximates f. Thus, if f is assumed to be hard to approximate by such circuits, the distribution induced by the generator is pseudo-random for $Size_m$.

The observation that the proof of Nisan and Wigderson relativizes was made in [KvM99]. This observation is important for Trevisan's extractor, [Tre99]. The observation added here is that the argument of Nisan and Wigderson connects the complexity of f and its restrictions.

Theorem 2.21. *[Yao82] There is a polynomial time oracle Turing Machine M^A with the following property. Assume a distribution $b = (b_1, \cdots, b_m)$ on $\{0,1\}^m$ does not ϵ-fool a test A. Choose uniformly an i and an m bit string α. Then if M is run on $(b_1, \cdots, b_{i-1}; \alpha)$ using oracle A,*

$$|Prob[M^A(b_1, \cdots, b_{i-1}; \alpha) = b_i] - 1/2| = \Omega(\epsilon/m)$$

Proof. (of lemma 2.20) Theorem 2.21 asserts that if a test A is not fooled by a distribution $(b_1, \cdots, b_m)$ then A could be used as oracle to predict b_i (or 1-b_i) given $(b_1, \cdots, b_{i-1})$. We are now ready to construct the required Turing Machine. In our case the distribution $b_1, \cdots, b_m$ is the distribution of outputs of the generator. We use x and β to generate it as follows: We start by using x and β to construct a seed s for the generator. Let us denote the elements of S_i by $\{a_1 < \cdots < a_l\}$. For each $p \notin S_i$, we set $s_p = \beta_p$. We fill the remaining l places with x by setting $s_{a_v} = x_v$. We now set $b_1, \cdots, b_m$ to be the output of $NW^{f,\Delta}(s)$. (Note that s is uniformly distributed when i, x, and β are uniformly distributed).

We now have the following equalities.

- $b_i = f(x)$.
- For $j < i$, $b_j = f^{\Delta}_{i,j,\beta}(s|_{S_i \cap S_j})$.

We get that for all $j < i$, b_j can be computed from x and β using $f^{\Delta}_{i,j,\beta}$. Thus, having oracle to $f^{\Delta}_{i,\beta}$ we can compute $b_1, \cdots, b_{i-1}$. We now use the Turing machine of theorem 2.21 to give a good estimate on $b_i = f(x)$. □

The above lemma can be used to connect the complexity of the original function, that of the restricted functions, and the power of the generator.

Corollary 2.22. *If $NW^{f,\Delta}$ does not ϵ-fool A, then there are β and i so that:* $S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$.

Proof. By lemma 2.20 if $NW^{f,\Delta}$ does not ϵ-fool A, then there are α, β and i so that

$$|Prob_x[M^{f^{\Delta}_{i,\beta},A}(x, \alpha, \beta) = f(x)] - 1/2| = \Omega(\epsilon/m)$$

Thus, the output on a uniformly chosen x is correlated with $f(x)$. The equation above can be expressed in the following form: There is a bit $b \in \{0,1\}$ so that

$$Prob_x[M^{f^{\Delta}_{i,\beta},A}(x, \alpha, \beta) \oplus b = f(x)] \geq 1/2 + \Omega(\epsilon/m)$$

By "hard-wiring" α, b and the circuit for $f^{\Delta}_{i,\beta}$ to M, we get the circuit promised in the corollary. □

Corollary 2.22 gives that if $NW^{f,\Delta}$ does not fool A, then one of the restrictions can be used to show that f is "easy", (at least in the sense that there exists a small circuit that uses A gates and "approximates" f). This is useful since all the restrictions are on only $u + \log m$ bits. Thus, if we start with a function f that is "hard" for A, either the $NW^{f,\Delta}$ fools A or we obtain a "smaller" function that requires roughly the same circuit size as the original one. This observation can be used recursively. Using corollary 2.22 in the case that A is not fooled only *one* of the restrictions could be shown to be hard. This caused the initial version of this work, [ISW99] to use a costly exhaustive search to "find" the hard restriction.

However, it follows just the same from lemma 2.20 that in the case that A is not fooled by $NW^{f,\Delta}$ a non-negligible fraction of the restrictions are hard. This will enable us to select a small number of these restrictions using a randomized procedure such that with high probability we will obtain a hard restriction.

Corollary 2.23. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i:*
$S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$

An analogous argument can be used to state corollary 2.23 for Kolmogorov Complexity. It turns out that by considering Kolmogorov complexity the same argument gives much better parameters! This happens because in the Kolmogorov complexity setting, the running time of M doesn't count. We only have to specify α and b. Thus, we get an additive term and not a multiplicative term.

Corollary 2.24. *If $NW^{f,\Delta}$ does not ϵ-fool A, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i:*
$K_{A,1/2-\Omega(\epsilon/m)}(f) \leq K_A(f^{\Delta}_{i,\beta}) + 2m$

2.3.6 Xoring generators

We will recursively construct a sequence of candidate pseudo-random generators/extractors, one of which will fool the relevant predicate A. We will need a way of combining them. Fortunately, we will be able to guarantee that the number of such candidate generators is small. We'd like to "xor" all the generator's into one generator. A technical inconvenience is that the "xor generator" does not necessarily fool A. The following definition and trivial lemma enable us to overcome this difficulty.

Definition 2.25. *For a predicate A on m bits and $y \in \{0,1\}^m$, define a predicate $A^{\oplus y}$ on m bits by having $A^{\oplus y}(x) = A(x \oplus y)$. Define $A^{\oplus}$ to be the class of all predicates $A^{\oplus y}$.*

Lemma 2.26. *Let $P_1, \cdots, P_r$ be distributions on $\{0,1\}^m$ and $A : \{0,1\}^m \to \{0,1\}$ be a predicate. Suppose that one of the P_i's is ϵ-pseudo-random for $A^{\oplus}$. Consider the distribution $\bar{P} = P_1 \oplus \cdots \oplus P_r$, which samples independently $z_i \in_{P_i} \{0,1\}^m$, and outputs $z_1 \oplus \cdots \oplus z_r$. Then the distribution $\bar{P}$ ϵ-fools A.*

Proof. Let P_i be ϵ-pseudo-random for $A^{\oplus}$. Without loss of generality, assume $i = r$.

$$Pr_{z \in_{\bar{P}} \{0,1\}^m}(A(z) = 1) =$$

$$E_{z_1 \in_{P_1} \{0,1\}^m, \cdots, z_{r-1} \in_{P_{r-1}} \{0,1\}^m}, \left[Pr_{z_r \in_{P_r} \{0,1\}^m}(A(z_1 \oplus \cdots \oplus z_r) = 1) \right]$$

Fix $z_1, \cdots, z_{r-1}$, and let $y = z_1 \oplus \cdots \oplus z_{r-1}$. We know that P_r ϵ-fools $A \oplus y$, and therefore

$$|Pr_{z_r \in_{P_r} \{0,1\}^m}(A(z_1 \oplus \cdots \oplus z_r) = 1) - Pr_{z \in_R \{0,1\}^m}(A(z) = 1)| \leq \epsilon$$

Taking expectation over $z_1, \cdots, z_{r-1}$, we get that $\bar{P}$ ϵ-fools A. □

The price of lemma 2.26, is that if you have two candidate generators. $G_1 : \{0,1\}^{d_1} \to \{0,1\}^m$, $G_2 : \{0,1\}^{d_2} \to \{0,1\}^m$ the $\oplus$-generator $G(x_1, x_2) = G_1(x_1) \oplus G_2(x_2)$ takes a seed of length $d_1 + d_2$. This means that "xoring" many generators blows up the seed length. We want to only increase the seed length of a single generator linearly. We will be able to avoid increasing the total seed length by more than a constant factor over that of G_1, by making sure that the seed lengths are decreasing exponentially. In particular, this means that there can be only logarithmically many (in the seed length) elements in the sequence.

Let us rephrase corollaries 2.23, 2.24, and replace A by $A^{\oplus}$. This change does not affect the parameters by much. To convert a circuit using $A^{\oplus y}$ gates to one using A gates, we can replace the $A^{\oplus y}$ gates with A gates, and negate wires going to the i'th input of an $A^{\oplus y}$ gate if $y_i = 1$. This gives:

Corollary 2.27. *If $NW^{f,\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i: $S_{A,1/2-\Omega(\epsilon/m)}(f) \leq S_A(f^{\Delta}_{i,\beta})(m^{O(1)})$*

As to Kolmogorov complexity, note that by giving y, we can convert a machine with oracle access to $A^{\oplus y}$ into one with oracle access to A.

Corollary 2.28. *If $NW^{f,\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i: $K_{A,1/2-\Omega(\epsilon/m)}(f) \leq K_A(f^{\Delta}_{i,\beta}) + 4m$*

2.3.7 Hardness Amplification

The above connections relate the quality of the generator and the complexity of the specified restrictions to the complexity of *approximating* the function f, i.e., computing a function f' that has non-negligible correlation to f. Much of the work on improving the results in [NW94] concerns constructing a hard to approximate function from one that is hard to compute in the worst-case ([BFNW93, Imp95, IW97, STV99]). This process is usually called hardness amplification. Here, we'll use the hardness amplification from [STV99], which is nearly optimal.

Theorem 2.29. *[STV99] There exists a polynomial time algorithm that given a function $f : \{0,1\}^l \to \{0,1\}$ (encoded as a 2^l bit truth table) and ρ, produces the truth table of a function $\hat{f} : \{0,1\}^{4l} \to \{0,1\}$, with the following properties for any predicate A:*

1. $S_A(f) \leq S_{A,1/2-\rho}(\hat{f})(\frac{l}{\rho})^{O(1)}$
2. $K_A(f) \leq K_{A,1/2-\rho}(\hat{f}) + O(\log \frac{1}{\rho})$

Combining this with the corollaries from the last section gives the following analogous results for circuit complexity and Kolmogorov complexity. Note that again the same argument produces more efficient parameters in the Kolmogorov complexity setting.

Corollary 2.30. *If $NW^{\hat{f},\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i: $S_A(f) \leq S_A(\hat{f}^{\Delta}_{i,\beta})(m/\epsilon)^{O(1)}$*

Corollary 2.31. *If $NW^{\hat{f},\Delta}$ is not ϵ-pseudo-random for $A^{\oplus}$, then for an $\Omega(\epsilon/m)$ fraction of pairs β and i: $K_A(f) \leq K_A(\hat{f}^{\Delta}_{i,\beta}) + 4m + O(\log \frac{m}{\epsilon})$*

The above lemmas give the same intuitive connection between the hardness of the function f and its restrictions. The difference is that now we measure "worst-case" hardness in both sides of the inequality. This enables us to use these corollaries recursively.

2.4 Our construction

In our construction, we use the techniques from the previous sections in a recursive way. The idea is that if the NW generator fails on $\hat{f}$, then *many* of the restrictions are almost as complex as f was, and much smaller than f. Instead of producing candidate generators from all the restrictions, we give a randomized procedure which constructs a single smaller function by combining many randomly sampled restrictions. Since, the only way our construction can get random bits is by taking them from the seed, we use pairwise independence to minimize the number of bits actually used. At the end, we will have a relatively small number of candidate generators (one for each level of the recursion), and we combine them using the xor-construction of section 2.3.6.

We use the same construction for both extractors and pseudo-random generators, using our standard convention that boolean functions over l bits are equivalent to strings of length $n = 2^l$. Our construction is given as input the following ingredients:

- The length of strings in the source distribution n. (Or alternatively for pseudo-random generators $n = 2^l$ where l is the number of inputs of the hard function).
- The length of the output m.
- An explicit $(k', 1/m)$-extractor (if we are aiming to construct an extractor) or an explicit $(k', 1/m)$-pseudo-random generator scheme (if we are aiming to construct a pseudo-random generator) , $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \rightarrow \{0,1\}^m$. To be used as the base of the recursion. (We will soon connect k' which measures the quality of $Base$ to that of our construction).

We construct a function $Rec : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$. ($d$ will soon be determined). We now describe the computation of Rec:

It begins by computing three sequences of integers $l_1, \cdots, l_r, u_1, \cdots, u_r, d_1, \cdots, d_r$ recursively as follows:

1. $d_1 = c_3 \log n$, where c_3 is a constant to be fixed later.
2. $d_{t+1} = \frac{d_t}{2}$.
3. $l_1 = 4 \log n$.
4. $u_t = max(\frac{c_1(l_t)^2}{d_t}, \log m)$
5. $l_{t+1} = 4(u_t + 5 \log m)$.
6. r is such that $u_{r-1} = \log m$, (We will soon prove that such an r exists).

We view the first input of Rec as a boolean function f over l bits. It will be convenient to view the second input as composed of three parts: The first is used to obtain seeds for "candidate generators". We denote this part by $s = (s_1, \cdots, s_{r-1})$

where $s_t \in \{0,1\}^{d_t}$. (Note that the total length of s is bounded by $\sum d_t \leq 2d_1$. The second part which we denote by s' will be a d' bits string which is used as a seed for $Base$. The third part which is of length $8d_1$ will allow Rec to perform randomized computations. Thus, in this presentation, we allow Rec to use randomization as long as the number of random bits used does not exceed $8d_1$. This makes the seed length $d = 2d_1 + d' + 8d_1 = O(\log n) + d'$.

For $1 \leq t \leq r$, Rec constructs a (l_t, u_t)-design $\Delta_t = (S_1, \cdots, S_m)$ with $S_i \subseteq [d_t]$, using theorem 2.17.

Rec then constructs truth tables for r functions $f_1, \cdots, f_r$ defined as follows: $f_1 = f$. For $t = 1$ to $r-1$, we pick m^4 pairs $(i_q, \beta_q) \in [m] \times \{0,1\}^{d_t}$ in a pairwise independent way. This requires $2(d_t + \log m) \leq 4d_t$ random bits using the method from [CG88]. Thus, the total number of random bits used by Rec is no more than $4\sum_t d_t \leq 8d_1$ as required.

Define f_{t+1} by:

$$f_{t+1}(q,j,z) = (\hat{f}_t)^{\Delta_t}_{(i_q,\beta_q)}(j,z)$$

Note that the truth table of each f_t can be computed in polynomial time given the truth table of f_{t-1}, and that the size of inputs to f_t are $u_{t-1} + 5\log m$ bits. Thus, inputs to $\hat{f}_t$ are 4 times this length, or length l_t, as required.

It then sets $z_t = NW^{\hat{f}_t,\Delta_t}(s_t)$ for $t = 1, \cdots, r-1$, and $z_r = Base^{f_r}(s_r)$.

Finally, it outputs $z_1 \oplus z_2 \oplus ... \oplus z_r$.

In the remainder of this section we will show that our construction produces extractors and pseudo-random generators when given a good base. We start by showing that the recursion halts after very few steps.

Lemma 2.32. *For every n and m, the recursion computing l_t, u_t, d_t terminates, and when it does*

$$r \leq \log\log\frac{4\log n}{\log m} + 1$$

and $l_r = 24\log m$.

Proof. We'll count the number of steps in which u_t decreases, until it reaches the limitation of $u_t \geq \log m$. For $i < r-1$, $u_t = \frac{c_1(l_t)^2}{d_t}$. Using this rule, and the fact that $u_t \geq \log m$, we get that $u_t \leq (\frac{c_1}{c_3})^{2^{t-1}} 11^{a_t} 2^{b_t} 4\log n$, Where the sequences $\{a_i\}$ and $\{b_i\}$ are defined by the following recurrences: $a_1 = 2$, $a_{t+1} = 2(a_t+1)$, $b_1 = 0$, $b_{t+1} = 2b_t + t - 1$. One can compute that $a_t = O(2^{t-1})$, and $b_t = O(2^{t-1})$. Substituting this in the above equation we get that $u_t < (\frac{c_1 d}{c_3})^{2^{t-1}} \cdot 4\log n$, for some constant d. Choosing $c_3 = 2dc_1$, we get that: $u_t < 2^{-2^{t-1}} 4\log n$. This gives that $r \leq \log\log\frac{4\log n}{\log m} + 1$. Note that since $u_{r-1} = \log m$, $l_r = 24\log m$ $\square$

The above lemma implies that our construction can be computed in time polynomial in n (which is exponential in $l < d$). Our main lemma shows that this construction produces pseudo-random generators and extractors.

Lemma 2.33. *There exists some constant c such that for every k', if $Base$ is an explicit $(k', 1/m)$-pseudo-random generator scheme $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, then Rec is an explicit $(k'm^{cr}, 4/m)$-pseudo random generator scheme.*

Lemma 2.34. *There exists some constant c such that for every k', if $Base$ is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, then Rec is an explicit $(k'+crm, 4/m)$-extractor Rec : $\{0,1\}^n \times \{0,1\}^{O(\log n)+d'} \to \{0,1\}^m$.*

The argument proving both lemmas 2.33,2.34 is identical. We therefore give only the proof of lemma 2.34. (For the proof of lemma 2.33 one should only replace any use of corollary 2.31 by 2.30, and this difference causes the difference in the final value of m).

We use the Kolmogorov complexity characterization of extractors given in definition 2.9. Lemma 2.34 follows from the following lemma and theorem 2.10.

Lemma 2.35. *There is some constant c such that Rec is a $(k' + crm, 2/m)$-extractor scheme.*

Proof. Fix $f \in \{0,1\}^n$ and a predicate A on m bits such that $K_A(f) > k'$. We first need:

Lemma 2.36. *Let $1 \leq t \leq r-1$. With probability at least $1 - 1/m^2$ (over the choice of the m^4 pairs (i_q, β_q)), one of the following two events occur:*

- E_t^1 **:** *$NW^{\hat{f}_t, \Delta_t}$ is $1/m^2$-pseudo-random for $A^{\oplus}$.*
- E_t^2 **:** *$K_A(f_{t+1}) \geq K_A(f_t) - O(m)$.*

Proof. If E_t^1 does not occur, then by corollary 2.31, for a random (i, β), $K_A(f_t) \leq K_A((\hat{f}_t)^{\Delta_t}_{(i,\beta)}) + 5m$ with probability at least $\Omega(1/m^2)$. Let B be the set of such pairs. Recall that f_{t+1} "contains" each $(\hat{f}_t)^{\Delta_t}_{(i_q,\beta_q)}$ for $1 \leq q \leq m^4$. Therefore, if any $(i_q, \beta_q) \in B$, then $K_A(f_t) \leq K_A(f_{t+1}) + O(m)$. For each q, there is a chance of $\Omega(1/m^2)$ that $(i_q, \beta_q) \in B$. Having chosen m^4 such pairs we expect $\Omega(m^2)$ of them to fall in B. Since these events are pairwise independent we can use Chebyshev's inequality and get that the probability that all the pairs (i_q, β_q) "miss" B, is $O(1/m^2)$. □

We define $E^1 = \cup_{1 \leq t \leq r-1} E_t^1$ and $E^2 = \cap_{1 \leq t \leq r-1} E_t^2$. Note that for all $1 \leq t \leq r-1$ the events E_t^1 and E_t^2 in the above lemma are determined before the seeds to the NW-generators are chosen. By the above lemma the probability that none of the events E^1,E^2 occurred, is bounded by $O(r/m^2) < 1/m$. If E^1 occurs then the conditional bias of Rec^f for A is at most $1/m$ by Lemma 2.26. If E^2 occurs then we can connect the hardness of f_t and f_{t+1} for all $1 \leq t \leq r-1$, and get that f_r is hard. Specifically, there exists a constant c such that $K_A(f_r) \geq K_A(f_1) - c(r-1)m \geq k' + 2m$. Using lemma 2.11 we get that when E^2 occurs, $Base^{f_r}$ $1/m$-fools A. (Verify that by lemma 2.32 $l_r = 24 \log m$, so f_r is an appropriate input for $Base$). However, in order to use lemma 2.26, we need to prove that $Base^{f_r}$ is $1/m$-pseudo-random for $A^{\oplus}$. This follows just the same since for every $y \in \{0,1\}^m$, $K_{A \oplus y}(f_r) \geq K_A(f_r) - m - O(1)$. By lemma 2.26 we get that if E^2 occurred then the conditional bias of Rec^f on A is at most $1/m$. This makes the total bias of Rec^f on A at most $1/m$ plus the probability that non of the events E^1,E^2 occurred, and the lemma follows. $\square$

2.5 Deriving the results

In this section we show how "plugging" different bases to our construction gives the previously stated theorems. The generator of theorems 2.1, 2.7 is achieved by Using the generator of [STV99] as $Base$.

Proof. (of theorems 2.1, 2.7) We use the pseudo-random generator scheme of [STV99] (theorem 2.6) as $Base$. We choose $k' = m^a$ (where a is the constant "hidden" in the Omega notation of theorem 2.6). Indeed, the generator of [STV99] is a $(k', 1/m)$-pseudo random generator scheme, and is $1/m$-pseudo-random for size m when given a function $g : \{0,1\}^{24 \log m} \rightarrow \{0,1\}$ which cannot be computed by circuits of size k'. The seed length of the generator is $d' = O(l)$. Using theorem 2.33 we get that Rec is a $(k, 1/m)$-pseudo random generator scheme for $k = m^a m^{cr} = m^{O(\log\log(l/\log k))}$ and in particular Rec^f is $4/m$-pseudo-random for size m when given a function $f : \{0,1\}^l \rightarrow \{0,1\}$ which cannot be computed by circuits of size k. The seed length of Rec is $O(l) + d' = O(l)$. $\square$

Theorem 2.12 follows using Trevisan's extractor as $Base$.

Proof. (of theorem 2.12) We use the extractor of [Tre99] as $Base$. Given a constant δ, we pick $k' = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}}$. The extractor of [Tre99] is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \rightarrow \{0,1\}^m$, where $d' = O(\log m)$. Using theorem 2.34 we get that Rec is an explicit $(k, 4/m)$-extractor, for $k = \frac{1}{2} \cdot m^{\frac{1}{1-\delta}} + crm < m^{\frac{1}{1-\delta}}$. The seed length of Rec is $O(\log n) + d' = O(\log n)$. $\square$

Theorem 2.13 follows using the extractor of [RRV99b] as $Base$.

Proof. (of theorem 2.13) We use the extractor of [RRV99b] as $Base$. We pick $k' = 2m$. The extractor of [RRV99b] is an explicit $(k', 1/m)$-extractor $Base : \{0,1\}^{m^{24}} \times \{0,1\}^{d'} \to \{0,1\}^m$, where $d' = O(\log^2 m)$. Using theorem 2.34 we get that Rec is an explicit $(k, 4/m)$-extractor, for

$$k = 2m + O(rm) = O(m \log\log(\log n / \log k))$$

as required. The seed length of Rec is $O(\log n) + d' = O(\log n + \log^2 k)$. $\square$

Chapter 3

Extracting randomness via repeated condensing

3.1 Introduction

On an input probability distribution which contains some randomness an *extractor* outputs a distribution with a (near) maximum entropy rate (namely the uniform distribution). We use the term "entropy rate" to denote the ratio between k (the number of random bits the source contains) and n (it's length). A natural weakening of this concept is a *condenser*, whose output distribution has a higher entropy rate than the input distribution[1].

