Chapter 1

DERANDOMIZING COMPLEXITY CLASSES

Peter Bro Miltersen
BRICS
University of Aarhus
bromille@brics.dk

1. INTRODUCTION

Consider a randomized algorithm, such as the *Rabin primality test* of Rabin, 1976. This algorithm is given as input an integer n, and, as auxiliary input, a sequence of *coin tosses*, i.e., a vector of unbiased, independent random bits. The test gives as output either "n is a prime" or "n is not a prime". For every input n, the probability that the output is a false statement is bounded from above by a small constant, say, $\frac{1}{4}$. The test runs in time polynomial in the size of the binary representation of n. In particular, the number r of random bits needed is polynomial in $\log n$.

Randomized algorithms such as the Rabin primality test are immensely useful, also in practice. An obstacle for using them is that they are not very suited for implementation on conventional hardware because of the auxiliary input, a sequence of unbiased, independent random bits. If the reader has never before considered the task of obtaining such a sequence, or even just a *single* unbiased random bit, he is invited to spend a few minutes pondering about possible ways of performing this task. It is certainly a non-trivial task, an expensive one (compared to other atomic operations, such as taking the AND of two bits), and, in many settings where no special hardware is available, arguably even an impossible one. The fact of life we face is that *independent random bits are scarce*. Thus, we want to use as few as possible. In this chapter, we describe theoretical techniques for saving on independent random bits. In particular, we ask the following three related questions.

1. It is well known that if we are unhappy about the error probability of a randomized algorithm, we can decrease it at an exponential

rate by running the algorithm a number of times. Running it k times increases the random bits used by a factor of k. Can we save on the number of random bits used to achieve a certain error probability?

2. Independent random unbiased bits are hard to obtain, but we may be able to get our hands on a "messier" source of randomness, such as a bit sequence $x \in \{0,1\}^r$ sampled from a source we don't quite know how to model, but which presumably "contains some randomness". Can we simulate our algorithm with such a bit sequence instead of unbiased independent bits?

3. Can we replace the randomized algorithm with a deterministic one, thus avoiding using randomness altogether?

The purpose of this chapter is to give an account of the state-of-the-art answers to these three questions, as of around the turn of the millennium. Before making the questions more precise, there are three general qualifications we should make.

First, we concentrate on randomized algorithms solving *decision problems*, with a small probability of error on every instance, such as the Rabin primality test. The important property is that there is a single well defined answer we are trying to obtain and the probability that we will *not* obtain it is small[1]. There are other ways in which an algorithm may use randomness and we won't be considering derandomizing those. For instance, we may want an algorithm to actually produce an output distributed according to some distribution, i.e., we may want the output to *contain* some randomness. Clearly, such an algorithm could never be completely derandomized[2]. As another example, there are randomized algorithms which generate, with high probability, a particular kind of combinatorial object. For instance, a randomized algorithm for generating, with very high probability, the truth table of a Boolean function $\{0,1\}^n \to \{0,1\}$ with circuit complexity $(1-o(1))\frac{2^n}{n}$ (i.e., almost maximum possible) is by picking each of the 2^n entries in the truth table independently at random. Clearly, we'll get a different truth table every time we run the algorithm, so we shall not attempt to derandomize this algorithm either (but the example is somewhat relevant to our techniques after all, as we shall see).

Second, we shall care much more about the number of random bits used than about other computational resources, such as time. Of course, if we do not care about time at all, everything is trivialised. Indeed, we can deterministically simulate a randomized algorithm using r random bits by running it on all possible 2^r bit sequences. Thus, an exponential slowdown is uninteresting, but we shall routinely accept any polynomial slowdown of our algorithm to save, say, a constant factor on the random bits used to achieve a certain error probability. We shall even consider a slowdown which is only just non-trivial, i.e., slightly subexponential, as interesting (though of course less interesting than a polynomial slowdown). Thus, we take a very fine-grained point of view when it comes to randomness, but a very coarse-grained point of view when it comes to time. Though this point of view of course has a bearing of the practicality of the results obtained (an issue we do not address in this chapter), it seems the right point of view from a scientific perspective: To get a basic understanding of the power of one resource, in this case randomness, we have to consider other resources as of secondary importance, to avoid cluttering up the big picture. After a basic understanding has been obtained, a more complete picture of the finer trade-offs between time and randomness can be studied (and such a picture is indeed emerging in current literature).

Third, we shall concentrate on *generic*, general-purpose derandomization, i.e., techniques that will work on *any* randomized algorithm, *without any knowledge necessary about how the algorithm operates*, except, possibly, some bound on the computational complexity of the algorithm, such as its time or space bound (hence, the title of the chapter). For example, we should be able to give good answers to the three questions above for, say, Rabin's primality algorithm, based *only* on the account of the algorithm given above. An alternative and very important strategy for derandomization which we will not cover, is to use all we know about the algorithm and the analysis leading to our estimate on its probability of error in order to answer the three questions above, for the particular algorithm in question. For the specific case of Rabin's primality test, there are reasonable (but unproven) number theoretic assumptions that allow a complete derandomization. The more general tools available for special-purpose derandomization may or may not work for a given randomized algorithm and usually require non-trivial adaptation to each particular case. Important cases of such special-purpose techniques are the techniques of *conditional probabilities* and *small sample spaces*[3]. We refer to Motwani and Raghavan, 1995, for an exposition of these techniques.

With these points in mind, let us define the three basic tasks a bit more precisely and state which results we shall cover about them in this chapter.

Randomness-efficient amplification. Suppose we have a randomized polynomial algorithm using r bits to obtain an error probability of

$\frac{1}{10}$. Suppose we want a smaller error probability than this. By repeating the algorithm k times and taking the majority of the answers, we can reduce the error probability to $2^{-O(k)}$ using rk random bits. We can also think of it as *amplifying* the success probability to $1 - 2^{-O(k)}$. If k is polynomial, the resulting algorithm is a polynomial algorithm. However, if random bits are very expensive, using a factor of k more may be unacceptable. The task of randomness-efficient amplification is to *obtain the same probability of error using fewer random bits*, while still preserving the efficiency.

It is not at all obvious that this task is possible at all. However, we shall see that it is, using no unproven assumptions. In particular we shall discuss the following simulations, all slowing down the randomized algorithm at most polynomially:

1. We can obtain an error probability of $\frac{1}{r^k}$ for any desired constant k using r bits, i.e. using *no* additional randomness.

2. We can obtain an error probability of 2^{-R} for any desired R, using (a) $r + O(R)$ random bits, and even (b) $(1+\epsilon)(r+R)$ random bits, for any desired constant $\epsilon > 0$.

3. Given any constant $\epsilon > 0$, we can, using R random bits, where R is a certain polynomial in r, obtain an error probability of 2^{-R+R^ϵ}.

4. If the algorithm has one-sided error (i.e., has error probability 0 on negative instances), we can, using R random bits for any parameter $R \geq r$, obtain an error probability of $2^{-R+r+\text{polylog } R}$.

Note that the flavor of the last two results is that *each* random bit used reduces the error probability of the algorithm by a factor of $2-o(1)$. Since the error probability of an algorithm using R random bits must be an integer multiplum of 2^{-R}, one cannot do much better without derandomizing the algorithm completely.

Computing with weak random sources. Again, suppose we have a randomized algorithm using r random bits to achieve an error probability of $\frac{1}{10}$. Suppose that we are happy with the error probability, but at a loss on how to supply the r unbiased, independent coin tosses needed. How do we construct hardware to flip coins? Many physical systems can reasonably be assumed to have a behavior exhibiting some randomness. A radioactive source is an obvious example. We can consider making a measurement of such a system, encoding the measurement as a bit string of length r, and using this bit string as the sequence of coin tosses to our algorithm. However, it is not very likely that we can easily find and build into our hardware a physical system where such a direct approach would give us independent, unbiased coin flips. It is far more likely that we would not able to state the exact distribution of the output at all. A source that is assumed to contain some randomness but with an unknown distribution is called a *weak random source*. Our objective is to simulate our randomized algorithm *using the weak random source instead of independent, unbiased coin flips*, while preserving (roughly) the efficiency and (precisely) the error probability of our algorithm.

The above task needs to be formalized before we can solve it. Obviously, we need some measure on the amount of randomness contained in a source, to be able to prove a theorem stating that any source containing a certain amount of randomness can be used to simulate our algorithm.

One classical measure for the amount of randomness contained in a source is the *Shannon entropy* of the source. Thus, it would be nice to prove a theorem of the form: Any source with a Shannon entropy above a certain threshold can be used to simulate our r-bit algorithm. However, we cannot obtain such a theorem as a source can have high Shannon entropy and yet only exhibit randomness with low probability. For instance, consider the following source. Let R be a value much bigger than r. With probability $\frac{99}{100}$, the source produces the value $(0, 0, \dots, 0) \in \{0, 1\}^R$. With probability $\frac{1}{100}$, the source produces a uniformly chosen value in $\{0, 1\}^R$. Then, the source has Shannon entropy more than $\frac{R}{100}$. Thus, by letting R grow, we can make a source with Shannon entropy exceeding r as much as we want. However, it does not seem likely that we can use it to simulate our randomized algorithm with success probability $\frac{9}{10}$, as we only get something containing randomness at all with probability $\frac{1}{100}$! Indeed, if we were able to do so, we would also be able to derandomize our algorithm completely.

A different measure for the amount of randomness contained in a source turns out to be much more useful. We say that a source has *min-entropy* at least t if any outcome of the source has probability at most 2^{-t}. Thus, a high min-entropy source is a source where no particular outcome is very likely to occur. It is easy to see that the min-entropy of a source is smaller than its Shannon entropy, i.e, a high min-entropy source is a stricter notion than a high Shannon entropy source. In particular, the problem described above is avoided, and it turns out that we *can* simulate our randomized algorithm using any such source.

5. For any constant $\epsilon > 0$ and any randomized polynomial algorithm A using r bits to obtain an error probability of $\frac{1}{3}$, there is a randomized polynomial algorithm A' which, instead of r random bits,

takes as auxiliary input $y \in \{0,1\}^R$, where R is polynomial in r. If y is sampled from any R-bit source with min-entropy at least R^ϵ, the error probability of the original algorithm is maintained.

Deterministic simulation. Finally, consider the task of removing the randomness requirement of our randomized algorithm all together, simulating it with an efficient deterministic one. Two (oddly contradictory) points make this task somewhat different from the two previously mentioned ones. First, *in the most general setting we will only be able to do it under an unproven assumption.* Indeed, it is at the time of writing an open problem if every problem solvable in exponential time can be solved by an efficient randomized algorithm. If this is the case, no general subexponential derandomization is possible, as this would imply that deterministic exponential time equals deterministic subexponential time, contradicting the time hierarchy theorem. Second, *deterministic simulation is being done all the time in the real world and is part of routine software engineering.* Indeed, lacking a source of truly random bits, most implementations of randomized algorithms currently running on conventional computers are replacing this source with a deterministically generated sequence of bits in some ad hoc way, e.g., with the output of the "pseudorandom generator" built into the C language. Of course, doing this without further analysis changes a randomized algorithm with a well analyzed error probability, such as the Rabin primality test, into a heuristic which may, conceivably, fail completely and give incorrect answers on all inputs.

What we shall see is that it *is* possible to construct pseudorandom generators (different from the one built into the C language) which makes a variation of the above scheme a correct simulation, under a reasonable complexity theoretic assumption.

In particular, we shall see that a sufficient condition is that "truth tables of hard Boolean functions can be generated efficiently deterministically" and prove the following theorems.

6. *If* there is an algorithm which on input n runs in time $2^{n^{O(1)}}$ and generates the truth table of a Boolean function $g : \{0,1\}^n \to \{0,1\}$ with circuit complexity at least $s(n)$, for some superpolynomial function s, *then* any decision problem that can be solved by a randomized polynomial time algorithm can also be solved by a deterministic one running in subexponential time.

7. *If* there is a constant $\epsilon > 0$ and an algorithm which on input n runs in time $2^{O(n)}$ and generates the truth table of a Boolean function $g : \{0,1\}^n \to \{0,1\}$ with circuit complexity at least $2^{\epsilon n}$

then any decision problem that can be solved by a randomized polynomial time algorithm can also be solved by a deterministic one in polynomial time.

While the two statements above are conditional, the conditions seem quite likely to be true, especially the condition of Statement 6. Note that since we have time more than $2^{O(n)}$ to generate the truth table for $g : \{0,1\}^n \to \{0,1\}$, we can, say, let g be the truth table of an **NP**-complete problem such as SATISFIABILITY, i.e., let $g(x) = 1$ if and only if x is the encoding of a Boolean formula with a satisfying assignment. While it has not been proved that g such defined does not have circuits of size $2^{o(\frac{n}{\log n})}$ (this would imply **P**$\neq$**NP**), no such circuits are known.

If we move to classes of algorithms defined by a space constraint, rather than a time constraint, some unconditional non-trivial deterministic simulation results are possible (the space measure is the standard one of complexity theory, i.e., we do not count the space taken up by the input in the space bound of the machine).

8. A decision problem solved by a randomized algorithm using space $O(\log n)$ and $O(\text{polylog } n)$ random bits on inputs of length n can be solved by a deterministic algorithm running in space $O(\log n)$.

9. A decision problem solved by a randomized algorithm using space $O(\log n)$ can be solved by a deterministic algorithm running in space $O((\log n)^2)$ and in polynomial time.

10. A decision problem solved by a randomized algorithm using space $O(\log n)$ can be solved by a deterministic algorithm running in space $O((\log n)^{\frac{3}{2}})$.

1.1 OVERVIEW OF CHAPTER

The results 1-10 stated above are landmark results in the theory of general derandomization. The main purpose of this chapter is to give almost (but not quite) self-contained proofs of many (but not all) of them, concentrating on those results that have not been covered in previous surveys and texts, such as Alon and Spencer, 1992, Motwani and Raghavan, 1995, Saks, 1996, Clementi et al., 1998, Goldreich, 1999, and Nisan and Ta-Shma, 1999. Though all of the results have an associated single paper where the result first appeared, many are the culmination of years of work appearing in many papers. The proof we want to give for each result is not always the original proof, nor is our sequence of proofs the same as the historical sequence, as many important connections were only realised recently.

In Section 2 (*Preliminaries*), we discuss three preliminary issues before starting the main story. The first of these is basic computational complexity theory, especially that concerned with randomized complexity classes. There is no hope of giving a comprehensive overview here, so we only explain what "extra" basics we need, apart from standard texts, such as Papadimitriou, 1994. The second is universal hashing and the construction of small and efficient families of universal hash functions. The third is the theory of error correcting codes. Both universal hashing and error correcting codes play crucial parts in many places of our main story and are thus well worth singling out.

A main key to solving all the three derandomization tasks turns out to be the concepts of *dispersers* and *extractors*. In Section 3 (*Dispersers and extractors*) we define these concepts and show how such objects solve the tasks of randomness-efficient amplification and computing using weak random sources. In Section 4 (*Constructions of dispersers and extractors*), we then show how to build these objects, leading to Statements 1-5 stated above. Here, we also introduce *Nisan-Wigderson designs*, an important tool in many of the proofs in the chapter.

The rest of the chapter is concerned with deterministic simulation. In Section 5 (*Hitting set and pseudorandom generators*), we introduce these two concepts that are the keys to all known deterministic simulation results. The most important kind of hitting set and pseudorandom generators, *hardness based generators*, will be shown to be special cases of dispersers and extractors. In Section 6, (*A low end generator*) and Section 7 (*A high end generator*) we prove Statement 6 and 7. As important lemmas, we prove in Section 6 *Yao's XOR lemma* and in Section 7 *Impagliazzo's hardcore lemma*. In Section 8 (*The compression approach to deterministic simulation*), we present a different approach to deterministic simulation that can be used to obtain similar results somewhat more easily and in Section 9 (*Other hardness based deterministic simulation results*), we survey which other results that can be proved along the lines of the proofs of Section 6, 7 and 8. Finally, in Section 10 (*Unconditional derandomization of space-bounded computation*) we present the unconditional deterministic simulation results for randomized space bounded computation, proving Statements 8-10. Again, extractors turn out to be key components.

We use standard mathematical notation in the theory of computation tradition. For instance, log will denote $\log_2$, the size of a finite set A will be denoted $|A|$ and the density of a set $A \subset \{0,1\}^n$, i.e., $|A|/2^n$ will be denoted $\mu(A)$. Though many theorems will be given fairly detailed proofs, we shall, in an attempt to improve readability, generally ignore round-off errors, i.e., sometimes tacitly assume expressions such as $\log n$

to be integers, even though we cannot strictly speaking assume that n is a power of two, without loss of generality. Fixing the proofs to hold in general is usually straightforward, amounting to adding $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ as appropriately, but sometimes requires slightly more work.

1.2 NOTES

The first paper to point out that randomness-efficient amplification is possible was Karp et al., 1986. The notion of computing with weak random sources are due to Santha and Vazirani, 1986. The observation that the min-entropy of a source is the right measure to consider is due to Zuckerman, 1990. The heuristic idea of using pseudorandom generators to simulate randomized algorithm has a long history. See Knuth, 1981, for an excellent survey of this. Blum and Micali, 1984, and Yao, 1982, were first to find ways of putting pseudorandom generators on more rigorous ground.

The results labelled 1 to 10 above first appeared in Karp et al., 1986 (result 1 for one-sided error algorithms), Cohen and Wigderson, 1989 (result 1 for two-sided error algorithms), Impagliazzo and Zuckerman, 1989, and Cohen and Wigderson, 1989 (result 2a), Zuckerman, 1996 (result 2b), Saks et al., 1998 (results 3 and 5 for one-sided error algorithms), Andreev et al., 1999 (results 3 and 5), Ta-Shma, 1998 (result 4), Babai et al., 1991a (result 6), Impagliazzo and Wigderson, 1997 (result 7), Nisan and Zuckerman, 1996 (result 8), Nisan, 1994 (result 9), and Saks and Zhou, 1999 (result 10). As the papers cited here are journal papers (where available), the above dates do not necessarily say much about the dates of the results. However, all the results are from the period 1986-1998, though some of the proofs presented in this chapter are more recent.

2. PRELIMINARIES

2.1 COMPLEXITY THEORY

We assume that the reader is familiar with standard notation for complexity classes (a good textbook is Papadimitriou, 1994).

In particular, $\mathbf{TIME}(t(n))$ where $t(n)$ is a proper time bound, is the class of languages decided by a deterministic Turing machine running in time $O(t(n))$ for inputs of length n, $\mathbf{P}$ is $\cup_{k\geq 1}\mathbf{TIME}(n^k)$, $\mathbf{E}$ is $\cup_{k\geq 1}$ $\mathbf{TIME}(2^{kn})$ and $\mathbf{EXP}$ is $\cup_{k\geq 1}\mathbf{TIME}$ (2^{n^k}), while $\mathbf{SUBEXP}$ is $\cap_{\epsilon>0}\mathbf{TIME}(2^{n^\epsilon})$. Also, $\mathbf{SPACE}(s(n))$ where $s(n)$ is a proper space bound, is the class of languages decided by a deterministic Turing machine using space $O(s(n))$ for inputs of length n, **L** is **SPACE**$(\log n)$, **LINSPACE** is **SPACE**(n) and **PSPACE** is **SPACE**$(n^{O(1)})$.

Apart from the uniform Turing machine model, we shall also consider the non-uniform circuit model. Unlike a Turing machine, a circuit only decides a finite language $L \subseteq \{0,1\}^n$ for a particular n. We say that a circuit C with n Boolean inputs and one output decides $L \subseteq \{0,1\}^n$ if $C(x) = 1$ if and only if $x \in L$. Turing machines can be simulated by circuits with a polynomial overhead. In particular, if a language L is in **P**, the circuit complexity of $L \cap \{0,1\}^n$ is polynomial in n. The class of languages L for which the circuit complexity of $L \cap \{0,1\}^n$ is polynomial in n is called **P/poly**. For all languages L, the circuit complexity of $L \cap \{0,1\}^n$ is at most exponential. More precisely, if we by $M_{n,m}$ denote the biggest possible circuit complexity of functions from $\{0,1\}^n$ to $\{0,1\}^m$, we have the Shannon-Lupanov bound of Shannon, 1949; Lupanov, 1965,

Theorem 1 $M_{n,m} = (1 + o(1))\frac{2^n m}{n+\log m}$.

A family of circuits C_n of size at most $s(n)$ is said to be a *uniform* family of size $s(n)$ if there is a Turing machine operating in space $O(\log s(n))$ which on input n outputs a description of C_n.

A *nondeterministic* Boolean circuit C contains, in addition to AND, OR and NOT gates, *choice*-gates of fan-in 0. The circuit evaluates to 1 on an input x, and we say that $C(x) = 1$, if there is *some* assignment of truth values to the choice-gates that makes the circuit evaluate to 1.

All complexity classes defined by a time bound or circuit size bound have *relativized* versions, where we assume "free" access to an oracle deciding a language L. In particular, **TIME**$[L](t(n))$ is the class of languages decided in time $t(n)$ with an oracle for L and **P**$[L]$, **EXP**$[L]$, etc. are defined accordingly. If **C** is a complexity class, **TIME**$[\mathbf{C}](t(n))$ is $\cup_{L\in \mathbf{C}}$ **TIME**$[L](t(n))$ and **P**$[\mathbf{C}]$, **EXP**$[\mathbf{C}]$, etc. are defined accordingly. A relativized circuit with oracle L (an L-circuit) contains L-gates deciding membership in L. When we consider the size of L-circuits, an L-gate with m inputs is charged size $m - 1$.

A complexity theoretic theorem and/or proof is said to *relativize* if it holds for relativized complexity classes with any fixed oracle L. Most complexity theoretic theorems and proofs do relativize but there are exceptions.

The complexity classes relevant for understanding time-efficient randomized computation are **RP** and **BPP**. We present the basic facts about these classes in somewhat more detail.

A language L is in **RP** if there is a randomized polynomial time Turing machine M so that for all $x \in L, M$ accepts with probability at least $\frac{1}{2}$ and for all $x \notin L$, M accepts with probability 0. Such a randomized Turing machine is said to have *one-sided error*. An example of a language in **RP** not known to be in **P** is NON-PRIMALITY, the language of integers which are not primes. Rabin's primality test is an **RP**-machine for this language. If, actually, for all $x \in L, M$ accepts x with probability at least $1-\epsilon$, we say that M has error probability at most ϵ. If we have an **RP**-machine M for a language L with error probability at most ϵ, we can get an **RP**-machine for a language L with error probability at most ϵ^k, by doing k independent runs of M and accepting if at least one of the runs accepts.

A language L is in **BPP** if there is a randomized polynomial time Turing machine M so that for all $x \in L, M$ accepts with probability at least $\frac{2}{3}$ and for all $x \notin L$, M accepts with probability at most $\frac{1}{3}$. Such a randomized Turing machine is said to have *two-sided error*. If, actually, for all $x \in L(x \notin L), M$ accepts (rejects) x with probability at least $1-\epsilon$, we say that M has error probability at most ϵ. If we have a **BPP**-machine M for a language L with error probability at most $\epsilon \leq \frac{1}{3}$, we can get a **BPP**-machine with error probability $\epsilon^{O(k)}$, by doing k independent runs of M and accepting if most of the runs accepts. This is an easy corollary of the *Chernoff bound*, a proof of which can be found, for instance, in Motwani and Raghavan, 1995:

Lemma 1 *Let $X_1, X_2, \ldots, X_m$ be independent 0-1 random variables and let $X = \sum X_i$. Let $\mu = \mathrm{E}[X]$. Then, for any $\delta > 0$,* $\Pr[X > (1+\delta)\mu] < [\frac{e^\delta}{(1+\delta)^{1+\delta}}]^\mu$ *and* $\Pr[X < (1-\delta)\mu] < e^{\frac{-\mu\delta^2}{2}}$

An immediate consequence of Lemma 1 is that the error probability of a **BPP**-machine can be made smaller than 2^{-n} on instances of length n, while preserving polynomial efficiency. This means that there is one particular coin toss sequence which will be good for all inputs of length n. As the Turing machine can be simulated by a circuit with polynomial overhead and the random coin toss sequence can be frozen and build into the circuit, we have that **BPP** $\subseteq$ **P/poly**, a fact first noted by Adleman, 1978.