In this chapter we construct efficient explicit condensers. The condenser constructions combine (variants or more efficient versions of) ideas from several works, including the block extraction scheme of [NZ96], the observation made in [SZ99, NTS99] that a failure of the block extraction scheme is also useful, the recursive "win-win" case analysis of [ISW99, ISW00], and the error correction of random sources used in [Tre99].

As a natural byproduct (via repeated iterating of condensers), we obtain new extractor constructions. The new extractors give significant qualitative improvements over previous ones for sources of arbitrary min-entropy; they are nearly optimal *simultaneously* in the main two parameters - seed length and output length. Specifically, our extractors can make any of these two parameters optimal (up to a constant factor), only at a *poly-logarithmic* loss in the other. Previous constructions require *polynomial* loss in both cases for general sources.

[1] The notion of condensers was also used in [RR99]. While similar in spirit, that paper deals with a completely different set of parameters and uses different techniques.

3.1.1 Results

We give two constructions, each optimal in one of the parameters and losing only a poly-logarithmic factor in the other. This should be compared to previous constructions (see table 1.3) which lose a polynomial factor when trying to optimize one of the parameters. Thus, both constructions come closer to simultaneously optimizing both parameters. The results are stated for constant ϵ (see section 3.8 for the exact dependence on ϵ). In the first construction we extract any constant fraction of the initial randomness using a seed of length $O(\log n \cdot (\log\log n)^2)$. This improves the best previous such result by [RRV99b] which uses a seed of length $O(\log^2 n)$.

Theorem 3.1. *For every n, k and constant ϵ, there are explicit (k, ϵ)-extractors $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n \cdot (\log\log n)^2)} \to \{0,1\}^{(1-\delta)k}$, where $\delta > 0$ is an arbitrary constant.*

Using the method of [WZ99] (see section 1.5.7) we get the following corollary[2], which also improves the previous best construction which extract all the randomness by [RRV99b].

Corollary 3.2. *For every n, k and constant ϵ, there are explicit (k, ϵ)-extractors $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n \cdot (\log\log n)^2 \cdot \log k)} \to \{0,1\}^k$.*

Our second construction uses the optimal seed length (that is $O(\log n)$) to extract $m = \Omega(k/\log n)$ bits, this improves the best previous result by [ISW00] which could only extract $m = k^{1-\delta}$ bits.

Theorem 3.3. *For every n, k and constant ϵ, there are explicit (k, ϵ)-extractors $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^{\Omega(k/\log n)}$.*

Using the construction from chapter 2, we get the following corollary (in which the "loss" depends only on k)[3].

Corollary 3.4. *For every n, k and constant ϵ, there are explicit (k, ϵ)-extractors $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^{\Omega(k/(\log k \cdot \log\log k))}$.*

[2] In fact, the version of theorem 3.1 stated above does not suffice to conclude the corollary. To use the method of [WZ99] we need a version where the error is $1/polylogn$ which follows follows just the same from our analysis, see section 3.8.

[3] Theorem 2.34 show that an extractor $Ext : \{0,1\}^{k^{O(1)}} \times \{0,1\}^d \to \{0,1\}^m$ can be used to construct an extractor $Ext : \{0,1\}^n \times \{0,1\}^{d+O(\log n)} \to \{0,1\}^{\Omega(m/\log\log k)}$ for any n.

3.1.2 Organization of this chapter

In section 3.2 we give an informal overview of the technique used in this chapter. In section 3.3 we define block sources. In section 3.4 we construct a block extraction scheme. In section 3.5 we use the method of [NTS99] to show that when using the block extraction scheme either we get a block source or we condense the source. In section 3.6 we run the block extraction scheme recursively and obtain condensers. In section 3.7 we use the condensers to construct extractors. Section 3.8 gives the exact dependence of our extractors on the error parameter ϵ.

3.2 Overview of the technique

3.2.1 High level overview

In contrast to latest extractors papers [Tre99, RRV99b, ISW00] we do not use Trevisan's paradigm. Instead, we revisit [NZ96] and attempt to construct block sources (A special kind of sources which allow very efficient extraction). Following [SZ99, NTS99], we observe that when failing to produce a block source the method of [NZ96] "condenses" the source. This enables us to use a "win-win" case analysis as in [ISW99, ISW00] which eventually results in a construction of a condenser. Our extractors are then constructed by repeated condensing. The parameters of these extractors are not as good as those stated in section 3.1.1. In order to get the promised parameters, we improve the construction of [NZ96].

3.2.2 Block sources

One scenario in which very efficient extraction is possible, is when the source distribution X is made of two independent concatenated distributions $X = (X_1, X_2)$, where X_1 is a k_1-source and X_2 is a k_2-source. Extractors for this special scenario (which are called block-source extractors) can be constructed by composing two extractors: An extractor with optimal seed length can be used to extract random bits from X_2, and these bits (being independent of X_1), can be used as seed to extract all the randomness from X_1 using an extractor with large output. (Note, that with today's best extractors this argument uses the optimal seed length to extract all the randomness from X_1, as long as k_2 is at least $polylog(n)$). The requirement that X_1 and X_2 be independent could be relaxed in the following way (that was suggested in [CG88]): Intuitively, it is sufficient that X_1 contains k_1 random bits, and X_2 contains k_2 random bits which are not contained in X_1. Such sources are called block-sources[4]. Thus, extracting randomness from general sources can be achieved by giving a construction which uses few random bits to transform a general source into a block source, and using a block-source extractor.

This approach was suggested by Nisan and Zuckerman in [NZ96]. They constructed a "block extraction scheme", that is a scheme which given an arbitrary source X, uses few random bits to produce a new source B (called a block) which is shorter than the initial source, and contains a large fraction of the initial randomness. This means that the distribution (B, X) meets the first requirement of block sources: The first block contains randomness. Intuitively, to meet the second requirement one should give an upper bound on the amount of randomness contained in B, and conclude that there is some randomness in X which is not contained in B. However, in the construction of Nisan and Zuckerman such an upper bound can be achieved only "artificially", by making the length of B smaller than k. This has a costly effect, as the amount of randomness that is guaranteed to be in B is proportional to its length. In particular, when $k < \sqrt{n}$ choosing the length of B this way, it may be the case that B contains no randomness. As a result, the extractors of [NZ96] do not work when $k < \sqrt{n}$.

3.2.3 A "win-win" analysis

A way to go around this problem was suggested in [SZ99, NTS99]. The idea is to argue that when the block extraction scheme fails to produce a block source, it produces a block B which is more "condensed" than the initial source. More precisely, we will use the block extraction scheme to produce a block B (say of length $n/2$). Consider the distribution (B, X). Recall that for our purposes, it suffices that X contains very few random bits that are not contained in B. In other words, when we fail to construct a block source, this happens because (almost) all the randomness "landed" in B. In this case, we obtained a block that is more condensed than the initial source X. (It has roughly the same amount of randomness and half the length).

Using this idea recursively, at each step either we construct a block source (from which we can extract randomness), or we condense the source. There is a limit on how much we can condense the source. Certainly, when the length reduces to k, no further condensing is possible. This means that running this procedure recursively enough times we will obtain a block source.

The outcome of the above procedure is several "candidate" distributions, where one of them is a block source. Not knowing which is the "right one", we run

[4]More precisely, a (k_1, k_2)-block source is a distribution $X = (X_1, X_2)$ such that X_1 is a k_1-source, and for every possible value x_1 of X_1, the distribution of X_2, conditioned on the event that $X_1 = x_1$, is a k_2-source.

block source extractors on all of them (using the same seed). We know that one of the distributions we obtain is close to uniform. It turns out that the number candidate distributions is relatively small. This means that concatenating the output distributions produces a new source which is much shorter than the initial source and contains a large fraction of the initial randomness. Thus, we have constructed a condenser. Our extractors are then constructed by repeatedly condensing the source until it becomes close to uniform.

A very similar approach has already been used in [NTS99]. (Actually, the construction in that paper is far more complex, partly because the extractors they had available were not as good as the ones we have now). Still, (even when plugging in very good extractors), the method explained above is too costly and ends up constructing an extractor that uses a *large* seed and extracts a *small* fraction of the randomness. We overcome this problem by improving the main primitive in our construction: the block extraction scheme of Nisan and Zuckerman.

3.2.4 Improved block extraction

Let us delve into the parameters. The block extraction scheme of Nisan and Zuckerman spends $O(\log n)$ random bits when producing a block B of length $n/2$, and can guarantee that B is an $\Omega(k/\log(n/k))$-source. This turns out to be too costly to run our recursive construction.

One problem is that the number of random bits used by the block extraction scheme is too large for our purposes. Since the block extraction scheme already spends $O(\log n)$ random bits, we can only afford to run it a constant number of times if we want to shoot for optimal seed length. Using the strategy described above we will need to run it roughly $\log n$ times, resulting in a large seed length. We overcome this problem by derandomizing the construction of Nisan and Zuckerman. We reduce the number of random bits used from $\log n$ to $\log\log n$, allowing us to run it a larger number of times. We find it surprising that some non-trivial extraction task can be performed using a sub-logarithmic amount of random bits.

A second problem is that we want the block B to contain a constant fraction of the initial randomness. In the construction of Nisan and Zuckerman this happens only when $k = \Omega(n)$. To overcome this problem we show how to reduce the case of $k = o(n)$ into the case of $k = \Omega(n)$. This is done by error correcting the source prior to using the block extraction scheme. We give a non-constructive argument to show that every error corrected random source has a "piece" of length k which is an $\Omega(k)$-source. Intuitively, this enables the block extraction scheme to be carried out on the condensed piece, where it performs best[5].

[5] As far as we know, the idea of error correcting random sources (in the context of extracting

3.3 Block sources

Block sources are random sources which have a special structure. The notion of block sources was defined in [CG88].

Definition 3.5. *[CG88] Two random variables (X_1, X_2) form a (k_1, k_2)-block source if X_1 is a k_1-source, and for every possible value x_1 of X_1 the distribution of X_2, given that $X_1 = x_1$, is a k_2-source.*

Block source extractors are extractors which work on block sources.

Definition 3.6. *A (k, t, ϵ)-block source extractor is a function $Ext : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^d \to \{0,1\}^m$, such that for every (k,t)-block source (X_1, X_2), (where the X's are of length n_1,n_2 respectively), the distribution $Ext(X_1, X_2, U_d)$ is ϵ-close to U_m.*

Block sources allows the following composition of extractors.

Theorem 3.7. *(implicit in [NZ96]) If there exist an explicit (k, ϵ_1)-extractor $Ext_1 : \{0,1\}^{n_1} \times \{0,1\}^{d_1} \to \{0,1\}^m$, and an explicit (t, ϵ_2)-extractor $Ext_2 : \{0,1\}^{n_2} \times \{0,1\}^{d_2} \to \{0,1\}^{d_1}$, then there exists an explicit $(k, t, \epsilon_1 + \epsilon_2)$-block-source extractor $Ext : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^{d_2} \to \{0,1\}^m$, where $Ext(x_1, x_2; y) = E_1(x_1, E_2(x_2, y))$.*

We can use the above theorem to compose two extractors: one which optimizes the seed length and another which optimizes the output length. The resulting block-source extractor will "inherit" the nice properties of both its component extractors. Particularly, taking Ext_1 to be the extractor of [RRV99b] and Ext_2 to be the extractor of [ISW00], we get the following block-source extractor:

Corollary 3.8. *For every n_1, n_2, k and $t \geq \log^4 n_1$ there exists an explicit $(k, t, \frac{1}{n_1} + \frac{1}{n_2})$-block source extractor $BE : \{0,1\}^{n_1} \times \{0,1\}^{n_2} \times \{0,1\}^{O(\log n_2)} \to \{0,1\}^k$.*

Remark 3.9. *In the rest of the chapter we assume that $k \geq \log^4 n$. We need this assumption since we're going to transform a k-source into a block-source in which the second block contains $\log^4 n$ bits. We can make this assumption because for $k = 2^{O(\sqrt{\log n})}$, theorems 3.1,3.3 follow from the constructions in chapter 2 (see theorem 2.3 and the discussion after it).*

Thus, to construct extractors which achieve short seed length and large output simultaneously, it suffices to use few random bits, and convert any k-source into a $(k', \log^4 n)$-block source such that k' is not much smaller than k.

This turns out to be a tricky problem, no such (efficient in terms of random bits spent) scheme is known when $k < \sqrt{n}$.

randomness) first appeared in [Tre99]. Interestingly, we use error correcting codes with slightly different properties, and use a different method in analyzing them.

3.4 Improved block extraction

Current constructions of block sources rely on a building block which we will call a "block extraction scheme". Intuitively, this is a scheme which given an arbitrary source X, uses few random bits to produce a new source B (called a block) which is shorter than the initial source, and contains a large fraction of the initial randomness. As explained in the introduction, given such a scheme, we may hope that B does not contain all the randomness of X, which intuitively means that the distribution (B, X) is a block source.

The following definition formalizes the notion of a block extraction scheme.

Definition 3.10. *A $(n, k, u, r, \gamma, \rho)$-block extraction scheme is a function:*

$$B : \{0,1\}^n \times \{0,1\}^u \to \{0,1\}^{\frac{n}{r}}$$

Such that for every k-source X on $\{0,1\}^n$, the distribution $B(X, Y)$ (obtained by sampling X from the source, and y uniformly from $\{0,1\}^u$), is ρ-close to a $\frac{\gamma k}{r}$-source.

Intuitively, the block extraction scheme spends u random bits in order to produce a block B of length n/r, which is guaranteed to be ρ-close to a $\gamma k/r$-source[6].

Using this notation, Nisan and Zuckerman proved the following theorem:

Lemma 3.11. *[NZ96] There exists an explicit $(n, k, O(\log n \log \frac{1}{\rho}), r, \Omega(\frac{1}{\log \frac{n}{k}}), \rho)$-block extraction scheme, as long as $\rho \geq 2^{-ck}$ for some constant c.*

The following lemma improves lemma 3.11.

Lemma 3.12. *There exists an explicit $(n, k, O(\log\log n + \log(1/\rho) + \log r), r, \Omega(1), \rho)$-block extraction scheme as long as $\rho \geq c\sqrt{\frac{r}{k}}$ for some constant c.*

Lemma 3.12 improves lemma 3.11 in two respects (as long as one settles for small r and large ρ).

1. We reduce the number of random bits spent by the block extraction scheme. In [NZ96] the number of random bits is logarithmic in n, whereas in lemma 3.12 the number of random bits is double logarithmic in n.

   This is achieved by derandomizing the proof of Nisan and Zuckerman using small bias spaces. In section 3.4.2 we describe the property that a distribution needs to have in order to allow the Nisan-Zuckerman analysis, and construct a small sample space with this property.

[6] The block extraction scheme of [NZ96] has a stronger property, namely that for a $(1-\rho)$-fraction of the y's, $B(X, y)$ is ρ-close to a $\frac{\gamma k}{r}$-source. This property is also shared by our new construction, but is not needed later in the analysis. Another interesting property of both constructions is that they do not depend on k.

2. We increase the amount of randomness guaranteed in the output block. In [NZ96] the amount of randomness guaranteed in the output block B is $\Omega(\frac{k}{r\log(n/k)})$. Lemma 3.12 guarantees that B contains $\Omega(\frac{k}{r})$ random bits.

   Note that the two quantities are the same when $k = \Omega(n)$. Indeed, our improvement is achieved by reducing the case of $k = o(n)$ to that of $k = \Omega(n)$. In section 3.4.1 we show that once a random source is error corrected, there are some k indices (to the error corrected source) which induce an $\Omega(k)$-source. Thus, by error correcting the source we can focus on a small portion of it in which the ratio between min-entropy and length is constant. The argument of Nisan and Zuckerman is then carried out only on this portion of the source where it performs best. The exact analysis is given in section 3.4.3.

3.4.1 Error corrected random sources

In this subsection we show that if we apply an error correcting code to an arbitrary k-source, we obtain a k-source which has k indices which induce an $\Omega(k)$-source.

We use the following construction of error correcting codes.

Theorem 3.13. *[Jus72] There exist constants a, b and an explicit error correcting code $EC : \{0,1\}^n \to \{0,1\}^{an}$ such that for every $x_1 \neq x_2 \in \{0,1\}^n$, $d(EC(x_1), EC(x_2)) \geq bn$. ($d(z_1, z_2)$ denotes the Hamming distance between z_1, z_2).*

In the remainder of this section we fix a, b and EC to be these of theorem 3.13. For a vector $x \in \{0,1\}^n$ and a set $T \subseteq [n]$ we use $x|_T$ to denote the restriction of x to T.

Lemma 3.14. *Let X be a k-source on $\{0,1\}^n$. There exists a set $T \subseteq [an]$ of size k, such that $EC(X)|_T$ is an $\Omega(k)$-source.*

Lemma 3.14 is an immediate corollary of lemma 3.15 which was mentioned to us by Russell Impagliazzo and seems to be folklore.

Lemma 3.15. *[Imp99] Let X be a k-source on $\{0,1\}^n$. For every v, there exists a set $T \subseteq [an]$ of size v, such that $EC(X)|_T$ is a $\frac{1}{2} \cdot \log 1/(2^{-k} + (1 - \frac{b}{a})^v)$-source.*

For this lemma we need another notion of measuring the amount of randomness in a distribution.

Definition 3.16. *For a distribution P over Ω, define the L_2-norm of P: $C(P) = \sum_{\omega \in \Omega} P(\omega)^2$. In words, $C(P)$ is the probability that two independent samples from P gave the same outcome. We refer to $C(P)$ as the collision probability of P.*

A distribution with low min-entropy has an element which gets large probability. This element has a large chance of appearing in two consecutive independent samples. This gives the following connection.

Fact 3.17. *If $C(P) \leq 2^{-k}$ then P is a $k/2$-source.*

Our goal becomes showing that there exists a subset of $[an]$ on which the error corrected source has low collision probability. We will show that a random (multi)-set has this property.

Proof. (of lemma 3.15) Consider the following probability space: x_1, x_2 are independently chosen from the distribution X, and $T = (i_1, .., i_v)$ are chosen independently where each i_j is uniformly distributed in $[an]$. Denote

$$B = \{(x_1, x_2, T) | EC(x_1)|_T = EC(x_2)|_T\}$$

We now bound the probability of B in the above sample space.

$$\Pr(B) = \Pr(B|x_1 = x_2)\Pr(x_1 = x_2) +$$

$$\sum_{a_1 \neq a_2} \Pr(B|x_1 = a_1, x_2 = a_2)\Pr(x_1 = a_1, x_2 = a_2) \tag{3.1}$$

X is a k-source, and therefore $\Pr(x_1 = x_2) \leq 2^{-k}$. For given $a_1 \neq a_2$, we know that the distance between $EC(a_1)$ and $EC(a_2)$ is at least bn. Thus, any of the i_j's has a chance of $\frac{b}{a}$ to "hit" a coordinate where $EC(a_1)$ and $EC(a_2)$ disagree. Having chosen v such coordinates the probability that none of them differentiated between $EC(a_1)$ and $EC(a_2)$ is bounded by $(1 - \frac{b}{a})^v$. Plugging this in 3.1 we get that

$$\Pr(B) \leq 2^{-k} + (1 - \frac{b}{a})^v$$

In the sample space we considered, T was chosen at random. Still, there is a fixing T' of the random variable T for which the above inequality holds. For this T' we have that the probability that independently chosen x_1 and x_2 have $EC(x_1)|_{T'} = EC(x_2)|_{T'}$ is small. In other words we just proved that

$$C(EC(X)|_{T'}) \leq 2^{-k} + (1 - \frac{b}{a})^v$$

The lemma immediately follows from fact 3.17. $\square$

3.4.2 A small sample space for intersecting large sets

The block extraction scheme of Nisan and Zuckerman is obtained by restricting the source to some subset of the indices which is selected using few random bits. More precisely, they construct a small sample space of small subsets of $[n]$ (having a property that we immediately describe) and prove that the distribution obtained by sampling an element from a k-source and restricting it to the indices in a random set from the sample space contains a large fraction of the initial randomness. In this section we construct a smaller such sample space which enables us to spend less random bits to construct a block extraction scheme.

It will be convenient to have a notion of efficiently constructible distributions.

Definition 3.18. *Call a distribution P on n bits, polynomially constructible using $u(n)$ bits[7], if there exists an algorithm $A : \{0,1\}^{u(n)} \to \{0,1\}^n$ which runs in time polynomial in n, such that the distribution $A(Y)$ where Y is chosen uniformly from $\{0,1\}^{u(n)}$ is identical to P.*

Intuitively, a k-source has k random bits "hidden" somewhere. This suggests that the property needed is that the distribution of sets will "intersect" any set of k indices.

Definition 3.19. *A distribution S over subsets of $[n]$ is called (n,k,r,δ)-intersecting if for every $G \subseteq [n]$ with $|G| \geq k$, $\Pr_S(|S \cap G| < \frac{k}{8r}) < \delta$.*

The following is implicit in [NZ96][8].

Lemma 3.20. *[NZ96] There exists some constant c such that if X is a k-source on $\{0,1\}^n$ and S is a distribution over subsets of $[n]$ which is $(n, \frac{ck}{\log(n/k)}, r, \delta)$-intersecting then the distribution $X|_S$ (obtained by sampling x from X and s from S and computing $x|_s$) is $(4\sqrt{\delta} + 2^{-\Omega(k)})$-close to a $\Omega(\frac{k}{r\log(n/k)})$-source.*

Nisan and Zuckerman use a construction based on $O(\log(1/\delta))$-wise independence to prove the following lemma.

Lemma 3.21. *[NZ96] For every n, k, r and $\delta > 2^{-O(k/r)}$ there is a (n,k,r,δ)-intersecting distribution on subsets of size n/r which is polynomially constructible using $O(\log n \cdot \log(1/\delta))$ bits.*

---

[7] Naturally, one should speak about a sequence $P = \{P_n\}$ of distributions for this to make sense.

[8] It should be noted that the argument of [NZ96] is not as straightforward as the above intuition suggests. In fact, the proof is quite involved and a hint to that may be found in the fact that the distribution needs to intersect sets of size $\frac{k}{\log(n/k)}$ and not k.

Using lemma 3.20 this immediately implies the block extraction scheme of lemma 3.11. We will be mostly interested in the case when r is small (say $r \leq \log n$) and δ is large (say $\delta \geq \log^{-O(1)} n$). For this setup we can save random bits and make the dependence on n double logarithmic.

Lemma 3.22. *There exists a constant c such that for every n, k, r and $\delta > cr/k$, there is a (n, k, r, δ)-intersecting distribution on subsets of size n/r which is polynomially constructible using $O(\log \log n + \log r + \log(1/\delta))$ bits.*

In the rest of this section we prove lemma 3.22. We will view distributions over n bit strings as distributions over subsets of $[n]$. (Using the standard correspondence $(W_1, \cdots, W_n) \leftrightarrow \{i | W_i = 1\}$). We will construct a distribution $(W_1, \cdots, W_n)$ with the following properties:

- For every $1 \leq i \leq n$, $\Pr(W_i = 1) \approx 1/2r$.
- For every set $G \subseteq [n]$ with $|G| \geq k$, the probability that the sum of the W_i's for $i \in G$ is far from the expected $|G|/2r$ is small. (It is important to note that we allow the "small probability" to be quite large, since we are shooting for large δ).

Note that the second condition gives both the intersecting property and the fact that the selected sets are rarely of size larger than n/r. We are interested in constructing such a distribution using as few as possible random bits. A pairwise independent distribution has these properties but takes $\log n$ random bits to construct. We can do better by using the "almost l-wise dependent" distributions of [NN90]. A technicality is that we want a distribution in which the probability that a single bit evaluates to 1 is close to $1/2r$ and not to $1/2$.

Let us define almost l-wise dependent distributions.

Definition 3.23. *([NN90]) A distribution $(P_1, \cdots, P_n)$ over $\{0,1\}^n$ is said to be (ϵ, l)-wise dependent with mean p if for every subset $\{i_1, \cdots, i_l\}$ of $[n]$, the distribution $(P_{i_1}, \cdots, P_{i_l})$ is ϵ-close to the distribution over l bit strings where all bits are independent and each of them takes the value 1 with probability p.*

Naor and Naor showed that almost l-wise dependent distributions can be constructed using very few random bits.

Theorem 3.24. *[NN90] For every n, l and ϵ, an (ϵ, l)-wise dependent distribution with mean $1/2$ is polynomially constructible using $O(\log \log n + l + \log(1/\epsilon))$ bits.*

The following corollary follows easily by taking an almost $2q$-dependent distribution over qn bits (with mean close to $1/2$), viewing it as n blocks of q bits, and setting the variable of the i'th block to the value 1 if all the variables in the corresponding block are 1.

Corollary 3.25. *For every n, ϵ and q, an $(\epsilon, 2)$-wise dependent distribution with mean 2^{-q} is polynomially constructible using $O(\log\log n + q + \log(1/\epsilon))$ bits.*

We are ready to construct our intersecting distribution.

Definition 3.26. *Let q be an integer such that $1/4r < 2^{-q} \leq 1/2r$, and $\epsilon = min(c\delta^2, c/r^2)$ where c is a constant which will be determined later. Let $W = (W_1, \cdots, W_n)$ be the $(\epsilon, 2)$-wise dependent distribution with mean 2^{-q} guaranteed in corollary 3.25.*

Almost pairwise dependent distributions are very much like pairwise independent distributions in the sense that they allow the use of Chebychev's inequality.

Lemma 3.27. *[NN90] If $(X_1, \cdots, X_n)$ is a $(\epsilon, 2)$-wise dependent distribution with mean p, then for every $0 < \lambda < 1$*

$$\Pr(|\sum_{1\leq i\leq n} X_i - pn| > \lambda pn) < O(\frac{1}{\lambda^2 pn} + \sqrt{\epsilon})$$

as long as $\epsilon < \frac{\lambda^2 p^2}{4}$.

The next lemma follows:

Lemma 3.28. *The constant c in definition 3.26 can be fixed in a way in which the distribution W has the following properties:*

1. *For every set $G \subseteq [n]$ such that $|G| \geq k$, $\Pr(\sum_{i\in G} W_i < \frac{k}{8r}) < \delta/3$.*
2. $\Pr(\sum_{1\leq i\leq n} W_i > \frac{n}{r}) < \delta/3$.

Proof. We use lemma 3.27 to deduce both parts of the lemma. To meet the condition in lemma 3.27 we need to make sure that $\epsilon < \frac{\lambda^2 p^2}{4} = \Theta(1/r^2)$. The requirement that $\epsilon < c/r^2$ takes care of this condition for small enough constant c. Applying lemma 3.27 we get that the probability of deviation from the mean (in both parts of the lemma) is bounded by $O(r/k + \delta\sqrt{c}) = O(\delta\sqrt{c})$. (The last equality follows from the condition of lemma 3.22). This is bounded from above by $\delta/3$ for small enough c. □

We are ready to prove lemma 3.22

Proof. (of lemma 3.22) The first item of lemma 3.28 shows that W is $(n, k, r, \delta/3)$-intersecting, and the second item shows that W could be transformed into a distribution over subsets of size exactly n/r without changing it by much. This change is done by adding arbitrary indices to the set if its size is smaller than n/r and deleting arbitrary indices if its size is larger than n/r. Adding indices will not spoil the intersecting property, and the probability that we need to delete indices is bounded by $\delta/3$. The lemma follows. □

3.4.3 Construction of block extraction scheme

In this subsection we put everything together and prove lemma 3.12. We are ready to define our block extraction scheme.

Definition 3.29. *(block extraction scheme) Given r, ρ and a constant e (which will be fixed later) let $u = O(\log\log n + \log r + \log(1/\rho))$ be the number of bits used by lemma 3.22 to construct a $(an, ek, ar, (\frac{\rho}{4})^2)$-intersecting distribution. For $y \in \{0,1\}^u$, let S_y be the set defined by y in the intersecting distribution. We now define:*

$$B(x,y) = EC(x)|_{S_y}$$

We are finally ready to prove lemma 3.12.

Proof. (of lemma 3.12) Let V denote the distribution $EC(X)$. Lemma 3.14 implies that there exists a set $T \subseteq [an]$ of size k such that $V|_T$ is a dk-source (for some constant d). Consider the distribution $S \cap T$ (the restriction of the intersecting distribution to the coordinates of T). It is trivial (from the definition) that this distribution is $(k, ek, ar, (\frac{\rho}{4})^2)$-intersecting. We now use lemma 3.20 on $V|_T$. Let us first check that the conditions of lemma 3.20 are met. We fix the constant e of definition 3.29, setting $e = (cd)/(-\log d)$, where c is the constant from lemma 3.20. The conditions of lemma 3.20 are met since $V|_T$ is a dk-source of length k and we have a distribution which is intersecting sets of size $ek = \frac{c(dk)}{\log(k/(dk))}$. We conclude that $V|_{S\cap T}$ is ρ-close to an $\Omega(k/r)$-source. Revealing more indices does not decrease the amount of randomness and thus $B(X,Y) = V|_S$ is ρ-close to an $\Omega(k/r)$-source. □

3.5 Partitioning to two "good" cases

In this section we show that using the block extraction scheme B (defined in section 3.4), essentially one of two "good" things happen. Either $(B(X,\cdot), X)$ is a block source or $B(X,\cdot)$ is much more condensed than X. Unfortunately, the above claim is not true. For some sources it may be the case that none of the two "good" cases applies[9]. However, this problem has already been solved in [NTS99]. The solution is to prove a weaker statement which has the same flavor, and can be used the same way later on. The exact statement is that every source can be partitioned into three

[9] An example is a source which flips a bit b and depending on the outcome decides weather to sample from a distribution X_1 in which the first k bits are random and the remaining $n-k$ bits are fixed, or from a distribution X_2 which is k-wise independent. X_1 corresponds to the first good case whereas X_2 corresponds to the second. However, for any of the two cases, with probability $1/2$ the distribution corresponding to the other case was chosen.

sets: The first has negligible weight and can be ignored. On the second, the block extraction scheme produces a block source, and on the third, the block extraction scheme condenses the source. To make this formal, we introduce the following notation:

Definition 3.30. *For a distribution P over a set Ω and $A \subseteq \Omega$ such that $P(A) \neq 0$, we define the conditional distribution P_A on Ω which gives weight $P(\omega)/P(A)$ for $\omega \in A$, and zero otherwise.*

We say that a random variable U on Ω, is a k-source in A, if the distribution $U(\omega)$, where ω is chosen according to P_A, is a k-source.

We say that two random variables U, V on Ω form a (k_1, k_2)-block source in A, if the distribution $(U(\omega), V(\omega))$, where ω is chosen according to P_A form a (k_1, k_2)-block source.

Recall that the current setup is the following: We have a k-source X on $\{0,1\}^n$, and an explicit $(n, k, u, r, \gamma, \rho)$-block extraction scheme B (guaranteed from lemma 3.12 for some constant γ). For the remainder of this section we fix $t = \log^4 n$. (t is the amount of randomness needed in the second block of a block source in the block source extractor of corollary 3.8). We are ready to state the main lemma of this section.

Lemma 3.31. *(Following [NTS99]) There exist a partition of $\{0,1\}^n \times \{0,1\}^u$ into three sets BAD, BLK, CON with the following properties:*

1. $\Pr(BAD) \leq 2(\rho + 2^{-t})$
2. *(B, X) is a $(\frac{\gamma k}{r} - t, t)$-block source in BLK.*
3. *B is a $(k - 2t)$-source in CON.*

(To be strictly formal, the probability space Ω in this lemma is $\{0,1\}^n \times \{0,1\}^u$ and the distribution P is $X \times U_u$. In part 2 of the lemma, X stands for the random variable $X(x, y) = x$).

In the remainder of this section we use the technique of [NTS99] to prove lemma 3.31. The idea is to partition the elements into three sets according to their "weight": The "small weight" elements will form the set CON. Intuitively the small weight elements induce a source of high min-entropy. The "medium-weight" elements will form the set BLK. Intuitively the medium weight elements induce a source of medium min-entropy. Thus, they contain some (but not all!) of the min-entropy of the initial source. The fraction of "large weight" elements is bounded by ρ (the error parameter of the block extraction scheme). These elements form the set BAD and can be ignored because of their small fraction.

The following definition is motivated by the above intuition. (The partition of lemma 3.31 will be a refinement of the following partition).

Definition 3.32. *We partition $\{0,1\}^n \times \{0,1\}^u$ according to the "weight" of the elements.*

$$\begin{array}{lllllll} LRG & = & \{(x,y) & | & 2^{-\frac{\gamma k}{r}} < & \Pr_{X,Y}(B(X,Y)=B(x,y)) & & \} \\ MED & = & \{(x,y) & | & 2^{-(k-t)} < & \Pr_{X,Y}(B(X,Y)=B(x,y)) & \leq 2^{-\frac{\gamma k}{r}} & \} \\ SML & = & \{(x,y) & | & & \Pr_{X,Y}(B(X,Y)=B(x,y)) & \leq 2^{-(k-t)} & \} \end{array}$$

We will use the following lemma to prove lemma 3.31.

Lemma 3.33. *The sets LRG, MED and SML have the following properties:*

1. $\Pr(LRG) \leq 2\rho$
2. (B,X) *is a* $(\frac{\gamma k}{r} - \log \frac{1}{\Pr(MED)}, t)$*-block source in* MED.
3. B *is a* $(k-(t+\log \frac{1}{\Pr(SML)}))$*-source in* SML.

(To be strictly formal, the probability space Ω in this lemma is $\{0,1\}^n \times \{0,1\}^u$ and the distribution P is $X \times U_u$. In part 2 of the lemma, X stands for the random variable $X(x,y)=x$).

Proof. To prove the first item of lemma 3.33 we show that in any block source the weight of the "error" elements is proportional to the error.

Claim 1. *Let V be ρ-close to a k-source. Define $L = \{v | \Pr_V(V=v) > 2^{-(k-1)}\}$. It follows that $\Pr_V(L) < 2\rho$.*

Proof. Let V' be a k-source such that V and V' are ρ-close. We have that $|\Pr_V(L) - \Pr_{V'}(L)| < \rho$. However, V' assigns small probability to all elements in L, whereas V assigns large probability to these elements. This gives that $\Pr_V(L) - \Pr_{V'}(L) > \Pr_{V'}(L)$, Which means that $\Pr_{V'}(L) < \rho$. Using the first inequality we get that $\Pr_V(L) < 2\rho$. ☐

The remaining two items follow by just calculating the appropriate conditional probabilities. For a pair (x,y) in MED (or SML) we have an appropriate upper bound on the "weight" of $B(x,y)$ in the distribution $B(X,Y)$. When we consider the distribution of $B(X,Y)$ conditioned on MED (or SML), we need to divide by the probability of MED (or SML). Thus, we "lose" an amount of bits which is inversely proportional to the logarithm of the probability of MED (or SML). This argument immediately gives that B is a $(\frac{\gamma k}{r} - \log \frac{1}{\Pr(MED)})$-source in MED, and a $(k-(t+\log \frac{1}{\Pr(SML)}))$-source in SML. To complete the proof we need to show that when conditioned on MED, the second block contains t random bits. We know that the distribution $(B(X,Y),X)$ is a k-source, which means that every pair (b,x) has small probability (at most 2^{-k}). For pairs $(x,y) \in MED$, we have a lower bound of $2^{-(k-t)}$ on the weight of $b = B(x,y)$. This means that the "second part" is responsible to bringing the probability of pairs (b,x) down to 2^{-k}. Formally, we have that for $(x,y) \in MED$.

$$\Pr_{X,Y}(X = x|B(X,Y) = B(x,y)) = \frac{\Pr_{X,Y}(X = x, B(X,Y) = B(x,y))}{\Pr_{X,Y}(B(X,Y) = B(x,y))} < 2^{-t}$$

All this happens in the initial (unconditioned) probability space. Still conditioning on MED does not change the ratio between the two probabilities in the above fraction, since both are multiplied by the same factor. This completes the proof. $\square$

Lemma 3.31 follows by slightly changing the above partition. The sets LRG, MED and SML are almost the partition we want. We only need to avoid the setup in which the sets MED or SML are too small, since in this case the effect of conditioning is too costly. Still, if one of the sets is small we can safely add it to the "bad" elements and ignore it. This is the intuition behind the following partition, which partitions $\{0,1\}^n \times \{0,1\}^u$ into three sets:

1. The set BAD will contain all $(x,y) \in LRG$. It will also contain all $(x,y) \in SML$ if $\Pr(SML) < 2^{-t}$, and all $(x,y) \in MED$ if $\Pr(MED) < 2^{-t}$.

2. The set BLK (which corresponds to the set MED) contains all $(x,y) \in MED$ if $\Pr(MED) \geq 2^{-t}$.

3. The set CON (which corresponds to the set SML) contains all $(x,y) \in SML$ if $\Pr(SML) \geq 2^{-t}$.

lemma 3.31 immediately follows from lemma 3.33.

3.6 Constructing condensers

Condensers are a generalization of extractors. They are functions which use few random bits to "condense" an arbitrary random source. In this section we define and construct condensers. In 3.6.1 we use lemma 3.31 recursively to construct few distributions such that one of them is a block source. In 3.6.2 we show that in such a case applying block source extractors to the "candidate" distributions results in a condenser.

3.6.1 Getting a block source

In this subsection we implement the idea presented in the introduction. Namely, that running the block extraction scheme recursively, will eventually produce a block source. (The intuition is that in every failure we condense the source, and this will no longer be possible once the source's length drops below k).

To run this scheme, we will need to use "fresh" random bits for each instantiation of the block extraction scheme. We use l to denote the number of iterations, and assume that the construction process is given l uniformly chosen elements in $\{0,1\}^u$ for running block extraction schemes. The exact recursive construction is given in the following definition.

Definition 3.34. *For $0 \le i \le l$, (where l will be fixed later), we recursively define a sequence of functions, $B_i : \{0,1\}^n \times (\{0,1\}^u)^l \to \{0,1\}^{\frac{n}{r^i}}$, in the following manner:*

- $B_0(x; y_1 \cdots y_l) = x$.
- *For $i > 0$, $B_i(x; y_1, \cdots, y_l) = B(B_{i-1}(x; y_1, \cdots, y_l), y_i)$. (We chose to be a bit informal and ignore the fact that different blocks have different lengths and therefore different u's. Certainly the first block is the longest and therefore the most costly).*

It is easy to see that B_i does not depend on $y_{i+1}, \cdots, y_l$, and that all these computations can be done in polynomial time.

Following the intuition in the introduction, we want to argue that there exists a small l and an $1 \le i \le l$ such that $(B_i(X;\cdot), B_{i-1}(X;\cdot))$ is a block source. This is essentially the case, yet formally we will have to partition the space $\{0,1\}^n \times (\{0,1\}^u)^l$ into $l+1$ subsets in an analogous way to lemma 3.31.

Lemma 3.35. *There exists a partition of $\{0,1\}^n \times (\{0,1\}^u)^l$ into $l+1$ sets: $BLK_1, \cdots, BLK_l$ and BAD with the following property:*

1. $\Pr(BAD) \le 2l(\rho + 2^{-t})$
2. *(B_i, B_{i-1}) is a (k', t)-block source in BLK_i, (where $k' \ge \frac{\gamma(k-2lt)}{r}$).*
3. $l = O(\log_r(n/k))$

The remainder of this section is devoted to proving lemma 3.35. We start by recursively defining the appropriate sets BAD, BLK, CON.

Definition 3.36. *For $0 \leq i \leq l$, we recursively define sets $BAD_i, BLK_i, CON_i \subseteq \{0,1\}^n \times (\{0,1\}^u)^l$, and integers k_i.*

We define $BAD_0 = BLK_0 = \emptyset$, $CON_0 = \{0,1\}^n \times (\{0,1\}^u)^l$, and $k_0 = k$. For $i > 0$, suppose $BAD_{i-1}, BLK_{i-1}, CON_{i-1}$ have already been defined, and B_{i-1} is a k_{i-1}-source in CON_{i-1}.

Lemma 3.31 gives a partitioning of $B_{i-1}(CON_{i-1}) \times \{0,1\}^u$ into three sets, BAD, BLK, CON. We now "pull these sets back to the original probability space" and view them as subsets of $\{0,1\}^n \times (\{0,1\}^u)^l$. Define:

1. $BAD_i = \{(x; y_1, \cdots, y_l) | (B_{i-1}(x; y_1, \cdots, y_l), y_i) \in BAD\}$
2. $BLK_i = \{(x; y_1, \cdots, y_l) | (B_{i-1}(x; y_1, \cdots, y_l), y_i) \in BLK\}$
3. $CON_i = \{(x; y_1, \cdots, y_l) | (B_{i-1}(x; y_1, \cdots, y_l), y_i) \in CON\}$

Note that this is a partition of CON_{i-1}. We set $k_i = k_{i-1} - 2t$. Let l be the first integer such that $\frac{n}{r^i} < k_i$, (we will prove that such an i exists).

Using lemma 3.31 we get that:

Lemma 3.37. *The following holds for all $0 < i \leq l$.*

1. $\Pr(BAD_i) \leq 2(\rho + 2^{-t})$.
2. *(B_i, B_{i-1}) is a $(\frac{\gamma k_{i-1}}{r} - t, t)$-block source in BLK_i.*
3. *B_i is a k_i-source in CON_i.*

The sets $BAD_1, \cdots, BAD_l; BLK_1, \cdots, BLK_{l-1}$ and CON_l form a partition of $\{0,1\}^n \times (\{0,1\}^u)^l$. The following lemma shows that l cannot be too large, and when the process stops the set CON_l is empty.

Lemma 3.38. *The process described above stops after $l = O(\log_r(n/k))$ steps, and $CON_l = \emptyset$.*

Proof. After $l = \Theta(\log_r(n/k))$ steps, the length of the l'th block is $n/r^l < k/2$. In the last application of lemma 3.33, the set of small weight elements, SML has probability at most $2^{k/2-(k_l-t)} < 2^{-t}$. When this happens, the set CON is by definition the empty set, and therefore in the last iteration $CON_l = \emptyset$. □

Lemma 3.35 follows by Defining $BAD = \cup_{1 \leq i \leq l} BAD_i$.

3.6.2 Condensers

A condenser is a generalization of an extractor. Rather than requiring that it outputs a distribution that is close to uniform it is only required to improve the entropy rate.

Definition 3.39. *A (k, k', ϵ)-condenser is a function $Con : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^{n'}$, such that for every k-source X of length n, the distribution $Con(X, U_d)$ is ϵ-close to a k'-source, and $k'/n' > k/n$.*

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

Had we been able to construct a single block source, we could have used the block source extractor of corollary 3.8 to get an extractor. Having several candidates, we can run block source extractors on all of them (using the same seed) and obtain a condenser.

Definition 3.40. *We define a function $Con : \{0,1\}^n \times \{0,1\}^{ul+O(\log n)} \to \{0,1\}^{n'}$. Given inputs $x \in \{0,1\}^n$ and $y \in \{0,1\}^{ul+O(\log n)}$, Con interprets its second argument as l strings $y_1, \cdots, y_l \in \{0,1\}^u$ and an additional string s of length $O(\log n)$. For $0 \leq i \leq l$ it computes $b_i = B_i(x; y_1, \cdots, y_l)$, (where B_i is taken from definition 3.34), and $o_i = BE(b_i, b_{i-1}, s)$, (where BE is the block source extractor of corollary 3.8 taking $m = k'$). The final output is $(o_1, \cdots, o_l)$, (which makes $n' = lk'$).*

Intuitively, it is always case that one of the o_i's is close to uniform. Formally, it is straightforward to see that The distribution $(o_1, \cdots, o_l)$ is $(\frac{1}{n} + 2l(\rho + 2^{-t}))$-close to a k'-source. Let us compare the entropy rates of this source and the initial source. The new source has min-entropy k' which is approximately k and length approximately $k \cdot \log \frac{n}{k}$, whereas the initial source had length $n = k \cdot \frac{n}{k}$. This means that Con indeed improves the entropy rate and is a (k, k', ρ)-condenser.

Remark 3.41. *Actually, the distribution $(o_1, \cdots, o_l)$ is a source of a special kind called a "somewhere random source" by Ta-Shma in [TS96]. In [TS96] it was shown that extracting randomness from such sources is easier using special extractors which are called "somewhere random mergers". At this point we could have used Ta-Shma's "somewhere random mergers", to extract the randomness from $(o_1, \cdots, o_l)$. Instead, we use different methods which are more efficient and exploit the fact that l is relatively small.*

We will fix ρ and r to get two condensers with specific parameters. For both of them we fix $\rho = \log^{-3} n$. For the first one we choose $r = 2$. This gives that the condenser maintains a constant fraction of the initial randomness.

Lemma 3.42. *For every k, there exists an explicit $(k, \Omega(k), 1/\log^2 n)$-condenser $Con : \{0,1\}^n \times \{0,1\}^{O(\log \frac{n}{k} \log \log n + \log n)} \to \{0,1\}^{O(k \log \frac{n}{k})}$.*

For the second condenser we choose $r = 2\log n$. This reduces the number of recursive calls and decreases the number of random bits spent.

Lemma 3.43. *For every k, there exists an explicit $(k, \Omega(\frac{k}{\log n}), 1/\log^2 n)$-condenser $Con : \{0,1\}^n \times \{0,1\}^{O(\log n)} \rightarrow \{0,1\}^{\frac{k}{2}}$.*

Again, we encounter the tradeoff between the length of the seed and the amount of randomness in the output. The first condenser loses only a constant fraction of the randomness, but spends more then $O(\log n)$ random bits, whereas the second spends only $O(\log n)$ random bits, but is able to maintain only a smaller fraction of the randomness.