Clearly **P** $\subseteq$ **RP** $\subseteq$ **BPP** $\subseteq$ **EXP**. We know that **P** $\neq$ **EXP** by the time hierarchy theorem, but at present, we don't know that **RP** $\neq$ **EXP**, though many people suspect **P** = **BPP**. In complexity theory, we often seek to understand a class through its *complete* problems, using, say, logspace reductions. For instance, to prove that **P**=**NP**, it is enough to find an efficient algorithm for a particular **NP**-complete problem. Similarly, if we wish to prove that **P**=**BPP**, it would seem like a good concrete strategy to first find a **BPP**-complete problem and focus on finding an efficient algorithm for that. Unfortunately, the classes

BPP and **RP** do *not* seem to have complete problems. This may seem somewhat surprising at first, as we would expect a complexity class to have at least the corresponding machine simulation problem as a complete problem. For instance, the problem NONDETERMINISTIC TURING MACHINE SIMULATION given by the language $\{\langle M, x\rangle | M$ is a nondeterministic Turing machine accepting x in at most $|x|^2$ steps $\}$ is complete for **NP**. The corresponding problem for two-sided error randomized computation would be RANDOMIZED TURING MACHINE SIMULATION, given by the language $\{\langle M, x\rangle | M$ is a two-sided error randomized Turing machine accepting x in at most $|x|^2$ steps $\}$. While this problem is *hard* for **BPP**, it is not *complete* for **BPP**, as it is not *in* **BPP**. The problem is that while being a nondeterministic Turing machine is a *syntactic* property, being a randomized one is a *semantic* one: Deciding whether a given Turing machine is a proper two-sided error randomized machine, i.e., whether it on every input accepts either with probability at least $\frac{2}{3}$ or with probability at most $\frac{1}{3}$, is not even decidable.

This somewhat annoying problem is really only an artifact of our definitions. In particular, it is caused by the decision to model all computational problems in terms of *languages*. The situation is much nicer if we extend the kind of problems we look at to include *partial languages* as well. Partial languages are also known in complexity theory as *promise problems*. A partial language or promise problem is simply a language $L \subseteq U$, where U is some subset of Σ^*. A Turing machine solves a promise problem L if it accepts all instances of L and rejects all instances of $U - L$. On every instance $x \notin U$, the machine can have arbitrary behavior. The trick is that the Turing machine does not have to be able to decide membership in U or even "know what U is". Informally speaking, we *promise* that we will only give the machine inputs from U. We can extend each complexity classes **C** considered above to a class of promise problems by changing its definition in the obvious way. This "promise-version" of a class **C** will be denoted **pC**, with **p** meaning either "promise" or "partial"[4].

For "syntactic classes" such as **P** or **NP**, there is not much reason to consider the promise-versions of the classes, as they behave exactly like the original classes in all important respects. For instance, **P**=**NP** if and only if **pP**=**pNP**. For "semantic classes" such as **RP** or **BPP**, the promise-versions are much better behaved than the original classes. For instance, the promise problem RANDOMIZED TURING MACHINE SIMULATION given by the partial language $\{\langle M, x\rangle | M$ is a two-sided error randomized Turing machine accepting x in at most $|x|^2$ steps $\}$ defined on $U = \{\langle M, x\rangle | M$ is a two-sided error randomized Turing machine $\}$ *is* complete for **pBPP**, as is easily seen.

The problems above are somewhat artificial. The classes **pBPP** and **pRP** also have natural complete problems, i.e. problems not mentioning simulating Turing machines.

Let CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION be the following promise problem. Given a Boolean circuit C of size n with n inputs and one output with a promise that *either* (a) $\Pr[C(x) = 1] \leq \frac{1}{3}$ *or* (b) $\Pr[C(x) = 1] \geq \frac{2}{3}$ for a randomly chosen $x \in \{0,1\}^n$, we should tell which of (a) or (b) is the case.

Let ONE-SIDED CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION be the following promise problem. Given a Boolean circuit C of size n with n inputs and one output with a promise that *either* (a) $\Pr[C(x) = 1] = 0$ *or* (b) $\Pr[C(x) = 1] \geq \frac{1}{2}$ for a randomly chosen $x \in \{0,1\}^n$, we should tell if (b) is the right case.

Lemma 2 *CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION is complete for* **pBPP** *and ONE-SIDED CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION is complete for* **pRP***.*

Proof We only describe the **pBPP**-case. The **pRP**-case is similar.

First, CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION is *in* **pBPP**: If the promise is kept we can see which of two possibilities is the case by randomly picking a sufficient number of assignments and seeing if the majority is accepted. The error probability can be made smaller than any desired constant.

Second, let any partial language L in **pBPP** be given and let M be a **pBPP**-machine for L with running time $p(n)$ on inputs of length n. Given an input length n, we can in space $O(\log n)$ build a circuit C of size $p(n)^{O(1)}$ with two inputs, the real input x, and a random bit sequence y, so that M accepts x when the random bit sequence is y if and only if $C(x, y) = 1$. Now, given an input x, we can build it into the circuit C, i.e. we let C_x be the circuit given by $C_x(y) = C(x, y)$. If x is in the domain of definition of L, C_x is a yes-instance of CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION if and only if $x \in L$. Similarly, C_x is a no-instance of CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION if and only if $x \notin L$. This completes the description of the reduction, except that the circuit constructed may have a size different from its number of inputs. To get a circuit with the same number of inputs as its size, we either add a number of “dummy inputs” that the circuit does not read, or add a number of “dummy gates” to the circuit that do not contribute to the output. This does not change the acceptance probability of the circuit. ∎

Most known randomized algorithms for natural decision problems have one-sided error. Also, technically, derandomizing one-sided error algorithms is easier than derandomizing two-sided error algorithms. Thus, it seems to be a good strategy to focus on derandomizing one-sided error algorithms. It can even be formally shown that this is sometimes a sufficient strategy, even if we are ultimately interested in two-sided error algorithms. This is a corollary of the following theorem, implicit in the work of Sipser, 1983, and Lautemann, 1983, and made explicit by Buhrman and Fortnow, 1999.

Theorem 2 $\mathbf{pBPP} = \mathbf{pRP}[\mathbf{pRP}]$.

Proof It is easy to see that $\mathbf{pBPP} = \mathbf{pBPP}[\mathbf{pBPP}]$. Thus, $\mathbf{pRP}[\mathbf{pRP}] \subseteq \mathbf{pBPP}$.

For the other direction, take a problem $L \subseteq U$ in **pBPP** by a randomized machine M. Assume that M uses $r(n)$ random bits on inputs of length n. By repeating the machine $O(r(n))$ times and taking majority of answers, we obtain a machine M' using $R = R(n) = O(r(n)^2)$ random bits and with error probability $2^{-O(r(n))} = 2^{-O(R^{1/2})}$.

Fix a particular $x \in U$. Let $A \subseteq \{0,1\}^R$ be the set of coin toss sequences that make M' accept on x. Either the density of A is greater than $\geq 1 - 2^{-O(R^{1/2})}$, in which case $x \in L$ or smaller than $2^{-O(R^{1/2})}$, in which case $x \notin L$.

Given A, and a vector $y \in \{0,1\}^R$, let $A \oplus y = \{x \oplus y | x \in A\}$, where $\oplus$ is bitwise XOR. Consider picking $y_1, y_2, \ldots, y_R$ at random. Let $A' = (A \oplus y_1) \cup (A \oplus y_2) \cup \cdots \cup (A \oplus y_R)$.

We claim that if $x \in L$, with very high probability, over the choice of $y_1, y_2, \ldots, y_R$, we have $A' = \{0,1\}^R$. On the other hand, if $x \notin L$, we immediately have that the density of A' is at most $R2^{-O(R^{1/2})}$, no matter which y_i's are chosen.

Suppose the claim is true. Then we show that we have the desired $\mathbf{pRP}[\mathbf{pRP}]$ algorithm for L. Given $y_1, y_2, \ldots, y_R$, we can easily construct a small circuit deciding membership in A': The circuit checks, on input z if there is a y_i so that the coin toss sequence $y_i \oplus z$ makes M' accept x. Given a random $y_1, y_2, \ldots, y_R$ we have that with very high probability either A' is very small or everything. Thus, if we negate the circuit for A', we have a circuit which either accepts with probability 0 or with very high probability. Deciding which is the case is in **pRP** by Lemma 2, in particular, we can take a **pRP** oracle O solving it. Then the following is a $\mathbf{pRP}[O]$ algorithm for L: Pick $y_1, y_2, \ldots, y_l$ at random, compute the circuit for A' and ask the oracle O if A' is small or everything. If the oracle says that A' is small, we say that $x \notin L$. Otherwise we say that $x \in L$. Note that $x \notin L$, then no matter which random

$y_1, y_2, \ldots, y_l$ we pick, A' will be small, so we always keep our promise to the oracle and thus always return the right answer. On the other hand, if $x \in L$, A' will be large with high probability, so we keep our promise to the oracle with high probability, and thus answer correctly with high probability. Thus, our algorithm is indeed an $\mathbf{pRP}[O]$ algorithm.

We now prove the claim. Assume that the density of A is at least $1-2^{-O(R^{1/2})}$. Fix a z in $\{0,1\}^R$ and consider the probability that it is not included in A'. Note that $z \in A \oplus y_i$ if and only if $z \oplus y_i$ is in A and $z \oplus y_i$ is uniformly distributed in $\{0,1\}^R$ and mutually independent of $z \oplus y_j$ for $j \neq i$. Thus, the probability that $z \notin A \oplus y_i$ is at most $2^{-O(R^{1/2})}$ and the probability that $z \notin A'$ is at most $(2^{-O(R^{1/2})})^R = 2^{-O(R^{3/2})}$ Thus, the probability that *some* z is not in A' is at most $2^{-O(R^{3/2})+R} = 2^{-O(R^{3/2})}$.

■

As $\mathbf{pP}[\mathbf{pP}] = \mathbf{pP}$, we have the following corollary.

Corollary 1 *If* **pRP**=**pP***, then* **pBPP**=**pP** *and in particular,* **BPP** = **P**.

Thus, if our aim is to prove that **pBPP**=**pP**, it is sufficient to prove the, a priori, easier statement **pRP**=**pP**. Corollary 1 is only known for promise problems. It is not known if **RP**=**P** implies **BPP**=**P**. In fact, Buhrman and Fortnow show that there are oracles relative to which this implication fails. One should be aware that Corollary 1 does not mean that we can answer any question about **pBPP** by considering the corresponding question for **pRP**. For instance, the statement "$\mathbf{pRP} \subseteq \mathbf{pSUBEXP}$ implies $\mathbf{pBPP} \subseteq \mathbf{pSUBEXP}$" has not been proven and its validity is not currently known (it does not even seem to be known if there is an oracle relative to which it fails). Also, the corollary is only relevant for deterministic simulation and not for randomness-efficient amplification nor for computing with weak random sources.

We now move to discuss classes defined by randomized computation with a bound on space, rather than a bound on time. There are certain subtleties and traps that one should be aware of when defining randomized space bounded computation. We refer to Saks, 1996, for an excellent exposition on this. We say that a language L is in $\mathbf{BP_H SPACE}(s(n))$ if there is a randomized machine M using space $O(s(n))$ on inputs of length n *and* with every computation eventually halting, so that for all $x \in L$, M accepts x with probability at least $\frac{2}{3}$ and for all $x \notin L$, M accepts x with probability at most $\frac{1}{3}$. We let $\mathbf{BP_H L}$ be $\mathbf{BP_H SPACE}(\log n)$. Simon, 1981, showed that $\mathbf{BP_H SPACE}(\log n)$ is included in $\mathbf{P}$ as well as $\mathbf{SPACE}(\log^2 n)$. An example of a language in $\mathbf{BP_H L}$ which is not known to be in **L** is UNDIRECTED GRAPH CONNECTIVITY, by the result of Aleliunas et al., 1979.

We saw that estimating the acceptance probability of a Boolean circuit is complete for **pBPP**. We can get a complete problem for $\mathbf{pBP_H L}$ by replacing "circuit" with a simpler computational device. Let AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION be the following promise problem. Given a deterministic finite automaton A over the alphabet $\{0,1\}$ and with n states, and with a single accepting state and with a promise that *either* (a) $\Pr[A \text{ accepts } x] \leq \frac{1}{3}$ *or* (b) $\Pr[A \text{ accepts } x] \geq \frac{2}{3}$ for a randomly chosen $x \in \{0,1\}^n$, we should tell which of (a) or (b) is the case.

Lemma 3 *AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION is complete for* $\mathbf{pBP_H L}$.

Proof First, AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION is *in* $\mathbf{pBP_H L}$: If the promise is kept we can see which of the two possibilities is the case by randomly picking a sufficient number of assignments and seeing if the majority is accepted (simulating the automaton on a particular input is easily done in logspace).

Second, let any partial language L in $\mathbf{pBP_H L}$ be given and let M be a $\mathbf{pBP_H L}$-machine for L using space $O(\log n)$ and with running time $p(n) = n^{O(1)}$ on inputs of length n. We can without loss of generality assume that M has a single accepting and halting configuration and a single rejecting and halting configuration. We modify the machine so that once it reaches one of these two configurations, it stays there rather than halting. Given an input x of length n, we can in space $O(\log n)$ build x into the machine, getting a finite state automaton with $s = n^{O(1)}$ states. The input of the automaton is the sequence of coin tosses made by the original machine. Now $x \in L$, if and only if this automaton accepts a random input of length s with probability more than $\frac{2}{3}$ and $x \notin L$, if and only if this automaton accepts a random input of length s with probability less than $\frac{1}{3}$. ■

An arguably even more natural complete problem for $\mathbf{pBP_H L}$ is APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION: Given an $n \times n$ stochastic matrix M with entries taken from $0, 1, \frac{1}{2}$ and indices i, j, with a promise that $(M^n)_{ij}$ is either at least $\frac{2}{3}$ or at most $\frac{1}{3}$, tell which of the two is the case.

Lemma 4 *APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION is complete for* $\mathbf{pBP_H L}$.

Proof We present two reductions, one in each direction, between APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION and AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION.

Given an instance M, i, j of APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION, where M is an $n \times n$ matrix, we can build a finite automaton A with n states $\{1, 2, \dots, n\}$. The transition function δ of the automaton is defined as follows. Each row r of M either contains a single 1 in one entry (say, s) and the rest of the entries 0, or it contains two $\frac{1}{2}$'s (say, in entries $s_1 < s_2$) and the rest 0. In the first case, we let $\delta(i, 0) = \delta(i, 1) = j$. In the second case, we let $\delta(i, 0) = s_1$ and $\delta(i, 1) = s_2$. We let the start state of the automaton be i and the single accepting state be j. It is easy to see that the probability that the automaton so constructed accepts a random input $x \in \{0, 1\}^n$ is exactly $(M^n)_{ij}$, so the reduction is valid.

For the other direction, given an instance A of AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION with n states and labelled $1, 2, \dots, n$ with start state 1 and single accepting state n, we can build an $n \times n$ matrix M with M_{rs} being the probability A goes from state r to state s in a single step. Then $M, 1, n$ is an instance of APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION and the reduction is easily seen to be valid. ■

2.2 UNIVERSAL HASHING

The notion of a hash function is a 1950's notion, originating from the programming of space-efficient lookup tables. Originally, the concept was heuristic rather than rigorous: A hash function was a single fixed function that was assumed, in all respects, to behave like a random function. This is of course an impossibility and thus somewhat unsatisfactory from a theoretical point of view. The "modern" point of view was introduced in the seminal paper of Carter and Wegman, 1979. In modern terminology, we rarely speak about single hash functions in isolation, but only about families of hash functions. What we aim for is a small family of functions so that if one function is picked uniformly at random from the family, certain *specific* events occur with roughly the same probability as they would have, had the functions been picked uniformly at random from the space of all functions.

An important event is a *collision* between two elements x, y introduced by the hash function h, i.e., the event that $h(x) = h(y)$. Concentrating on such events leads to Carter and Wegman's definition of a universal class of hash functions. To be precise, let $\mathcal{H}$ be a family of functions with each $h \in \mathcal{H}$ mapping $\{0, 1\}^n$ to $\{0, 1\}^m$. We say that $\mathcal{H}$ is a *universal* *family of hash functions*, if, for all fixed $x, y \in \{0,1\}^n$, $x \neq y$ implies that if h is picked uniformly at random from $\mathcal{H}$, $\Pr[h(x) = h(y)] \leq 2^{-m}$.

We say that $\mathcal{H}$ is *strongly universal* if, for all fixed $x, y \in \{0,1\}^n$ and $a, b \in \{0,1\}^m$, we have that $x \neq y$ implies that if h is picked uniformly at random from $\mathcal{H}$, $\Pr[h(x) = a \wedge h(y) = b] = 2^{-2m}$.

A reformulation of the notion of a strongly universal family of hash functions is the notion of a pairwise independent sample space: The definition of strong universality is equivalent to stating that if h is picked uniformly at random from $\mathcal{H}$, then all random variables in the set $\{h(x)\}_{x \in \{0,1\}^n}$ are uniformly distributed and pairwise independent. Thus, constructing a small strongly universal family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$ is equivalent to constructing a small uniform probability space with 2^n associated random variables, each uniformly distributed in $\{0,1\}^m$ and pairwise independent.

Sometimes more than pairwise independence is needed. Given an integer $k \geq 2$, a family $\mathcal{H}$ is a *k-wise* independent family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$ if the random variables in the set $\{h(x)\}_{x \in \{0,1\}^n}$ are k-wise independent and uniform on $\{0,1\}^m$ when h is picked uniformly at random from $\mathcal{H}$.

Finally we say that $\mathcal{H}$ is *k-weakly universal* if for each $h \in \mathcal{H}$, if, for all fixed $x, y \in \{0,1\}^n$, $x \neq y$ implies that if h is picked uniformly at random from $\mathcal{H}$, $\Pr[h(x) = h(y)] \leq k2^{-m}$.

We shall need small, efficiently computable families of universal hash functions. A straightforward way of constructing such families is by using the properties of arithmetic over finite fields. In particular, if $\mathbf{F}$ is a finite field then the family of functions $\mathcal{H} = \{h_{a,b} : \mathbf{F} \to \mathbf{F} | a, b \in \mathbf{F}, h_{a,b}(x) = ax + b\}$ is strongly universal as is easily seen. If $|\mathbf{F}| = 2^n$, we can identify $|\mathbf{F}|$ with $\{0,1\}^n$. This identification can be made so that arithmetic is polynomial time and polylog space efficient, using a construction of Shoup, 1994. If we desire a universal family mapping $\{0,1\}^n$ to $\{0,1\}^m$ for $n \neq m$, we can easily modify the family above: If $n > m$, we take the family above and throw away $n - l$ bits of the outputs of the functions. If $n < m$, we take a family mapping $\{0,1\}^m$ to $\{0,1\}^m$ and take the restriction of every function in the family to a subdomain of size 2^n.

The construction easily generalizes to k-wise independent families: We take $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_{k-1}} : \mathbf{F} \to \mathbf{F} \mid a_0, a_1, \ldots, a_{k-1} \in \mathbf{F}, h_{a_0,a_1,\ldots,a_{k-1}}(x) = a_0 + a_1 x + a_2 x^2 + \ldots a_{k-1} x^{k-1}\}$.

Summing up, we have

Lemma 5 *For every k, and every n, m, there is a k-wise independent family $\mathcal{H}$ of size at most $2^{k \max(n,m)}$ mapping $\{0,1\}^n$ to $\{0,1\}^m$, with* *each member in $\mathcal{H}$ described as a bit string of length $k \max(n,m)$, so that members in $\mathcal{H}$ can be evaluated in polynomial time and linear space.*

Sometimes we need a bit better performance than just polynomial time and linear space. For pairwise independence, we can easily do somewhat better: For any integer n, and $x, y \in \mathrm{GF}(2)^n$, the wrapped convolution $x \circ y \in \mathrm{GF}(2)^n$ of x and y is given by $(x \circ y)_i = \sum_{j=1}^{n} x_j y_{(i-j) \bmod n}$. Impagliazzo and Zuckerman, 1989, and Mansour et al., 1993, observed that $\mathcal{H} = \{h_{a,b} : \{0,1\}^n \to \{0,1\}^n | a \in \mathbf{F}, b \in \mathbf{F}, h_{a,b}(x) = a \circ x + b\}$ is strongly universal and, for $m \leq n$ $\mathcal{H} = \{h_a : \{0,1\}^n \to \{0,1\}^m | a \in \mathbf{F} - \{0\}, h_a(x) = \text{first } m \text{ bits of } a \circ x\}$ is universal. Unlike the family based on finite field, we can easily compute this family in logspace. Furthermore, wrapped Boolean convolution has nearly linear sized circuits by the classical result of Schönhage and Strassen, 1971. Summing up, we have

Lemma 6 *For every $m \leq n$, there is a strongly universal logspace computable family of size 2^{2n} mapping $\{0,1\}^n$ to $\{0,1\}^n$ with circuits of size $O(n \log n \log \log n)$, and a universal logspace computable family of size 2^n mapping $\{0,1\}^n$ to $\{0,1\}^m$ with circuits of size $O(n \log n \log \log n)$.*

Dietzfelbinger, 1996, presented a similar construction, yielding a k-wise independent family, for any $k \geq 2$ (his family is based on integer multiplication, rather than wrapped convolution). Thus, even k-wise independence can be done in logspace and with almost linear sized circuits. We shall not need this fact, however.

It is an interesting open problem if there is a strongly universal family with linear sized circuits. However, Andreev et al., 1997, noted that if we only want universality and not strong universality, and furthermore want the size of the co-domain to be only slightly smaller than the size of the domain, linear sized circuits are possible.

Lemma 7 *For any $n \geq 1$, there is a universal family $\mathcal{H}$ mapping $\{0,1\}^n$ to $\{0,1\}^{n-1}$ with the following properties.*

- *Each $h \in \mathcal{H}$ is two-to-one, i.e., each $y \in \{0,1\}^{n-1}$ has exactly two values x_1, x_2 in $\{0,1\}^n$, so that $h(x_1) = h(x_2) = y$.*
- *For each $h \in \mathcal{H}$, h and h^{-1} have linear sized circuits (the latter circuit returning two values).*

Proof Let h be given by $\mathcal{H} = \{h_a\}_{a \in \{0,1\}^n, a \neq 0}$ where h_a is the natural group homomorphism $h_a : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n/(a)$. Obviously, each member of $\mathcal{H}$ is two-to-one, and it is easily checked that $\mathcal{H}$ is universal.

Thus, we just need to find a representation of $\mathrm{GF}(2)^n/(a)$ as bit vectors of length $n-1$ so that h and h^{-1} have linear circuits.

The representation we choose is as follows. Let i be any index in $\{1,\ldots,n\}$ for which $a_i \neq 0$. Let $\tilde{x}$ be a member of $\mathrm{GF}(2)^n/(a)$. We can view $\tilde{x}$ as a two-element subset of $\mathrm{GF}(2)^n$ and may note that exactly one element of the subset, say x, has $x_i = 0$. We represent $\tilde{x}$ as the bit string x with the i'th bit removed. It is easily checked that with this representation, h_a and h_a^{-1} have linear sized circuits and we are done. ■

When considering weakly universal family, we clearly want to gain something compared to the bounds above. It turns out that we can make the size of weakly universal families much smaller than the size of universal families. A simple way to construct a small k-weakly universal family for an integer $k \geq 2$ is to use univariate polynomials over a finite field again, but switch the roles of the hash function and the value to be hashed. To be precise, if we take $\mathcal{H} = \{h_x : \mathbf{F}^k \to \mathbf{F} \mid x \in \mathbf{F}, h_x(a_0, a_1, \ldots, a_k) = a_0 + a_1 x + a_2 x^2 + \ldots a_{k-1} x^{k-1}\}$, we get a k-weakly universal family, so we have

Lemma 8 *For every k, and every n, there is a k-weakly universal family $\mathcal{H}$ of size 2^n mapping $\{0,1\}^{kn}$ to $\{0,1\}^n$ that can be evaluated in polynomial time and linear space.*

By a somewhat more involved construction one can get small k-weakly universal families of comparable size with k very close to 1. Goldreich and Wigderson, 1994, construct such a family which we state without proof.

Lemma 9 *For every $\epsilon > 0$, and every n, m, there is a $(1+\epsilon)$-weakly universal family $\mathcal{H}$ of size $(n2^m/\epsilon)^{O(1)}$ mapping $\{0,1\}^n$ to $\{0,1\}^m$ that can be evaluated in polynomial time and linear space.*

2.3 ERROR CORRECTING CODES

The original motivation of the theory of error correcting codes is data transmission over a noisy channel. We consider the following setup. A *sender* wants to transmit a bit string x to a *receiver*. However, a *meddler* is allowed to introduce a limited amount of noise in any message sent from the sender to the receiver by flipping a certain fraction of the bits in the message. How does the sender transmit the message reliably?

The solution is to use an *error correcting code*. Though we are mainly interested in Boolean codes, we need the concept of a code over an arbitrary set in order to construct good Boolean codes. Let n, m be integers with $n < m$ and let S be a set of size at least 2. An *error correcting* code over S with *relative minimum distance* δ is a map $e : S^n \to S^m$, so that for any $x \neq y$, the fraction of entries where $e(x)$ and $e(y)$ differ (i.e., the relative Hamming distance between $e(x)$ and $e(y)$), is at least δ. The set $\{e(x)|x \in S^n\}$ is the set of *code words* of e. The *rate* of the code is the value n/m. For the applications we consider, it is often more convenient to speak of the *blowup* of the code rather than the rate. The blowup is simply the value m, as a function of n. For example, if $m = n^2$, we speak about a code with quadratic blowup. If $S = \{0,1\}$, the code is a *Boolean* code. If the original message x is a prefix of the encoded message $e(x)$, i.e. $e(x) = x \circ e'(x)$ for some map e', we say that the code is *systematic*. If S is a field and the set of code words form an n-dimensional subspace of the vector space over S of dimension m, we say that the code is *linear*. All codes considered in this chapter will be linear.

Let us see that error correcting codes solve the data transmission problem. Let e be a Boolean error correcting code with relative minimum distance δ. Given $x \in \{0,1\}^n$, if the sender computes $\tilde{x} = e(x)$ and the meddler flips less than a $\delta/2$ fraction of the bits of $\tilde{x}$, and transmits the result x^* to the receiver, the receiver can reconstruct the original message as the value x minimizing the Hamming distance between $e(x)$ and x^*. This procedure is called *maximum likelihood decoding*. Note that it as stated is an exponential procedure. Sometimes we can find more efficient procedures.

Coding theory is concerned with the construction of codes with small blowup and large relative minimum distance. In the traditional applications of coding theory, often a linear blowup $n \to O(n)$ or even $n \to n + o(n)$ is required and lots of effort is spent constructing such codes. For the applications to derandomizing complexity classes, it is sufficient to consider codes with polynomial blowup. Such codes are rather easily constructed, as we see next.

Let $\mathbf{F}$ be a finite field of size $m > n$. The *Reed-Solomon* code over $\mathbf{F}$ is the code $e_{\mathrm{RS}} : \mathbf{F}^n \to \mathbf{F}^m$ defined as follows. Let $i_1, i_2, \ldots, i_n$ be arbitrary distinct members of $\mathbf{F}$. Given $x \in \mathbf{F}^n$, let p be the unique polynomial of degree at most $n-1$, so that $p(i_j) = x_j$ for all $j \in \{1, \ldots, n\}$. Let $e_{\mathrm{RS}}(x)$ be $(p(y))_{y \in \mathbf{F}}$, i.e., the enumeration of p on all members of $\mathbf{F}$. As every two polynomials of degree at most $n-1$ agree on at most $n-1$ points, we find that the Reed-Solomon code has relative minimum distance $\frac{m-n+1}{m}$. Thus, if we let $m = n^c$ for a constant c, we get a code with polynomial blowup and relative minimum distance close to 1.

The Reed-Solomon code has an efficient maximum likelihood decoding procedure. More generally, such a decoding procedure exists even if we modify the code so that the encoding of x is not the enumeration of p over *all* members of $\mathbf{F}$ but only on some subset containing $i_1, i_2, \ldots, i_n$. This was shown by Berlekamp and Welch, 1986. We state the existence of this procedure as a lemma.

Lemma 10 *There is a polynomial procedure, which, given as input a subset $M \subseteq \mathbf{F}$ of size m and the tabulation of a function $f : M \to \mathbf{F}$ which disagree with some polynomial p of degree at most d on less than a $\frac{1}{2} - \frac{d}{2m}$ fraction of M, outputs the unique such polynomial p.*

The Reed-Solomon code is a great code. Its only problem is that it is not Boolean. For Boolean codes, the *Plotkin bound* states that any Boolean code with relative minimum distance $d > \frac{1}{2}$ has at most $\frac{2d}{2d-1}$ code words, i.e., if d is bounded away from $\frac{1}{2}$ by a constant, the number of code words is constant. Thus, for asymptotically non-trivial Boolean codes, the best relative minimum distance we can hope for is $\frac{1}{2}$. The *Hadamard* code achieves this. Given any integer value l, the Hadamard code $e_{\mathrm{H}} : \{0,1\}^l \to \{0,1\}^{2^l}$ is defined $e_{\mathrm{H}}(x) = (\langle x, y\rangle)_{y \in \{0,1\}^l}$, i.e., we encode x as the inner products of x with all possible vectors y. It is easy to see that the relative minimum distance of the Hadamard code is $\frac{1}{2}$: Given fixed vectors $x_1 \neq x_2$, the probability that a randomly chosen vector y has $\langle x_1 - x_2, y\rangle = 0$ is exactly $\frac{1}{2}$. Thus, the Hadamard code has exponential blowup but a good minimum distance.

To get a Boolean code with satisfactory blowup and relative minimum distance, we introduce the concept of *code concatenation*, which we define by the example relevant to us: Given the Reed-Solomon code $e_{\mathrm{RS}} : \mathbf{F}^n \to \mathbf{F}^m$ for a field of size m, where $m = 2^l$ take any encoding c of field elements as bit vectors of length l, and consider the Hadamard code e_{H} with parameter l. Now we define the concatenation of the Reed-Solomon code with the Hadamard code $e_{\mathrm{RS}} \circ e_{\mathrm{H}}$ as the Boolean code $e_{\mathrm{RS}} \circ e_{\mathrm{H}} : \{0,1\}^n \to \{0,1\}^{m2^l}$, defined by

$$e_{\mathrm{RS}} \circ e_{\mathrm{H}}(x) = e_{\mathrm{H}}(c(y_1)) \circ e_{\mathrm{H}}(c(y_2)) \circ \cdots \circ e_{\mathrm{H}}(c(y_m))$$

where $y = y_1 y_2 \ldots y_m = e_{\mathrm{RS}}(x)$ and $\circ$ denotes concatenation of strings. In short, we take the Reed-Solomon code of x (viewing the bits of x as the field elements 0 and 1 of $\mathbf{F}$) and encode each of the field elements of the resulting code word in the Hadamard code.

Lemma 11 *Given any parameters $n \leq m$ with m a power of two, the concatenation of the Reed-Solomon code with the Hadamard code $e = e_{\mathrm{RS}} \circ e_{\mathrm{H}} : \{0,1\}^n \to \{0,1\}^{m^2}$ is a Boolean code with minimum distance $\frac{1}{2} - \frac{n-1}{2m}$.*

Proof Take any two different code words of e. The corresponding Reed-Solomon code words differ on at least $m - n + 1$ entries, i.e., a $\frac{m-n+1}{m}$ fraction of all the entries. Where the Reed-Solomon code words differ, the Boolean Hadamard encoding of the corresponding field element differ on half the entries. Thus, the code word of e differ on at least a $\frac{m-n+1}{m} \cdot \frac{1}{2}$ fraction of the bits. ■

The code e of Lemma 11 will satisfy all the "generic" coding needs of this chapter. In fact this is the case for the following specific setting of the parameters. Note that the code can be made systematic (so that x is a prefix of $e(x)$) by an appropriate renumbering of the bit positions of the code words.

Lemma 12 *For n a power of two, there is a polynomial time computable, linear, systematic code $e : \{0,1\}^n \to \{0,1\}^{n^{10}}$ with relative minimum distance at least $\frac{1}{2} - \frac{1}{n^4}$.*

If an error correcting code has minimum distance d, and we are given a vector which has distance less than $d/2$ to a code word, maximum likelihood decoding finds this code word. Thus, if $d = \frac{1}{2} - \epsilon$, we can correct up to a $\frac{1}{4} - \epsilon$ fraction of errors. For some applications we shall see, this correction distance is not good enough. For such applications we can consider *list decoding*: If we are content with finding not one original code word, but a small list of possibilities, we can do much better. The following lemma, due to Bellare et al., 1998, Lemma A.1, and stated here without proof (the proof is a clever, but elementary, counting argument), expresses this fact.

Lemma 13 *Let $0 < \alpha < \frac{1}{2}$ and let a Boolean code of relative minimum distance $\frac{1}{2} - \alpha$ be given. The number of code words within distance $\frac{1}{2} - \alpha^{\frac{1}{2}}$ from any given string is at most $\frac{1}{2\alpha}$.*

For the special case of the code of Lemma 12, we get

Lemma 14 *Any Hamming ball in $\{0,1\}^{n^{10}}$ of relative radius $\frac{1}{2} - \frac{1}{n^2}$ contains at most $\frac{n^4}{2}$ code words of e, where e is the code of Lemma 12.*

Finally, we shall need a connection between error correcting codes and universal hashing. In Miltersen, 1998, it was observed that there is a very simple weakly universal family of hash functions for the set of code words for an error correcting code. Let the family $\mathcal{H}$ of *projections* be the family of functions of the form $h_S : \{0,1\}^m \to \{0,1\}^l$, where S is an l-subset of $\{1, 2, \ldots, m\}$ and where $h(x)$ is the concatenation of the bits of x, indexed by S. For example, $h_{\{2,3,5\}}(011000101) = 110$. Unlike the hash functions of the previous subsection, projections are trivial to compute and intuitively, "does not mess up the data". Still, we have the following lemma.

Lemma 15 *Let $C \subseteq \{0,1\}^m$ be the set of code words of an error correcting code of relative minimum distance $\frac{1}{2} - \epsilon$. The projection family $\mathcal{H}$ is a $(1+2\epsilon)^l$-weakly universal family from C to $\{0,1\}^l$.*

Proof Let $\delta = \frac{1}{2} - \epsilon$. Given $x, y \in \{0,1\}^m$, we must show that a random element h_S of $\mathcal{H}$ has $h_S(x) = h_S(y)$ with probability at most $(1+2\epsilon)^l 2^{-l} = (1-\delta)^l$.

We can pick a random element $h_S \in \mathcal{H}$ by picking each element of S at random, one at a time, without replacement. The final hash function h_S will have $h_S(x) \neq h_S(y)$ if at least one of the bits chosen is a bit where x and y differ. Suppose we have picked elements $i_1, i_2, \ldots, i_j$ of S and have not separated x and y. We pick i_{j+1} at random from $\{1, \ldots, m\} - \{i_1, i_2, \ldots, i_j\}$. We can bound from below the conditional probability that the inclusion of i_{j+1} will separate x and y, by $\frac{\delta m}{m-j} \geq \delta$. Thus, the probability that none of the chosen elements in the final set S will separate x and y is at most $(1-\delta)^l$, as desired. ■

3. DISPERSERS AND EXTRACTORS

3.1 DISPERSERS

Consider the following, purely combinatorial, two-player game. Let N be a positive integer and let $\epsilon > 0$. The first player, Alice, secretly chooses at least $(1-\epsilon)N$ cards from a deck of N cards and puts a mark on the face of the cards chosen. Then she places all the cards in the deck, face down, before the second player, Bob. Now Bob must turn over at most d cards. If a mark is revealed, Bob wins the game.

If $d \leq \epsilon N$, no *deterministic* strategy for Bob can be good: If Alice knows the strategy, she can simply mark a set of cards she knows that Bob will not choose and Bob loses. On the other hand, a good and simple *randomized* strategy for Bob would be to choose the d cards to turn over randomly, without replacement, yielding a loosing probability for Bob which can be bounded from above by ϵ^d.

For choosing d cards independently at random Bob needs to use approximately $d \log N$ random bits. We now add a twist to the game: We want Bob to adhere to a randomized strategy *using much fewer random bits*. Formally, we define an *R-bit strategy* for Bob to be a map ϕ from $\{0,1\}^R$ to the set of subsets of $\{1, \ldots, N\}$ of size at most d. Bob executes the strategy by picking $x \in \{0,1\}^R$ uniformly at random and turning over the cards of $\phi(x)$.

The general task we now face is, for given values of N, ϵ, R, and d to find an R-bit strategy minimizing Bob's chance of loosing the game, or at least making this chance strictly smaller than some value p. Let us (temporarily, until Proposition 1) denote such a strategy an (N, ϵ, R, d, p)-*strategy.*

It turns out that we can conveniently rephrase the notion of a strategy in the language of bipartite graphs.

Definition 1 *A bipartite graph on vertex sets U and V and with edges $E \subseteq U \times V$ is called an $(1-\epsilon)$-hitting* disperser *with* threshold T *if, for all subsets S of U of size $|S| \geq T$, we have that the size of the set of neighbors of S in V, $|\Gamma(S)|$, is more than $\epsilon|V|$.*

Given an R-bit strategy ϕ, we can form a bipartite graph with $U = \{0,1\}^R$ and $V = \{1, \dots, N\}$ and edges defined by $\Gamma(x) = \phi(x)$ for $x \in U$, where $\Gamma(x)$ is the set of neighbors in V of x. We now have

Proposition 1 *A strategy is an (N, ϵ, R, d, p)-strategy if and only if the corresponding graph is a $(1-\epsilon)$-hitting disperser with threshold $p|U|$.*

Proof First, we show that the graph of a good strategy is a good disperser. So, let the graph of an (N, ϵ, R, d, p)-strategy be given. We must show that in the corresponding graph, every U-subset of size at least $p|U|$ has more than $\epsilon|V|$ neighbors. Suppose not. Then some U-subset S of size at least $p|U|$ has at most $\epsilon|V|$ neighbors. This means that there is a subset S' of $|V|$ of size at least $(1-\epsilon)|V|$ and no edges between S and S'. Now we play the game between Alice and Bob with S' being the set of marked cards. By definition of an (N, ϵ, R, d, p)-strategy, if Bob chooses a random vertex $x \in U$, the probability that the set of neighbors of x does not intersect S' is less than p. Thus, this is the case for less than $p|U|$ vertices x in U. On the other hand, it is the case for all the vertices of S, and $|S| \geq p|U|$, a contradiction.

Next, we show that a good disperser defines the graph of a good strategy. So let G be a $(1-\epsilon)$-hitting disperser on $U \times V$ with threshold $p|U|$ and appropriate degree and size of U and V. We want to show that if we take any subset S' of V of size at least $(1-\epsilon)|V|$ and a random vertex x of U, the probability that S' does not contain a neighbor of x is less than p. So suppose it is at least p, and let S be the subset of U which is not neighboring S'. As $|S| \geq p|U|$, by the disperser-property, we have that the set of neighbors of S has size more than $\epsilon|V|$. But this is impossible, as $|S'| \geq (1-\epsilon)|V|$. ∎

It is sometimes convenient to describe a disperser as a *map* $D : U \times \{1, 2, \ldots, d\} \to V$ instead of a bipartite graph, with the natural correspondence between maps and graphs in mind, i.e., $D(x, i)$ is the i'th neighbor of x, relative to some arbitrary enumeration of its neighbors.

It is immediate how we can use strategies developed for the card game to solve the randomness-efficient amplification problem for a one-sided error randomized algorithm deciding a language L: If we have a randomized algorithm for L using r random bits on instances of a certain size and with error probability at most ϵ, we let $N = 2^r$, and let the cards of Alice be the possible sequences of r random coin tosses. For a positive instance $x \in L$, we let the marked cards be the coin toss sequences leading to acceptance of x. Now we get the following randomized algorithm deciding L and using R random bits: Take a random element $y \in \{0, 1\}^R$. Compute the set $\phi(y)$. Execute the original algorithm on input x with the set of coin toss sequences in $\phi(x)$. If one of them leads to acceptance, we accept, otherwise we reject. The error probability of our algorithm is at most Bob's risk of loosing the game.

To solve the amplification problem in the way we described above while maintaining efficiency, we need the computation of $\phi(y)$, given y to be efficient. The notion of an efficient strategy, translated into the language of graphs, becomes the notion of an *explicit disperser*.

Definition 2 *A family of dispersers $E_r \subseteq U_r \times V_r$ with $|V_r| = 2^r$ is* explicit*, if there is an algorithm which on input r and $x \in U_r$ enumerates the neighbors of x in V_r, in time polynomial in $r, \log |U_r|$ and the number of such vertices.*

The discussion above yields the following "master lemma".

Lemma 16 *If there is an explicit family of $\frac{1}{2}$-hitting dispersers $E_r \subseteq U_r \times V_r$, $r = 1, 2, \ldots$ with $|V_r| = 2^r$, $|U_r| = 2^{R(r)}$, degree $d(r)$ for each vertex in U_r, and threshold $T(r)$, then a one-sided error algorithm running in time t, using r random bits to achieve error probability $\frac{1}{2}$ can be converted to a one-sided error algorithm, running in time $(R(r) + d(r) + t)^{O(1)}$ using $R(r)$ random bits to achieve error probability $T(r)/2^{R(r)}$.*

We will next show that, perhaps somewhat surprisingly, dispersers not only immediately solve the amplification problem, but also the weak random source problem, for one-sided error algorithms.

Lemma 17 *If there is an explicit family of $\frac{1}{2}$-hitting dispersers $E_r \subseteq U_r \times V_r$, $r = 1, 2, \ldots$ with $|V_r| = 2^r$, $|U_r| = 2^{R(r)}$, degree $d(r)$ for each* *vertex in U_r, and threshold $T(r)$, then a one-sided error algorithm running in time t, using r random bits to achieve a error probability $\frac{1}{2}$ can be simulated by an algorithm running in time $(R(r)+d(r)+t)^{O(1)}$ and using any $R(r)$-bit weak random source with min-entropy at least $\log T(r) + k$ to achieve error probability 2^{-k}.*

Proof By lemma 16, the disperser enables us to reduce the error probability of the algorithm to $T(r)/2^{R(r)}$, using $R(r)$ unbiased random bits. Now consider just giving this modified algorithm a sample of the weak random source instead of the coin toss sequence. As the error probability is at most $T(r)/2^{R(r)}$, at most $T(r)$ specific coin toss sequences make the algorithm give the wrong output. As the source has min-entropy at least $\log T(r) + k$, each of these bad sequences will be given with probability at most $\frac{1}{2^k T(r)}$. Thus, the probability of getting a bad sequence is at most 2^{-k}. This is also a bound on the error probability of the modified algorithm using the weak random source. ∎

3.2 EXTRACTORS

We shall next see what the corresponding definitions would look like for two-sided error algorithms. We should consider a modified card game between Alice and Bob. Again, Alice marks at least a $1-\epsilon$ fraction of the cards, but now, it is not sufficient for Bob that one of the cards he turns over is marked: *More than half of them should be marked.* By the Chernoff bound, choosing cards uniformly at random is a good strategy for Bob. Again, we want to find strategies using fewer random bits. We can use a strategy for Bob to do randomness-efficient amplification for two-sided error algorithms, as the strategy would also enable Bob to distinguish between a situation where Alice marks at least a $1-\epsilon$ fraction of the cards from a situation where she marks at most an ϵ fraction of the cards. As we did above, we can turn the definition of such a strategy into a graph theoretic property of bipartite graphs. This would lead to the definition of what is known as a *majority disperser.*

While explicit majority dispersers are sufficient for both randomness-efficient amplification and computing with weak random sources, we choose in this chapter to focus on a somewhat stronger property, leading to the graph theoretic notion of an *extractor.* An extractor is also a majority disperser and a disperser, but extractors have many other applications as well. In particular, they are better suited than dispersers and majority dispersers to being used as building blocks in other constructions.

Before defining extractors, let us introduce and analyze a closeness measure for probability distributions. We say that two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ are ϵ-close if their L_1-distance $\|\mathcal{D}_1 - \mathcal{D}_2\|_1$ is at most ϵ, i.e., if the sum, over all possible outcomes, of the distance between the two probabilities of the outcome according to the two distributions, is at most ϵ. The following lemma is easily proved and very useful.

Lemma 18 *Two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ on domain S are ϵ-close if and only if, for any event $T \subseteq S$, $|\Pr_{x\in\mathcal{D}_1}[x \in T] - \Pr_{x\in\mathcal{D}_2}[x \in T]| \leq \epsilon/2$*

In Lemma 18, one should think of T as a *statistical test*, trying to distinguish between $x \in \mathcal{D}_1$ and $x \in \mathcal{D}_2$. Thus, we also say that $\mathcal{D}_1$ and $\mathcal{D}_2$ are *statistically close*.

Statistical closeness is preserved under deterministic and random transformations.

Lemma 19 *If x_1 is a random variable with distribution $\mathcal{D}_1$ and x_2 is a random variable with distribution $\mathcal{D}_2$ and $\mathcal{D}_1$ and $\mathcal{D}_2$ are ϵ-close, then for any function f, $f(x_1)$ and $f(x_2)$ have distributions which are ϵ-close. Furthermore if y is a random variable independent of x_1 and x_2, $f(x_1, y)$ and $f(x_2, y)$ have distributions which are ϵ-close.*

We present two lemmas which are useful for proving closeness when one of the two distributions is the uniform distribution.

Let $\mathcal{D}$ be a probability distribution. The *collision probability* of $\mathcal{D}$ is the probability that two independent samples from $\mathcal{D}$ are identical. The first lemma is due to Impagliazzo et al., 1989.

Lemma 20 *Let $\mathcal{D}$ be a probability distribution on a domain of size s. If the collision probability of $\mathcal{D}$ is at most $(1+\epsilon)/s$, then $\mathcal{D}$ is $\sqrt{\epsilon}$-close to uniform.*

Proof Let $v(x) = \mathcal{D}(x) - 1/s$. Then, if we view v as a real vector of length s, we want to bound the one-norm of v, as this is the distance between $\mathcal{D}$ and the uniform distribution.