3.7 Constructing extractors

In this section we use the condensers constructed in the previous section to prove the two main theorems, (theorems 3.1,3.3).

For theorem 3.1 we use the condenser of lemma 3.42 repeatedly (with fresh seeds) to condense the source until we achieve constant entropy rate. (This is guaranteed to happen after no more than $\log^* n$ iterations). For constant entropy rate, Zuckerman's extractor ([Zuc97] see table 1.3), uses the optimal seed length to extract a constant fraction. This procedure loses some randomness in the iteration process, and results an extractor which extracts a sub-constant fraction of the initial randomness. We then use [WZ99] to increase this fraction to an arbitrary constant.

Proof. (of theorem 3.1) It is easy to check that given a (k, k', ϵ)-condenser $Con_1 : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^{n'}$ and a (k', k'', ϵ')-condenser $Con_2 : \{0,1\}^{n'} \times \{0,1\}^{d'} \rightarrow \{0,1\}^{n''}$, composing the condensers produces a $(k, k'', \epsilon+\epsilon')$-condenser $Con : \{0,1\}^n \times \{0,1\}^{d+d'} \rightarrow \{0,1\}^{n''}$.

Let us denote the entropy rate of a source by $R(X) = k/n$. The condenser of lemma 3.42 produces a source X' with $R(X') = \Theta(1/\log(1/R(X)))$ (or in other words $R^{-1}(X') = \Theta(\log(R^{-1}(X))))$. Thus, after $\log^* R^{-1}(X) \leq \log^* n$ iterations the entropy rate becomes constant. Once the ratio is constant Zuckerman's extractor ([Zuc97], see table 1.3), can be used to extract a constant fraction (say half) of the randomness. The problem with this procedure is that our condenser loses a constant fraction of the randomness in every iteration. Thus, after $\log^* n$ iterations we can extract only $k/2^{O(\log^* n)}$ random bits from the source, and produce an extractor which extracts a $1/2^{O(\log^* n)}$ fraction of the initial randomness. To get to a constant fraction we use the method of Wigderson and

Zuckerman, [WZ99].[10] We use at most $\log n \cdot \log\log n$ random bits for every of the $\log^* n$ iterations. Implementing the technique of Wigderson and Zuckerman multiplies this amount by $2^{O(\log^* n)}$. Thus, the total number of random bits is $\log n \cdot \log\log n \cdot \log^* n \cdot 2^{O(\log^* n)} = O(\log n \cdot (\log\log n)^2)$ as required. □

Remark 3.44. *The condenser of lemma 3.42 uses only $O(\log n)$ random bits when $k = \Omega(\frac{n}{2^{\log n/\log\log n}})$. Thus, for this case we get an extractor that uses only $O(\log n \cdot 2^{O(\log^* n)})$ random bits. We can reduce the seed length to $O(\log n(\log^* n)^2)$ by applying another recursion. Here's a rough sketch of the idea: In each of the $\log^* n$ applications of the condenser we are guaranteed to "extract" a constant fraction of the randomness. However, just as we did previously we can consider two cases: Either we extracted (almost) all the randomness, (and in this case we don't lose randomness when doing $\log^* n$ repetitions), or we get a block source, and can extract a constant fraction of the randomness. Applying block source extractors to all the "candidate block sources" and concatenating the outputs, we get a distribution of length roughly $k \log^* n$ which is an $\Omega(k)$-source. Using methods from [NZ96] on such a condensed distribution extracts a $\Omega(k/\log^* n)$ bits using seed $O(\log n)$. We now use the method of [WZ99] to extract $\Omega(k)$ bits, at the cost of multiplying the seed by $O(\log^* n)$.*

In the case of theorem 3.3 we are shooting for the optimal seed length and cannot afford the condenser of lemma 3.42 or repeated condensing. Instead we use the condenser of lemma 3.43 interpreting it as a block extraction scheme. Viewed this way the condenser extracts a block B of length $k/2$, therefore the distribution (B, X) forms a block source, since B is too short to "steal" all the randomness from X. (This intuition is formalized in the next lemma). All that is left is to use the block source extractor of corollary 3.8.

Lemma 3.45. *Let Con be the condenser of lemma 3.43. If X is a k-source then the distribution $Con(X, U_{O(\log n)}), X)$ is $O(1/\log^2 n)$-close to an $(\Omega(k/\log n), \log^4 n)$-block source.*

Proof. For this proof we view Con as a $(n, k, O(\log n), 2n/k, \Omega(1/\log n), 1/\log^2 n)$-block extraction scheme. Con is a very good block extraction scheme in the sense that the length of the block is only $k/2$. Intuitively, this means that X must contain randomness which is not in B, and therefore (B, X) is a block source. Formally,

[10] Wigderson and Zuckerman suggested to repeatedly extract randomness from the source (using fresh seeds), until one extracts the desired fraction. This gives that if $m = k/p$ then m could be increased to $(1-\delta)k$ (where δ is an arbitrary constant), at the cost of multiplying d by $O(p)$. (An exact formulation of the Wigderson and Zuckerman technique can be found for example in [Nis96, NTS99]).

we consider the partition of $X \times \{0,1\}^{O(\log n)}$ into three sets defined in definition 3.32 (recall that $t = \log^4 n$). The set SML corresponds with pairs (x, y) which go to strings $b \in \{0,1\}^{k/2}$ with weight smaller than $2^{-(k-t)}$. Thus, $\Pr(SML) \leq 2^{k/2-(k-t)} < 1/\log^2 n$. We have that both the sets SML and LRG have small probability, which in turn means (using lemma 3.33) that $\Pr(MED) \geq 1 - 3/\log^2 n$. Lemma 3.33 also guarantees that $(Con(X, U_{O(\log n)}), X)$ is a $(\Omega(k/\log n), \log^4 n)$-block source in MED. $\square$

Proof. (of theorem 3.3) Given a k-source, we use lemma 3.45 to get a distribution that is close to a block-source and use the block-source extractor of corollary 3.8. $\square$

Remark 3.46. *The output length in theorem 3.3 (and corollary 3.4) can be improved to $k/\log^\delta n$, ($k/\log^\delta k$), for any constant $\delta > 0$. This is done by choosing $r = \log^{\delta/2} n$ instead of $r = 2\log n$ in lemma 3.43. This will make the condenser maintain a larger fraction of the initial randomness while still using the optimal seed length. The proof of theorem 3.3 will now fail, since we used the fact that the condenser has output smaller than k. However, at this point the source becomes so condensed that standard techniques can be used to construct a block source while loosing only another $\log^{\delta/2} n$ fraction of the randomness.*

3.8 Achieving small error

The statement of theorems 3.1,3.3 is for constant error ϵ. The analysis provided in this chapter gives a slightly better result and allows to replace the requirement that ϵ be a constant with $\epsilon = 1/(logn)^c$ for any constant c. Still, our technique does not give good dependence of the seed length on the error[11]. We get better dependence on ϵ using the error reduction transformation of [RRV99a], which transforms an extractor with large (say constant) error into an extractor with arbitrary small error, while losing only a little bit in the other parameters. More precisely, after undergoing the transformation, a factor of $O(\log m (\log\log m)^{O(1)} + \log(1/\epsilon))$ is added to d, and the fraction extracted decreases by a constant. The latter loss makes no difference from our point of view since we are only able to extract constant fractions. The first loss isn't significant in the case of theorem 3.1, since the seed size is

[11]Although the block extraction scheme of lemma 3.12 has the required dependence on the error ρ, there are two problems. The first is that we run the block extraction scheme roughly $\log n$ times (which makes the error the multiplication of the two factors). The second is that because we use Chebychev's inequality our block extraction scheme works only when $\rho = 1/k^{O(1)}$.

already larger than the optimal one by a multiplicative $polyloglog(n)$ factor. However, it completely spoils theorem 3.3 and makes it inferior to theorem 3.1. Here is theorem 3.1 rephrased using the error reduction transformation of [RRV99a]:

Theorem 3.47. *(theorem 3.1 rephrased for non-constant ϵ) For every n, k and $\epsilon > exp(\frac{-n}{(\log^* n)^{O(\log^* n)}})$, there are explicit (k, ϵ)-extractors $Ext : \{0,1\}^n \times \{0,1\}^{O(\log n \cdot (\log \log n)^{O(1)} + \log(1/\epsilon))} \to \{0,1\}^{(1-\delta)k}$, where $\delta > 0$ is an arbitrary constant.*

Chapter 4

Simple extractors for all min-entropies and a new pseudo-random generator

4.1 Introduction

In this chapter we present a new, simple, self-contained, and versatile construction of both extractors and pseudo-random generators that achieves good results for the whole range of parameters. Such a construction is important from a practical point of view, and also may advance the technical effort to construct extractors matching the lower bounds by introducing new ideas and stripping away the complicated recursions, iterations, and compositions that characterized much of the previous work. We build on a new technique introduced in [TSZS01] for building extractors, contributing a number of key ideas that allow it to be used for PRGs, and for extractors over a broad parameter range (the extractors in [TSZS01] only work when the min-entropy is quite high, and even then they lose quite a bit of the source randomness).

Additional randomness t	Output length m	which k	Reference
$O(\log n)$	$k^{1-\delta}$	$k > \log^{O(1/\delta)} n$	Theorem 4.1
$(1+\alpha)\log n$	$k^{\Omega(\alpha)}$	any k	Theorem 4.2
$(1+\alpha)\log n$	$k/(\log^{O(1/\alpha)} n)$	any k	Theorem 4.4

Table 4.1: extractors constructed in this paper ($\delta > 0$ is any constant and $\alpha < 1$).

In terms of actual parameters, our extractor constructions do not break any important records, but do match constructions that were considered breakthroughs only a few years ago. In the area of PRGs, we obtain some concrete improvements, solving open problems raised by [ISW99] and [MV99]. Below, we outline our results in more detail:

- Our extractors (summarized in Table 4.1) all come from the same simple polynomial-based construction, which we tune to achieve a variety of parameters. Our first extractor achieves a relatively large output length, while retaining the asymptotically optimal seed length. This matches the seed length and output length of Trevisan's extractor (which requires $k = n^{\Omega(1)}$), but also works for small k, giving a more direct solution to the problem addressed by [ISW00, TSUZ01] (see chapter 2). Our second extractor achieves a seed length with near-optimal leading constant for all min-entropies. This extends the result of [TSZS01] which works only for large min-entropy. Our third extractor almost matches the output length of [RSW00] (the current best construction for general k with seed length $O(\log n)$), and *simultaneously* achieves a seed length with near-optimal leading constant. Unlike the first two extractors it is not self-contained – it relies on another extractor construction from [TSZS01, NZ96].

- Prior to this paper, all known PRG constructions were based on the original NW PRG (with the exception of the Blum-Micali-Yao-style PRGs [BM84, Yao82], which are based on cryptographic hardness assumptions). Coming up with an alternate construction has long been an open problem. Our construction does *not* use the NW PRG, and is arguably simpler than previous constructions. In particular, there is no explicit hardness-amplification component: we transform worst-case hardness directly into pseudo-randomness. The parameters of our PRG match [IW97, STV99] (that is, seed length $O(\log^2 n/\log s)$ and output length $s^{\Omega(1)}$) and consequently we obtain a new proof that $BPP = P$ if there exists a function family in E which requires exponential size circuits.

- A hitting set generator (HSG) is the one-sided variant of a PRG, and the canonical construct for derandomizing RP (instead of BPP). We give the first construction of an *optimal* HSG; that is, a HSG with seed length $O(\log n)$ that outputs $s^{\Omega(1)}$ bits when given a function over $\log n$ bits which requires size s circuits. Our construction is optimal in the sense made formal in [ISW99]; namely, any construction which does significantly better also produces a harder function than the one initially supplied to it (see the discussion in section 1.4.4).

An optimal HSG immediately gives an optimal hardness versus randomness tradeoff for RP. Using the result of [ACR96] that HSGs suffice to derandomize BPP,[1] we obtain an optimal hardness versus randomness tradeoff for BPP. Specifically, we get the following tradeoff: if there exists a family of functions in E requiring circuits of size $s(n)$, then BPTIME$(t) \subseteq$ DTIME$(2^{O(s^{-1}(t^{O(1)}))})$. This completes a line of research begun in [ISW99] and in particular improves the best previous result (due to [ISW99, ISW00] see chapter 2) which obtained the weaker conclusion that BPTIME$(t) \subseteq$ DTIME$(2^{O(s^{-1}(t^{O(\log\log t)}))})$.

- Specific properties of our proof technique allow us to prove that both our PRG and HSG, with identical parameters, can be used to fool *nondeterministic circuits* when given a function that is hard for *single-valued-nondeterministic* circuits. This gives an optimal hardness versus randomness tradeoff for AM (the $AM \stackrel{?}{=} NP$ problem is the nondeterministic analog of the $BPP \stackrel{?}{=} P$ problem), and improves and extends [KvM99] (which relied on a stronger hardness assumption) and [MV99] (which does not work for all hardnesses).

4.1.1 A note regarding notation

In this chapter we use slightly different notation than previous chapters. In particular we use the letter t for the *seed length* of pseudo-random generators and extractors (instead of d). The letter d is taken for the dimension of the vector-space.

4.1.2 Results

In this chapter we give the following constructions of extractors.

Theorem 4.1. *For all n, all constants $\epsilon, \delta > 0$, and all $k \geq \log^{4/\delta} n$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k^{1-\delta}$.*

Theorem 4.2. *For all n, k, ϵ and δ, our construction gives a (k, ϵ)-extractor with seed length $t = (1 + O(\delta)) \log n + O(\delta) \left(\frac{\log n}{\log m} \log \left(\frac{1}{\epsilon} \right) \right)$ and output length $m = k^{\Omega(\delta)}$.*

[1] In our setting it is also possible to use the much simpler "tournament of generators" from [ISW99].

Theorem 4.3. *For all n, k, constant ϵ and constant $\delta > 0$ our construction gives an explicit (k, ϵ)-extractor with seed length $t = O(\log n)$ and output length $m = k/(\log n)^{2+\delta}$.*

Theorem 4.4. *For all n, k, constant ϵ and any $0 < \delta < 1$ and our construction gives an explicit (k, ϵ)-extractor with seed length $t = (1 + \delta)\log n$ and output length $m = k/(\log n)^{O(1/\delta)}$.*

We give the following constructions of generators:

Theorem 4.5. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ which is computable in time $n^{O(1)}$ with $S(g) \geq s$ then there exists a $1/m$-PRG for size m with seed length $t = O(\log^2 n/\log s)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in time $n^{O(1)}$.*

Theorem 4.6. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ which is computable in time $n^{O(1)}$ with $S(g) \geq s$ then there exists a $1/m$-HSG for size m with seed length $t = O(\log n)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in time $n^{O(1)}$.*

We give the following constructions of nondeterministic generators:

In the next theorem S_{SV} is used to denote the *single-valued nondeterministic circuit size* of a function, see definition 4.31.

Theorem 4.7. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ in nondeterministic time $n^{O(1)}$ with $S_{SV}(g) \geq s$ then there exists a $1/m$-nondeterministic PRG for size m with seed length $t = O(\log^2 n/\log s)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in nondeterministic time $n^{O(1)}$.*

By computable in nondeterministic time t we mean that each output bit is in $NTIME(t) \cap coNTIME(t)$ which means that it is easy to compute the output of the generator with nondeterministic advice. The corollary also follows when the function and generator are computed in deterministic time as the generator can be computed in deterministic time $n^{O(1)}$ given oracle access to the function.

Theorem 4.8. *For any s, if there exists a function $g : \{0,1\}^{\log n} \to \{0,1\}$ which is computable in nondeterministic time $n^{O(1)}$ with $S_{SV}(g) \geq s$ then there exists a $1/m$-nondeterministic HSG for size m with seed length $t = O(\log n)$ and output length $m = s^{\Omega(1)}$. Furthermore, this generator can be computed in nondeterministic time $n^{O(1)}$.*

The same remark about nondeterministic time that follows theorem 4.7 applies.

While the constructions in [TSZS01] and the present work are simple, the proofs are more involved. The common thread in the proof techniques of [STV99], [TSZS01] and this work is the use of specific properties of error-correcting codes, and ideas for decoding them. In the Section 4.2 we describe the construction of [TSZS01] and our improvements at a high level.

4.2 Overview of the Technique

We first outline the relevant prior work on extractors in section 4.2.1, then our new ideas in section 4.2.2, and finally we describe the additional ideas needed to build PRGs and HSGs in section 4.2.3.

4.2.1 Previous work

Trevisan [Tre99] showed that a powerful proof technique that at first seems natural only for PRGS can in fact be used for extractors. The proof technique works by contradiction. One assumes that the extractor's output is not close to uniform and therefore does not pass some prediction test; that is, there exists a function f (the *predictor*) whose value is with non-negligible probability the i-th symbol output by the extractor when it is given the first $i-1$ symbols. A contradiction is derived by giving a "reconstruction procedure" based on the predictor f. This procedure is able to "reconstruct" the string sampled from the weak random source from a short advice string, for a large fraction of source strings. If the source has large min-entropy, this is a contradiction, because it implies that a large number of source strings have short descriptions (and there are only a few short descriptions).

Ta-Shma, Zuckerman and Safra [TSZS01] proposed a new extractor construction based on polynomials that uses this proof technique in a new way. Their construction is very simple. One thinks of the string sampled from the weak random source as a low-degree multivariate polynomial p over a finite field and of the seed as a random evaluation point. The extractor computes m "successive" points (starting from the random evaluation point) and outputs the evaluation of the polynomial at these points. The meaning of "successive" is purposely left a bit vague in this presentation and we will elaborate on it later.

Their proof describes a reconstruction procedure that attempts to reconstruct a source string (which is a polynomial p) using a predictor f and a short advice string which contains the value of p at a fixed small set of points. The basic idea is to use $i-1$ successive points from within this set to predict the value of p at the "next" point (using f). Now the predictor can be "advanced" by one step and use $i - 2$ successive points plus the one just predicted to predict the value of p at the next point, and so on, until all of p is reconstructed. However, there are two complications.

First, the predictor is only correct with a small advantage over random guessing. To overcome this, at each step the predictor is used to predict in parallel all points along a random line L. Because points on a random line are pairwise independent, with high probability the predictor is correct on the same fraction of points in L as its total advantage over the whole space. The collection of values along the line can now be error-corrected since p restricted to L is a low-degree univariate polynomial.

Second, the relative number of errors is so large that unique decoding is impossible, and one must use list-decoding to obtain a small number of possible options, one of which is correct. To pin down the correct one, the advice string includes the evaluation of p at a random point on L, which with high probability agrees with *only* the correct item in the list.

We note that in this summary the extractor and the predictor output field elements instead of bits; however there is an easy reduction of the binary case to this one (see Section 4.3.4).

4.2.2 Improvements of this paper

The key to improving the quality of the [TSZS01] extractor is reducing the length of the advice string. To do this one has to decrease the degree of p, which increases the dimension. But, in [TSZS01] increasing the dimension *increases* the length of the advice string. Our improvements enable us to increase the number of dimensions without this deleterious effect.

An algebraic approach. What does "successive" mean? In [TSZS01] (say in two dimensions over finite field F), the successor of (a_1, a_2) is $(a_1 + 1, a_2)$. The rational is that starting from any point (a_1, a_2) and taking successive steps one covers the line $F \times a_2$. Doing this for all a_2's covers the plane. However, this geometric approach succumbs to the "curse of dimensionality" as the dimension is increased.

The main source of our improvement comes from taking an algebraic instead of a geometric approach. The polynomials we wish to learn are defined over the vector-space F^d. Our insight is to view this space as an extension field of F. The multiplicative group of this field is cyclic and has generator g, and for us, the successor of an element is obtained by *multiplication* by g. This indeed has the essential property that by repeatedly taking successors, we cover the whole space. We also use critically that multiplication by g corresponds to a *linear transform* in the vector-space, so that lines get mapped to lines. Replacing the geometric approach by an algebraic approach avoids the geometric structure of F^d and now the dimension does not come into play.

Curves instead of lines. In [TSZS01], each prediction step fails with probability inversely proportional to the size of the field F, and they use a union bound to argue that *no* prediction steps fails during the entire reconstruction process.

Increasing the number of dimensions decreases the number of points in a line and means that it takes many more prediction steps to reconstruct the whole space and therefore many more events are in the union bound. It also decreases the field size which increases the failure probability of each individual event. Together these effects overwhelm us.

We overcome this by predicting along *degree r curves* instead of lines. Since the collection of points on such curves is r-wise independent, we can use higher moment tail inequalities to argue that the failure probability of each prediction step is *exponentially small* in r, which permits us to use the union bound even for our much larger collection of events. Using curves instead of lines is also necessary for the improvement described next.

Interleaved reconstruction procedures. There's an additional cost to using more prediction steps. Recall that in [TSZS01], the advice string must include the value of the polynomial at a random point on the line, *for each prediction step*. Having to include these will blow up the size of the advice string.

To overcome this problem we run two "interleaved" reconstruction procedures. Each uses its own random curve but we arrange it so that the two curves intersect at a few random points. The two reconstruction procedures works on their own. However, when one needs the value of the polynomial at a random point on its curve, it can use the value *already calculated* by the other reconstruction procedure instead of relying on the advice string. Thus, no additional information is required in the advice string beyond the values needed to get the two interleaved reconstruction procedures started.

We stress that almost all of what we have described in the preceding subsections relates to the *proof* that our construction is indeed an extractor. The extractor construction itself remains very simple.

4.2.3 Constructing PRGs and HSGs

Since the proof technique outlined above was originally applied to PRG constructions, it is easy to adapt to that setting. To convert our extractor into a PRG, we fix the "source string" x to be the truth table of a hard function, use the seed as before to pick an evaluation point, and output m successive evaluations. If this in *not* a PRG, then there is an *efficient* predictor f, and we wish to use f to produce a *small* circuit C that computes the function $x(j) = x_j$, contradicting the hardness of x.

Two things contribute to the size of C: the length of the advice string (which must be hardwired into C), and the number of prediction steps (since each step invokes f and requires computation to perform the list-decoding). Because the advice string must be small, the improvements we get over [TSZS01] in this area are essential.

However, the natural adaptation of our extractor into a PRG suffers from an inherent problem of the method of [TSZS01]: It is highly sequential. Specifically, computing x_j at a position j that is "far away" from the points specified by the advice string (one that takes many successive prediction steps to get to) takes too many steps and makes C too large to derive a contradiction.

The problem is that we can't have a "successor function" defined over F^d in such a way that very few applications of the function can get to *every* point from a fixed starting location. A helpful idea is to allow *several* such functions so that short sequences of applications of the different functions can reach every point. To achieve this we will take the first successor function to be the one we used for the extractor, the second successor function to be m *applications* of the first, the third to be m^2 applications of the first and so on. By first taking a few small strides, then a few larger strides, then a few even larger strides, etc., we can reach every point in a small number of steps.