Viewing $\mathcal{D}$ as a vector, the collision probability of $\mathcal{D}$ is $\langle\mathcal{D},\mathcal{D}\rangle$, so we have $\langle\mathcal{D},\mathcal{D}\rangle \leq (1+\epsilon)/s$. But $\langle\mathcal{D},\mathcal{D}\rangle = \langle v+1/s, v+1/s\rangle = \langle v,v\rangle + \langle 1/s, 1/s\rangle + 2\langle v, 1/s\rangle = \langle v,v\rangle + 1/s \leq (1+\epsilon)/s$, so $\langle v,v\rangle \leq \epsilon/s$. By the Cauchy-Schwartz inequality $\|v\|_1 \leq \sqrt{s}\sqrt{\langle v,v\rangle} \leq \sqrt{\epsilon}$. ■

The second lemma is due to Trevisan, 1999, building on Yao, 1982. Here, we consider a probability distribution $\mathcal{D}$ on $\{0,1\}^m$, rather than an arbitrary domain and take the structure of $\{0,1\}^m$ into account. We say that a probability distribution $\mathcal{D}$ on $\{0,1\}^m$ is *ϵ-predictable* if for some i

between 1 and m, there is a function (the *predictor*) $f : \{0,1\}^{i-1} \to \{0,1\}$ which predicts the i'th bit of $x \in \mathcal{D}$ well, given the previous bits, in the following sense. If $x \in \{0,1\}^n$ is chosen randomly according to $\mathcal{D}$, then

$$\Pr[f(x_{1..(i-1)}) = x_i] \geq \frac{1}{2} + \epsilon.$$

The value ϵ is called the *advantage* of the predictor.

Lemma 21 *If a distribution $\mathcal{D}$ on $\{0,1\}^m$ is not ϵ-close to uniform, then it is $\frac{\epsilon}{2m}$-predictable.*

Proof We consider the $m+1$ distributions $\mathcal{D}_0, \ldots, \mathcal{D}_m$, defined as follows. To pick a random element of $\mathcal{D}_i$, we pick $x \in \{0,1\}^m$ according to $\mathcal{D}$ and $y \in \{0,1\}^m$ according to the uniform distribution $\mathcal{U}$ and return $x_1, \ldots, x_i, y_{i+1}, \ldots, y_m$. Thus, $\mathcal{D}_0$ is the uniform distribution, $\mathcal{D}_m$ is $\mathcal{D}$, and the $\mathcal{D}_i$'s in between are *hybrids* of the uniform distribution and $\mathcal{D}$.

If $\| \mathcal{D} - \mathcal{U} \|_1 > \epsilon$, then, by the triangle inequality, there must be i so that $\| \mathcal{D}_{i+1} - \mathcal{D}_i \|_1 > \epsilon/m$. Let $\mathcal{D}'_{i+1}$ and $\mathcal{D}'_i$ be the distribution induced by $\mathcal{D}_{i+1}$ and $\mathcal{D}_i$ by removing all entries in the vectors except the first $i+1$ ones. Clearly, $\| \mathcal{D}'_{i+1} - \mathcal{D}'_i \|_1 = \| \mathcal{D}_{i+1} - \mathcal{D}_i \|_1 > \epsilon/m$. Also note that we can sample $\mathcal{D}'_i$ by sampling $\mathcal{D}'_{i+1}$ and replacing the last bit of the sample with an unbiased random bit.

Now, a predictor $f : \{0,1\}^i \to \{0,1\}$ is given as follows: Given input $y \in \{0,1\}^i$, the value 0 is predicted if for random $x \in \{0,1\}^{i+1}$ according to $\mathcal{D}'_{i+1}$, $\Pr[x_{i+1} = 0 | x_1 = y_1 \wedge x_2 = y_2 \wedge \ldots \wedge x_i = y_i]$ is bigger than $\frac{1}{2}$. Otherwise 1 is predicted. A calculation shows that the advantage of the predictor is exactly $\frac{1}{2} \| \mathcal{D}'_{i+1} - \mathcal{D}'_i \|_1$. ∎

Having introduced the notion of closeness of distributions, we are ready to define an extractor.

Definition 3 *A bipartite (multi-)graph on vertex sets U and V and with edges $E \subseteq U \times V$ so that every vertex x in U has exactly d neighbors is called an* extractor *with* error ϵ *and* min-entropy threshold t *if the following holds: If $x \in U$ is chosen according to a any distribution with min-entropy at least t, and a vertex $y \in V$ is then chosen uniformly at random from the neighbors of x, the resulting distribution of $y \in V$ is ϵ-close to uniform.*

The right way to think of an extractor is as a *randomness-refiner*, taking as input a weak random input (i.e., $x \in U$) and a short uniformly random *seed* (i.e., a random outgoing edge from x) and outputs a value, nearly uniformly distributed (i.e., y). The extractor *extracts* the min-entropy from the source U, using $\log d$ truly random bits, and producing

$\log |V|$ bits of pure randomness. We shall often describe an extractor as a *map* $E : U \times \{1, 2, \ldots, d\} \to V$ instead of a bipartite graph (with the natural correspondence between maps and graphs in mind).

It is easy to see that extractors are also dispersers.

Proposition 2 *An extractor with error strictly less than ϵ and min-entropy threshold t is also a $\frac{\epsilon}{2}$-hitting disperser with threshold 2^t.*

Furthermore, extractors are also majority dispersers. This means that they solve the randomness-efficient amplification and weak random source problems for two-sided error algorithms.

Lemma 22 *If there is an explicit family of extractors $E_r \subseteq U_r \times V_r$, $r = 1, 2, \ldots$ with $|V_r| = 2^r$, $|U_r| = 2^{R(r)}$, degree $d(r)$, min-entropy threshold $t(r)$, and error less than $\frac{1}{3}$, then a two-sided error algorithm running in time T, using r random bits to achieve a error probability of $\frac{1}{3}$ can be converted to a two-sided error algorithm, running in time $(R(r)+d(r)+T)^{O(1)}$ using $R(r)$ random bits to achieve error probability $2^{t(r)-R(r)}$.*

Proof The simulating algorithm uses its $R(r)$ bit input as an index to a node $x \in U_r$. The neighbors of x in V_n is $y_1, y_2, \ldots, y_{d(r)} \in \{0,1\}^r$. The simulating algorithm simulates the original algorithm on coin toss sequences $y_1, y_2, \ldots, y_r$ and accepts if the majority of these computations accept. We now show that this has the desired probability of error. Suppose without loss of generality that the right behavior is "accept", i.e., that at least a $\frac{2}{3}$ fraction of V_r corresponds to accepting computations of the original algorithm. We need to show that the error probability is less than $2^{t(r)-R(r)}$, i.e., that less than $2^{t(r)}$ choices of x causes reject. Suppose that the set of choices of x leading to reject has size at least $2^{t(r)}$. Then the uniform distribution on these choices has min-entropy at least $t(r)$, and thus picking a random one of the choices x and a random neighbor y of x is $\frac{1}{3}$- close to the uniform distribution on V_r. But now consider the statistical test $T \subset V_r$ consisting of all accepting coin toss sequences of the original algorithm. The probability that a uniform sample from V_r is in T is at least $\frac{2}{3}$. Thus, by Lemma 18 the probability that y generated as above is in T is at more than $\frac{1}{2}$. But this is a contradiction, as for *each* of the choices of x leading to reject, the majority of neighbors are not in T. ■

By a similar proof, we have the corresponding result for computing with weak random sources.

Lemma 23 *If there is an explicit family of extractors $E_r \subseteq U_r \times V_r$, $r = 1, 2, \ldots$ with $|V_r| = 2^r$, $|U_r| = 2^{R(r)}$, degree $d(r)$, min-entropy threshold*

$t(r)$, and error less than $\frac{1}{3}$, then a two-sided error algorithm running in time T, using r random bits to achieve an error probability $\frac{1}{3}$ can be simulated by an algorithm running in time $(R(r) + d(r) + T)^{O(1)}$ and using any $R(r)$-bit weak random source with min-entropy at least $t(r)+k$ to achieve an error probability of 2^{-k}.

3.3 NOTES

The notion of dispersers and majority disperser and the observation that they capture the problems of randomness efficient amplification and computing with weak random sources is due to Cohen and Wigderson, 1989. Even earlier, Sipser, 1988, had used a disperser graph for randomness-efficient amplification, without giving his graph a name. The stronger notion of extractors and the realization of their importance is due to Nisan and Zuckerman, 1996, though extractors were implicitly used in constructions beginning with Impagliazzo et al., 1989.

We have shown that dispersers can be used for randomness efficient amplification for one-sided error algorithm and extractors for randomness efficient amplification for two-sided error algorithms. A clever construction by Andreev et al., 1999 shows that dispersers can be used for randomness efficient amplification for two-sided error algorithms as well.

4. CONSTRUCTIONS OF DISPERSERS AND EXTRACTORS

There are essentially three known ways of constructing explicit dispersers and extractors. The first is based on explicit expander constructions. The second is based on iterated universal hashing. The third is based on error correcting codes and Nisan-Wigderson designs. We shall describe all three ways in the next three subsections, but only the technique using error correcting codes and Nisan-Wigderson designs will be described in full detail. This technique was only discovered recently, by Trevisan, 1999. It is simpler than the other two techniques, and for some important settings of the parameters it is the only known technique that will work.

4.1 EXPANDER BASED DISPERSERS

The explicit expander constructions of Margulis, 1973, Gabber and Galil, 1981, and Lubotzky et al., 1986, form an important way of building explicit dispersers and extractors. Historically, it was also the first way.

Let $G = (V, E)$ be an undirected d-regular graph. We say that G is a c-expander, for $c > 0$, if for any subset V_1 of V with $|V_1| \leq |V|/2$, we have $|V_1 \cup \Gamma(V_1)| \geq (1+c)|V_1|$, where $\Gamma(V_1)$ are vertices adjacent to some vertex in V_1.

In the papers mentioned above, families of explicit expanders with $|V| \to \infty$ and the parameters $c > 0$ and d being constants were constructed. The definitions of the expanders are generally not very involved. Particularly easy to describe is the 5-regular Gabber-Galil expander (V, E) with size $V = 2m^2$, for any desired integer m. Identifying V with $\{0,1\} \times (\mathbf{Z}/m\mathbf{Z})^2$, each vertex (b, x, y) has edges to $(1-b, x, y)$, $(1-b, x, x+y)$, $(1-b, x, x+y-1)$, $(1-b, x+y, y)$ and $(1-b, x+y+1, y)$. The graph is a c-expander for a small constant $c > 0$.

The proofs that the graphs defined are in fact expanders are, on the other hand, very involved, generally proceeding as follows. First, one appeals to a correspondence between the expansion properties of a graph and the second largest eigenvalue of its adjacency matrix discovered and described by Tanner, 1984, Alon and Milman, 1985, and Alon, 1986, and then one analyzes the eigenvalues of the adjacency matrix of the graph described above using various tools, such as Fourier analysis and number theory. The details are most certainly beyond the scope of this chapter!

The definition of an expander has the flavor: *Any set of vertices has "many" neighbors.* Note that the definition of a disperser has the same flavor (though with a different interpretation of "many" and a restriction of "any"), so it is not too surprising that we can use expanders to construct dispersers.

There are two known ways of constructing a disperser (U, V, E) from an expander $G = (V', E')$.

The first construction, due to Karp et al., 1986, is direct and very easy (once the expansion property of the graph is known). We let $U = V = V'$ and, for some parameter $l \geq 1$, we let $E = \{(u, v)|$ there is a path of length at most l from u to v in $G\}$. It follows directly from the definitions that, for any $\epsilon > 0$, (U, V, E) is an $(\frac{1}{2}+\epsilon)$-hitting disperser with threshold $\frac{1}{2}(1+c)^{-l}$ and degree at most d^{l+1}

Using this construction on, for instance, the Gabber-Galil expanders, we get the following theorem.

Theorem 3 *For any desired constant k, there is an explicit $\frac{1}{2}$-hitting disperser $E_r \subset V_r \times V_r$ with $|V_r| = 2^r$, degree $r^{O(1)}$ and threshold $2^r/r^k$.*

Actually, when proving the theorem, we can only use the Gabber-Galil expander if r is odd, as we then have that 2^r is of the form $2m^2$. However, we can always reduce the number of random bits used by an additive constant, by trying all possibible settings of the bits removed. Combining Theorem 3 with Lemma 16, we get Statement 1 of the introduction (for one-sided error). A slightly more involved argument gives majority dispersers and extractors with similar parameters, establishing Statement 1 for two-sided error algorithms. We refer to Cohen and Wigderson, 1989, Alon and Spencer, 1992, and Goldreich and Wigderson, 1994 for details. In particular, the extracting property of the power of an expander graph follows from the *expander mixing lemma* appearing in Goldreich and Wigderson, 1994 as Lemma 2.2.

The second construction gives us the possibility of varying the size of U. For each of the vertices of V', we label the outgoing edges $1, 2, \ldots, d$ in an arbitrary way. Thus, each edge gets labelled twice, once for each "direction", but the two labels do not have to agree. Then, for some parameter l, we let $U = V' \times \{1, \ldots, d\}^l$ and we let $V = V'$. Now we let the edges of the disperser be $E = \{((u, e_1, e_2, \ldots, e_l), v)|$ The walk starting in u and following edges labelled $e_1, e_2, \ldots, e_l$ (in that order) encounters v at some point. $\}$.

Thus, the disperser defined has $|U| = |V|d^l$ and degree l. It was shown by Cohen and Wigderson, 1989, and Impagliazzo and Zuckerman, 1989, (based on earlier work of Ajtai et al., 1987), that it is a $\frac{1}{2}$-hitting disperser with threshold $2^{-\Omega(l)}|U|$, and a majority disperser with similar parameters. This gives us Statement 2(a) of the introduction.

Unlike the simpler way of using expanders to construct dispersers, there is no known proof of these results based directly on the expansion property of the graph. The only known proofs uses again the relationship between the expansion of a graph and the second largest eigenvalue of its adjacency matrix. Once this relationship is known, the proof is clever, but not difficult. A good exposition can be found in the text of Alon and Spencer, 1992.

4.2 CONSTRUCTIONS BASED ON UNIVERSAL HASHING

The second way of constructing dispersers and extractors is using universal families of hash functions. We first describe a simple disperser construction of Chor and Goldreich, 1989.

Given parameters $V = \{1, \ldots, m\}$ and $\epsilon > 0$. Let D be a set of size more than $\lceil 1/\epsilon \rceil$. Let $\mathcal{H}$ be an explicit strongly universal family of hash functions mapping D to V with $|\mathcal{H}| = (|D||V|)^{O(1)}$. We let $U = \mathcal{H}$ and let E be given by $\{(h, h(x))|h \in U, x \in D\}$. Thus, each vertex of U corresponds to a hash function and each outgoing edge from such a vertex corresponds to applying this hash function to a particular value.

We shall show that this is a $\frac{1}{2}$-hitting disperser with threshold ϵ. This, combined with Lemma 16 means that it can be used to reduce the error probability of any one-sided error randomized algorithm using r bits to achieve error probability $\frac{1}{2}$ to at most $1/r^k$, for any constant k with only a polynomial blow up in the running time and using only a constant factor more random bits. We saw in the last subsection that we can use expanders to achieve the same error probability using *no* extra random bits, but the construction here is much more elementary.

Theorem 4 *For any value of $|V|$ and any $\epsilon > 0$, the above is an explicit $\frac{1}{2}$-hitting disperser with $|U| = (|V|/\epsilon)^{O(1)}$, degree $O(1/\epsilon)$, and threshold $\epsilon|U|$.*

Proof We need to show that for any subset S of U of size at least $\epsilon|U|$, we have that $|\Gamma(S)|$ is greater than $|V|/2$. Equivalently, we must show that if we take any subset V' of V of size at least $|V|/2$ and a randomly chosen vertex h of U, the probability that there is an edge between h and V' is more than $1 - \epsilon$.

For fixed V' and $i \in D$, define a random variable Y_i which is 1 if $h(i) \in V'$. Thus, there is an edge between h and V' if and only if $\sum_{i \in D} Y_i > 0$. The expectation of Y_i is $\mathrm{E}[Y_i] = |V'|/|V| \geq \frac{1}{2}$. Thus the variance of Y_i is $\mathrm{V}[Y_i] = \mathrm{E}[Y_i](1 - \mathrm{E}[Y_i]) \leq 1/4$. By linearity of expectation, $\mathrm{E}[\sum_i Y_i] \geq |D|/2$ and as the variables Y_i are pairwise independent, we have $\mathrm{V}[\sum Y_i] = \sum \mathrm{V}[Y_i] \leq |D|/4$.

Thus, $\Pr[$ There is *not* an edge between h and V' $] = \Pr[\sum_{i \in D} Y_i = 0]$ $= \Pr[\sum_{i \in D} Y_i \leq \mathrm{E}[\sum_{i \in D} Y_i] - |D|/2] \leq (|D|/4)/(|D|/2)^2) = 1/|D| < \epsilon$,

where the first inequality is Chebyshev's, stating that if X is a random variable with expectation μ and variance σ^2, then for any real number $t > 0$, $\Pr[|X - \mu| \geq t] \leq \sigma^2/t^2$. ■

A similar argument establishes that the same construction yields a majority disperser. Thus, it can also be used as an amplifier for two-sided error algorithms.

In the disperser construction of Chor and Goldreich, each hash function in the strongly universal family corresponds to a vertex in U and the elements we apply them to corresponds to outgoing edges from U. By switching the two roles, we get an extractor, as stated in the *leftover hash lemma* by Impagliazzo et al., 1989.

Lemma 24 *Let $\mathcal{H}$ be a universal family of hash functions of size 2^n indexed by and identified with $\{0,1\}^n$, mapping $\{0,1\}^n$ to $\{0,1\}^l$. Let $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{n+l}$ be defined by $E(x,h) = (h(x), h)$. Then, for any a, E is an extractor with min-entropy threshold $l + a$ and error at most $2^{-a/2}$.*

Proof Let $x_1, x_2 \in \{0,1\}^n$ be independent samples from a source with min-entropy at least $l+a$ and let $h_1, h_2 \in \{0,1\}^s$ be independent uniform samples. By Lemma 20, we just need to show that the collision probability $\Pr[E(x_1,h_1) = E(x_2,h_2)]$ is very close to $1/2^{n+l}$. Note that $\Pr[E(x_1,h_1) = E(x_2,h_2)] = \Pr[h_1 = h_2 \wedge h_1(x_1) = h_1(x_2)] \le \Pr[h_1 = h_2 \wedge x_1 = x_2] + \Pr[h_1 = h_2 \wedge h_1(x_1) = h_1(x_2) | x_1 \neq x_2]$. As $\mathcal{H}$ is universal, the latter probability can be bounded from above by $2^{-n} \cdot 2^{-l}$, i.e., exactly $1/2^{n+l}$. Thus, we just need to bound the former, which is equal to $2^{-n}\Pr[x_1 = x_2] \le 2^{-n}2^{-(l+a)}$, as x_1, x_2 were from a source of min-entropy at least $l+a$. Thus, the collision probability is at most $(1+2^{-a})2^{-(n+l)}$, and, by Lemma 20, $E(x_1,h_1)$ is $2^{-a/2}$ close to uniform. ■

The lemma is called the leftover hash lemma, because h appears both in the input and the output of the extractor. Thus, h is "recycled".

The leftover hash lemma does not immediately define an extractor useful for deterministic amplification or computing with weak random sources because of the high degree: Viewing the extractor as a graph, each vertex in U has 2^n neighbors and the corresponding algorithm would thus run in exponential time.

However, the extractor can be used as a component in the construction of dispersers and extractors with parameters that are directly useful for randomness-efficient amplification.

The most simple way of doing this was described by Impagliazzo and Zuckerman, 1989, who called their technique *recycling of random bits*. They showed how to use the extractor of the leftover hash lemma to get, given any size of V, a $\frac{1}{2}$-hitting disperser $E \subseteq U \times V$ with $|U| = |V|^{O(1)}$, and degree $(\log|V|)^{O(1)}$ with threshold $2^{-\Omega(\sqrt{\log|V|})}|U|$.

Thus, it can be used to convert any **RP** algorithm using r random bits to achieve an error probability of $\frac{1}{3}$ to an **RP** algorithm using $O(r)$ random bits to achieve an error probability of $2^{-\Omega(\sqrt{r})}$. Note that this is not as good as the expander based disperser of the last subsection, but a self-contained proof that it works is much easier (in particular, it can be given here).

Let $U = \{0,1\}^{3n}$ and $V = \{0,1\}^n$. Let $E : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^{2n-\sqrt{n}}$ be the extractor with error $2^{-(\sqrt{n}-1)/2}$ and min-entropy threshold $n-1$ of Lemma 24 (i.e., we set $l = n-\sqrt{n}$, $a = \sqrt{n}-1$).

The disperser D is defined as follows. Given $x \in U = \{0,1\}^{3n}$ split x into two strings x_1 and y_1 of length n, and $\sqrt{n}$ strings $s_1, s_2, \ldots, s_{\sqrt{n}}$, each of length $\sqrt{n}$. Then recursively define $(x_2,y_2), (x_3,y_3), \ldots (x_{\sqrt{n}}, y_{\sqrt{n}})$, with $x_i, y_i \in \{0,1\}^n$ by $(x_{i+1}, y_{i+1}) = E(x_i, y_i) \circ s_i$, where $\circ$ denotes concatenation. The neighbors of x in V should then be $x_1, x_2, \ldots, x_{\sqrt{n}}$.

The proof that this works is very typical of the way extractors are used in other constructions. The intuition is the following. We are trying to hit a set S, using the uniformly random, but not independent, elements $x_1, x_2, \ldots, x_{\sqrt{n}}$. Suppose we have already tried $x_1, x_2, \ldots, x_i$, without hitting S. Then the conditional distribution of x_i *given this* still contains a lot of min-entropy. Letting $(x_{i+1}, y_{i+1}) = E(x_i, y_i)$ makes the conditional distribution of x_{i+1}, given that we did not yet hit S, close to uniform, and thus our chance of hitting S with x_{i+1} is almost as good, as if x_{i+1} had been independent from $x_1, x_2, \ldots, x_i$.

In the formal proof, we have to be a bit more careful.

Theorem 5 *The graph defined above is a* $\frac{1}{2}$*-hitting disperser with threshold* $2^{-\Omega(\sqrt{n})}2^{3n}$.

Proof We should show that if S is a subset of V of density at least $\frac{1}{2}$, then for random x, the probability that $x_1, x_2, \ldots, x_{\sqrt{n}}$ does not intersect S at most $2^{-O(\sqrt{n})}$. We assume without loss of generality that the density of S is exactly $\frac{1}{2}$.

Let $b_1, b_2, \ldots, b_{\sqrt{n}}$ be indicator variables, with b_i indicating if $x_i \in S$.

Let $\mathcal{D}_i$ be the distribution on $\{0,1\}^{i+2n}$, induced by the variables $b_1, b_2, \ldots, b_i, x_{i+1}, y_{i+1}$. Let $\mathcal{U}_i$ be the uniform distribution on $\{0,1\}^{i+2n}$. Given any distribution $\mathcal{D}'$ on $\{0,1\}^{i-1+2n}$, let $\mathrm{extend}(\mathcal{D}')$ be the distribution on $\{0,1\}^{i+2n}$ obtained as follows: Take a sample $(j_1, j_2, \ldots, j_{i-1}, u_i, v_i)$ from $\mathcal{D}'$, let $j_i = 1_{u_i \in S}$ and $(u_{i+1}, v_{i+1}) = E(u_i, v_i) \circ s$, where s is a vector of $\sqrt{n}$ unbiased independent bits, and output the vector $(j_1, j_2, \ldots, j_i, u_{i+1}, v_{i+1})$. Thus, we have that $\mathcal{D}_i = \mathrm{extend}(\mathcal{D}_{i-1})$.

We'll show by induction that $\|\mathcal{U}_i - \mathcal{D}_i\|_1 \leq i2^{-\frac{\sqrt{n}-1}{2}}$.

Note that $\|\mathcal{U}_{\sqrt{n}} - \mathcal{D}_{\sqrt{n}}\|_1 \leq \sqrt{n}2^{-\frac{\sqrt{n}-1}{2}}$ implies the statement of the lemma. Indeed, by Lemma 18 we have

$\Pr[\forall i, x_i \notin S] \leq 2^{-\sqrt{n}} + \frac{1}{2}\sqrt{n}2^{-\frac{\sqrt{n}-1}{2}} = 2^{-\Omega(\sqrt{n})}$.

For $i = 0$, the statement is true, as $U_0 = D_0$.

Suppose that the statement holds for $i-1$, i.e. that the distribution of the bit vector $b_1, b_2, \ldots b_{i-1}, x_i, y_i$ is $(i-1)2^{-\frac{\sqrt{n}-1}{2}}$-close to the uniform distribution $\mathcal{U}_{i-1}$.

We will bound $\|\mathcal{D}_i - \mathcal{U}_i\|_1 = \|\mathrm{extend}(\mathcal{D}_{i-1}) - \mathcal{U}_i\|_1$ from above by $\|\mathrm{extend}(\mathcal{D}_{i-1}) - \mathrm{extend}(\mathcal{U}_{i-1})\|_1 + \|\mathrm{extend}(\mathcal{U}_{i-1}) - \mathcal{U}_i\|_1$.

By Lemma 19, we can bound $\|\mathrm{extend}(\mathcal{D}_{i-1}) - \mathrm{extend}(\mathcal{U}_{i-1})\|_1$ from above by $\|\mathcal{D}_{i-1} - \mathcal{U}_{i-1}\|_1$ which is at most $(i-1)2^{-\frac{\sqrt{n}-1}{2}}$ by the induction hypothesis. Thus we only have to bound $\|\mathrm{extend}(\mathcal{U}_{i-1}) - \mathcal{U}_i\|_1$.

Now, let $u_1, u_2, \ldots u_{i-1}, x_i', y_i'$ be a sample from $\mathcal{U}_{i-1}$. Letting $b_i' = 1_{x_i' \in S}$ and $(x_{i+1}', y_{i+1}') = E(x_i, y_i) \circ s_i$, we should prove that the distribution of $u_1, u_2, \ldots, u_{i-1}$ b'_i, x'_{i+1}, y'_{i+1} is close to uniform. The bits $u_1, u_2, \ldots, u_{i-1}$ are uniformly distributed and independent of what comes after. The i'th bit, b'_i is unbiased, as the density of S is $\frac{1}{2}$. Also, for any setting of $b'_i \in \{0,1\}$, the conditional distribution of x'_i *given* b'_i, loses at most 1 bit of min-entropy compared to the unconditional distribution of x'_i, as the conditional probability of any outcome, conditioned by an event of probability $\frac{1}{2}$ is at most a factor of two bigger than the unconditional probability. So, the conditional min-entropy of x'_i, given b'_i, for any outcome of b'_i, is at least $n-1$. By the extractor property of E, the conditional distribution of $(x'_{i+1}, y'_{i+1}) = E(x'_i, y'_i)$, given the value of b'_i, is $2^{-\frac{\sqrt{n}-1}{2}}$- close to uniform. We conclude that $(u_1, u_2, \ldots, u_{i-1}, b'_i, x'_{i+1}, y'_{i+1})$ is $2^{-\frac{\sqrt{n}-1}{2}}$-close to uniform.

Thus, $\|\mathcal{D}_i - \mathcal{U}_i\|_1 \leq (i-1)2^{-\frac{\sqrt{n}-1}{2}} + 2^{-\frac{\sqrt{n}-1}{2}} = i2^{-\frac{\sqrt{n}-1}{2}}$, and we are done. ■

The above constructions were relatively straightforward. Following them, in a series of papers, Zuckerman, 1990, Zuckerman, 1991, Nisan and Zuckerman, 1996, Wigderson and Zuckerman, 1999, Goldreich and Wigderson, 1994, Srinivasan and Zuckerman, 1999, Saks et al., 1998, Zuckerman, 1996, Ta-Shma, 1996, Ta-Shma, 1998, expanded and refined the ideas. It is beyond the scope of this chapter to give a detailed account here. An excellent survey of this line of research is given by Nisan and Ta-Shma, 1999. We merely state some of the important ingredients and then state the landmark results obtained. Many of the results can also be obtained using the technique using error correcting codes of the next subsection, but there are exceptions.

One important ingredient is the family of hash function used. First, note that the proof of the leftover hash lemma works, even if the family used is only $(1+\epsilon)$-weakly universal, for some small $\epsilon > 0$. Thus, we can use the family of Lemma 9 instead of the family based on arithmetic over a field. This won't do us any good in the construction of Impagliazzo and Zuckerman described above, but in other settings, where the parameters are different, it will.

Another important ingredient is the notion of *extractor composition* by Wigderson and Zuckerman, 1999: If E_1 and E_2 are extractors with certain parameters, then E defined by $E(x, y_1 \circ y_2) = E_1(x, y_1) \circ E_2(x, y_2)$ is also an extractor. The proof sketch of this is somewhat similar to the proof of Theorem 5: If x has high min-entropy, then $E_1(x, y_1)$ is close to uniform. Furthermore, if a is a particular outcome of $E_1(x, y_1)$ with high probability, then the min-entropy of x given that $E_1(x, y_1) \circ y_2 = a$ is still high, so even given this event, $E_2(x, y_2)$ has distribution close to uniform.

The error introduced by outcomes a of low probability is low, so we can safely deal with these outcomes and conclude that $E_1(x, y_1) \circ E_2(x, y_2)$ is close to uniform.

A third important ingredient is the notion of a *blockwise source.* A blockwise source is a source $(X_1, X_2, \ldots, X_k)$ on $S_1 \times S_2 \times \cdots \times S_k$, for some value k and some sets S_i, so that the conditional marginal distribution of X_i, conditioned on any particular outcome of $X_1, X_2, \ldots, X_{i-1}$, still has high min-entropy. Nisan and Zuckerman, 1996, show that by picking out subsets of bits from a weak random source on $\{0,1\}^R$ randomly, one can obtain a blockwise source. The bits can be taken out randomness-efficiently by using k-wise independent families of hash functions (similar to the use of pairwise independence in Theorem 4 - intuitively, one is trying to *hit* some *really* random bits in the weak random sample). Saks et al., 1998 describes a different way of obtaining a blockwise source, based on splitting the input in blocks in an "adaptive" way. This technique was also used by Ta-Shma, 1996 and Ta-Shma, 1998.

Having obtained a block-wise source $(X_1, X_2, \ldots, X_k)$, one can extract a value Y_0 with close to uniform distribution from it, in the following iterative way, using extractors of appropriate parameters: Given a truly random seed y_k, one lets $y_{k-1} = E(X_k, y_k)$, $y_{k-2} = E(X_{k-1}, y_{k-1})$, $\ldots$, $y_0 = E(X_1, y_1)$. Again, the proof that this works is similar to the proof of Theorem 5. It is, however, rather fragile: It is important that one does the construction "backwards", starting with X_k rather than X_1.

Finally, Ta-Shma, 1996, introduced the notion of a *merger*, an important component in the latest constructions: Loosely speaking, a merger is a deterministic algorithm which, given a number of sources, an unknown one of them uniform, outputs a nearly uniform distribution.

Best results obtained using these techniques include the following, the first by Zuckerman, 1996.

Theorem 6 *For any constants* $0 < \alpha < \delta < 1$*, there is an explicit extractor* E *for all parameters* $n \geq 1, \epsilon > 0$*, with*

$$E : \{0,1\}^n \times \{0,1\}^{O(\log n + \log \epsilon^{-1})} \rightarrow \{0,1\}^{(\delta - \alpha)n},$$

error ϵ*, and min-entropy threshold* δn*.*

The theorem of Zuckerman, combined with Lemma 22 implies Statement 2(b) of the introduction.

The second by Ta-Shma, 1996.

Theorem 7 *There is an explicit extractor* E *for all parameters* n, m, ϵ *with* $E : \{0,1\}^n \times \{0,1\}^{(\log n + \log \epsilon^{-1})^{O(1)}} \rightarrow \{0,1\}^m$*, error* ϵ*, and min-entropy threshold* m*.*

Note that Ta-Shma's extractor extracts *all* the min-entropy from the source: The min-entropy threshold equals the number bits of the output domain. On the other hand, the degree of the extractor is $2^{\mathrm{polylog} n}$, so using the extractor for deterministic amplification would induce a superpolynomial slowdown.

The third by Ta-Shma, 1998.

Theorem 8 *There is an explicit ϵ-hitting disperser D for all parameters n, m, ϵ with $D : \{0,1\}^n \times \{0,1\}^{O(\log n + \log \epsilon^{-1})} \to \{0,1\}^{m - \mathrm{polylog} n}$, with threshold 2^m.*

Combined with Lemma 16, this theorem implies Statement 4 of the introduction.

Missing from the above theorems is an extractor, capable of extracting randomness from a source of sublinear min-entropy, using a seed of logarithmic length. It is not known how to obtain such an extractor using iterated universal hashing, but it can be obtained using Trevisan's technique of error correcting codes, to be explained in the next subsection.

4.3 THE TREVISAN EXTRACTOR

In this subsection, we shall give a detailed account of the extractor of Trevisan, 1999, proving the following theorem.

Theorem 9 *For any constants $k > c > 1$ there is an explicit extractor for any parameter r with $E : \{0,1\}^{r^k} \times \{0,1\}^{O(\log r + \log \epsilon^{-1})} \to \{0,1\}^r$, with error ϵ and min-entropy threshold r^c.*

For the presentation, we shall be content with proving the theorem for $c = 3$, rather than c arbitrarily close to one, but with k being as large as desired. We shall also deal with constant error $\epsilon = \frac{1}{10}$ only. Still, combining the theorem thus proved with Lemma 22, we get Statement 3 of the introduction, and combining it with Lemma 23, we get Statement 5 of the introduction. Also, this version of the theorem is enough for the important applications to pseudorandom generators we describe later.

Trevisan's extractor construction was a major breakthrough. Curiously, the entire construction was in some sense already known when Trevisan found it, as a *hardness based pseudorandom generator* with similar parameters was already constructed by Impagliazzo and Wigderson, 1997. In Section 5, we introduce this concept and show that it is a special case of an extractor. Nobody realized this, however, until Trevisan's work. Trevisan essentially took the generator of Impagliazzo and Wigderson, 1997, and threw away what was not needed to keep the extractor property.

The construction has two parts: The use of error correcting codes and the use of Nisan-Wigderson designs. We explain each component separately.

Min-entropy and source coding. To prove that a map $E : U \times A \to V$ is an extractor, we have to prove that if we pick $x \in U$ distributed according to $\mathcal{D}$, where $\mathcal{D}$ is a distribution of high min-entropy and $y \in A$ is uniformly distributed, then $E(x, y)$ has a distribution which is close to uniform. If we make such a proof by way of contradiction, we would assume that $E(x, y)$ is *not* close to uniform and then prove that $\mathcal{D}$ does not have high min-entropy. Thus, it is useful to find strategies for proving that a distribution does not have high entropy. Concepts from *source coding* turns out to be useful for this.

A *lossless source code* for $\{0,1\}^m$ is a pair of maps, the *encoder* $c : \{0,1\}^m \to \{0,1\}^*$ and the *decoder* $d : \{0,1\}^* \to \{0,1\}^m$ so that $\forall x \in \{0,1\}^m, d(c(x)) = x$. The set $\{c(x)|x \in \{0,1\}^m\}$ is the set of *code words*. A *prefix* code is a lossless source code with the extra requirement that no code word is a prefix of another code word. A *fixed length* code has the extra requirement that all code words have the same length.

Let $\mathcal{D}$ be a distribution on $\{0,1\}^m$. Shannon's lossless source coding theorem states that any prefix code $c : \{0,1\}^m \to \{0,1\}^*$ must have $E_{x\in\mathcal{D}}(|c(x)|) \geq H(\mathcal{D})$. Thus we can prove that a distribution does not have high *Shannon* entropy by providing a source code for the distribution with small average code length.

We can attempt to adopt a similar strategy to prove that a distribution does not have high min-entropy using the following simple observation: If a distribution $\mathcal{D}$ has min-entropy t, then any *fixed length* code for $\mathcal{D}$ must have code length at least t. Thus we can prove that a distribution does not have high min-entropy by providing a source code for the distribution with small worst case code length. While valid, this strategy is not generally useful as it can only be used in very special cases.

To get a more useful strategy, we have to consider *lossy source codes*. A *lossy* fixed length source code with *rate l* is a pair of maps, the encoder $c : \{0,1\}^m \to \{0,1\}^l$ and the decoder $d : \{0,1\}^l \to \{0,1\}^m$. Now there is no requirement that $\forall x \in \{0,1\}^m d(c(x)) = x$. Instead, we refer to $d(c(x))$ as the *quantized* value of x and, given a distribution $\mathcal{D}$ on $\{0,1\}^m$, we define the *distortion* of the source code relative to $\mathcal{D}$ as the probability $\Pr[d(c(x))_i \neq x_i]$ where x is chosen randomly according to $\mathcal{D}$ and i is chosen uniformly at random in $\{1, \ldots, m\}$.

We cannot directly relate the min-entropy of a distribution to the rate-distortion performance of a lossy fixed length code for the distribution, as some distributions with high entropy or high min-entropy admit lossy codes with low rate and distortion (indeed, industrial lossy data compression, such as the JPEG standard, is based on this fact). However, if we first encode our distribution in a good error correcting code, and consider source codes for the encoded distribution, high min-entropy of the original distribution (and thus for the encoded distribution) *does* imply that we cannot have low rate and distortion simultaneously, *even if we by "low distortion" mean a distortion just below the trivially obtainable distortion level of* $\frac{1}{2}$. This simple but crucial fact is reflected in the following lemma.

Lemma 25 *Let* $e : \{0,1\}^m \to \{0,1\}^{m^{10}}$ *be the error correcting code of Lemma 12. Let* $\mathcal{D}$ *be a distribution on* $\{0,1\}^m$ *with min-entropy at least* t *and let* $e(\mathcal{D})$ *be the induced distribution on* $\{0,1\}^{m^{10}}$. *Then any lossy fixed length source code for* $\{0,1\}^{m^{10}}$ *with distortion less than* $\frac{1}{2} - \frac{2}{m^2}$ *relative to* $e(\mathcal{D})$, *has rate at least* $t - 6\log m$.

Proof Let the source encoder be c and the decoder d. Consider the event G that a randomly chosen sample y of $e(\mathcal{D})$ has Hamming distance smaller than $\frac{1}{2} - \frac{1}{m^2}$ to its quantized value $d(c(y))$. We have that $\Pr[y \in G] \geq \frac{1}{m^2}$, for otherwise the distortion of the source code would be more than $(1 - \frac{1}{m^2})(\frac{1}{2} - \frac{1}{m^2}) > \frac{1}{2} - \frac{2}{m^2}$.

Assume that the rate of the source code is l. Thus, there are at most 2^l source code words. Let w be the most popular value of $c(y)$, given that $y \in G$. Let G' be the event $y \in G \wedge c(y) = w$. For a randomly chosen y of $e(\mathcal{D})$ we have that $\Pr[y \in G'] \geq \frac{1}{2^l m^2}$.

On the other hand, all $y \in G$ are e-code words and if $c(y) = w$, y is in the Hamming ball of radius $\frac{1}{2} - \frac{1}{m^2}$ with center $d(w)$. Thus, by Lemma 14, there are at most $m^4/2$ outcomes of G', and since $\mathcal{D}$ and thus $e(\mathcal{D})$ has min-entropy t, each outcome has probability at most 2^{-t}. Thus $\Pr[G'] \leq \frac{m^4}{2} 2^{-t}$.

Combining the two inequalities, we have $\frac{1}{2^l m^2} \leq \frac{m^4}{2} 2^{-t}$, so $l \geq t - 6\log m$. ■

In the extractor construction to follow, the first step of the extractor (viewed as a function) will be to apply an error correcting code to its first input, i.e. the output of the weak random source. Then another transformation, to be outlined next will follow. We will prove the extractor construction correct by showing that if a distribution does *not* yield a distribution ϵ-close to uniform when sent through the extractor, we can make a low rate lossy source code with distortion bounded away from $\frac{1}{2}$ for the encoded distribution.

Nisan-Wigderson designs. A Nisan-Wigderson design, introduced in Nisan and Wigderson, 1994, is a set system with small pairwise intersections. More precisely, an (m, n, s, l)-Nisan-Wigderson design is a system of m sets $S_1, S_2, \ldots, S_m$, each an s-subset of $\{1, 2, \ldots, n\}$, so that for $i \neq j, |S_i \cap S_j| \leq l$.

Explicit Nisan-Wigderson designs can be constructed using weakly universal families of hash functions.

Lemma 26 *For any parameters m, n which are powers of two, there is an explicit Nisan-Wigderson design with parameters (n^m, n^2, n, m).*

Proof Let $\mathcal{H}$ be the explicit m-weakly universal family of Lemma 8 of size n mapping $\{1, 2, \ldots, n\}^m \to \{1, 2, \ldots, n\}$, and let the system consist of the sets $S_x = \{(h, h(x)) | h \in \mathcal{H}\}, x \in \{1, 2, \ldots, n\}^m$. ∎

This construction is the best explicit one known, if we define an explicit set system to be a set system where the corresponding bipartite graph is explicit. Unfortunately, it does not quite give us the parameters necessary for the application at hand. Fortunately, we do not really need explicit designs, but only polynomial time constructible ones.

Lemma 27 *For any constant $c \geq 1$, there is a deterministic algorithm which, given as input m, outputs an $(m, O(\log m), c \log m, \log m)$-Nisan-Wigderson design, using time polynomial in m.*

Proof The algorithm constructs the set system on a domain of size $100c^2 \log m$. The system is constructed in a "greedy" way by picking one set at a time. Suppose we have already picked $S_1, S_2, \ldots, S_i$ of the correct size and with the correct bound on the size of the pairwise intersections. If we can prove that there *exists* a set S_{i+1} we can add to the system without violating the constraints, we are done, as the algorithm can exhaustively search for such a set in time polynomial in m, since $\binom{100c^2 \log m}{c \log m} = m^{O(1)}$, as c is constant.

To show the existence of an appropriate set S_{i+1}, we pick randomly, with replacement, $2c \log m$ members of $1, 2, \ldots, 100c^2 \log m$. Let the resulting multi-set be denoted S. Lemma 1 implies that

1. With very high probability, the multi-set S has at least $c \log m$ different members.
2. With very high probability, the size of the intersection of S with any of the previously chosen sets, is less than $\log m$.

Thus, the desired set S_{i+1} can be obtained by taking an appropriate subset of S. ∎

A design constructed using weakly universal families would have domain size $O((\log m)^2)$, rather than $O(\log m)$.

Putting things together. The Trevisan extractor is most conveniently described as an efficiently computable map $E : U \times A \to V$ with $U = \{0,1\}^{r^k}$, $V = \{0,1\}^r$, and $A = \{0,1\}^{c \log r}$. Here, k is any constant greater than 5 and c is a constant dependent on k. For sufficiently large r, we shall prove the construction to be an extractor with error at most $\frac{1}{10}$ and min-entropy threshold at least r^3, thus proving the special case of Theorem 9 outlined below the statement of the theorem.

We describe the operation of E, viewed as an algorithm. First, given the parameter r, the extractor uses Lemma 27 to construct an $(r, c\log r, 10k\log r, \log r)$-Nisan-Wigderson design $S_1, S_2, \dots, S_r$, where c is a constant depending on k. This constant c is also the constant c in the definition of the set of seeds A.

Now, given the first input $x \in U$, the extractor computes the value $e(x) \in \{0,1\}^{r^{10k}}$. For notational convenience, we represent the sequence $e(x)$ by its characteristic function $g : \{0,1\}^{10k\log r} \to \{0,1\}$.

Then, given the second input $y \in A$, the final output $E(x, y)$ is computed as follows. Each set in the Nisan-Wigderson design is a subset of $\{1, \dots, c\log r\}$. View y as a bit sequence of length $c\log r$, and for S being a set in the design, let y_S be the subsequence consisting of the bits indexed by S. For example, if $v = 10111001011$ and $S = \{2, 4, 8\}$, $y_S = 011$. Now, we give as output $E(x, y) = g(y_{S_1})g(y_{S_2})\dots g(y_{S_r})$.

We claim that the map E so defined is an extractor with min-entropy threshold r^3 and error at most $\frac{1}{10}$.

To see this, let $\mathcal{D}$ be some distribution on $U = \{0,1\}^{r^k}$ with min-entropy at least r^3. Pick $x \in \mathcal{D}$ at random and $y \in A$ uniformly at random. We must show that $E(x, y) \in \{0,1\}^r$ has a distribution which is $\frac{1}{10}$-close to the uniform distribution on $\{0,1\}^r$. Suppose not. Then, by Lemma 21, there is a predictor with advantage $\epsilon = \frac{1}{20(r+1)}$ for one of the bits of $E(x, y)$, say the i'th bit, given the previous bits. We will use this predictor to construct a lossy source code for $c(x)$ with a better distortion than possible by Lemma 25, and thus reach a contradiction. The i'th bit of $E(x, y)$ is $g(y_{S_i})$. Our predictor predicts $g(y_{S_i})$ on input $g(y_{S_1}), g(y_{S_2}), \dots, g(y_{S_{i-1}})$ with advantage ϵ when x is chosen from $\mathcal{D}$ and y is chosen uniformly at random.

It does not seem immediately useful to be able to predict $g(y_{S_i})$ on input $g(y_{S_1}), g(y_{S_2}), \dots, g(y_{S_{i-1}})$. Suppose, on the other hand, that we could predict $g(y_{S_i})$ from y_{S_i} with a positive advantage. Then we could reconstruct the entire vector $e(x)$ (for which g is the characteristic function) by cycling through all possible settings of y_{S_i}. The advantage would give us a reconstruction of $e(x)$ with a distortion bounded away from $\frac{1}{2}$.

To accomplish this, we *freeze* the bits of $y_{S_i^c}$ (i.e., all bits except the bits of y_{S_i}) to fixed values while preserving the advantage of our predictor for a random setting of y_{S_i}. Since our predictor predicts $g(y_{S_i})$ from $g(y_{S_1}), g(y_{S_2}), \ldots, g(y_{S_{i-1}})$ rather than y_{S_i}, our strategy will be to *compute* $g(y_{S_1}), \ldots, g(y_{S_{i-1}})$ given y_{S_i}. As $S_1 \cap S_i$ only contains $\log r$ bit positions and the bits of $y_{S_1 \setminus S_i}$ have been frozen, then for any fixed x (and thus fixed g), we can *tabulate* the value of $g(y_{S_1})$ as a function of $y_{S_1 \cap S_i}$. The table can be represented as a bit string of $2^{\log r} = r$ bits. Similarly, we can tabulate the value of $g(y_{S_j})$ as a function of $y_{S_j \cap S_i}$ for each $j = 2, \ldots, i-1$.

The value i in binary representation and the concatenation of these $i-1$ tables defines our source code word $c(e(x))$ for the value $e(x)$. We pad the code word with zeros to make it a fixed length code. The rate of the source code so defined is $r^2 + O(\log r)$.

To decode a code word $y = c(e(x))$ (without knowing $e(x)$), we do the following. We cycle through all possible values of y_{S_i}, and use the tables of the code word to look up the corresponding values of $g(y_{S_1}), \ldots, g(y_{S_{i-1}})$. These we feed into the predictor, to get a prediction for $g(y_{S_i})$, i.e., for the bit of $e(x)$ indexed by y_{S_i}. Concatenating all the bits predicted gives us $d(y) = d(c(e(x))$. As the predictor has advantage ϵ, we see that for random $x \in \mathcal{D}$ and random i, the probability that $d(c(e(x))_i \neq e(x)_i$ is at most $\frac{1}{2} - \epsilon$, i.e. the distortion of the source code we have defined is at most $\frac{1}{2} - \epsilon = \frac{1}{2} - \frac{1}{20(r+1)}$.