Each one of these successor functions corresponds to a construction very similar to our extractor construction. We show that *at least one* of these constructions must be a PRG, since if none of them are, then we have predictors for all of them which gives us the predictors with differing strides needed to contradict the hardness of x.

By running all of these "candidate" PRGs with independent seeds, and XORing their output, we obtain the desired single PRG. Because we need so few candidate generators (a *constant* number if x has exponential hardness), the seed length of the XORed generator is still relatively short, and we match the parameters of [STV99]. By taking the *union* of these candidate PRGs, we obtain the HSG with optimal seed length described in the introduction.

Generators for nondeterministic circuits

The difficulty in using the reconstruction paradigm to construct pseudo-random generators for nondeterministic (or co-nondeterministic) circuits is that when given a small nondeterministic prediction test which the generator does not pass the reconstruction procedure can only access it in a one-sided way: It can efficiently verify that the answer of the predictor is “one”, but there’s not necessarily an efficient way to verify that the output is “zero”. This is a problem if the task of the reconstruction procedure is to construct a small nondeterministic circuit for the hard function. Klivans and van-Melkebeek [KvM99] bypassed this problem by assuming a hardness assumption for circuits with SAT-oracle. Under this assumption NW-based constructions can be used to fool circuits with SAT-oracle. (Our improvements over NW-based generators immediately translate to this framework). Miltersen and Vinodchandran [MV99] gave an alternate construction which works using SV-circuit hardness, which is weaker than nondeterministic circuit hardness. However, their construction gives only hitting set generators, and works only for the “high end”, meaning that it has to assume that the circuit lower bound of the hard function is at least $2^{\sqrt{l}}$ where l is the input size of the function.

Both our pseudo-random generators and our hitting set generators can be adapted to fool nondeterministic (co-nondeterministic circuits) using SV-circuit hardness. The main observation is that we use our predictors along random curves. When used this way the number of “ones” we expect to find in the output of the predictor is close to its number in the whole space, and the probability of deviation is very small. The reconstruction will get the fraction of “ones” in the whole space as a non-uniform advice. We verify that there are at least this number of “ones” in our random curve and assume the rest to be “zeroes”. With very high probability this is close enough to the real values on the curve to allow the list decoding phase to proceed unchanged.

It should be noted that the presentation in this section is over-simplified and the reader is referred to the technical sections for exact details. the actual construction also involves a non-standard version of the “low degree extension” encoding which is suitable for our application.

4.3 Extractors

In [Yao82], Yao showed that to prove that a distribution is close to uniform it is sufficient to prove that it ”passes all prediction tests”. More precisely, to show that a function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is a (k, ϵ)-extractor it is sufficient to prove (4.1) for all distributions X on $\{0,1\}^n$ with min-entropy at least k:

$$\forall \ 1 \le i \le m \text{ and all functions } f : \{0,1\}^{i-1} \to \{0,1\},$$
$$\Pr[f(E(X,U_t)_{1\dots i-1}) = E(X,U_t)_i] \le \frac{1}{2} + \frac{\epsilon}{m} \tag{4.1}$$

and indeed the proofs for many recent extractor constructions follow this route. Property (4.1) requires that each successive bit of output be "unpredictable" based on the previous bits.

As our construction of extractors is algebraic, we will be working over finite fields with q elements. It will therefore be useful to define a "q-ary" extractor. Such an extractor is required to satisfy an unpredictability property analogous to (4.1); however, in the larger field we allow the prediction function f to output a small list of possible next elements, instead of just one.

Definition 4.9 (q-ary extractor). *Let F be the field with q elements. A (k, ρ) q-ary* extractor *is a function $E : \{0,1\}^n \times \{0,1\}^t \to F^m$ such the that for all random variables X with $H_\infty(X) \geq k$:*

$$\begin{aligned}&\forall \ 1 \leq i \leq m \text{ and all functions } f : F^{i-1} \to F^{(\rho^{-2})},\\ &\qquad \Pr[\exists j \ f(E(X,U_t)_{1\ldots i-1})_j = E(X,U_t)_i] \leq \rho.\end{aligned} \tag{4.2}$$

In [TSZS01] it was shown how to transform q-ary extractors into (regular) extractors, and the details are given in section 4.3.4. This allows us to focus on building good q-ary extractors.

We need the following list-decoding bound due to [Sud97]:

Lemma 4.10 ([Sud97]). *Given n distinct pairs (x_i, y_i) in field F and parameters k and d, with $k > \sqrt{2dn}$, there are at most $2n/k$ degree d polynomials g such that $g(x_i) = y_i$ for at least k pairs. Furthermore, a list of all such polynomials can be computed in time poly$(n, \log |F|)$.*

We also need the following tail inequality for t-wise independent random variables:

Lemma 4.11 ([BR94]). *Let $t > 4$ be an even integer. Suppose $X_1, X_2, \ldots, X_n$ are t-wise independent random variables taking values in $[0, 1]$. Let $X = \sum X_i$, $\mu = E[X]$ and $A > 0$. Then:*

$$\Pr[|X - \mu| \geq A] \leq 8 \cdot \left(\frac{t\mu + t^2}{A^2}\right)^{t/2}$$

4.3.1 The construction

Our construction is very simple. Like [TSZS01] our first step is to encode the string x from the weak random source with a q-ary Reed-Muller (d-variate polynomial) code. The coordinates of such a code are in one-to-one correspondence with the vectors in the vector-space F^d. Our q-ary extractor uses its truly random bits to pick a random $\vec{v} \in F^d$. It outputs the $\vec{v}$-th symbol of the encoded string and $m-1$ successive symbols. The successor of $\vec{v}$, is $A\vec{v}$, where A is a special $d \times d$ matrix, on which we elaborate below.

More formally, let F be the field with q elements, and fix the dimension d. Let h be the smallest integer such that

$$\binom{h+d}{d} \geq \frac{n}{\log q}. \tag{4.3}$$

For $x \in \{0,1\}^n$, let $\hat{x}$ denote the d-variate polynomial of total degree $h-1$ whose coefficients are specified by x. Such a polynomial has $\binom{h+d}{d}$ coefficients, so (4.3) implies that distinct x give rise to distinct $\hat{x}$.

The important property that we require of matrix A is that it "generates" $F^d \setminus \{0\}$. That is, for any non-zero vector $\vec{v} \in F^d$,

$$\{A^i \vec{v}\}_i = F^d \setminus \{0\}. \tag{4.4}$$

It is easy to show that such a matrix exists and can be found efficiently, using the algebraic structure of F^d:

Lemma 4.12. *Matrix A satisfying (4.4) exists and can be found in time $q^{O(d)}$.*

Proof. There is a natural isomorphism between the field $GF(q^d)$ and the vector-space F^d. Let g be a generator of the multiplicative group of $GF(q^d)$ (which is cyclic). Multiplication in the field corresponds to linear transforms in the vector-space, so the linear transform A corresponding to multiplication by g satisfies (4.4). We can find g by exhaustive search[2] in time $q^{O(d)}$. $\square$

We can now define our q-ary extractor $E : \{0,1\}^n \times \{0,1\}^{d \log q} \rightarrow F^m$. We interpret the second input of the q-ary extractor as a vector $\vec{v} \in F^d$.

$$E(x, \vec{v}) = \hat{x}(A^1 \vec{v}) \circ \hat{x}(A^2 \vec{v}) \circ \cdots \circ \hat{x}(A^m \vec{v}) \tag{4.5}$$

Our main theorem is the following:

Theorem 4.13 (extractor main). *For every n, q, d and h satisfying (4.3), E is a (k, ρ) q-ary extractor, provided that*

1. $k > \Omega(mhd \log q) + \log\left(\frac{1}{\rho}\right)$ and $q > \Omega\left(\frac{(hd)^2}{\rho^4}\right)$, or

[2] More precise bounds, and significant improvements in certain cases can be found in [Sho92] and [Shp96].

2. $k > \Omega(mhd \log^2 q) + \log\left(\frac{1}{\rho}\right)$ *and* $q > \Omega\left(\frac{hd \log q}{\rho^4}\right)$.

The proof of theorem 4.13 is given in the next two sections. In section 4.3.4 we explain how to choose parameters to derive the extractors in Table 4.1 from the main theorem.

4.3.2 The reconstruction proof paradigm

To prove that E is a q-ary extractor, we use ideas that originated in [Tre99] and are refined in [TSZS01], which one might label the "reconstruction proof paradigm".

Suppose E is *not* a (k, ρ) q-ary extractor. We fix a random variable X over $\{0,1\}^n$ with $H_\infty(X) \geq k$ and a function f that violates property (4.2). We then describe a "reconstruction function" $R : \{0,1\}^a \to \{0,1\}^n$ that has oracle access to f, and we design R to have the following property:

$$\Pr_{x \leftarrow X}[\exists z \;\; R^f(z) = x] \geq p \tag{4.6}$$

for some not too small p. Intuitively, this implies that for a large fraction of strings x in the support of X, there is a short "advice string" z from which we can reconstruct x. Typically the advice string consists of small portions of an encoded version of x, and R uses ideas from decoding error-correcting codes to "reconstruct" x from this partial information. We think of z as a "short description" of x. For a fixed f, there are at most 2^a strings in $\{0,1\}^n$ which can be "described" using an a-bit long advice string (i.e., at most 2^a strings are in the range of R^f). The total weight of these strings sampled according to X is at most $2^a \cdot 2^{-k}$. Thus, we get a contradiction if $2^{a-k} < p$. We conclude that E is a (k, ρ) q-ary extractor provided $a + \log(\frac{1}{p}) < k$.

Our main task is to use the "next-element predictor" function f in a clever way to allow many strings x in the support of X to be reconstructed from very short advice strings.

To obtain R, we use essentially the framework of [TSZS01], which we now describe more precisely. We use f in a sequence of "prediction steps." In each prediction step we learn the value of $\hat{x}$ evaluated at some points in F^d. In the end, we know $\hat{x}$ evaluated at *all* points in F^d, and we can interpolate to recover the coefficients of $\hat{x}$ and finally x.

Step $i > t$ requires the evaluations of $\hat{x}$ at a set of "known points" $P_{i-t}, \cdots P_{i-1}$ and uses f to evaluate $\hat{x}$ at a set of points P_i. We will prime the pump for the first step by supplying an advice string that can be used to obtain $\hat{x}$ evaluated at $P_1, \cdots, P_t$. After that, we can evaluate $\hat{x}$ on all the points in $\cup P_i$ by making a series of prediction steps.

In the next section we will use a probabilistic argument to show that there *exists* a sequence P_i's and prediction steps that works for many x in the support of X, and requires very few bits of advice to get started.

4.3.3 Proof of the main extractor theorem

In this section we prove Theorem 4.13. As outlined above, we fix a random variable X with $H_\infty(X) \geq k$. We assume for the purpose of contradiction that E is not a (k, ρ) q-ary extractor. Thus, there exists a predictor function function $f : F^{i^*-1} \rightarrow F^{(\rho^{-2})}$ violating property (4.2). We start by observing that the predictor gains advantage for many fixed x's

Definition 4.14. *A string $x \in \{0,1\}^n$ is* good *with respect to E and f if*

$$\Pr_y[\exists j \; f(E(x,y)_{1\ldots i^*-1})_j = E(x,y)_{i^*}] > \rho/2,$$

The following lemma is immediate by an averaging argument.

Lemma 4.15. $\Pr_{x \leftarrow X}[x \textit{ is good}] > \rho/2$.

The following lemma describes the "reconstruction function".

Lemma 4.16. *Let n, q, d, and h be as in the statement of Theorem 4.13 (1) (resp. Theorem 4.13 (2)), and fix a function $f : F^{i^*-1} \rightarrow F^{(\rho^{-2})}$. There exists a randomized procedure $R : \{0,1\}^a \rightarrow \{0,1\}^n$ with $a = O(mhd \log q)$ (resp. $a = O(mhd \log^2 q)$) such that for every good x:*

$$\Pr[\exists z \quad R^f(z) = x] \geq 1/2$$

where the probability is over the random choices of R.

Proof. Fix a good x. Our goal is to reconstruct x from a short advice string. The predictor function f can "attempt to predict" the evaluation of $\hat{x}$ at z, when given the evaluation of $\hat{x}$ at the points

$$A^{-(i^*-1)}z, A^{-(i^*-2)}z, \cdots, A^{-1}z.$$

This is because $E(x, A^{-i^*}z)_{1\cdots i^*-1} = \hat{x}(A^{-(i^*-1)}z), \cdots, \hat{x}(A^{-1}z)$. Thus, if $y = A^{-i^*}z$ is one of the $\rho/2$ seeds for which f correctly predicts $E(x,y)_i^*$ given $E(x,y)_{1\cdots i^*-1}$ then f computes $\hat{x}$ at z given the previous evaluations.

This prediction is only guaranteed to succeed on a $\rho/2$-fraction of the points z. The crux of the proof is a randomized choice of low-degree curves with special intersection properties which will allow us to "error correct" wrong answers of the predictor f and evaluate $\hat{x}$ correctly at every point. More formally, we use the coin tosses of R to define sets $\{P_i\}_i$ and $\{K_i\}_i$ as in Section 4.3.2 in the following way:

- Set $r = \Theta(d)$ (resp. $r = \Theta(d \log q)$)
- Pick $2r$ random points $\vec{y_1}, \vec{y_2}, \dots \vec{y_{2r}}$ from F^d, and $2r$ values $t_1, t_2, \dots t_{2r}$ from F.
- Define the degree $2r-1$ polynomial $p_1 : F \to F^d$ so that $p_1(t_i) = \vec{y_i}$, for $i = 1, 2, \dots, 2r$.
- Define the degree $2r-1$ polynomial $p_2 : F \to F^d$ so that $p_2(t_i) = A\vec{y_i}$ for $i = 1 \dots r$ and $p_2(t_i) = \vec{y_i}$ for $i = r+1 \dots 2r$.

We also define $P_{2j+1} = \{A^j p_1(z)\}_{z \in F}$ and $P_{2j+2} = \{A^j p_2(z)\}_{z \in F}$ for all integers j. Since A "generates" $F^d \setminus \{\vec{0}\}$ we have that $\cup P_i = F^d \setminus \{\vec{0}\}$. Using the fact that A is a non-singular linear transform we have for all i:

- The set P_i is a $2r$-wise independent collection of points.
- P_i and P_{i+1} intersect at r random points.
- The polynomial $\hat{x}$ restricted to P_i (denoted $\hat{x}_{|P_i}$) is a univariate polynomial of degree at most $2hr$.

We will set up the reconstruction function by supplying it with an advice string which will allow it to compute the evaluation of $\hat{x}$ on all the points in $\{P_1, \cdots, P_{2(i^*-1)}\}$. This is done by giving as advice the $2hr$ coefficients of $\hat{x}|_{P_i}$ for $1 \leq i \leq 2(i^*-1)$. Thus, the length of the advice string is at most $4mhr \log q \leq a$ as stated. From these evaluations we can use the predictor f to attempt to predict the evaluations of $\hat{x}$ at the points P_i for $i > 2(i^*-1)$. In general, for $i > 2(i^*-1)$ we assume that we already computed the evaluations of $\hat{x}$ at points:

$$P_{i-2(i^*-1)}, \cdots, P_{i-1}$$

we will now show how to use these evaluations to compute the evaluations of $\hat{x}$ at P_i.

We first invoke f once for attempting to predict every points $z \in P_i$. That is we run f on the evaluations of $\hat{x}$ at $\{A^{-1}z, \cdots, A^{-i^*-1}z\}$ to obtain a list of ρ^{-2} "candidates" for $\hat{x}(z)$. (Note that $A^{-j}z \in A^{-j}P_i = P_{i-2j}$). We will show that with high probability these predicted values and the evaluations of $\hat{x}$ on P_{i-1} completely determine the values of $\hat{x}$ on P_i. The first step is to show that with high probability, many of the points z in P_i are predicted correctly.

Claim 2. *With probability at least $1 - 1/8q^d$ over the coin tosses of R:*

$$\Pr_{\vec{z} \in P_i}[\exists j \; f(\hat{x}(A^{-i^*+1}\vec{z}), \dots, \hat{x}(A^{-1}\vec{z}))_j = \hat{x}(\vec{z})] \geq \rho/4.$$

Proof. (of Claim 2) Let Y_ℓ be the indicator random variable for the event that the set of predicted values for ℓ-th point in P_i contains the evaluation of $\hat{x}$ at that point, and let $Y = \sum_{\ell=1}^{q} Y_\ell$. Since x is good, we have that for every ℓ, $E[Y_\ell] > \rho/2$. It follows that $\mu = E[Y] > (\rho/2)q$. Since P_i is a $2r$-wise independent set of points, we can apply Lemma 4.11, and get that the probability that $Y \leq (\rho/4)q$ is at most:

$$\Pr[|Y - E[Y]| \geq E[Y]/2] \leq \left(\frac{O(r)}{\rho q}\right)^r < \frac{1}{8q^d},$$

where the final inequality holds by our choice of r and q. $\square$

Therefore, with high probability, we have $(\rho/4)q$ "good" evaluations that agree with the degree $2rh$ univariate polynomial $\hat{x}_{|P_i}$, out of a total of $\rho^{-2}q$. By our choice of q and r, Lemma 4.10 applies, and we conclude that at most $8\rho^{-3}$ degree $2rh$ polynomials agree with out evaluations on this number of points.

Now, P_i intersects P_{i-1} at r random positions, and we know the evaluation of $\hat{x}$ at the points in P_{i-1}. Two distinct degree $2rh$ polynomials can agree on at most $2rh/q$ fraction of their points, so the probability that an "incorrect" polynomial from among our candidates agrees with $\hat{x}|_{P_i}$ on all r random points is at most:

$$(8\rho^{-3})\left(\frac{2rh}{q}\right)^r < \frac{1}{8q^d}$$

where the final inequality holds by our choice of r and q.

So, with probability at least $1 - \frac{1}{4q^d}$ over the random coins of R, we learn the evaluations of $\hat{x}$ on the points in P_i successfully.

After $2q^d$ such prediction steps, we have learned $\hat{x}$ on $F^d \setminus \{\vec{0}\}$, by property (4.4) of matrix A. By the union bound, the probability that every step of this reconstruction is successful is at least $1/2$. These evaluations uniquely determine $\hat{x}$, and the reconstruction function R then outputs x (which can be easily computed from $\hat{x}$). $\square$

Proceeding with the proof of the main theorem, by Lemma 4.15 we know that a $\rho/2$-fraction of x's are good. It follows by Lemma 4.16 that for a $\rho/2$-fraction of the x's the randomized procedure R succeeds with probability at least $1/2$. Therefore, there must be a fixing of the coin tosses of R for which we have:

$$\Pr_{x \leftarrow X}[\exists z \quad R^f(z) = x] > \frac{1}{2} \cdot \rho/2 = \rho/4.$$

Following the outline in section 4.3.2, this implies that $\exp(O(mdh \log q) - k) \geq \rho/4$ (resp. $\exp(O(mdh \log^2 q) - k) \geq \rho/4$), which contradicts the lower bound on k in the statement of the theorem. This completes the proof of the main theorem.

4.3.4 From q-ary extractors to (regular) extractors

We are now left with the task of converting a q-ary extractor into a regular one. The standard way to achieve this is to use "list decodable" error correcting codes. The transformation described in Lemma 4.18 below is essentially the information-theoretic analog of the hard-core bit constructions of Goldreich-Levin [GL89]. In the following definition Δ is the Hamming distance function.

Definition 4.17. *A binary code $C : \{0,1\}^{\bar{k}} \to \{0,1\}^{\bar{n}}$ is* (ρ, ℓ)-list-decodable*, if for all $r \in \{0,1\}^{\bar{n}}$, the set $S_r = \{x : \Delta(C(x), r) \leq (1/2 - \rho)\bar{n}\}$ has size at most ℓ. The code is* efficiently encodable *if C is computable in time poly$(\bar{n})$, and* efficiently list-decodable *if S_r can be computed from r in time poly$(\bar{n}, \ell)$.*

By the Johnson bound, any binary code with relative distance at least $1/2 - \rho^2$ is (ρ, ρ^{-2})-list-decodable.

Lemma 4.18 ([TSZS01]). *Let F be the field with q elements and let $C : \{0,1\}^{\bar{k}=\log q} \to \{0,1\}^{\bar{n}}$ be a (ρ, ρ^{-2})-list-decodable code. If $E : \{0,1\}^n \times \{0,1\}^t \to F^m$ is a (k, ρ) q-ary extractor, then $E' : \{0,1\}^n \times \{0,1\}^{t+\log \bar{n}} \to \{0,1\}^m$ defined by:*

$$E'(x; (y, j)) = C(E(x; y)_1)_j \circ \cdots \circ C(E(x; y)_m)_j$$

is a $(k, 2\rho m)$ extractor.

Proof. Suppose E' is not an extractor. Then there exists a function f violating property (4.1) with $\epsilon = 2\rho m$. It follows from an averaging argument that for a ρ-fraction of pairs (x, y)

$$\Pr_z[f(E'(x; (y, z))_{1,\cdots,i-1} = E'(x; (y, z))_i] \geq 1/2 + \rho$$

We now design a q-ary predictor f' for E. Given $i - 1$ q-ary inputs $w_1, \cdots, w_{i-1}$, we compute

$$r_j = f(C(w_1)_j, \cdots, C(w_{i-1})_j)$$

for $1 \leq j \leq \bar{n}$. Predictor f' outputs a list of size ρ^{-2} of those codewords that differ from r in at most $(1/2 - \rho)\bar{n}$ positions. For a ρ fraction of pairs (x, y) at least a $1/2 + \rho$ fraction of the r_js are predicted correctly, and hence this list contains $E(x, y)_i$. The existence of predictor f' contradicts E being a (k, ρ) q-ary extractor. ☐

There are explicit constructions of codes with the required minimum distance and short blocklength $\bar{n} = (\frac{\log q}{\rho})^{O(1)}$ (see, e.g., [GS00]). Thus, this transformation has minimal effect on the seed length. Even relatively simple codes, like a Reed-Solomon code concatenated with a Hadamard code yield the desired parameters.

If we are not optimizing constants in the seed length, we can even afford to use perhaps the simplest binary code, the Hadamard code[3], which has relative distance 1/2.

We now combine Lemma 4.18 with Theorem 4.13 to obtain the constructions reported in section 4.1.2.

Proof. (of theorem 4.1) We choose:

- $h = k^{\delta/4}$
- $d = \log n / (\log h - \log\log n)$
- $\rho = \Theta(\epsilon/m)$
- $q = \Theta(\rho^{-4}(hd)^2)$

We verify that the conditions of Theorem 4.13 (1) are met:

$$\binom{h+d}{d} > \left(\frac{h}{d}\right)^d \geq \left(\frac{h}{\log n}\right)^d = n$$

By the lower bound on k,

$$O(mhd\log q) + \log(1/\rho) = O(k^{1-\delta}k^{\delta/4}\log^2 n) < k$$

The lower bound on k also gives that $h \geq \log n > d$ and thus the extractor runs in time $q^{O(d)} = (hd)^{O(d)} = h^{O(d)} = n^{O(1)}$.

Using the Hadamard code for the conversion from q-ary extractors, the seed length $t = d\log q + \log q$ is as stated in the corollary. □

One of the advantages of the extractors constructed in [TSZS01] are that they optimize the leading constant in the seed length. By picking parameters appropriately, we achieve a similar result and prove theorem 4.2.

Proof. (of theorem 4.2) Note that when $m \leq \log n$ we can just output the seed, so we assume $m > \log n$, and consequently that $k > \log^{1/\delta} n$. We choose the following parameters:

- $h = \sqrt{k}$
- $d = \log n / (\log h - \log\log n)$
- $\rho = \Theta(\epsilon/m)$

[3] The Hadamard encoding of a $\log q$-bit string x is $C(x) = \{\sum x_i y_i \bmod 2\}_{y \in \{0,1\}^{\log q}}$.

- $q = \Theta(\rho^{-4} hd \log n)$

We verify that the conditions of Theorem 4.13(2) are met. As before, $\binom{h+d}{d} > n$ and $d < \log n$.

$$O(mhd \log^2 q) = O(k^{1/2+\delta} \log^3 n) < k$$

holds by the lower bounds on k and δ. Using a code with blocklength $(\log q/\rho)^{O(1)}$ for the conversion from q-ary extractors, the seed length is:

$$t = d \log q + O\left(\log \frac{\log q}{\rho}\right) \leq d \log h + d \cdot O(\log m + \log \log n).$$

By the lower bound on k, we have $h > \log^{1/(2\delta)} n$, and thus $d \log h \leq (1 + O(\delta)) \log n$ and $\log \log n / \log h = 2\delta$ from which we conclude $d \log \log n \leq O(\delta) \log n$. Finally, $d \leq \log n/((1-2\delta)\log h) = O(\delta) \log n / \log m$, and so altogether, we obtain the stated bound on t. □

Notice that when $k = n^{\Omega(1)}$, the seed length of both of the extractors in theorem 4.1,4.2 is $O(\log(n/\epsilon))$, which gives the asymptotically optimal dependence on ϵ. Achieving this dependence was a focus of [RRV99b, RRV99a], and we obtain it directly.

The quality of our extractors can be significantly improved by using a more complex transformation of q-ary extractors to (regular) extractors. Such a transformation was given in [TSZS01].

Theorem 4.19. *[TSZS01] Let F be the field with q elements. For every k, ρ, and m, there is a polynomial time computable function*

$$B : F^m \times \{0,1\}^{O(\log \log q) + \log^{O(1)}(1/\rho)} \to \{0,1\}^{(1-O(\sqrt{\rho}))m - O(\log^* m(\log^* m + \log(1/\rho)))}$$

such that for any (k, ρ) q-ary extractor E with output length m,

$$E'(x; (y, j)) = B(E(x, y), j)$$

is a $(k, O(\rho \log^* m))$-extractor.

The expressions above are a bit complicated. The important thing to notice is that when ρ is not too small, the q-ary extractor is converted into a regular extractor with roughly the same seed length and output length. While this is also the case with Lemma 4.18 the important difference is the relation between the error of the q-ary extractor and the final extractor. In Lemma 4.18 the error of the final extractor is $O(\rho m)$, as compared to $O(\rho \log^* m)$ in Theorem 4.19. In Theorems 4.1 and 4.2

we had to choose $\rho = O(\epsilon/m)$ to get ϵ error when applying Lemma 4.18. As the seed length of the q-ary extractor is at least $d \log(1/\rho) \geq d \log m$ we had to make sure that $d \log m$ was small, i.e., $O(\log n)$. Since $d \approx \log n / \log h$ (so that $\binom{h+d}{d} > n$), this forces us to choose h to be $m^{\Omega(1)}$. The effect of this choice is that we extract only a small fraction of the randomness in the source as $k \geq mh$. However using Theorem 4.19 we can choose $\rho = O(\epsilon/\log^* m)$. This allows us to choose much smaller h, (say $h = \log^{O(1)} n$), and extract a larger fraction of the randomness in the source.

Proof. (of theorem 4.3) We choose $h = \log^{1+\delta/2} n$, $d = \frac{\log n}{\log h - \log\log n}$, $\rho = \Theta(\epsilon/(\log^* m))$, and $q = \Theta(\frac{(hd)^2}{\rho^4})$. These choices meet the requirements of Theorem 4.13, and give the required parameters. The computations are similar to those made in Theorem 4.1. □

This almost matches the parameters of the current best explicit construction for general min-entropy [RSW00], and is by far a simpler construction.

We can further reduce the seed length to $(1+\delta)\log n$ at the cost of extracting slightly fewer bits as is the case in theorem 4.4.

Proof. (of theorem 4.4) We choose $h = \log^{\Theta(1/\delta)} n$, $d = \frac{\log n}{\log h - \log\log n}$, $\rho = \Theta(\epsilon/(\log^* m))$, and $q = \Theta(\frac{hd \log q}{\rho^4})$. The computations we have to make use the fact that $h > \log^{\Theta(1/\delta)} n$, and are similar to those made in Theorem 4.2. □

4.4 A new pseudo-random generator

In this section we exploit the connection Trevisan noticed between extractors and PRG's "the other way": we build PRGs (and HSGs) using the ideas outlined in section 4.2.3. Throughout, we identify the string $x \in \{0,1\}^n$ with the function $x : \{0,1\}^{\log n} \to \{0,1\}$ by setting $x(i) = x_i$. We denote by $S(x)$ the size of the smallest circuit computing function x. In this chapter it will be convenient to differ between the output length of the generator and the size of circuits it fools.

Definition 4.20 (PRG). *An* ϵ-PRG for size s *is a function* $G : \{0,1\}^t \to \{0,1\}^m$ *such that*

$$\forall\, 1 \leq i \leq m \text{ and all functions } f : \{0,1\}^{i-1} \to \{0,1\} \text{ with size } s \text{ circuits },$$
$$\Pr[f(G(U_t)_{1\ldots i-1}) = G(U_t)_i] \leq \frac{1}{2} + \frac{\epsilon}{m}. \quad (4.7)$$

Property (4.7) implies the more standard requirement that $|\Pr[C(G(U_t)) = 1] - \Pr[C(U_m) = 1]| \leq \epsilon$ for all size $s - O(1)$ circuits C, and in particular implies definition 1.4. As in Section 4.3 we can define the q-ary version of PRGs.

Definition 4.21 (q-ary PRG). *Let F be the field with q elements. A ρ-q-*ary PRG for size s *is a function $G : \{0,1\}^t \to F^m$ such that* $\forall\ 1 \leq i \leq m$ *and all functions $f : F^{i-1} \to F^{(\rho^{-2})}$ with size s circuits*

$$\forall\ 1 \leq i \leq m \textit{ and all functions } f : F^{i-1} \to F^{(\rho^{-2})} \textit{ with size } s \textit{ circuits },$$
$$\Pr[\exists j\ f(G(U_t)_{1\ldots i-1})_j = G(U_t)_i] \leq \rho. \quad (4.8)$$

As in section 4.3 we will focus on constructing q-ary generators and later transform them into (binary) generators.

4.4.1 Changes to the extractor construction

Given a function $x : \{0,1\}^{\log n} \to \{0,1\}$ which cannot be computed by circuits of size s, our goal is to construct a q-ary PRG $G_x : \{0,1\}^d \to F^m$. The construction will be quite similar to the extractor construction, and to prove correctess, we will derive a contradiction from the existence of a "predictor" f violating (4.8) above. As with the extractor this is done by describing a "reconstruction procedure" R. There are two important differences in what we require of R in the PRG setting as compared to the extractor setting. First, R takes an additional input i (as well as the short advice string a which is the same for all i) and should output $x(i)$ as opposed to simply outputting all of x. Second, R should be *efficient*; that is, it should run in time polynomial in m. Since f also has a small circuit, we can compute $x(i)$ efficiently by evaluating $R^f(a, i)$, which for the proper choice of parameters will contradict the hardness of the function x. In order to meet these new requirements, we need to make some changes to the construction and the proof.

The encoding: In the extractor setting, $\hat{x}$ is the Reed-Muller encoding of x, and in order to determine $x(i)$ from the encoding, we need to learn enough of $\hat{x}$ to be able to interpolate and find its coefficients. The efficiency demands on R in the PRG setting preclude being able to learn this many evaluations of $\hat{x}$ (we would need n evaluations, but are allowed only poly(m) time). Therefore, we use an alternate "low-degree extension" encoding for the PRG. In this scheme we again encode x as a low-degree polynomial $\hat{x} : F^d \to F$, but we also ensure that there is an efficiently computable function $\ell : [n] \to F^d$ such that $x(i) = \hat{x}(\ell(i))$. Thus, we can determine $x(i)$ by learning only the specified evaluation of $\hat{x}$.

The standard way to produce $\hat{x}$ from x is to pick an arbitrary set $H \subseteq F$ with $|H|^d = n$ and any efficient one-to-one function $\ell : [n] \to H^d$, and define $\hat{x}$ to be any polynomial with degree at most h in each variable for which $\hat{x}(\ell(i)) = x(i)$. Our reconstruction procedure operates on the cyclic group with generator A (which corresponds to F^d), and if we use this standard low-degree extension, we are stuck having to compute the integer j such that $A^j\vec{1} = \ell(i)$ whenever we want to determine $x(i)$. Finding such a j is a discrete-log problem that we don't know how to solve efficiently. Instead, we use a specific embedding of x into $\hat{x}$ that avoids this problem: $x(i)$ is embedded at location $A^{ip}\vec{1} \in F^d$, for a fixed integer p; thus simply knowing i gives us the required exponent. Such an embedding is somewhat delicate: we need to arrange for $\{A^{ip}\vec{1}\}_{i\in\mathbb{Z}}$ to coincide with H^d; the details on how to achieve this are given in the technical section..

The reconstruction: As explained in the introduction, the main idea is to use *several* predictors with varying strides when performing the reconstruction. This allows us to travel quickly from a fixed point (like $\vec{1}$) to any given point in F^d. To implement this idea we need two new ingredients. First, the curves C_1 and C_2 need to have the intersection properties we used in the extractor setting *for each stride* that we will use for the PRG. This is achieved by generalizing the idea used to obtain the intersections for the extractor setting, which in turn requires slightly larger degree curves. Second, for every $\vec{v} \in F^d$ we need to describe an efficiently computable short sequence of prediction steps (with varying strides) which starts from $\vec{1}$ and reaches $\vec{v}$ – this is used by the reconstruction procedure to rapidly learn $x(i)$ using only the predictor and the initial evaluations of $\hat{x}$ supplied by the advice string.

4.4.2 The construction

Our construction uses a hard function $x : \{0,1\}^{\log n} \to \{0,1\}$ to build a PRG $G_x : \{0,1\}^t \to \{0,1\}^m$. Like in the extractor setting, we first encode x into $\hat{x}$ using a multivariate polynomial code. Here however, instead of using standard Reed-Muller codes, we need to take $\hat{x}$ to be the *low-degree extension* of x, and we need to be careful how we do even that.

Just as before, the major parameters are the field size q, the dimension d and the degree h. As in the previous section we will think of F^d as both a vector-space and the extension field of F. However, we also need a new ingredient. We will require that F has a subfield F' of size h (which forces h to be a prime power, and q to be a power of h). We note that F'^d is a subset of F^d, and we have the following lemma which extends Lemma 4.12.

Lemma 4.22. *Let h, q and d be such that: h is a prime power, q is a power of h, and d and $\log_h q$ are relatively prime. Then there exist invertible $d \times d$ matrices A*

and A' with entries from F which satisfy:

- *For any non-zero vector $\vec{v} \in F^d$: $\{A^i\vec{v}\}_{i\in\mathbb{Z}} = F^d \setminus \{0\}$*
- *For any non-zero vector $\vec{v} \in F'^d$ $\{A'^i\vec{v}\}_{i\in\mathbb{Z}} = H^d \setminus \{0\}$.*
- *$A' = A^{(q^d-1)/(h^d-1)}$.*
- *A can be found in time $q^{O(d)}$.*
- *A' can be found in time $h^{O(d)}$.*

Proof. We first need a polynomial of degree d *with coefficients in* H that is irreducible over F. Let $c = \log_h q$ and let $p(z)$ be a polynomial of degree d that is irreducible over $H = GF(h)$. Let α be a root of p and notice that the field $GF(h^c)[\alpha]$ contains both F and $GF(h^d)$. Furthermore, $GF(h^c)[\alpha]$ is contained in $GF(h^{cd})$. However, no *proper* subfield of $GF(h^{cd})$ can contain both $F = GF(h^c)$ and $GF(h^d)$ because c and d are relatively prime. Therefore $GF(h^c)[\alpha] = GF(h^{cd})$, which implies that p is irreducible over $F = GF(h^c)$, as desired. We note that p can be found by exhaustive search in time $h^{O(d)}$.

We now construct $GF(q^d)$ by considering its elements to be polynomials over F modulo $p(z)$. Let $\{1, z, z^2, \ldots z^{d-1}\}$ be the standard basis for $GF(q^d)$ over F. In this basis, the set H^d is exactly the following subset of $GF(q^d)$:

$$\{\sum_{i=0}^{d-1} \beta_i z^i | \beta_i \in F'\}.$$

Since $p(z)$ has all coefficients in H, this subset is closed under multiplication, and hence $H^d \setminus \{0\}$ is a subgroup of order $h^d - 1$ of the multiplicative group of $GF(F^d)$. Therefore, if we find A corresponding to a generator of the multiplicative group of $GF(F^d)$ (as in Lemma 4.12), then $A' = A^{(q^d-1)/(h^d-1)}$ generates the unique subgroup of order $h^d - 1$, whose elements are $H^d \setminus \{0\}$. As before A can be found by exhaustive search. □

For our low-degree-extension of x, we require (compare to (4.3)):

$$h^d > n. \tag{4.9}$$

Now, in our low degree extension, we will "embed" x into a polynomial defined over the vector-space F^d. The embedding works as follows: we want $\hat{x}(A'^i\vec{1}) = x(i)$. Here, $\vec{1}$ is the all-ones vector, which is in $F'^d \subseteq F^d$ since $1 \in F' \subseteq F$, and which serves as a reference vector throughout the construction. Since $h^d - 1 \geq$

n, there are enough "slots" to embed all of x, and since A' generates $F'^d \setminus \{0\}$, we have embedded all of x in a d dimensional cube of size h^d. Therefore there exists a low degree polynomial $\hat{x}$ over F^d with degree h in each variable such that $\hat{x}(A'^i\vec{1}) = x(i)$. Note that in this section h denotes the degree in individual variables and the total degree is at most hd. The computation of $\hat{x}$ from x can be done in $\text{poly}(n, q^d) = q^{O(d)}$ time.

We now describe d "candidate" PRGs. For $0 \leq j < d$ we define functions $G_x^{(j)} : \{0,1\}^{d \log q} \to F^m$ as follows. We think of the input $\vec{v}$ as a vector in F^d.

$$\begin{aligned}
G_x^{(0)}(\vec{v}) &= \hat{x}(A^1\vec{v}) \circ \hat{x}(A^2\vec{v}) \circ \cdots \circ \hat{x}(A^m\vec{v}) \\
G_x^{(1)}(\vec{v}) &= \hat{x}(A^{q\cdot 1}\vec{v}) \circ \hat{x}(A^{q\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q\cdot m}\vec{v}) \\
G_x^{(2)}(\vec{v}) &= \hat{x}(A^{q^2\cdot 1}\vec{v}) \circ \hat{x}(A^{q^2\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^2\cdot m}\vec{v}) \\
\vdots \quad & \quad \vdots \\
G_x^{(j)}(\vec{v}) &= \hat{x}(A^{q^j\cdot 1}\vec{v}) \circ \hat{x}(A^{q^j\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^j\cdot m}\vec{v}) \\
\vdots \quad & \quad \vdots \\
G_x^{(d-1)}(\vec{v}) &= \hat{x}(A^{q^{d-1}\cdot 1}\vec{v}) \circ \hat{x}(A^{q^{d-1}\cdot 2}\vec{v}) \circ \cdots \circ \hat{x}(A^{q^{d-1}\cdot m}\vec{v}) \qquad (4.10)
\end{aligned}$$

Note that each $G_x^{(j)}$ corresponds to using our q-ary extractor construction with the "successor function" A^{q^j}. Our main theorem will show that at least one of these functions is a q-ary PRG, provided x is a sufficiently hard predicate.

Theorem 4.23 (PRG main). *For every n, d, h and q satisfying (4.9) and the conditions of Lemma 4.22, at least one $G_x^{(j)}$ is a ρ-q-ary PRG for size s, provided that $S(x) > s \cdot poly(m, q)$ and $q > max(\Omega(\rho^{-4}hd^2 \log^2 q), 2d^4 \log^2 q)$. Furthermore, all the $G_x^{(j)}$s are computable in time $poly(q^d, n)$ with oracle access to x.*

We prove theorem 4.23 in section 4.4.4. We now show how to construct a *single* (binary) PRG. It will be convenient to fix all parameters as functions of n, m. We will require that $\log n \leq m \leq n$ and set:

- $\rho = m^{-3}$.
- $h = m$.
- $d = \log n / \log m$.
- $q = \Omega(\rho^{-4}hd^2 \log^2 q) < m^{O(1)}$.

By these choices the seed length of each "candidate generator" is $O(\log n)$. Our next goal is to transform the q-ary candidates into binary ones, and for this we use a computational analogue of Lemma 4.18. In this case, it is not sufficient that the encoding procedure is efficient. We also require that the code has efficient (list)-decoding.

Lemma 4.24. *Let F be the field with q elements, Let $C : \{0,1\}^{\bar{k}=\log q} \to \{0,1\}^{\bar{n}}$ be a $(\rho, 1/\rho^2)$-efficiently list decodable error-correcting code. If $G : \{0,1\}^t \to F^m$ is a ρ-q-ary generator for size $s\bar{n}+m\bar{n}^{O(1)}+(\bar{n}/\rho)^{O(1)}$, then $G' : \{0,1\}^{t+\log \bar{n}} \to \{0,1\}^m$ defined by:*

$$G'(y,j) = C(G(y)_1)_j \circ C(G(y))_2)_j) \circ \cdots \circ C(G(y))_m)_j$$

is a $O(\rho m)$-PRG for size s

Lemma 4.24 follows from the proof of lemma 4.18 by using the additional efficiency requirements. Using, for example, [GS00], we can obtain such a code with $\bar{n} = \text{poly}(\bar{k}, \rho^{-1})$. In our setting $\bar{k} = \log q = O(\log m)$ and we only require that

$$\bar{n} = m^{O(1)} \tag{4.11}$$

Thus, we can use the much simpler Hadamard code and the list decoding procedure of [GL89].

We use lemma 4.24 to transform each of the $G_x^{(j)}$'s into a binary function ${G'_x}^{(j)} : \{0,1\}^{O(\log n)} \to \{0,1\}^m$. We conclude that:

Corollary 4.25. *At least one ${G'_x}^{(j)}$ is a $1/m$-PRG for size s provided that $S(x) > sm^{O(1)}$. Furthermore, all the ${G'_x}^{(j)}$'s are computable in time $n^{O(1)}$ with oracle access to x.*

Our PRG is obtained by "XOR-ing" the d candidates functions with independent seeds:

$$G_x(y_0, \cdots, y_{d-1}) = {G'_x}^{(0)}(y_0) \oplus \cdots \oplus {G'_x}^{(d-1)}(y_{d-1}) \tag{4.12}$$

It is standard that "XOR-ing" many candidates where one of them is a PRG for size s indeed produces a PRG for size $s - m$; for a proof, see [ISW00]. The seed length of G_x is $O(d \log n) = O(\frac{\log^2 n}{\log m})$. This proves theorem 4.5 and matches the parameters of the PRG construction of [STV99].

The most important implication of this corollary is a new proof of the Impagliazzo-Wigderson Theorem [IW97], which states that $BPP = P$ if there exist a function family in E that requires exponential size circuits.

4.4.3 Hitting set generators and an optimal hardness vs. randomness tradeoff

Hitting set generators are designed to derandomize RP.

Definition 4.26 (HSG). *An ϵ-HSG for size s is a function $H_x : \{0,1\}^t \rightarrow \{0,1\}^m$ such that for all functions $C : \{0,1\}^m \rightarrow \{0,1\}$ computable by size s circuits that have $\Pr_z[C(z) = 1] \geq \epsilon$, there exists some y for which $C(H_x(y)) = 1$.*

One of the ${G'_x}^{(j)}$'s is a PRG and therefore if we choose a random candidate, we will hit the "right one" with positive probability. We define:

$$H_x(y, j) = {G'_x}^{(j)}(y)$$

It is standard that choosing a random candidate from a collection of functions where one of them is a PRG produces an HSG; for a proof see [ISW99]. Very few bits (at most $O(\log \log n)$) are needed to choose j and thus we get an *optimal* HSG and prove theorem 4.6

In [ACR96] (see also [ACRT97, BF99, GVW00]) it was shown how to derandomize two-sided error probabilistic algorithms using an HSG. Applying this, we extend the Impagliazzo-Wigderson Theorem [IW97] to any hardness assumption.

Corollary 4.27. *If there exists a function family $g = \{g_n\} \in E$ which require size $s(n)$ circuits, then BPTIME(ℓ) $\subseteq$ DTIME($2^{O(s^{-1}(\ell^{O(1)}))}$).*

4.4.4 Proof of the main PRG theorem

In this section we prove Theorem 4.23. We fix a string $x \in \{0,1\}^n$, and let $\hat{x} \in F^d$ be the encoding of x described in the previous section. Assume for the purpose of contradiction that *no* $G_x^{(j)}$ is an ρ-q-ary PRG. Then by definition we have integers $i^{(j)}$ and next-element predictors $f^{(j)} : F^{i^{(j)}-1} \rightarrow F^{(\rho^{-2})}$ violating property (4.8) for each $G_x^{(j)}$, respectively. WLOG we will assume that $i^{(j)} = m$ for all j, as predictor $f^{(j)}$ can simply ignore its first $m - i^{(j)}$ inputs. Each of these predictors can be implemented by a size s circuit. As in the proof of Theorem 4.13 our task is to use these predictors and a short advice string to reconstruct x. However, in this setup reconstructing x means constructing a small circuit that computes $x(i)$ from input i.

It will be helpful to abstract the process used in the proof of Theorem 4.13 to learn a new curve and present it as a procedure. In the remainder of this section $deg(\hat{x})$ denotes the total degree of polynomial $\hat{x}$.