By Lemma 25, the rate of the source code must be at least $r^3 - O(\log r)$. This is a contradiction, as we already established the rate to be $r^2 + O(\log r)$. Thus, our assumption that the distribution of $E(x, y)$ is not ϵ-close to uniform must be false, and we are done.

4.4 TOWARDS OPTIMAL EXTRACTORS

The extractor construction technique of Trevisan was further refined by Raz et al., 1999b, Raz et al., 1999a, Impagliazzo et al., 1999, and Impagliazzo et al., 2000.

Some of the ingredients in the improvements of Raz et al., 1999b are the following:

First, a weakening of the notion of Nisan-Wigderson design is introduced. A *weak* Nisan-Wigderson designs is defined similarly to a Nisan-Wigderson design, except that, for each set S in the design, one demands a bound on $\sum_T 2^{|S \cap T|}$ rather than $\max_T |S \cap T|$, where the sum and max are over the other sets in the design. As is apparent in the proof of the correctness of the Trevisan extractor, it is enough to bound the first quantity. This makes better parameters possible.

Second, by using a special error correcting code, rather than a "generic" error correcting code, Raz et al., 1999b showed that the tabulation of the functions $g(S_j)$ used in the source code can be replaced with a more succinct representation, making the source code shorter.

Third, an improved version of extractor composition is introduced, where two extractors E_1, E_2 are composed as $E(x, y_1 \circ y_2) = E_1(x, y_1) \circ E_2(x \circ y_1, y_2)$ rather than $E(x, y_1 \circ y_2) = E_1(x, y_1) \circ E_2(x, y_2)$, as done by Wigderson and Zuckerman, 1999.

These improvement makes it possible, for instance, to match the parameters of the extractor of Theorem 7.

To further describe the improvement obtained, one should first note that in the extractor constructions stated so far, we have only considered rough tradeoffs between the five parameters (U, V, the degree d, the error ϵ, and the min-entropy threshold). When looking at the finer tradeoffs possible, for extractors extracting all or almost all the min-entropy of a source, the notion of *entropy loss* of the extractor becomes relevant. The entropy loss of an extractor is the difference between the number of almost unbiased bits produced by the extractor and the sum of its min-entropy threshold and the number of truly random bits given as input, i.e. $\log d$. Radhakrishnan and Ta-Shma, 1997, prove that the entropy loss of any extractor must be at least $\log \epsilon^{-1}$. Unlike any previous extractors, an extractor of Raz et al., 1999b matches this bound within an additive constant, with d being quasipolynomial.

Further improvements and refinements were made in Raz et al., 1999a, Impagliazzo et al., 1999, and Impagliazzo et al., 2000. It is beyond the scope of this chapter to discuss the various tradeoffs obtained. Also, presumably any information given would quickly be obsolete, as the possible combinations of the available techniques do not seem to have been fully explored yet.

We shall here just mention that it is, at the time of writing, not known how to match the parameters of Theorem 6 or Theorem 8 using error correcting codes and designs.

The ultimate goal and the most important open problem is to construct, for parameters $r < R$ and $\epsilon > 0$ an explicit extractor $E : \{0,1\}^R \times \{0,1\}^{O(\log R + \log \epsilon^{-1})} \to \{0,1\}^r$ with error ϵ and min-entropy threshold r. Such an extractor would have as consequence that for any parameter $R > r$, a randomized polynomial algorithm using r random bits could be simulated with polynomial overhead, using R random bits, and achieving an error probability of 2^{-R+r}. Note that Theorem 7 (or the alternative construction given by Raz et al., 1999b) makes such a simulation possible if we allow a quasipolynomial, rather than polynomial, overhead in time.

5. HITTING SET AND PSEUDORANDOM GENERATORS

For the rest of this chapter, we shall deal with the third derandomization task: Simulating randomized algorithms deterministically with a non-trivial overhead in time. As mentioned earlier, it is, at the time of writing, an open problem if **BPP**=**EXP**. Thus, we can only hope for conditional results, such as reasonable conditions implying **BPP**=**P**, or weaker statements such as **BPP** $\subseteq$ **SUBEXP**.

All known proofs of such implications proceed by the construction of either *hitting set generators* or *pseudorandom generators*.

To motivate these concepts, let us go back to the card game between Alice and Bob of Section 3.1, used to define dispersers, and try to make it match the problem of deterministic simulation. As in the case of randomness-efficient amplification, each card of Alice corresponds to the coin toss sequence in a particular computation of a one-sided error algorithm with error probability bounded by ϵ. The marked cards correspond to accepting computations. To efficiently derandomize the randomized algorithm using the card game, it seems that we would need a *deterministic* efficient strategy for Bob, i.e., a small set of cards to turn over, with a guarantee that no matter how Alice marked her cards, one of the marked cards would be turned over. But as we pointed out when we first discussed the card game, such strategies do not exist, and we seem to be stuck.

The key observation that makes progress possible is the fact that for the intended application, the set of marked cards is *not* an arbitrary set of the given size: *it is the set of coin toss sequences that make an efficient randomized algorithm accept a certain input.* Thus, we should consider a modified card game, where Alice must choose her set of marked cards from a quite restricted class $\mathcal{C}$ of possibilities. This changes the game completely! Now it is very possible that a good deterministic strategy for Bob exists. Such a strategy we call a *hitting set* for $\mathcal{C}$.

Definition 4 *Let $\mathcal{C}$ be a set system over a finite universe. A $(1-\epsilon)$-hitting set for $\mathcal{C}$ is a set H so that $H \cap S \neq \emptyset$ for every $S \in \mathcal{C}$ of density at least $1-\epsilon$.*

To see which class $\mathcal{C}$ to consider for the task at hand, it is easiest to use Lemma 2: To derandomize **RP**, it is sufficient to be able to find satisfying assignments to circuits $C : \{0,1\}^r \rightarrow \{0,1\}$ with acceptance probability at least $\frac{1}{2}$. Thus, we should define $\mathcal{C}$ to be the family of subsets of $\{0,1\}^r$ recognized by circuits of size r and consider $\frac{1}{2}$-hitting sets for $\mathcal{C}$. Abusing terminology slightly, we shall refer to such a hitting set as a $\frac{1}{2}$-hitting set "for circuits of size r" rather than as a $\frac{1}{2}$-hitting set "for the sets decided by circuits of size r". To see if an instance of CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION satisfies at least half its assignments, we simply check if one of the elements of the hitting set satisfies the circuit. For this to be an efficient deterministic procedure, we need to be able to efficiently generate, on input r a small hitting set for circuits of size r. This leads us to the notion of a *hitting set generator.*

Definition 5 *A $(1-\epsilon)$-hitting set generator for $\mathcal{C}$ is a deterministic algorithm which on input r outputs a $(1-\epsilon)$-hitting set $H_r \subseteq \{0,1\}^r$ for $\mathcal{C}$.*

The discussion above is encapsulated in the following Lemma.

Lemma 28 *If a $\frac{1}{2}$-hitting set generator for circuits of size r, running in time $t(r)$ exists, then* **RP** *is contained in* **TIME**$(t(n^{O(1)})n^{O(1)})$.

In particular, if a hitting set generator running in polynomial time exists, then **RP**=**P**.

It is not so easy to directly construct a generator with the parameters of Lemma 28, even making reasonable unproven assumptions. To make the task of eliminating the use of random bits easier, it seems a good strategy to first try to use as few as possible. Thus, the work we have already done on randomness-efficient amplification should be very useful to us. That this intuition is true, is expressed in the following lemma.

Given an integer q and a real value $\epsilon > 0$, the problem (q,ϵ)-STRONG ONE-SIDED CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION is defined as follows: Given a Boolean circuit C of size n^q with n inputs and one output, with a promise that *either* (a) $\Pr[C(x) = 1] = 0$ *or* (b) $\Pr[C(x) = 1] \geq 1 - 2^{-n+n^\epsilon}$ for a randomly chosen $x \in \{0,1\}^n$ we should tell which of (a) or (b) is the case.

Lemma 29 *For any constant $\epsilon > 0$, there is a constant q, so that (q,ϵ)-STRONG ONE-SIDED CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION is complete for* **pRP**.

Proof Given an **pRP** problem solved by a machine M, we reduce it to (q,ϵ)-STRONG ONE-SIDED CIRCUIT ACCEPTANCE PROBABILITY ESTIMATION as follows. First, using Theorem 9, combined with

Lemma 22, convert M to a polynomial machine M' using $R(n) > n$ random bits and achieving error probability $2^{-R(n)+R(n)^\epsilon}$. Then, as in the proof of Lemma 2, given an input x, convert M' to a circuit for instances of length $|x|$ with two inputs, the real input x and the $R(|x|)$ random bits and build the input x into the circuit, leaving only the random bits as input. The circuit has size $R(|x|)^q$, for some constant q depending on ϵ. Note that unlike the proof of Lemma 2, we cannot make the circuit linear size by adding dummy inputs, as we now measure the error probability as a function of the number of random bits. ■

Using Lemma 29 instead of Lemma 2, we get the following variation of Lemma 28.

Lemma 30 *For any constant $\epsilon > 0$, there is a constant $q \geq 1$, so that if a $(1 - 2^{-r+r^\epsilon})$-hitting set generator for circuits $C : \{0,1\}^r \rightarrow \{0,1\}$ of size r^q exists, running in time $t(r)$, then* **RP** *is contained in* $\mathbf{TIME}(t(n^{O(1)})n^{O(1)})$.

Thus, we have shown that we only have to construct a $(1 - 2^{-r+r^\epsilon})$-hitting set generator, rather than a $\frac{1}{2}$-hitting set generator to derandomize **RP**. A slightly different way of seeing this is by using the following lemma, which states that we can simply convert a $(1 - 2^{-r+r^\epsilon})$-hitting set into a $\frac{1}{2}$-hitting set.

Lemma 31 *For any constant $\epsilon > 0$, there are constants $q \geq 1$ and $\delta > 0$ so that the following holds. There is a polynomial time procedure which, on input H where H is $(1 - 2^{-r+r^\epsilon})$-hitting set in $\{0,1\}^r$ for circuits of size r^q, outputs a $\frac{1}{2}$-hitting set in $\{0,1\}^{r'}$ for circuits of size r', where $r' = r^\delta$.*

The polynomial time procedure of Lemma 31 does the following: It interprets H as a subset of U, where $E \subseteq U \times V$ is the explicit extractor (viewed as a disperser) of Theorem 9 and outputs the neighbors of H in V. We omit the details, as they are essentially just another way of looking at lemma 29 and its proof.

Lemmas 28 and 30 hold with **RP** replaced with **pRP**. Thus, Theorem 2 implies that if a hitting set generator running in polynomial time exists, then also **BPP**=**P**. On the other hand, Theorem 2 does not give us a more general theorem of the form: "If a hitting set generator running in time $t(r)$ exists, then **BPP** is contained in $\mathbf{TIME}(t(n^{O(1)})n^{O(1)})$". The proof of Theorem 2 would yield only (roughly) that **BPP** is contained in $\mathbf{TIME}(t(t(n^{O(1)}))n^{O(1)})$. However, Andreev et al., 1998, showed by a different proof that the stronger conclusion does in fact hold. The proof was simplified significantly by Goldreich and Wigderson, 1999, the latter proof using extractors.

A more direct way of derandomizing **BPP** is to consider a stronger generator than a hitting set generator. The definition of a $(1-\epsilon)$-hitting set generator is strongly analogous to the definition of a $(1-\epsilon)$-hitting explicit disperser: In the disperser case, we want to hit each subset of V of density at least $1-\epsilon$ with high probability. In the hitting set generator case, we want to hit every subset of V of density at least $1-\epsilon$, recognized by a small circuit. To derandomize two-sided error algorithms, we want to make a definition similarly analogous to the definition of an extractor. To get the correct definition, it is useful to consider Lemma 18. An extractor produces a distribution which looks, to every statistical test, almost as the uniform distribution. Analogously, a *pseudorandom generator* produces a distribution which looks, to every statistical test given by a small circuit, almost as the uniform distribution. Now we get the following definitions, analogous to the definitions of hitting sets and hitting set generators.

Definition 6 *If $\mathcal{C}$ is a set system over $\{0,1\}^r$, a pseudorandom set for $\mathcal{C}$ with error ϵ is a (multi-)set P in $\{0,1\}^r$ so that for every set $S \in \mathcal{C}$, $|\Pr_{x\in P}[x \in S] - \Pr_{x\in\{0,1\}^r}[x \in S]| \leq \frac{\epsilon}{2}$.*

We use the terminology "A pseudorandom set with error ϵ for circuits of size r" a short hand for "A pseudorandom set with error ϵ for the subsets of $\{0,1\}^r$, decided by circuits of size r".

A pseudorandom generator with error ϵ for circuits of size r is a deterministic algorithm which on input r, outputs a pseudorandom set in $\{0,1\}^r$ for circuits of size r.

Now, Lemma 2 immediately yields

Lemma 32 *If a pseudorandom generator with error less than $\frac{1}{3}$ for circuits of size r, running in time $t(r)$ exists, then* **BPP** *is contained in* $\mathbf{TIME}(t(n^{O(1)})n^{O(1)})$.

As mentioned, it is not known how to make efficient hitting set or pseudorandom generators without appealing to unproven assumptions. In the rest of this section, we discuss three ways of constructing hitting set and pseudorandom generators based on different assumptions.

5.1 CRYPTOGRAPHIC PSEUDORANDOM GENERATORS

The first constructions of pseudorandom generators, done by Blum and Micali, 1984, and Yao, 1982, were intended primarily for cryptography, with derandomization as an interesting secondary application. This means that the parameters of interest were somewhat different.

Recall that we want our pseudorandom generator to efficiently output, on input r, a pseudorandom set for circuits of size r. If the set has size $d(r)$, we should of course be allowed time at least $d(r)$ to output the set. For cryptographic applications it turns out that we want something stronger. We want the generator, on input r *and* an index $i \in \{1, 2, \ldots, d(r)\}$ to output the i'th member of the pseudorandom set (relative to some arbitrary enumeration of the set) in time polynomial in the length of i, i.e., $\log d(r)$. Thus, we should be able to *sample* from the set very efficiently. This is not necessary for deterministic simulation, as we intend to run our original randomized algorithm on all coin toss sequences in the set (or use them as assignments to the circuit of Lemma 2). Furthermore, in the cryptographic case, we want the set to be pseudorandom for circuits of size r^k, for all constant k, simultaneously, rather than just circuits of size r. In particular, the sampling procedure will have shorter time to produce a member of the set that the size of the circuits the resulting distribution should be pseudorandom for.

Clearly, the cryptographic requirements are much more restrictive than the requirements for derandomization and it is only known how to construct cryptographic generators under much harsher assumptions than the assumptions we are aiming for (such as the assumptions of Statement 6 and 7 in the introduction). Blum and Micali, 1984, and Yao, 1982, constructed cryptographic generators under average case hardness assumptions about specific number theoretic functions (discrete log and factoring). Later, Goldreich and Levin, 1989 showed that such generators can be constructed under the assumption that a *cryptographic one-way permutation* exists, and Impagliazzo et al., 1989, showed that they can be constructed under the weaker assumption that a cryptographic one-way *function* exists. This latter condition can also be shown to be necessary. A cryptographic one-way function is a polynomial time computable function f mapping $\{0,1\}^*$ to $\{0,1\}^*$, with $|f(x)| \geq |x|$ for all x, so that for any polynomial time computable function, g does a very bad job inverting f. More precisely, for any polynomial time function g, for a random $x \in \{0,1\}^n$, the probability that $g(f(x))$ is an inverse of $f(x)$, i.e., the probability that $f(g(f(x))) = f(x)$, is negligible, i.e., smaller than the inverse of any polynomial in n.

Impagliazzo et al., 1989, building on Blum and Micali, 1984, Yao, 1982, and Goldreich and Levin, 1989, prove the following theorem.

Theorem 10 *If a cryptographic one-way function exists, then* **BPP** *is included in* **SUBEXP**.

Today, this theorem is mainly of historical interest, as we can get a (provably) stronger statement using the hardness based generators to be described below. On the other hand, there is no known way of using the constructions we shall concentrate on in this chapter directly for cryptography, so cryptographic pseudorandom generators remain extremely important for such applications.

5.2 SIPSER'S HITTING SET GENERATOR

Sipser proved the existence of a hitting set generator using the assumption that some computations need almost as much space as time.

Theorem 11 *There is a constant $\delta > 0$ so that the following holds. If, for some constant $c \geq 1$, there is a language L in* $\mathbf{TIME}(2^{cn})$ *so that any Turing machine for L must use space at least $2^{(c-\delta)n}$ on all sufficiently large input lengths n, then* **pP**=**pRP** *(and hence* **P**=**BPP***).*

Proof By Lemma 30, it is enough to show that there is a generator which on input r runs in time polynomial in r and outputs a set $H_r \subseteq \{0,1\}^r$ which is a $(1-2^{-r+r^{1/2}})$-hitting set for circuit of size r^q, for a certain constant q. The value of δ for which we will show the theorem will be $\delta = \frac{1}{5q}$.

Let L be a language in $\mathbf{TIME}(2^{cn})$ and let M be a Turing machine witnessing this fact. We can assume that the work tapes of M are all over the Boolean alphabet $\{0,1\}$, and that each work tape has length at most 2^{cn}. Given a parameter r, we split each work tape into $2^{cn}/r$ blocks of length r. Given an input x to M, during the computation of M, each block of each work tape will attain a sequence of configurations, each configuration being a bit vector in $\{0,1\}^r$. Let H_x be the union of all these configurations, taken over all blocks of all the work tapes. Let $H_{n,r}$ be the union of all sets H_x, for all $x \in \{0,1\}^n$.

Now consider the set $H_r := H_{2q\log r, r}$. Observe that the set can be generated in polynomial time (in r). We claim that, for sufficiently large r, this is a $(1-2^{-r+r^{1/2}})$-hitting set in $\{0,1\}^r$ for circuits of size r^q.

Suppose not. Then there are arbitrarily large values of r and circuits $C_r : \{0,1\}^r \to \{0,1\}$ of size r^q with $|Z(C_r)| \leq 2^{r^{1/2}}$ and $H_r \subseteq Z(C_r)$, where $Z(C_r) = \{x | C_r(x) = 0\}$.

The crucial observation is that we can now define a *compression* function $c : H_r \to \{0,1\}^{r^{1/2}}$ and a decompressor $d : \{0,1\}^{r^{1/2}} \to \{0,1\}^r$, so that $d(c(x)) = x$ for every $x \in H_r$ by letting $c(x)$ be the *rank* of x in $Z(C_r)$. If C_r is given as advice, the functions c and d can be computed in space $O(r^q)$, as this is the size of the circuit C_r.

We now describe a simulation of M on inputs of length $n = 2q \log r$ using space $2^{(c-\delta)n}$, contradicting the assumption about L. We first assume that the machine is given as advice the circuit C_r on a special tape. At any given time, the simulating machine M' stores on its work tapes the compressed values $c(y)$ for each block y on each work tape of M, except for those blocks containing a tape head of M. These are stored in uncompressed, "verbatim" form. Whenever a tape head of M is to be moved from one block u to another v, M' simulates this by compressing u, replacing it with $c(u)$ and decompressing the value $c(v)$ stored on its tapes, replacing it with v. Thus, the entire computation of M can be simulated.

The space use is $O(r^q)$ for the compression/decompression operations, $O(r)$ for the uncompressed blocks and $O(r^{\frac{1}{2}} \cdot 2^{cn}/r)$ for the compressed ones, i.e., a total of $O(r^q + 2^{cn}/r^{\frac{1}{2}})$ which is less than $2^{(c-\delta)n}$, as desired.

We now just need to show that we don't need the advice C_r. This follows from the fact that the above simulation can be done using *any* circuit $C : \{0,1\}^r \to \{0,1\}$ instead of C_r and will work correctly if we have $|Z(C)| \leq 2^{r^{\frac{1}{2}}}$ and any block of the computation turns out to be a zero of C. We can test for these properties when we do the simulation and abort the simulation if one of them fails. Thus to simulate M without the advice, we try the simulation of *all* possible C and if they all fail, we do a naive simulation. For each r where H_r is not a hitting set, our simulation works in space $2^{(c-\delta)n}$, as it will eventually find C_r or another circuit that works. This contradicts the assumption about L.

■

We can optimize the value of δ by using the best available dispersers to get as good a tradeoff between ϵ and q as possible in Lemma 29. The value $\delta = 0.01$ is certainly possible by the above argument. However, there is not much reason to do so, as we can obtain a stronger theorem as a corollary to the hardness based generator of Impagliazzo and Wigderson, described in the next section.

5.3 HARDNESS BASED GENERATORS

Hardness based generators are the most important kind of hitting set and pseudorandom generators for applications in derandomization. The next two sections of the chapters will be devoted to constructing them. In Section 6, we prove the following Theorem, first proved by Babai et al., 1991a, building on Nisan and Wigderson, 1994.

Theorem 12 *If there is a language L in* **EXP** *so that, for all n, the circuit complexity of $L \cap \{0,1\}^n$ is $n^{\omega(1)}$, then* **BPP** $\subseteq$ **SUBEXP**.

This establishes Statement 6 of the introduction: Note that the assumption stating that a language in **EXP** of superpolynomial circuit complexity for all n exists is just a reformulation of the assumption that the truth table of a Boolean function on n inputs with superpolynomial circuit complexity can be generated in time $2^{n^{O(1)}}$.

Also note that the assumption of the theorem is *not* equivalent to "$\mathbf{EXP} \not\subseteq \mathbf{P/poly}$"., i.e., to the statement that **EXP** does not have polynomial sized circuits. Indeed it is conceivable that any language in **EXP** has polynomial sized circuits, except on some "sparse" infinite sequence of input lengths. If this is the case, the statement "**EXP** does not have polynomial sized circuits" could be true, but the assumption of theorem 12 would be false. However, the proof of Theorem 12 also yields a proof of the following theorem.

Theorem 13 *If* **EXP** *does not have polynomial circuits then* $\mathbf{BPP} \subseteq \cap_{\epsilon>0}\mathbf{TIME}_{i.o}(2^{n^\epsilon})$

Here, $\mathbf{TIME}_{i.o}(t)$ is the class of languages decided by a deterministic Turing machine running in time $t(n)$ and deciding the correct language for infinitely many input lengths n.

The theorems above provide a relatively weak conclusion from a relatively weak assumption, and are often referred to as *low end* simulation theorems. A *high end* simulation theorem is the theorem of Impagliazzo and Wigderson, 1997, which we shall prove in Section 7.

Theorem 14 *If there is a language L in* **E** *and an $\epsilon > 0$, so that for all n, the circuit complexity of $L \cap \{0,1\}^n$ is at least $2^{\epsilon n}$, then* $\mathbf{BPP} = \mathbf{P}$.

This yields Statement 7 of the introduction, as the assumption of the theorem is, again, just a reformulation of the assumption of that statement.

The proofs of the theorems above work by constructing *hardness based* pseudorandom or hitting set generators. Such a generator works as follows. First the generator efficiently generates the truth table of a Boolean function f with big circuit complexity. This is possible by the assumption of the theorem. Then, using a series of transformation, the generator transforms this truth table into a hitting set or pseudorandom set for circuits of the appropriate size.

In Trevisan, 1999, it was observed that it is useful to think of the truth table as an extra input to the generator, rather than being generated inside the generator. Thus we define:

Definition 7 *A hardness based $(1-\epsilon)$-hitting set generator with hardness threshold $s(\cdot)$ is an algorithm which takes as input the truth table of* *a Boolean function* $f : \{0,1\}^{\log n} \rightarrow \{0,1\}$ *(i.e. n bits), with a promise that the circuit complexity of f is at least* $s(\log n)$ *and outputs a hitting set* $H \subseteq \{0,1\}^{n'}$ *for circuits of size n', for some n' depending on n.*

A similar definition can be made for the case of a hardness based pseudorandom generator. The crucial observation of Trevisan, 1999 is that when we look at it this way, the syntactic similarity to the definitions of dispersers and extractors are immediate. Indeed, given a hardness based hitting set generator, we can define a graph on $U \times V$, where $U = \{0,1\}^n$ and $V = \{0,1\}^{n'}$ by putting an edge between $x \in \{0,1\}^n$ and $y \in \{0,1\}^{n'}$, if y is a member of the set produced by the generator if it is given as input the truth table x. Now, this graph has the property that if we take a *fixed* vertex in U with big circuit complexity, its set of neighbors intersects any set $S \subseteq V$ of density $1 - \epsilon$ which is recognized by a small circuit. Recall that the disperser property states that for any set $S \subseteq V$ of density $1 - \epsilon$, this will be the case for *the vast majority of vertices of* U, but not any particular fixed one. On the other hand, the disperser does not require S to be recognized by a small circuit. Thus, it seems that we are getting something more from the disperser (we are hitting all sets), by paying something more (we pick a random vertex from U, rather than a fixed one with a particular property).

We now show that the similarity is not merely syntactic: Hardness based hitting set generators *are* dispersers, if the proof that they work *relativizes*, i.e., holds under any oracle.

Definition 8 *A relativizable hardness based* $(1-\epsilon)$*-hitting set generator with hardness threshold* $s(\cdot)$ *is an algorithm which takes as input the truth table of a Boolean function* $f : \{0,1\}^{\log n} \rightarrow \{0,1\}$ *(i.e. n bits), with a promise that the circuit complexity of f is at least* $s(\log n)$ *(with respect to some arbitrary oracle A) and outputs a* $(1-\epsilon)$*-hitting set* $H \subseteq \{0,1\}^{n'}$ *for circuits of size n' (containing oracles for A), for some n' depending on n.*

The hardness based generators we shall construct will be relativizable, as they use no special properties of Boolean circuits that are not satisfied by oracle circuits as well. The reason they are dispersers is, intuitively, that if a construction relativizes, it cannot use any particular property of "big sets recognized by small circuits" as opposed to just "big sets". Thus, even though the hitting set generator as opposed to a disperser allows us to just hit the big sets recognized by small circuits, we can only take advantage of this to a limited extend in the proof of the correctness of the generator. Formally, we have the following theorem.

Theorem 15 *A relativizable hardness-based* $(1-\epsilon)$*-hitting set generator with hardness threshold* $s(\cdot)$ *is, viewed as a graph, an explicit* $(1-\epsilon)$*-hitting disperser on* $\{0,1\}^n \times \{0,1\}^{n'}$ *with threshold* $2^{2s(\log n)\log s(\log n)}$.

Proof We need to prove that given any subset S of U with $|S| \geq 2^{2s(\log n)\log s(\log n)}$, more than ϵn of the vertices of V are adjacent to S. Suppose not. Let S be a set for which this is not the case, and let A be the non-neighbors of S, so we have $|A| \geq (1-\epsilon)|V|$. Viewed as a subset of $\{0,1\}^{\log |V|}$, we can use A as an oracle and consider circuit complexity relative to A. By Shannon's counting argument, viewed as truth tables for Boolean functions on $\log |U|$ variables, at least one of the members of S must have oracle circuit complexity with oracle gates for A at least $\frac{1}{2}\log|S|/\log\log|S| > s(\log n)$. Let this element of S be denoted a. Thus, by the hitting set generator property, the vertices in V adjacent to a will intersect every set in V which is the characteristic (accepted) set of an oracle circuit with oracle gates for A of size at most n *and* has size at least $(1-\epsilon)|V|$.

But then consider the oracle circuit defined by $x \rightarrow A(x)$. It has size n, density at least $1-\epsilon$ and the neighbors of a do not intersect its characteristic set (as this set is the non-neighbors of S). A contradiction. ■

By an analogous proof, one can show that relativizable hardness based pseudorandom generators are extractors. Thus, we cannot build these generators without building dispersers and extractors as well.

This leads to two strategies for building the generators: We can either refine the disperser/extractor constructions, making them obtain the stronger generator properties. This is the strategy used in Section 6 where we shall present a proof of Theorem 12, based on extending the extractor construction of Trevisan (as we mentioned earlier, historically, it was the other way around). Or, alternatively, we can build dispersers or extractors with the appropriate parameters *into* the constructions, using them as "black boxes" and then try to "add on" the stronger generator property. This is the strategy used in Section 7 where we shall present a proof of Theorem 14. It is also very close to the way we proved Theorem 11, even though the generator of that theorem was not hardness-based.

First however, we shall note that Theorems 12 and 14 give stronger results that either the cryptographic generators or Sipser's generator. Indeed, Theorem 10 can also be obtained as a corollary of Theorem 12, as the assumption of a cryptographic one-way function easily yields the appropriate language in **EXP**. Also, as corollary of Theorem 14, we get the following stronger version of Theorem 11.

Corollary 2 *If there for any constant $\epsilon > 0$ there is a language L in* **E** *so that any Turing machine for L must use space at least $2^{\epsilon n}$ on all sufficiently large input lengths n, then* **P**=**BPP**.

Proof We show that the assumption of Corollary 2 implies the assumption of Theorem 14. So let a (without loss of generality, Boolean) language L in **TIME** (2^{cn}) be given so that any machine for L must use space $2^{\epsilon n}$. We shall construct a language L' in **E** so that $L' \cap \{0,1\}^n$ has no circuit of size $2^{\frac{\epsilon}{10c}n}$.

Given $L \in \mathbf{E}$ by a one-tape Turing machine M, we define the language $L' = \{\langle x, i, j, \sigma, b, \tau \rangle |$ the contents of the i'th cell at time j during the computation of M on x is σ and b is a Boolean which is true if the tape head is on cell i at time j. Furthermore, if b is true, then the state of the machine at time j is $\tau\}$. Then L' is *locally checkable*, i.e., we can tell if circuit C decides $L' \cap \{0,1\}^n$ for some input length n correctly if an exponential number of easily generated local conditions of the form $C(y) \Leftrightarrow C(y_1) \wedge C(y_2) \ldots \wedge C(y_l)$, where l is a constant, are satisfied. This means that we can check if a certain circuit C of size $2^{\frac{\epsilon}{10c}n}$ computes $L' \cap \{0,1\}^n$, using space linear in the description of the circuit. Now suppose such a circuit actually exists, for infinitely many n. We show how to evaluate L space-efficiently on infinitely many input lengths. On input $x \in \{0,1\}^n$, we reduce the question $x \in L$? to a question $x' \in L'$? for x' of length $n' \leq 10nc$. Now we do an exhaustive search for a circuit of size $2^{\frac{\epsilon}{10c}n'}$ deciding L', testing each possibility as described above. If we find one, we use it to decide $x' \in L'$ and we are done. If we fail to find a correct circuit, we run the original machine M to decide $x \in L$. As we will find a circuit for infinitely many input lengths, we have an algorithm in space $2^{\epsilon n}$ for infinitely many n, contradicting the assumption on L. ∎

Even though we can get the stronger statement of Corollary 2, a self-contained proof, including the proof of Theorem 14 would be much harder than the proof of Theorem 11 given before. To get the easiest known self-contained proof of Corollary 2, we can use Theorem 20 to be described later, instead of Theorem 14.

6. A LOW END GENERATOR

In this section, we prove Theorem 12. For this, we describe, for any constant $k \geq 1$, a procedure running in time $2^{O(n^6)}$, converting the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ with no circuits of size $s(n)$, for a superpolynomial function s, into a pseudorandom set in $\{0,1\}^{n'}$ with small error, for circuits of size n' with $n' = n^k$. The construction relativizes. By the discussion of Section 5.3, this means that the procedure also defines an explicit extractor $E : U \times A \to V$ with $|U| = 2^{2^n}$, $|V| = 2^{n^k}$, $|A| = 2^{O(n^6)}$ and min-entropy threshold $s(n)^{1+o(1)}$. While we have not previously explicitly described an extractor construction with these parameters, having concentrated on extractors with polynomial degree, it is not difficult to see that by setting the parameters right in the Trevisan extractor, we can obtain such an extractor.

The actual construction of the pseudorandom generator follows the steps of the Trevisan extractor quite closely.

Recall that the Trevisan extractor performs two operations on its inputs $x \in U$ and $y \in A$, where x is randomly chosen with a distribution with min-entropy at least t, and y is uniformly chosen. First, it applies an error correcting code e to x, obtaining $e(x)$. If x has min-entropy at least t, any lossy source code for $e(x)$ with distortion bounded away from $\frac{1}{2}$ must have rate close to t. Second, the value y defines, through Nisan-Wigderson designs, a number of bit positions of $e(x)$ which are taken out and concatenated. This yields a value $E(x, y)$ with distribution close to uniform.

The pseudorandom generator goes through two completely analogous steps. Instead of starting with a value x sampled from a distribution on m bits with min-entropy at least t, we start with a *computational analogue* of this: The truth table of a function $f : \{0,1\}^n \to \{0,1\}$ with n roughly $\log m$ and with circuit complexity at least (roughly) t. Intuitively, if we think of a distribution with min-entropy at least t simply as a distribution with no lossless fixed length source code with rate t, we see that the relevant computational analogue of a lossless source code is a circuit. Then, rather than generating a value $e(x)$ distributed so that any lossy source code with rate roughly t has distortion close to one half, we generate a computational analogue of this: *The truth table of a Boolean function $\tilde{f}$ so that any circuit C of size close to t attempting to compute $\tilde{f}$ must disagree with $\tilde{f}$ on a fraction of inputs close to $\frac{1}{2}$.* Finally, we use the second input y to pick out, through Nisan-Wigderson designs, a number of inputs on which to evaluate $\tilde{f}$. As in the extractor case, we concatenate the outputs. This yields a value $G(x, y)$. The values $\{G(x, y) | y \in A\}$ now yield a computational analogue of a distribution ϵ-close to random: an ϵ-pseudorandom set for small circuits.

In the next subsections we formalize the above intuitive description. The second step causes the fewer problems and will be dealt with first, in Subsection 6.1. Essentially, we can just redo the corresponding information theoretic proof replacing "low rate source codes" with "small circuits" and "predictors" with "predictors with small circuits". To improve readability, we build the generator of the original truth table into our generators, rather than regard the truth table as an extra input. Thus, the last step of the construction will be given by the following lemma.

Lemma 33 *Let $\tilde{s}(n)$ be a superpolynomial function and let constant $c \geq 2$.* If *there is a procedure running in time $2^{O(n^c)}$ which on input n outputs the truth table of a function $\tilde{f} : \{0,1\}^n \to \{0,1\}$ so that no circuit of size $\tilde{s}(n)$ agrees with $\tilde{f}$ on more than a $\frac{1}{2} + \tilde{s}(n)^{-1}$ fraction of the inputs,* then*, for any constant $\epsilon > 0$, there is procedure running in time $2^{O(n^\epsilon)}$ which on input n outputs a pseudorandom set for circuits of size n with error $\frac{1}{10}$, and thus,* **BPP** *is in* $\mathbf{TIME}(2^{O(n^\epsilon)})$.

To successfully implement the first step, we need to do more work. While the key is still the use of error correcting codes, it turns out that generic error correcting codes will not do the trick for us. We need a special kind of error correcting codes, not traditionally studied in coding theory: *locally decodable codes*. Locally decodable error correcting codes will enable us to transform a function f with no circuit of size t to a function f' so that all circuits of size roughly t must disagree with f' on a non-trivial fraction of the inputs. Thus, in Section 6.2, we prove the following lemma.

Lemma 34 *Let $s(n)$ be a superpolynomial function and let constant $c \geq 2$.* If *there is a procedure running in time $2^{O(n^c)}$ which on input n outputs the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ with no circuits of size $s(n)$* then *there is a procedure running in time $2^{O(n^c)}$ which on input n outputs the truth table of a function $f' : \{0,1\}^n \to \{0,1\}$ so that any circuit of size $s'(n)$ disagrees with f' on at least a $1/n^2$ fraction of the inputs, where $s'(n)$ is another superpolynomial function.*

Unfortunately, locally decodable codes will not give us what we really desire to get in a position to apply Lemma 33: A function $\tilde{f}$ so that all small circuits disagree with $\tilde{f}$ on almost half the inputs. Instead we convert, in Section 6.3, our function f' to such a function $\tilde{f}$ using a different, non-coding theoretic, construction: If $f' : \{0,1\}^n \to \{0,1\}$, we let $\tilde{f} : \{0,1\}^{n^5} \to \{0,1\}$ be given by $\tilde{f}(x_1, x_2, \ldots, x_{n^4}) = f'(x_1) \oplus f'(x_2) \oplus \cdots \oplus f'(x_{n^4})$, where $\oplus$ denotes Boolean XOR, and each $x_i \in \{0,1\}^n$. A lemma, *Yao's XOR lemma* will tells us that this construction has the desired property. Thus, we can bridge the gap between Lemma 34 and Lemma 33, proving the following lemma.

Lemma 35 *Let $s'(n)$ be a superpolynomial function and let $c \geq 2$.* If *there is a procedure running in time 2^{n^c} which on input n outputs the truth table of a function $f' : \{0,1\}^n \to \{0,1\}$ so that all circuits of size*

$s'(n)$ disagrees with f' on at least a $1/n^2$ fraction of the inputs, then *there is a procedure running in time $2^{O(n^c)}$ which on input n outputs the truth table of a function $\tilde{f} : \{0,1\}^n \to \{0,1\}$ so that no circuit of size $\tilde{s}(n)$ agrees with $\tilde{f}$ on more than a $\frac{1}{2} + 1/\tilde{s}(n)$ of the inputs, where $\tilde{s}(n)$ is another superpolynomial function.*

Lemma 34, Lemma 35 and Lemma 33 together proves Theorem 12. Thus, the construction we give has 3 steps rather than 2. It is possible to do the construction without the middle step by using a different, more relaxed, kind of code than locally decodable codes, essentially capturing "local list decodability". This was done by Sudan et al., 1999. We present the three step proof here, as it arguably gives the easiest self-contained proof of Theorem 12. It was also the original proof given by Nisan and Wigderson, 1994, and Babai et al., 1991a.

6.1 UNPREDICTABILITY AND PSEUDORANDOMNESS

In this subsection, we prove Lemma 33. For this, we need a computational analogue of Lemma 21. This lemma was proved before Lemma 21, by Yao, 1982.

Lemma 36 *Given $\epsilon > 0$. If a set $S \subseteq \{0,1\}^m$ is not ϵ-pseudorandom for circuits of size m, then the uniform distribution $\mathcal{D}$ on S is $\frac{\epsilon}{2(m+1)}$-predictable, with the predictor given by a circuit of size m.*

Proof We mimic the proof of Lemma 21 closely. We consider the $m+1$ distributions $\mathcal{D}_0, \ldots, \mathcal{D}_m$, defined as follows. To pick a random element of $\mathcal{D}_i$, we pick $x \in \{0,1\}^m$ according to $\mathcal{D}$ and $y \in \{0,1\}^m$ according to the uniform distribution $\mathcal{U}$ and return $x_1, \ldots, x_i, y_{i+1}, \ldots, y_m$. Thus, $\mathcal{D}_0$ is the uniform distribution, $\mathcal{D}_m$ is $\mathcal{D}$, and the $\mathcal{D}_i$'s in between are hybrids of the uniform distribution and $\mathcal{D}$.

By definition, if S is not pseudorandom for circuits of size m, we have a circuit C of size m, so that $|\Pr_{x\in S}[C(x) = 1] - \Pr_{x\in\{0,1\}^m}[C(x) = 1]| > \frac{\epsilon}{2}$, i.e. $|\Pr_{x\in\mathcal{D}_m}[C(x) = 1] - \Pr_{x\in\mathcal{D}_0}[C(x) = 1]| > \frac{\epsilon}{2}$. By the triangle inequality, there must be i so that $|\Pr_{x\in\mathcal{D}_{i+1}}[C(x) = 1] - \Pr_{x\in\mathcal{D}_i}[C(x) = 1]| > \frac{\epsilon}{2(m+1)}$.

We now define a randomized predictor f for the $i+1$'st bit with good (expected) advantage. On input $x_1 x_2 \ldots x_i$ generate uniform random bits $r_{i+1}, \ldots, r_m$. Thus, $\tilde{x} = x_1 x_2 \ldots x_i r_{i+1} \ldots r_m$ is a sample from $\mathcal{D}_{i+1}$. Now, if $C(\tilde{x}) = 1$, predict x_{i+1} to be r_{i+1}, otherwise predict x_{i+1} to be $\neg r_{i+1}$. The expected advantage of the predictor is at least $\frac{\epsilon}{2(m+1)}$. We can freeze the random decisions of the predictor to get a predictor with the same advantage, given by a circuit of size m. ■

For a much more detailed coverage of the computational version of the hybrid technique, see Goldreich, 1999.

Now we are ready to prove Lemma 33. Let the parameter $\epsilon > 0$ be fixed. Given input n, we describe a procedure for generating a pseudo-random set for circuits of size n, using time $2^{O(n^\epsilon)}$.

We let $\epsilon' = \epsilon/c$, where c is the constant in the assumption of the Lemma. Then, using this assumption, we generate in time $2^{O(n^\epsilon)}$ the truth table of a function $f : \{0,1\}^{n^{\epsilon'}} \to \{0,1\}$ so that any circuit of size $s(n^{\epsilon'})$ agrees with f on at most a $\frac{1}{2} + \frac{1}{s(n^{\epsilon'})}$ fraction of the inputs.

Now, use Lemma 26 to construct an $(n, n^{2\epsilon'}, n^{\epsilon'}, \log n)$ Nisan-Wigderson design $S_1, S_2, \ldots, S_n$. The pseudorandom set for circuits of size n is defined to be $P = \{v_y\}_{y \in \{0,1\}^{n^{2\epsilon'}}}$, with v_y defined as follows: View y as a bit sequence of length $n^{2\epsilon'}$, and for S being a set in the design, let y_S by the subsequence consisting of the bits indexed by S. Then $v_y = f(y_{S_1})f(y_{S_2}) \ldots f(y_{S_n})$.

Suppose this is not a pseudorandom set for circuits of size n with error at most $\frac{1}{10}$. Then, if y is chosen at random, by Lemma 36, there is a predictor with advantage $\frac{1}{20(n+1)}$, given by a circuit of size n, for one of the bits of v_y, say the i'th bit $f(y_{S_i})$, given the previous bits.

Freeze the bits of $y_{S_i^c}$ (i.e., all bits except the bits of y_{S_i}) to fixed values while preserving the advantage of the predictor for a random setting of y_{S_i}. As $S_1 \cap S_i$ only contains $\log n$ bit positions and the bits of $y_{S_1 \setminus S_i}$ have been frozen, we can tabulate the value of $f(y_{S_1})$ as a function of $y_{S_1 \cap S_i}$ using a polynomial size table. Similarly, we can tabulate the value of $f(y_{S_j})$ as a function of $y_{S_j \cap S_i}$ for each $j = 2, \ldots, i-1$.

Now we have the following polynomial circuit approximating $f(y)$. The circuit has the tables above given as advice. On input y, the circuit looks up $f(y_{S_1}), \ldots, f(y_{S_{i-1}})$ in the tables above, These values are then fed into the predictor, to get a prediction for $f(y_{S_i})$ which we give as output. By the advantage of the predictor, the circuit is correct on random input with probability at least $\frac{1}{2} + \frac{1}{2(n+1)}$. The circuit has polynomial size. This contradicts the assumption on f. Thus, we have proved Lemma 33.

6.2 LOCALLY DECODABLE CODES

To introduce locally decodable error correcting codes, let us first recall the motivation for classical error correcting codes presented in Section 2.3: A sender wants to transmit a bit string x to a receiver reliably, even if a meddler is allowed to introduce a limited amount of noise in any message sent. Encoding x in an error correcting codes gave a satisfactory solution to this problem. Now consider a setting where the string x is rather long, say, an encoding of the Encyclopedia Britannica, and the receiver is really only interested in a small piece of x, say, the piece about the nesting habits of night owls. Thus, the receiver does not really want to read and decode the entire received string x^*. Ideally, he wants to read only a piece of x^* comparable in size to the piece about night owls in the original string x. Informally, a *locally decodable code* is an error correcting code that allows him to achieve this goal.

At first sight, it seems that locally decodable codes cannot exist for interesting settings of the parameters. In general, we would want the receiver to look at much fewer bits than the meddler is allowed to mix up. Thus, if the meddler knows which piece the receiver is interested in (and we assume that he does), the meddler will be able to mess up the communication completely by replacing all the bits the receiver is going to look at by random bits. Indeed, this impossibility proof is correct, but only if we assume that the receiver will choose which bits to look at *deterministically*. If we allow the receiver to pick the bits to look at using a *randomized* strategy and also allow him to retrieve an incorrect message piece with some small probability, locally decodable codes do exist, as we shall see in this section. First, we give a formal definition of a locally decodable error correcting code.

Definition 9 *A family of error correcting codes $e : \{0,1\}^n \to \{0,1\}^m$ is* locally decodable *with error ϵ and relative correction distance δ if there is an efficient randomized algorithm, the* decoding algorithm*, which is given as input an index $i \in \{1, \dots, n\}$, and, in a random access memory, a string x^*, so that the relative distance between x^* and $\tilde{x} = e(x)$ is at most δ, for some string $x \in \{0,1\}^n$. The algorithm runs in time polynomial in $\log n$ and returns a bit u, so that, with probability at least $1 - \epsilon$, over the random coin flips of the algorithm, we have $u = x_i$.*

Our intended application for a locally decodable code $e : \{0,1\}^n \to \{0,1\}^m$ is to convert the truth table of a function $f : \{0,1\}^{\log n} \to \{0,1\}$ with no small circuits to the truth table of a function $f : \{0,1\}^{\log m} \to \{0,1\}$ which cannot be approximated well by small circuits. We now verify that locally decodable codes indeed do this for us.

Lemma 37 *Let $e : \{0,1\}^n \to \{0,1\}^m$ be a locally decodable error correcting code with error less than $1/n$ and relative correction distance δ. Then for some constant k, the following is true: If x is the truth table of a function f on $\log n$ variables with no circuits of size s, then $e(x)$ is the truth table of a function f' on $\log m$ variables so that any circuit of size $s/(\log n)^k$ attempting to compute f' must make errors on more than a δ fraction of the inputs.*

Proof Let C' be a circuit of size s' computing f' incorrectly on at most a δ fraction of the inputs. The truth table of C', viewed as a bit string of length m differs from the truth table of f' on at most a δ fraction of the inputs. Thus, the local decoding procedure applied to the truth table of C' will reliably retrieve any bit of the truth table of f, i.e., it will evaluate f on any desired input with an error probability of at most $1/n$. Individual bits of the truth table of C' can be obtained by evaluating C'. Thus, by building C' into the local decoding procedure, we get a randomized circuit for f with an error probability of less than $1/n$ and of size $(\log n)^k s'$ for some constant k, as the time complexity of the procedure is polynomial in $\log n$. As the error probability is smaller than $1/n$ and there are only $2^{\log n} = n$ different inputs to give the circuit, we can freeze the random bits to get a deterministic circuit for f of the same size. Thus $(\log n)^k s' > s$ or $s' > s/(\log n)^k$, as desired. ■

We present two locally decodable codes. The first one, due to Babai et al., 1991b, is very simple, but has a rather bad blowup: It expands n bits to $2^{O(\log n \log\log n)}$ bits, i.e., the blowup is slightly superpolynomial. Also, it has a subconstant relative correction distance. It is, however, good enough to be used to prove Lemma 34 and we can give a self-contained proof of its local decodability without appealing to any non-trivial coding theory.