Procedure Learn Next Curve

- Input:
    - next curve $C : F \to F^d$: a degree v polynomial
    - reference points $R \subseteq F$: a set of elements of F
    - stride j: an integer in $[0 \ldots (d-1)]$
    - input evaluations $\{a_t^i\}_{t \in F, i \in [1 \ldots (m-1)]}$ and $\{b_t\}_{t \in R}$: elements of F whose *intended values* are $a_t^i = \hat{x}(A^{-iq^j} C(t))$ and $b_t = \hat{x}(C(t))$.
- Output:
    - output evaluations $\{c_t\}_{t \in F}$: elements of F whose *intended values* are $c_t = \hat{x}(C(t))$.
- Action:
    - For each $t \in F$, compute $f^{(j)}(a_t^{m-1}, a_t^{m-2}, \ldots, a_t^1)$, which gives a set S_t of ρ^{-2} values.
    - Apply Lemma 4.10 on the $q\rho^{-2}$ pairs $\{(t,e)\}_{t \in F, e \in S_t}$ to obtain a list of at most $8\rho^{-3}$ degree $deg(\hat{x})v$ univariate polynomials $p(t)$ which contains all polynomials for which $p(t) \in S_t$ for at least $\rho q/4$ values of t. If this list is empty, fail.
    - If the list contains a unique polynomial $p(t)$ for which $p(t) = b_t$ for all $t \in R$, output $\{p(t)\}_{t \in F}$; otherwise fail.

We say that **Learn Next Curve** *succeeds* on an input if its output evaluations are the intended values given that its input evaluations are the intended values. We now argue that for a random next curve C and a random set of reference points $R \subseteq F$, the procedure succeeds with high probability.

Lemma 4.28. *For all strides j,*

$$\Pr_{C,R}[\textbf{Learn Next Curve } \textit{succeeds}] \geq 1 - \left(\frac{O(v)}{\rho q}\right)^{v/2} - (8\rho^{-3})\left(\frac{v deg(\hat{x})}{q}\right)^r,$$

where $C : F \to F^d$ is a uniformly chosen degree $v \leq \rho q$ curve, and $R \subseteq F$ is a uniformly chosen subset of F of size r.

Proof. The argument is identical to that in Section 4.3.3. We first argue that with high probability $\hat{x}(C(t))$ is predicted correctly for many t's.

Claim 3. *With probability at least $1 - \left(\frac{O(v)}{\rho q}\right)^{v/2}$ over the choice of C*

$$\Pr_{t \in F}[\exists e \in S_t \; e = \hat{x}(C(t))] \geq \rho/4$$

Proof. (of Claim 3) The proof is identical to that of Claim 2. We argue that the collection of points $\{C(t)\}_{t\in F}$ is v-wise independent (over the choice of C), and then use Lemma 4.11. $\square$

It follows that with probability at least $1 - \left(\frac{O(v)}{\rho q}\right)^{v/2}$ the procedure applies Lemma 4.10 with enough "correct pairs", and therefore one of the polynomials in the list is the polynomial $p(t) = \hat{x}(C(t))$. Two polynomials of degree $vdeg(\hat{x})$ can agree on at most a $vdeg(\hat{x})/q$ fraction of their points, so the probability that an "incorrect" polynomial from the list agrees with p on r random points is at most

$$(8\rho^{-3})\left(\frac{vdeg(\hat{x})}{q}\right)^r.$$

If $\hat{x}(C(t))$ is predicted correctly for enough t's so that $p(t)$ appears in the list, and no "incorrect" polynomials in the list agree on the r random points in R, the procedure succeeds. $\square$

In the remainder of the proof, if $C : F \to F^d$ is a curve and A is a $d \times d$ matrix, then we denote by AC the function defined by $AC(t) = A \cdot C(t)$. In some contexts we also use AC to denote the set $\{AC(t)\}_{t\in F}$, and we adopt the shorthand $A(C_1 \cup C_2)$ for $AC_1 \cup AC_2$.

Lemma 4.29. *There exist degree $v = O(d^2 \log q)$ curves C_1 and C_2 for which the following hold:*

- *for all i and all j,* **Learn Next Curve** *succeeds given* next curve $A^{i+q^j}C_1$, reference points $A^{i+q^j}C_1 \cap A^iC_2$ *and* stride j, *and*
- *for all i and all j,* **Learn Next Curve** *succeeds given* next curve A^iC_2, reference points $A^iC_1 \cap A^iC_2$ *and* stride j.

Proof. We pick C_1 and C_2 randomly with certain intersection properties, and apply Lemma 4.28 to argue that with high probability each invocation of **Learn Next Curve** listed above succeeds. A union bound then shows that with non-zero probability *all* such invocations succeed, and the lemma follows.

Set $r = O(d \log q)$ and $u = d + 1$, and pick ur random points from F^d:

$$\vec{y_{11}}, \vec{y_{12}}, \ldots \vec{y_{1r}}, \vec{y_{21}}, \vec{y_{22}}, \ldots \vec{y_{2r}}, \quad \ldots \quad \vec{y_{u1}}, \vec{y_{u2}}, \ldots \vec{y_{ur}}$$

and ur random values from F:

$$t_{11}, t_{12}, \ldots t_{1r}, t_{21}, t_{22}, \ldots t_{2r}, \quad \ldots \quad t_{u1}, t_{u2}, \ldots t_{ur}.$$

By the the lower bound on q, the probability that these are all distinct is at least $1/2$. If they are distinct, we define the degree $ur-1$ curve C_1 so that $C_1(t_{ij}) = \vec{y_{ij}}$ for all i, j; similarly, we define the degree $ur-1$ curve C_2 so that $C_2(t_{1j}) = \vec{y_{1j}}$ for all j and $C_2(t_{ij}) = A^{q^{i-2}}\vec{y_{ij}}$ for $i \geq 2$ and all j. The curves C_1 and C_2 have the following properties for all i:

- The sets A^iC_1 and A^iC_2 are ur-wise independent collections of points.
- For all j the sets $A^i \cdot A^{q^j}C_1$ and A^iC_2 intersect at r random positions.
- The sets A^iC_1 and A^iC_2 intersect at r random positions.

Therefore, by Lemma 4.28, for each *individual* invocation of **Learn Next Curve** listed in the statement of the lemma, the procedure succeeds with probability at least:

$$1 - \left(\frac{\Omega(v)}{\rho q}\right)^{v/2} - (8\rho^{-3})\left(\frac{vdeg(\hat{x})}{q}\right)^r,$$

which is at least $1 - \frac{1}{8dq^d}$ by our choice of parameters. By the union bound, *all* $2dq^d$ invocations of **Learn Next Curve** listed in the statement of the lemma succeed simultaneously with probability at least $3/4$. Discounting the possibility that the t_{ij}'s are not distinct, we have that with probability at least $1/4$ over our random choices, we obtain C_1 and C_2 for which the lemma holds. □

Using **Learn Next Curve** with the "good" curves C_1 and C_2, we can now construct a small circuit which when given input i produces $x(i)$. The basic step involves *two* invocations of **Learn Next Curve** to learn the evaluation of $\hat{x}$ at the points $A^i(C_1 \cup C_2)$, for some i. Specifically, we first invoke **Learn Next Curve** with next curve A^iC_1, reference points $A^{i-q^j}C_2 \cap A^iC_1$ and stride j; then we invoke **Learn Next Curve** with next curve A^iC_2, reference points $A_iC_1 \cap A_iC_2$ and stride j. We will call this two-step process *interleaved learning* of $A^i(C_1 \cup C_2)$ using stride j. Notice that to supply **Learn Next Curve** with the correct input evaluations for interleaved learning of $A_i(C_1 \cup C_2)$ using stride j, we need to know the evaluation of $\hat{x}$ at points:

$$\bigcup_{k=1}^{m-1} A^{i-kq^j}(C_1 \cup C_2).$$

By our encoding, we know that $x(i) = \hat{x}(A'^i\vec{1}) = \hat{x}(A^{ip}\vec{1})$. Also, since A generates $F^d \setminus \{\vec{0}\}$, we know that $C_1(1) = A^a\vec{1}$ for some integer a between 0 and $q^d - 1$. Thus we need to "travel" $b = ip - a \pmod{q^d - 1}$ steps from curve C_1 to reach curve A^bC_1, and then we output the evaluation of $\hat{x}$ at $A^bC_1(1) =$

$A^{ip-a}A^a\vec{1} = A^{ip}\vec{1}$. Our circuit will be supplied with C_1, C_2, a, and the evaluation of $\hat{x}$ at:

$$\bigcup_{k=1}^{m-1} A^k(C_1 \cup C_2)$$

as non-uniform advice.

We now use **Learn Next Curve** in d phases. Write $b' = (b - mq^{d-1}) \bmod (q^d - 1)$ in its q-ary representation: $b' = \sum_{j=0}^{d-1} b_j q^j$. We maintain the invariant that after phase j, we have learned $\hat{x}$ at $A^w(C_1 \cup C_2)$ for an integer w for which w and b' agree on the least significant $j+1$ digits of their q-ary representation. Specifically, we execute the following sequence:

Phase 0**:** Perform interleaved learning of $A^{m+k}(C_1 \cup C_2)$ using stride 0, for $k = 0, 1, \ldots, (m+1)q$. Notice that the non-uniform advice provides the needed input evaluations.

Phase 1**:** Perform interleaved learning of $A^{mq+b_0+kq}(C_1 \cup C_2)$ using stride 1, for $k = 0, 1, \ldots, (m+1)q$. Notice that the values learned in phase 0 provide the needed input evaluations.

Phase 2**:** Perform interleaved learning of $A^{mq^2+b_0+qb_1+kq^2}(C_1 \cup C_2)$ using stride 2, for $k = 0, 1, \ldots, (m+1)q$. Notice that the values learned in phase 1 provide the needed input evaluations.

⋮

Phase j**:** Perform interleaved learning of $A^{mq^j+\sum_{t=0}^{j-1} b_t q^t + kq^j}(C_1 \cup C_2)$ using stride j, for $k = 0, 1, \ldots, (m+1)q$. The values learned in phase $j-1$ provide the needed input evaluations.

⋮

Phase $d-1$**.** Perform interleaved learning of $A^{mq^{d-1}+\sum_{t=0}^{d-2} b_t q^t + kq^{d-1}}(C_1 \cup C_2)$ using stride $d-1$, for $k = 0, 1, \ldots, b_{d-1}$. The values learned in phase $d-2$ provide the needed input evaluations. The last value of k yields $A^{mq^{d-1}+b'}(C_1 \cup C_2)$, and note that $mq^{d-1} + b' \equiv b \pmod{q^d - 1}$. Since $A^{q^d-1} = I$ we have learned $\hat{x}(A^b C_1(1)) = x(i)$, which we output.

Notice that we have invoked **Learn Next Curve** $O(mqd)$ times. Each invocation requires poly(m, q) computation time and invokes a predictor (with a circuit of size s) q times. The total computation time is therefore $O(mqd(sq + \text{poly}(m, q)))$, and the non-uniform advice has size $O(mdhur \log q)$ so altogether the circuit has size $s \cdot \text{poly}(m, q)$. Therefore, if x has hardness greater than this value, we have a contradiction, implying that some $G_x^{(j)}$ must be an ϵ-q-ary PRG. This concludes the proof of Theorem 4.23.

4.5 Pseudo-random generators for nondeterministic circuits

Just as BPP is a randomized version of P, the class AM (defined in [Bab85, BM88]) is a randomized version of NP. To derandomize BPP (ideally, prove $BPP = P$), we can use a function with large non-uniform (deterministic) complexity to build PRGs that "fool" small (deterministic) circuits; to derandomize AM (ideally, prove $AM = NP$), we use a function that requires large non-uniform *nondeterministic* complexity to build PRGs that "fool" small *nondeterministic* circuits. The precise meaning of "non-uniform nondeterministic complexity" is important here: Arvind and Kobler [AK97] observed that the NW PRG [NW94] also works in the nondeterministic setup. More precisely, given a *nondeterministic* predictor for the NW PRG the proof of [NW94] gives a nondeterministic circuit which computes the function correctly on a fraction of inputs slightly larger than one half. It follows that the NW-generator can be used to fool nondeterministic circuits given a function which is hard on average. In the standard setup when given a function which is hard on the worst case it can be converted via "hardness amplification" (as in [BFNW93, Imp95, IW97, STV99]) into a function which is hard enough on the average to run the NW PRG. Unfortunately, these transformations do not transform worst case nondeterministic hardness into average case nondeterministic hardness. Klivans and van Melkebeek observed in [KvM99] that the NW PRG (as well as the hardness amplification constructions) relativizes, and therefore functions that are hard for *circuits with SAT oracle gates* suffice for PRG's which fool circuits with SAT oracle gates, which in turn derandomize AM. Miltersen and Vinodchandran [MV99] used novel techniques to show that functions hard for *single-valued nondeterministic circuits* suffice (in this case to build a HSG for AM). The latter hardness assumption is weaker than that of Klivans and van Melkebeek and follows from standard nondeterministic hardness – to see this it is useful to view the two hardness assumptions as the non-uniform analogs of the classes P^{NP} and $NP \cap coNP$, respectively.

Like the NW PRG, the MV HSG does not give an optimal hardness vs. randomness tradeoff; in fact it fails altogether if the hard function has hardness less than $2^{\sqrt{\log n}}$ (on $\log n$ bits inputs). We show that given a function with large *single-valued nondeterministic hardness*, our HSG construction actually fools small *nondeterministic circuits*, and hence we derandomize AM. This is *not* a straightforward relativization, but instead relies on specific properties of our proof technique. As a result we obtain an *optimal* conversion of nondeterministic hardness into pseudo-randomness fooling nondeterministic circuits, solving an open problem raised by [MV99]. The same trick applies to our PRG construction, giving the first PRG that fools nondeterministic circuits using only a worst-case single-valued nondeterministic hardness assumption. nondeterministic hardness It is interesting to note that the technique we use here to deal with single-valued nondeterministic hardness actually applies to the NW generator when used in conjunction with hardness amplification from [STV99].

To state our result we need to briefly review some definitions of nondeterministic circuits.

Definition 4.30. *A* nondeterministic circuit C *(resp.* co-nondeterministic circuit C*) is a an ordinary circuit with a single output gate and two sets of inputs:* $x_1, x_2, \ldots, x_n$ *and* $y_1, y_2, \ldots, y_m$*. The function* $f : \{0,1\}^n \to \{0,1\}$ *computed by* C *is defined by* $f(x) = 1$ *iff* $\exists y\ C(x,y) = 1$ *(resp.* $\forall y\ C(x,y) = 1$*).*

Notice that if f is computed by a nondeterministic circuit of size s, then $\neg f$ is computed by a co-nondeterministic circuit of size s, and vice versa.

Definition 4.31. *A* single-valued (SV) nondeterministic circuit C *is an ordinary circuit with a single output gate, a single "flag" input* z*, and two sets of inputs:* $x_1, x_2, \ldots, x_n$ *and* $y_1, y_2, \ldots, y_m$*. A function* $f : \{0,1\}^n \to \{0,1\}$ *is computed by* C *if* $(C(0,x,y) = 1) \Rightarrow (C(1,x,y) = f(x))$*, and for all* x*,* $\exists y\ C(0,x,y)$*.*

Whereas nondeterministic circuits verify that the value of a function is "one", SV-nondeterministic circuits verify the value in both cases. Indeed, notice that if f is computed by a SV-nondeterministic circuit of size s, then $\neg f$ is also computed by a SV-nondeterministic circuit of size s. This means that every function which has a small SV circuit also has both small nondeterministic and co-nondeterministic circuits, and vice versa.

We now define the concepts analogous to that of section 4.4 for nondeterministic circuits.

Definition 4.32 (nondeterministic PRG). *An* ϵ-nondeterministic PRG for size s *is a function* $G : \{0,1\}^t \to \{0,1\}^m$ *such that*

$\forall\ 1 \le i \le m$ *and all functions* $f : \{0,1\}^{i-1} \to \{0,1\}$ *with size* s *nondeterministic or co-nondeterministic circuits*,

$$\Pr[f(G(U_t)_{1 \ldots i-1}) = G(U_t)_i] \le \frac{1}{2} + \frac{\epsilon}{m}. \quad (4.13)$$

It should be noted that the argument of Yao [Yao82] also works in this setup and an ϵ-nondeterministic PRG produces a distribution which cannot be distinguished from random by nondeterministic (or co-nondeterministic) circuits.

Our construction and results translate to the nondeterministic setup with exactly the same parameters. The only thing we need to change is the proof, and the additional argument used for the nondeterministic setup are outlined in section 4.5.2. In the next section we state our result which are analogous to those in section 4.4 with exactly the same parameters.

4.5.1 Our results

Let $S_{SV}(x)$ denote the size of the smallest SV nondeterministic circuit computing the function $x : \{0,1\}^{\log n} \to \{0,1\}$ and let ${G'}_x^{(j)} : \{0,1\}^{d \log q + \log \bar{n}} \to \{0,1\}^m$ be the functions defined by (4.10) after applying the transformation described in Lemma 4.24 with $\rho = \epsilon/(8m)$ to each $G_x^{(j)}$ (using, e.g., the Hadamard code, so $\bar{n} = q$). The next theorem is analogous to Theorem 4.23.

Theorem 4.33 (nondeterministic PRG main). *For every n, d, and h satisfying (4.9) and the conditions of Lemma 4.22, at least one ${G'}_x^{(j)}$ is a ϵ-nondeterministic PRG for size s, provided that $S_{SV}(x) > s \cdot poly(m, q)$ and $q > \Omega((m/\epsilon)^{-4} h d^2 \log^2 q)$. Furthermore, all the ${G'}_x^{(j)}$s are computable in time $poly(q^d, n)$ with oracle access to x.*

There is however, a slight difference, in theorem 4.33 we argue about the binary versions of the candidate generators, whereas in theorem 4.23 we argue about the q-ary versions. In the deterministic setup this makes no difference, however in the nondeterministic setup we do not know how to convert from q-ary to binary, and will rely on particular properties of our construction.

Fixing the parameters and using exclusive or to combine the candidate generators as in section 4.4, we obtain the theorem 4.7.

The notions of one-sided error and two-sided error coincide for AM and nondeterministic hitting set generators suffice to derandomize it.

Definition 4.34 (nondeterministic HSG). *An ϵ-nondeterministic HSG for size s is a function $H_x : \{0,1\}^t \to \{0,1\}^m$ such that for all functions $C : \{0,1\}^m \to \{0,1\}$ computable by size s nondeterministic or co-nondeterministic circuits that have $\Pr_z[C(z) = 1] \geq \epsilon$, there exists some y for which C accepts $H_x(y)$.*

It should be noted that the nondeterministic HSG of Miltersen and Vinodchandran [MV99] achieves an error ϵ which approaches 1 as a function of m. Using such an HSG for AM requires reducing the error of the AM protocol using explicit dispersers. Our construction achieves small ϵ. Another major difference is that our construction achieves the optimal tradeoff and in particular works when the hardness is small. Combining the candidate generators into an HSG as in section 4.4 gives theorem 4.8.

Finally, the HSG can be used to derandomize AM, giving the following optimal tradeoff:

Corollary 4.35. *If there exist a family of functions $f = \{f_n\} \in NE \cap coNE$ with $S_{SV}(f_n) > s(n)$, then $AM \subseteq NDTIME(2^{O(s^{-1}(n^{O(1)}))})$.*

4.5.2 Proof of the main nondeterministic PRG theorem

In this section we prove Theorem 4.33. The proof follows the outline of the proof of Theorem 4.23. We first explain why we need to modify the proof of Theorem 4.23 for it to work in the nondeterministic setting.

The proof of Theorem 4.23 constructs a small (deterministic) circuit C which computes x when given (deterministic) circuits which compute the predictors $f^{(j)}$'s. In the nondeterministic setting each one of the $f^{(j)}$'s has either a small nondeterministic circuit or a small co-nondeterministic circuit, and we want to construct a small SV nondeterministic circuit C computing x. Suppose that $f^{(j)}$ is a nondeterministic circuit, and that in the course of its computation, C wishes to evaluate $f^{(j)}$ on input a. If $f^{(j)}(a) = 1$ then there is a short proof which shows this, and C can use this short proof to justify its computation. However, if $f^{(j)}(a) = 0$ we cannot assume that there is a short proof of this fact. In this case, C cannot evaluate $f^{(j)}(a)$ as part of its computation. More generally, the problem is that an SV nondeterministic circuit cannot use a nondeterministic circuit as a black box.

To solve this problem we would like to find a short proof that $f^{(j)}(a) = 0$. We will use the fact that at each prediction step, C runs the predictor $f^{(j)}$ on points a which are on a random curve[4]. With high probability the fraction of a's on the curve for which $f^{(j)}(a) = 1$ and the fraction p of a's in F^d on which $f^{(j)}(a) = 1$ differ by at most some small δ. If C verifies that a $(p - \delta)$ fraction of the a's on the curve have $f^{(j)}(a) = 1$ (which can be done within an SV nondeterministic computation) then it can be sure that almost all remaining a's on the curve have $f^{(j)}(a) = 0$. In particular, if C assumes that *all* remaining a's have $f^{(j)}(a) = 0$ then it agrees with $f^{(j)}$ on at least a $(1 - 2\delta)$ fraction of the points in the curve. Thus we can view C as having access to a predictor which makes slightly more errors than $f^{(j)}$. By choosing the parameters appropriately C can perform the list

[4]This presentation is oversimplified and confuses between binary predictors and q-ary predictors. The actual proof deals with this problem.

decoding phase even with slightly more errors, and the proof goes on essentially unchanged. We also need to provide p to C as non-uniform advice.

In the remainder of the section we formally prove Theorem 4.33. Recall that the "candidate PRGs" $G_x'^{(j)} : F^d \times [\bar{n}] \to \{0,1\}^m$ have been obtained from the $G_x^{(j)}$ defined in (4.10) using an (ρ, ρ^{-2})-efficiently list decodable error-correcting code, with encoding function $E : \{0,1\}^{\log q} \to \{0,1\}^{\bar{n}}$, and that $\rho = \epsilon/(8m)$. (For clarity, we are splitting the seed of the $G_x'^{(j)}$ into two parts – the first being the vector in F^d and the second being the index into the codeword of the error-correcting code). We assume for the purpose of contradiction that no $G_x'^{(j)}$ is an ϵ-nondeterministic PRG. Then without loss of generality, we have next-bit predictors $f^{(j)} : \{0,1\}^{m-1} \to \{0,1\}$ violating property (4.13) for each $G_x'^{(j)}$, respectively. Furthermore, each of these predictor functions can be implemented by a size s nondeterministic or co-nondeterministic circuit.

Using these predictors, we produce a small SV nondeterministic circuit that computes $x(i)$ from input i. Our algorithm here differs from the algorithm used in the proof of Theorem 4.23 *only* by some additional steps at the beginning of **Learn Next Curve**, and a corresponding modification of the proof of Lemma 4.28. For clarity we present the modified version of **Learn Next Curve** in its entirety; the new actions are marked with "+".

Let $p_0^{(j)} = \Pr_{y \in F^d, z \in [\bar{n}]}[f^{(j)}(G_x'^{(j)}(y,z)_{1 \ldots m-1}) = 0]$ be the fraction of points on which the j-th predictor predicts 0, and let $p_1^{(j)} = 1 - p_0^{(j)}$ be the fraction of points on which it predicts 1. The new procedure will make use of values $n_j \in [q\bar{n}]$ and $\beta_j \in \{0,1\}$ for $j = 0, 1, \ldots, d-1$ defined as follows:

$$\beta_j = \begin{cases} 1 & \text{if } f^{(j)} \text{ is computed by a size } s \text{ nondeterministic circuit} \\ 0 & \text{if } f^{(j)} \text{ is computed by a size } s \text{ co-nondeterministic circuit} \end{cases} \tag{4.14}$$

$$n_j = \left\lfloor q\bar{n}\left(p_{\beta_j}^{(j)} - \frac{\epsilon}{4m}\right)\right\rfloor. \tag{4.15}$$

These values will ultimately be supplied as non-uniform advice to the circuit. For all j, let $D^{(j)}$ be the nondeterministic or co-nondeterministic circuit computing $f^{(j)}$.

Procedure Nondeterministic Learn Next Curve

- Input:
    - next curve $C : F \to F^d$: a degree v polynomial
    - reference points $R \subseteq F$: a set of elements of F
    - stride j: an integer in $[0 \ldots (\ell - 1)]$

- input evaluations $\{a_t^i\}_{t\in F, i\in[1\ldots(m-1)]}$ and $\{b_t\}_{t\in R}$: elements of F whose *intended values* are $a_t^i = \hat{x}(A^{-iq^j}C(t))$ and $b_t = \hat{x}(C(t))$.

- Output:
  - output evaluations $\{c_t\}_{t\in F}$: elements of F whose *intended values* are $c_t = \hat{x}(C(t))$.

- Action:
  + Guess a set T of n_j distinct pairs $(t_i, z_i) \in F \times [\bar{n}]$ and a "witness" string w_i for each.
  + Check that this is a "good guess"; i.e., for all

    $$(t_i, z_i) \in T \quad D^{(j)}(E(a_{t_i}^{m-1})_{z_i}, E(a_{t_i}^{m-2})_{z_i}, \ldots E(a_{t_i}^{1})_{z_i}; w_i) = \beta_j.$$

    If it is not, output "bad guess."
  + For all $t \in F$ and all $z \in [\bar{n}]$, set $r_z^t = \begin{cases} \beta_j & \text{if } (t,z) \in T \\ 1-\beta_j & \text{otherwise} \end{cases}$.
  + For all $t \in F$, set S_t to be the list of ρ^{-2} codewords that differ from r^t in at most $(1/2-\rho)\bar{n}$ places.
  - Apply Lemma 4.10 on the $q\rho^{-2}$ pairs $\{(t,e)\}_{t\in F, e\in S_t}$ to obtain a list of at most $8\rho^{-3}$ degree $deg(\hat{x})v$ univariate polynomials $p(t)$ which contains all polynomials for which $p(t) \in S_t$ for at least $\rho q/4$ values of t. If this list is empty, fail.
  - If the list contains a unique polynomial $p(t)$ for which $p(t) = b_t$ for all $t \in R$, output $\{p(t)\}_{t\in F}$; otherwise fail.

We say that **Nondeterministic Learn Next Curve** *succeeds* on an input if its output evaluations are the intended values given that its input evaluations are the intended values *for all "good" guesses, and there is at least one such "good" guess*. As in the proof of Theorem 4.23, we argue that for a random next curve C and a random set of reference points $R \subseteq F$, the procedure succeeds with high probability.

Lemma 4.36. *For all strides j,*

$$\Pr_{C,R}[\textbf{Nondeterministic Learn Next Curve } \textit{succeeds}] \geq 1 - O(2^{-v/2}) - (8\rho^{-3})\left(\frac{vdeg(\hat{x})}{q}\right)^r,$$

where $C : F \to F^d$ is a uniformly chosen degree $v \leq \left(\frac{\epsilon}{4m}\right)^2 (q/4)$ curve, and $R \subseteq F$ is a uniformly chosen subset of F of size r.

Proof. Fix j, and suppose that the input evaluations are the intended values. We first argue that the fraction of points on which $f^{(j)}$ predicts β_j along curve C is close the fraction of points on which $f^{(j)}$ predicts β_j in the whole space. Define the random variable

$$X_t = \Pr_{z \in [\bar{n}]}[f^{(j)}(E(a_t^{m-1})_z, E(a_t^{m-2})_z, \ldots E(a_t^1)_z) = \beta_j],$$

and let $X = \sum_{t \in F} X_t$. Notice that $E[X] = qp_{\beta_j}^{(j)}$. By Lemma 4.11, we have:

$$\Pr[|X - E[X]| \geq \frac{\epsilon}{4m} q] \leq O\left(2^{-v/2}\right). \tag{4.16}$$

This implies that with probability at least $1 - O\left(2^{-v/2}\right)$,

$$n_j \leq \bar{n}q\left(p_{\beta_j}^{(j)} - \frac{\epsilon}{4m}\right) \leq |\{(t, z) : f^{(j)}(E(a_t^{m-1})_z, \ldots E(a_t^1)_z) = \beta_j\}| \leq \bar{n}q\left(p_{\beta_j}^{(j)} + \frac{\epsilon}{4m}\right) \tag{4.17}$$

If the "bad" event in (4.16) does not occur, two important observations hold: (1) there exists at least one "good guess" of T and the witness strings w_i, and (2) for any such "good guess,"

$$\Pr_{t,z}[r_z^t = f^{(j)}(E(a_t^{m-1})_z, \ldots E(a_t^1)_z)] \geq 1 - \frac{\epsilon}{2m},$$

since the set T identifies all but an $\epsilon/(2m)$ fraction of the points on which $f^{(j)}$ predicts β_j.

Now we argue that $f^{(j)}$ is correct along curve C on almost the same fraction of points as the fraction of points in the whole space on which $f^{(j)}$ is correct. Define the random variable

$$Y_t = \Pr_{z \in [\bar{n}]}[f^{(j)}(E(a_t^{m-1})_z, \ldots, E(a_t^1)_z) = E(\hat{x}(C(t)))_z],$$

and define $Y = \sum_{t \in F} Y_t$. Notice that $E[Y] > q(1/2+\epsilon/m)$ since we are assuming that $f^{(j)}$ violates property (4.13). By Lemma 4.11, we have:

$$\Pr[|Y - E[Y]| \geq \frac{\epsilon}{4m} q] \leq O\left(2^{-v/2}\right). \tag{4.18}$$

If neither the event in (4.16) nor the event in (4.18) occurs, then we have:

$$\Pr_{t,z}[r_z^t = E(\hat{x}(C(t)))_z] \geq \frac{1}{2} + \frac{\epsilon}{4m}.$$

By an averaging argument we have that for at least an $\epsilon/(8m)$ fraction of the t's, $\Pr_z[r_z^t = E(\hat{x}(C(t)))_z] \geq 1/2 + \epsilon/(8m)$. For these t, the relative Hamming distance between r^t and $E(\hat{x}(C(t)))$ is at most $1/2 - \rho$, so S_t contains $\hat{x}(C(t))$. The remainder of the proof of Lemma 4.28 now goes through unchanged. $\square$

The remainder of the proof of Theorem 4.23 (following Lemma 4.28) goes through unchanged. Lemma 4.29 now shows that there exist curves C_1 and C_2 for which **Nondeterministic Learn Next Curve** succeeds on all steps. In the present context, this also means that on these curves **Nondeterministic Learn Next Curve** is an SV-nondeterministic "subroutine" (on "bad" curves there may be no "good guess,". Using this procedure repeatedly to compute $x(i)$ results in an SV-nondeterministic circuit computing x of size $s \cdot \text{poly}(m, q, \bar{n})$, a contradiction. This concludes the proof of Theorem 4.33.

Appendix A

Converting extractors into strong extractors

A.1 Introduction

It is sometimes helpful to have a stronger variant of extractors, called a *strong* extractor. A strong extractor is required to extract only randomness from the source.

Definition A.1. *A* (k,ϵ)*-extractor* $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *is* strong *if for every* k*-source* X*, the distribution* $(Ext(X,U_d) \circ U_d)$ *(obtained by concatenating the seed to the output of the extractor) is* ϵ*-close to a* U_{m+d}.

Intuitively, this is helpful since a strong extractor has the property that for any source a $1-\epsilon$ fraction of the seeds extract randomness from that source. It is interesting to note that the concept of strong-extractors preceded that of non-strong extractors, and the strong version was the one which was defined in the seminal paper of [NZ96]. Several extractors constructions (with examples being [TS96, ISW00] and the constructions of this paper) are non-strong or difficult to analyze.

The following theorem shows that every non-strong extractor can be transformed into a strong one with essentially the same parameters.

Theorem A.2. *Any explicit* (k,ϵ)*-extractor* $Ext : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ *can be transformed into an explicit strong* $(k, O(\sqrt{\epsilon}))$*-extractor* $Ext' : \{0,1\}^n \times \{0,1\}^{d+d'} \to \{0,1\}^{m-d-\Delta'-1}$ *for* $d' = polylog(d/\epsilon)$ *and* $\Delta' = 2\log(1/\epsilon) + O(1)$.

The output length of Ext' is shorter than that of Ext. This is unavoidable as the output of Ext may contain d bits of randomness from the seed. The additional loss of $\Delta' + 1$ can be recovered by enlarging the seed length. Recovering this loss is most significant when the initial extractor Ext has a small entropy loss[1]. For this case the loss of our construction can be recovered with essentially the same parameters. Exact details are given in the next remark.

Remark A.3. *In [RRV99b] it was shown that any strong extractor which has seed length* d *and entropy-loss* Δ *can be transformed into an extractor with seed length* $d + O(\Delta)$ *and an optimal entropy loss of* $2\log(1/\epsilon) + O(1)$. *Thus, if the initial extractor* Ext *had an entropy loss of* Δ*, we can use [RRV99b] to construct a strong extractor* Ext'' *with seed length* $d'' = d + d'O(\Delta)$ *and optimal entropy loss. This addition is affordable if* Δ *is small.*

A.2 Construction and proof

The intuition above also gives a hint for the construction. The output of Ext may contain d bits of randomness which "belong" to the seed. Still, it contains roughly $m - d$ bits which *do not* depend on the seed. Thus, fixing the seed, the output of Ext is a random source of length m which contains roughly $m - d$ random bits. We can now use another extractor to extract this randomness and "dismantle" the correlation between the seed and output. The extractor we need is one that works well when the source lacks only a very small amount of randomness. Such a construction was given by [GW97] and improved by [RVW00].

Theorem A.4. *[RVW00] There are explicit strong* (k, ϵ)*-extractors* $RVW : \{0,1\}^n \times \{0,1\}^{d'} \to \{0,1\}^{k-\Delta'}$ *For* $d' = polylog((n-k)/\epsilon)$ *and* $\Delta' = 2\log(1/\epsilon)$.

We use this extractor to construct Ext' as follows: Let $RVW : \{0,1\}^m \times \{0,1\}^{d'=\text{polylog}(d/\epsilon)} \to \{0,1\}^{m-d-1-\Delta'}$ be an $(m-d-1, \epsilon)$-extractor guaranteed by theorem A.4.

$$Ext'(x; (y, z)) = RVW(Ext(x, y), z)$$

The actual proof that Ext' has the desired properties is slightly more complicated than the above presentation. This is mainly because the above presentation ignores the error of Ext.

Proof. (of theorem A.2) Fix a k-source X on $\{0,1\}^n$. For a source element x and a seed $y \in \{0,1\}^d$ we define the *weight* of (x, y), denoted $w(x, y)$ in the following way.

$$w(x, y) = \Pr_{X,Y}(Ext(X, Y) = Ext(x, y))$$

[1]Entropy loss is defined to be $k + d - m$ in strong extractors and $k - m$ in regular extractors. In [RTS97] it was shown that extractors must suffer an entropy loss of $2\log(1/\epsilon) + O(1)$.

That is the weight of the string which is obtained by the extractor when running on x and y. The target distribution of an extractor is ϵ-close to uniform, therefore the fraction of "heavy" elements is small.

Claim 1. $\Pr_{X,Y}(w(X,Y) > 2^{-(m-1)}) < 2\epsilon$.

Proof. Consider the set $A = \{Ext(x,y) | \, w(x,y) > 2^{-(m-1)}\}$. Let us denote $\Pr_{X,Y}(A)$ by $P_{X,Y}$ and $\Pr_{U_m}(A)$ by P_U. From the definition of extractors we have that $|P_{X,Y} - P_U| \le \epsilon$. From the definition of A we have that $P_{X,Y} - P_U > 2P_U$. The claim follows. □

Call a seed $y \in \{0,1\}^d$ *bad* if $\Pr_X(w(X,y) > 2^{-(m-1)}) > \sqrt{2\epsilon}$. The following claim follows immediately from Markov's inequality.

Claim 2. *The fraction of bad seeds is at most* $\sqrt{2\epsilon}$.

The following claim shows that running the extractor with a good seed produces a source which lacks very few random bits.

Claim 3. *For a good seed* y*,* $Ext(X,y)$ *is* $\sqrt{2\epsilon}$*-close to an* $(m-d-1)$*-source.*

Proof. For a good y we know that at least a $1 - \sqrt{2\epsilon}$ fraction of the x's are "light" (meaning that $w(x,y) \le 2^{-(m-1)}$). For a light x,

$$\Pr_X(Ext(X,y) = Ext(x,y)) = \Pr_{X,Y}(Ext(X,Y) = Ext(x,y)|Y = y)$$

$$\le \frac{w(x,y)}{2^{-d}} \le 2^{-(m-d-1)}$$

□

On any fixed y, Ext' runs RVW on the source $Ext(X,y)$ using a fresh seed z. Thus, for a good y, Ext' extracts $m - d - \Delta' - 1$ bits. The following claim follows immediately from claim 3 and the fact that RVW is a strong extractor.

Claim 4. *For a good seed* y*,* $(Ext'(X;(y,Z)) \circ Z)$ *is* $(2\sqrt{\epsilon} + \epsilon)$*-close to uniform.*

To complete the proof we will need the following standard facts:

Fact A.5. *If* P *and* Q *are* ϵ*-close then for every function* f*,* $f(P)$ *and* $f(Q)$ *are* ϵ*-close.*

Fact A.6. *Let* X, Q *be distributions over domains* $\bar{X}, \bar{Q}$ *respectively. Suppose that for every* $x \in \bar{X}$ *there is a distribution* P_x *over the domain* $\bar{Q}$ *such that* P_x *and* Q *are* ϵ*-close. Then the distributions* (P_X, X) *and* (Q, X) *are* ϵ*-close.*

Consider the uniform distribution over good seeds which we denote by Y'. By claim 4 and fact A.6 we get that $(Ext'(X; (Y', Z)) \circ Z) \circ Y'$ is $2\sqrt{\epsilon} + \epsilon$-close to $(U_{m-d-\Delta-1} \circ U_{d'}) \circ Y'$. Claim 2 gives that Y (that is the uniform distribution over d bit strings) and Y' are $2\sqrt{\epsilon}$-close. We can now use fact A.5 to replace Y' by Y and get that $(Ext'(X; (Y, Z)) \circ (Y, Z))$ is $O(\sqrt{\epsilon})$-close to uniform. $\square$

Appendix B

A tournament of generators

B.1 Introduction

Our efforts to construct pseudo-random generators in chapters 2,4 lead to constructing *many* ”candidate generators” $G_1, \cdots, G_l$ where at least one of them is guaranteed to be pseudo-random. It is easy to construct a hitting set generator given such a collection of candidates. By defining $H(y, z) = G_z(y)$. This transformation increases the seed length by a $\log l$ factor which is affordable. There’s also a natural way to construct a *single* pseudo-random generator by defining $G(y_1, \cdots, y_l) = G_1(y_1) \oplus \cdots \oplus G_l(y_l)$. Unfortunately, this transformation blows up the seed length by a factor of l. We were able to bypass this problem in chapter 2 by using specific properties of the construction to combine the generators. In chapter 4 we had to pay the penalty of the ”xoring” method which led ro a difference in seed length between the hitting set generators and pseudo-random generators constructed there.

A way around this problem is given in [ACR96] (see also [ACRT97, BF99, GVW00]). In that paper it is shown how to estimate the fraction of inputs accepted by a given circuit using a hitting set generator. Thus, since solving this approximation problem suffices to derandomize two-sided error algorithms[1], hitting set generators suffice to derandomize BPP.

In this chapter we give a simple algorithm which solves the above approximation problem given a collection of candidate generators. This enables using our hitting set constructions to derandomize BPP using a far simpler construction and proof than that of [ACR96] and the works that followed it.

[1]See the argument in lemma 1.7.

B.2 Construction and proof

We prove the following theorem:

Theorem B.1. *There exists an algorithm for the following computational problem:*
Input:

- *A circuit C of size m.*
- *A collection of functions $G_1, .., G_l$, where $G_i : \{0,1\}^{d_i} \to \{0,1\}^m$ and $d_i \le d$ for all i. Moreover, at least one of the G_i's is a ϵ-pseudo-random generator for size m.*

Output: *A number α, such that*

$$|\Pr_x[C(x) = 1] - \alpha| \le 2\epsilon$$

The algorithm solves the problem. and runs in time polynomial in $m, l, 2^d$.

Proof. For $y \in \{0,1\}^m$, define $C_y(w) = C(w \oplus y)$. Note that for all $y \in \{0,1\}^m$, C_y is of the same size as C. For $1 \le i \le l$, define:

$$\alpha_i(y) = \Pr_{z \in U_{d_i}\{0,1\}^{d_i}}[C_y(G_i(z))) = 1]$$

For $i, j \in [l]$ define:

$$\alpha_{ij} = \mathrm{E}_{z \in U_{d_j}\{0,1\}^{d_j}}(\alpha_i(G_j(z)))$$

Let k be an index such that G_k is a ϵ-pseudo-random generator for size m. For all $y \in \{0,1\}^m$: $\Pr_x[C(x) = 1] = \Pr_x[C_y(x) = 1]$ and $|\alpha_k(y) - \Pr_x[C_y(x) = 1]| \le \epsilon$. From this we have that for all $1 \le j \le l$:

$$|\alpha_{kj} - \Pr_x[C(x) = 1]| \le \epsilon$$

Note that for all $i, j \in [l]$, $\alpha_{ij} = \alpha_{ji}$. This is because both amount to taking all pairs a_1, a_2 from the outputs of G_i, G_j and running $C(a_1 \oplus a_2)$. The algorithm computes α_{ij} for all $i, j \in [l]$. and picks a row r such that all the numbers in $I = \{\alpha_{rj} | j \in [l]\}$ lie on an interval of length 2ϵ. It then returns α, the middle of the interval of I. Such an r exists, because k has that property. For all i, we have that $|\alpha_{ik} - \Pr_x[C(x) = 1]| \le \epsilon$, and therefore all the numbers in I, are at a distance of at most 3ϵ from $\Pr_x[C(x) = 1]$. From this we have that $|\alpha - \Pr_x[C(x) = 1]| \le 2\epsilon$. ∎

Bibliography

[ACR96] A. Andreev, A. Clementi, and J. Rolim. Hitting sets derandomize BPP. In *ICALP*, 1996. Final version in J. ACM.

[ACRT97] Alexander E. Andreev, Andrea E. F. Clementi, José D. P. Rolim, and Luca Trevisan. Weak random sources, hitting sets, and BPP simulations. In *38th Annual Symposium on Foundations of Computer Science*, pages 264–272, Miami Beach, Florida, 20–22 October 1997. IEEE.

[Ade78] L. Adelman. Two theorems on random polynomial time. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, 1978.

[AK97] V. Arvind and Johannes Kobler. On resource-bounded measure and pseudorandomness. *17th International Conference on Foundations of Software Technology and Theoretival Computer Science*, 1997.

[Bab85] Laszlo Babai. Trading group theory for randomness. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, 1985.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *Theoretical aspects of computer science, 16th annual symposium*, 1999.

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

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

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

[BM88] Laszlo Babai and S. Moran. Arthur-merlin games: a randomized proof system and a hierarchy of complexity clasees. *JCSS: Journal of Computer and System Sciences*, 36, 1988.

[BO83] M. Ben-Or. Another advantage of free choice: Completely asynchronous byzantine agreement. In *2nd ACM symposium on Primciples of Distributed Computing*, 1983.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *35th Annual Symposium on Foundations of Computer Science*, 1994.

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

[CNS99] J.Y. Cai, A. Nerurkar, and D. Sivakumar. Hardness and hierarchy theorems for probabilistic quasi-polynomial time. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 726–735, 1999.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.

[GMR82] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, 1982.

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

[GS00] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[GVW00] Oded Goldreich, Salil Vadhan, and Avi Wigderson. Simplified derandomization of BPP using a hitting set generator. Technical Report TR00-004, Electronic Colloquium on Computational Complexity, January 2000.

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

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

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

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

[Imp99] Russell Impagliazzo. private communication, 1999.

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

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

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

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

[IW98] Russell Impagliazzo and Avi Wigderson. Randomness vs. time: Derandomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science*. IEEE, 1998.

[Jus72] J. Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Trans. Info. Theory*, 18:652–656, 1972.

[Kab00] Valentine Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. In *Fifteenth Annual IEEE Conference on Computational Complexity*, pages 150–157, 2000.

[KvM99] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, 1999.

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

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

[MV99] Peter Bro Miltersen and N. V. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[Nis91] N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, 11(1):63–70, 1991.

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

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

[NN90] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 213–223, Baltimore, Maryland, 14–16 May 1990.

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

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

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

[Pap94] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley publishing company, 1994.

[Rab83] M.O. Rabin. Randomized byzantine agreement. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, 1983.

[RR97] A. R. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Science*, 55(1):24–35, 1997.

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

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

[RRV99b] R. Raz, O. Reingold, and S. Vadhan. Extracting all the randomness and reducing the error in trevisan's extractors. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 149–158, 1999.

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

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

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

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

[Sho92] V. Shoup. Searching for primitive roots in finite fields. *Mathematics of Computation*, 58:369–380, 1992.

[Shp96] Igor Shparlinski. On finding primitive roots in finite fields. *Theoretical Computer Science*, 157:273–275, 1996.

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

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

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

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

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

[SW86] M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, 1986.

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

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

[TS96] A. Ta-Shma. On extracting randomness from weak random sources. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 276–285, 1996.

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

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

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

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

[Uma99] C. Umans. Hardness of approximating Σ_2^p minimization problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 465–474, 1999.

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

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

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

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

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

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

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