The Babai-Fortnow-Lund code (for shorts, the BFL-code) for $x \in \{0,1\}^n$ is defined as follows. Given $x \in \{0,1\}^n$, identify x with its characteristic function $x : \{0,1\}^{\log n} \to \{0,1\}$. Let $\mathbf{F}$ be a finite field of size roughly $(\log n)^2$. Let a *multilinear polynomial* over $\mathbf{F}$ be a polynomial where each variable occurs with degree at most 1 in each term. It is easy to see that there is a unique $\log n$-variate multilinear polynomial $p : \mathbf{F}^{\log n} \to \mathbf{F}$ so that p agrees with x on $\{0,1\}^{\log n}$. This polynomial is called the *multilinear extension* of x. Our code for x is the enumeration of all values of p, i.e., $\tilde{x} = (p(y))_{y \in \mathbf{F}^{\log n}}$. To make the code Boolean, we simply replace each field element $\mathbf{F}$ by an arbitrary Boolean encoding using $O(\log\log n)$ bits.

First we observe that the rate of the code is indeed $n \to 2^{O(\log n \log\log n)}$. Now we prove that the code is locally decodable.

Theorem 16 *The BFL-code is locally decodable with error* $1/n$ *and relative correction distance* $1/(\log n)^2$.

Proof Assume that we are a given a string x^* with distance at most $1/(\log n)^2$ to a BFL-code word $\tilde{x}$. Let $p : \mathbf{F}^{\log n} \to \mathbf{F}$ be the corresponding multilinear map. With random access to $\tilde{x}$, we can, for a given v, read off the value of $p(v)$ by reading $O(\log\log n)$ bits of $\tilde{x}$. If v is *randomly* chosen and we have random access to x^* instead of $\tilde{x}$, the probability that the same procedure does *not* give us the correct value of $p(v)$ is at most $O(\log\log n/(\log n)^2)$.

However, to prove locally decodability, we have to show that given some specific, i.e., *non-random*, value of v, we can retrieve the correct value of $p(v)$ with high probability, where the probability is over the choices of our decoding algorithm only. To achieve this, we reduce the non-random case to the random case as follows:

Let $i_1, i_2, \ldots, i_{\log n+1}$ be arbitrary non-zero members of $\mathbf{F}$. Given v, pick $u \in \mathbf{F}^{\log n}$ at random and read off from x^* the values $p(v+i_1 u), p(v+i_2 u), \ldots, p(v+i_{\log n+1} u)$, or, more precisely, what *would* have been these values, had x^* been uncontaminated. However, as each of $v + iu$ for $i \in \{1, \ldots, \log n + 1\}$ are uniformly chosen in $\mathbf{F}^{\log n}$ (though not independent) we have by the union bound that all of the retrieved values are correct with probability at least $1 - O((\log n + 1)(\log\log n/(\log n)^2)) = 1 - O(\log\log n/\log n)$. As p is multilinear, the map $f(t) = p(v+ut)$ is a univariate polynomial of degree at most $\log n$. Given that the retrieved values are all correct, we have found $f(i_1), f(i_2), \ldots, f(i_{\log n+1})$ and can uniquely interpolate the polynomial f. Then, we evaluate $f(0)$ which is the desired value $p(v)$.

The above procedure has error probability at most $O(\log\log n/\log n)$. To reduce the error probability to $1/n$, we repeat it $O(\log n)$ times. ■

Combining the BFL code with Lemma 37, we have a proof of Lemma 34: Using the assumption of the Lemma, we generate the truth table of a Boolean function with no small circuits, and return the BFL-encoding of this truth table. A slight technical problem is that to prove Lemma 34, we need, on any input n, to produce a truth table of length 2^n and this may not match the length of a BFL-codeword, for any setting of the parameters. A simple way to solve this which is easily seen to work if the parameters are set carefully, is to generate a BFL-codeword of length a power of two close to, but not exceeding the desired length 2^n, and concatenating it with itself a sufficient number of times to make the total length exactly 2^n.

For later use in the proof of Theorem 14 in Section 7, we shall also present a better locally decodable code due to Babai et al., 1991a which has only polynomial blowup $n \to n^{O(1)}$ and constant relative correction distance. Though similar to the above BFL-code, the decoding procedure of the BFLS-code is somewhat more complicated and uses as subroutine the Berlekamp-Welch decoder, presented in Section 2.3 as Lemma 10.

To define the BFLS-code, we first define a code $e' : \mathbf{F}^n \to \mathbf{F}^{n'}$, where $\mathbf{F}$ is a finite field of size roughly $(\log n)^4$. The final BFLS-code code is then the concatenated code $e'' : \{0,1\}^n \to \{0,1\}^{n'r}$. defined by $e''(x) = e' \circ e(x) = e(e'(x)_1) \circ e(e'(x)_2) \circ \cdots \circ e(e'(x)_{m'})$, where e is the code of Lemma 12. Note that the BFL-code is really a concatenated code too, but with a trivial encoding as the inner code.

The outer code e' can be seen both as a generalization of the multilinear extension to higher degree polynomials and as a generalization of the Reed-Solomon code to higher dimensions. It is called the *low degree extension* and is defined as follows.

Pick a subset S of $\mathbf{F}$ of size $\log n$ and identify the set $\{1, \ldots, n\}$ with the set $S^{\log n/\log\log n}$ in some arbitrary way. Thus, given a message $x \in \mathbf{F}^n$, we can view x as a map $x : S^l \to \{0,1\}$, where $l = \log n/\log\log n$. There is a unique l-variate polynomial p of degree at most $|S| - 1$ in each variable agreeing with f on S^l. The low degree extension $e'(x)$ is the enumeration of all values of this map, i.e. the vector $(p(v))_{v \in \mathbf{F}^l}$.

The sole advantage we get from using the low degree extension over the multilinear extension is the fact that it only blows up the size of its input by a polynomial. Indeed, the number of field elements in a code word is roughly $((\log n)^4)^{\log n/\log\log n} = n^4$.

Now we define the BFLS-code as the concatenation of the low degree extension with the code e of Lemma 12. Observe that the blowup of the code is indeed polynomial.

Theorem 17 *The BFLS-code is locally decodable with error* $1/n$ *and relative correction distance* $\frac{1}{20}$.

Proof Assume that we are a given a string x^* with distance at most 0.05 to a BFLS-code word $\tilde{x}$. Let $p : \mathbf{F}^l \to \mathbf{F}$ be the corresponding low degree polynomial. With random access to $\tilde{x}$, we can, for a given v, read off the value of $\tilde{p}(v)$ by reading $O(\log\log n)$ bits of $\tilde{x}$. If we have random access to x^* instead, a value so found may be incorrect. We can at least replace a value which is not an encoding of a field element by the closest valid encoding (an exhaustive search through all field elements is feasible, as $|\mathbf{F}|$ is only polylog n). Then, for a *randomly* chosen v, the probability that this procedure does *not* give us the correct value of $p(v)$ is less than 0.24. This is because less than a 0.24 fraction of the encodings of field elements can have more than a 0.24 fraction of the bits flipped - otherwise the relative distance between x^* and $\tilde{x}$ would be more than $(0.24)(0.24) > 0.05$.

Again, we reduce the non-random case to the random one in a similar way as before. Let $i_1, i_2, \ldots, i_{(\log n)^3}$ be arbitrary non-zero members of $\mathbf{F}$. Given v, pick $x \in \mathbf{F}^l$ at random and read off from x^* the values $p(v+$

$i_1x), p(v+i_2x), \ldots, p(v+i_{(\log n)^3}x)$, or rather, what *would* have been these values, had x^* been uncontaminated. However, as each of $v+ix$ for $i \in \{1, \ldots, (\log n)^3\}$ are uniformly chosen in $\mathbf{F}^l$ (though not independent), each is incorrect with probability at most 0.24, so the expected fraction of incorrect values is at most 0.24. Thus, the probability that more than a 0.501 fraction of values are correct is at least 0.501 (as $0.499 \cdot 0.499 > 0.24$). As p is a polynomial with a degree of at most $\log n$ in each variable, the map $f(t) = p(v+xt)$ is a univariate polynomial of degree at most $(\log n)^2/\log\log n$. Given that at least a 0.501 fraction of the retrieved values are correct, and n is sufficiently large, as we can assume without loss of generality, we can use the Berlekamp-Welch procedure of Lemma 10 to reconstruct this polynomial. Then, we evaluate $f(0)$ which is the desired value $p(v)$.

The above procedure gives us the correct answer with probability at least 0.501. To get the desired error probability, we repeat the procedure a number of times and take the most common answer. ∎

Locally decodable codes form an interesting topic in their own right. An important open problem is whether there are locally decodable codes with constant correction distance and with only linear blowup (rather than polynomial blowup). By doing the above construction more carefully, the blowup can be made $n^{1+\epsilon}$, for any desired constant $\epsilon > 0$. This is the best result currently known about locally decodable codes[5].

The BFLS code and Lemma 37 gives us the following improvement of Lemma 34 which will be used in the proof of the Impagliazzo-Wigderson theorem.

Lemma 38 *Given a constant $\epsilon > 0$. If there is a procedure which runs in time $2^{O(n)}$ and on input n generates a truth table of some Boolean function $f : \{0,1\}^n \to \{0,1\}$ with circuit complexity at least $2^{\epsilon n}$,* then *there is a constant $\epsilon' > 0$ and a procedure which runs in time $2^{O(n)}$ and on input n generates a truth table of some Boolean function $f' : \{0,1\}^n \to \{0,1\}$ so that any Boolean circuit of size $2^{\epsilon' n}$ fails to compute f' on at least a $\frac{1}{20}$ fraction of the inputs.*

6.3 THE XOR LEMMA

In this subsection we prove Lemma 35, thus completing the proof of Theorem 12. The statement of the lemma is a corollary of *Yao's XOR lemma*. For convenience, in the statement and proof of this lemma, we consider Boolean functions mapping $\{0,1\}^n$ to $\{-1,1\}$ rather than $\{0,1\}$. Then Boolean XOR is just integer multiplication. Furthermore, if we define the *correlation* of two functions f_1 and f_2 as the value

$2^{-n}\sum_{x\in\{0,1\}^n} f_1(x)f_2(x)$, we see that the correlation of f_1 and f_2 is δ, if and only if the probability over random input x that $f_1(x)=f_2(x)$, is $\frac{1}{2}+\frac{\delta}{2}$.

Yao's XOR-lemma, of Yao, 1982, is the following lemma.

Lemma 39 *Let $f:\{0,1\}^n \to \{-1,1\}$ and assume that all circuits of size at most s have correlation at most δ with f. Now define for any parameter $k \geq 1$*

$$f^{(k)}:\{0,1\}^{kn} \to \{-1,1\}$$
$$f^{(k)}(x_1,x_2,\ldots,x_k) = f(x_1)f(x_2)\ldots f(x_k),$$

where each $x_i \in \{0,1\}^n$. Then all circuits of size at most

$$\frac{s-10(\frac{n}{\epsilon})^{10}}{\left(\frac{n}{\epsilon}\right)^{10}}$$

have correlation at most $\delta^k + k\epsilon$ with $f^{(k)}$.

Before proving the Lemma, let us see that it implies Lemma 35. To generate the desired truth table, we generate a truth table f' using the assumption of Lemma 35 and let $\tilde{f}:\{0,1\}^{n^5} \to \{0,1\}$ be defined by $\tilde{f}(x_1,x_2,\ldots,x_{n^4}) = f'(x_1)\oplus f'(x_2)\oplus\cdots\oplus f'(x_{n^4})$. As in the proof of Lemma 34 in Subsection 6.2, we may not be able to match exactly the size of the truth table we are supposed to generate, but as we did there, we can solve this by simply concatenating a number of copies of the generated truth table.

The proof of the XOR lemma we shall present is due to Levin, 1987 (for much more information about the XOR lemma and other proofs, see Goldreich et al., 1995). It is a proof by induction in k, with the induction step being handled by the following lemma, the *isolation lemma.*

Lemma 40 *Let $f_1:\{0,1\}^{n_1} \to \{-1,1\}$ and $f_2:\{0,1\}^{n_2} \to \{-1,1\}$. Assume that all circuits of size at most s_1 have correlation at most δ_1 with f_1 and that all circuits of size at most s_2 have correlation at most δ_2 with f_2. Then all circuits of size at most*

$$s = \min\left(\frac{s_1-10(\frac{n_1}{\epsilon})^{10}}{\left(\frac{n_1}{\epsilon}\right)^{10}}, s_2\right)$$

have correlation at most $\delta=\delta_1\delta_2+\epsilon$ with $f(x,y)=f_1(x)f_2(y)$.

Proof Assume to the contrary that a circuit C of size s has correlation more than δ with f.

Then, if we define $T(x) = \mathrm{E}_y[C(x,y)f_2(y)]$, we have

$$\delta < \mathrm{E}_{x,y}[C(x,y)f(x,y)] = \mathrm{E}_{x,y}[C(x,y)f_1(x)f_2(y)] = \mathrm{E}_x[f_1(x)T(x)].$$

Assume that $\exists x : T(x) > \delta_2$. Then the circuit $y \to C(x,y)$ has a correlation with f_2 better than allowed by the lemma hypothesis. Similarly, assume that $\exists x : T(x) < -\delta_2$. Then the circuit $y \to \neg C(x,y)$ has a correlation with f_2 better than allowed by the lemma hypothesis.

Hence, $\forall x : \frac{T(x)}{\delta_2} \in [-1;1]$. If the function $T(x)/\delta_2$ was Boolean and computable by a small circuit, it would contradict the assumption about f_1. Our strategy is to modify the function so that this will be the case.

Let $(y_i, f_2(y_i))_{i=1,2,\ldots}$ be a sequence of randomly generated samples, with y_i uniformly chosen in $\{0,1\}^{n_2}$.

Define

$$\tilde{T}(x) = \sum_{i=1}^{(\frac{n_1}{\epsilon})^{10}} f_2(y_i)C(x,y_i)$$

With the random sequence given as advice, $\tilde{T}$ is computable by a circuit of size at most $s(n_1/\epsilon)^{10} + 5(n_1/\epsilon)^{10} \leq s_1 - 5(n_1/\epsilon)^{10}$. Furthermore, by the Chernoff bound of Lemma 1 we have that for fixed x,

$$\Pr[|\tilde{T}(x) - T(x)| > \epsilon] < 2^{-2n_1}.$$

So with high probability $|\tilde{T}(x) - T(x)| \leq \epsilon$ holds for all x. We now freeze the choice of random pairs so that this is in fact the case. Now consider the value $\frac{\tilde{T}(x)}{\delta_2}$. Let $\Delta(x) = \tilde{T}(x) - T(x)$.

$$\begin{aligned}
\mathop{\mathrm{E}}_x\left(f_1(x)\tfrac{\tilde{T}(x)}{\delta_2}\right) &= \mathop{\mathrm{E}}_x\left(f_1(x)\tfrac{T(x)+\Delta(x)}{\delta_2}\right) \\
&= \mathop{\mathrm{E}}_x\left(\tfrac{f_1(x)T(x)}{\delta_2}\right) + \mathop{\mathrm{E}}_x\left(\tfrac{f_1(x)\Delta(x)}{\delta_2}\right) \\
&\geq \tfrac{\delta_1\delta_2+\epsilon}{\delta_2} - \tfrac{\epsilon}{\delta_2} \\
&= \delta_1.
\end{aligned}$$

We thus have the desired violation of correlation, but the function we have defined is not Boolean. We now round the values $\tilde{T}(x)$ to values in $\{-1,1\}$ in a way that preserves correlation. Consider a randomized circuit doing the following on input x: Compute $\tilde{T}(x)$ and with probability $(\frac{\tilde{T}(x)}{\delta_2}+1)/2$ output 1, otherwise output -1. This circuit has the same expected correlation with f_1 as $\tilde{T}/\delta_2$. Now freeze the random choices made, while preserving the correlation. We now have a deterministic circuit of size at most $s_1 - 5(n_1/\epsilon)^{10} + 5n_1^2 \leq s_1$ with correlation at least δ_1 with f_1, contradicting the assumption about f_1. ■

Proof (of Lemma 39) By induction in k using the isolation lemma.

Induction step (assume true for $k-1$): Let $f_1 = f$ and $f_2 = f^{(k-1)}$, $s_1 = s$ and $s_2 = \frac{s-10(\frac{n}{\epsilon})^{10}}{\left(\frac{n}{\epsilon}\right)^{10}}$. We conclude by Lemma 40 that all circuits of size at most $\frac{s-10(\frac{n}{\epsilon})^{10}}{\left(\frac{n}{\epsilon}\right)^{10}}$ will have correlation at most

$$\delta\left(\delta^{k-1} + (k-1)\epsilon\right) + \epsilon = \delta^k + \delta(k-1)\epsilon + \epsilon \leq \delta^k + k\epsilon$$

with $f^{(k)}$. ■

7. A HIGH END GENERATOR

In this section, we prove Theorem 14. By now, there are several known ways of doing this. First observe that if one sets the parameters of the proof in Section 6 carefully, and in particular replaces Lemma 34 with Lemma 38 one gets very close to obtaining a proof of Theorem 14. In fact, one can prove the theorem with the conclusion **BPP** = **P** replaced with **BPP** $\subseteq$ **TIME**$(2^{\text{polylog} n})$. The only step in the proof causing the superpolynomial blowup is Lemma 35, the XOR lemma. To transform a truth table of a function so that all small circuits fails to compute the function on a constant fraction of the inputs into a truth table of a function so that all small circuits fail to compute the function on almost half the inputs, the XOR lemma as stated needs to blowup the size of the truth table quasipolynomially.

In the first proof of Theorem 14, by Impagliazzo and Wigderson, 1997, the use of the XOR lemma was replaced with a modified XOR lemma, where the function $f^{(k)}$ is not defined on *all* k-tuples of inputs, but only on a carefully selected subset. This makes a polynomial blowup possible. The proof is technically involved.

Two simpler proofs were given by Sudan et al., 1999. The second of the proofs consists of extending the notion of locally decodable codes to "locally list-decodable codes", thus eliminating the use of the second step altogether and obtaining a proof very close to the proof of Theorem 9.

The proof we shall present in this section is a further simplification of the first proof presented by Sudan et al., 1999. The proof is based on the second strategy for constructing hardness based generators we outlined in Section 5.3: We use the extractor of Theorem 9 as a black box in the construction and try to "add on" the generator property. Following this approach, Sudan et al., 1999 obtain a pseudorandom generator. The simpler version we present here only gives a hitting set generator.

This makes it possible to eliminate one of the steps performed in the construction of Sudan et al., 1999.

In Subsection 7.1, we prove an important lemma for the construction, Impagliazzo's hardcore lemma. In Subsection 7.2, we present the proof of Theorem 14.

7.1 THE HARDCORE LEMMA

Impagliazzo's hardcore lemma, due to Impagliazzo, 1995, states that any function f which disagrees with any small circuit on a constant fraction of the inputs, has a *hardcore*. Informally speaking, the hardcore is a set of hard instances of the function. More precisely, the hardcore is a set of inputs H of constant density so that any small circuit must fail to compute f on almost half the instances of H. The proof we present is attributed to Nisan in Impagliazzo, 1995. For much more information about the hardcore lemma and other proofs, see this reference, and Klivans and Servedio, 1999.

Lemma 41 *Let $0 < \epsilon < \frac{1}{10}$ be a fixed constant and let n be sufficiently large. Given a function $f : \{0,1\}^n \to \{0,1\}$ so that all circuits of size $2^{\epsilon n}$ compute f incorrectly on at least a $\frac{1}{20}$ fraction of the inputs. Then there is a set $H \subseteq \{0,1\}^n$ of density at least $\frac{1}{25}$, so that all circuits of size $2^{(\epsilon/2)n}$ compute f incorrectly on at least a $\frac{1}{2} - \frac{1}{s(n)}$ fraction of H, where s is a superpolynomial function. The set H is called a* hardcore *for f.*

Proof We consider the following two-player zero-sum game between Alice and Bob. Alice chooses a circuit C of size $2^{(\epsilon/2)n}$. Simultaneously, Bob chooses a subset S of $\{0,1\}^n$ of density at least $\frac{1}{20}$. The payoff of Alice is the probability, over a uniformly chosen input $x \in S$, that $C(x) = f(x)$. The payoff of Bob is this value negated.

Intuitively, a good deterministic move for Bob would be a hardcore set. On the other hand, it is easy to see that Alice does not have a good deterministic move: Whatever circuit Alice chooses, Bob can choose a set, so that the payoff of Alice is 0.

According to the min-max theorem of game theory, the game has a value, say v, so that there is a *randomized* strategy for Alice ensuring that her expected payoff is at least v no matter what Bob does, and there is *randomized* strategy for Bob ensuring that his expected payoff is at least $-v$, no matter what Alice does.

Our aim is to show that if $v > \frac{1}{2} + n^{-\log n}$, then there is a circuit of size $2^{\epsilon n}$ computing f incorrectly on less than a $\frac{1}{20}$ fraction of the inputs, contradicting the assumption of the lemma. On the other hand, we'll show that if $v \leq \frac{1}{2} + n^{-\log n}$, Bob's optimal randomized strategy can be made deterministic, yielding a hardcore set of the desired parameters.

First assume $v > \frac{1}{2} + n^{-\log n}$ and consider the optimal strategy of Alice. The strategy is a probability distribution over circuits of size $2^{(\epsilon/2)n}$. We must derive from it a single, deterministic circuit $\tilde{C}$ of size at most $2^{\epsilon n}$ computing f "too well". We construct this deterministic circuit probabilistically: We sample $2^{(\epsilon/4)n}$ circuits from the distribution of the optimal strategy of Alice and take their majority. This is the circuit $\tilde{C}$. We now argue that it, with high probability, has the correct property.

To see this, we partition $x \in \{0,1\}^n$ in two parts, G and B. G is the set of strings which are good for the strategy of Alice: $x \in G$ if and only if a random circuit C according to her strategy has $C(x) = f(x)$ with probability at least $\frac{1}{2} + n^{-\log n}$. B is the set of strings which are bad for the strategy of Alice, i.e., the rest of the strings. We argue that $\tilde{C}$ does well on G and that B is so small that it doesn't matter how well $\tilde{C}$ does on B.

First we argue that B is small, in particular that the density of B is less than $\frac{1}{20}$: Take the $\frac{2^n}{20}$ strings in $\{0,1\}^n$ on which the strategy of Alice does worst. If Bob plays this set, the expected payoff for Alice is still at least $\frac{1}{2} + n^{-\log n}$. Thus the set contains some element *not* in B and hence *all* of B as a proper subset.

Next fix any $x \in G$. By the definition of G, a random circuit according to the strategy of Alice answers correctly on x with probability at least $\frac{1}{2} + n^{-\log n}$. By the Chernoff bound of Lemma 1, the probability that $\tilde{C}$ is correct on x is overwhelming, certainly, it is bigger than $1 - 2^{-n}$. Thus, we can fix a choice of $\tilde{C}$ which is correct on all members of G. This circuit has size at most $2^{\epsilon n}$ and computes f except on B, a set of density less than $\frac{1}{20}$, contradicting the assumption about f. Thus the value of the game is less than $\frac{1}{2} + n^{-\log n}$.

Now consider an optimal strategy of Bob. Consider the distribution $\mathcal{D}$ on $\{0,1\}^n$ defined by picking a random set according to the strategy of Bob and then a random element of the set. For a randomly chosen element according to $\mathcal{D}$, any fixed circuit of Alice answers correctly with probability less than $\frac{1}{2} + n^{-\log n}$. After the set has been chosen, any particular element has probability at most $(2^n/20)^{-1}$ of being chosen, so $\mathcal{D}$ assigns probability at most $20/2^n$ to any particular element.

We now construct the desired hardcore set H probabilistically. For each $x \in \{0,1\}^n$, we make an independent decision about whether to include $x \in H$. We include x with probability $2^n\mathcal{D}(x)/20$, which is at most 1, by the above analyses.

By the Chernoff bound of Lemma 1, the probability that H has density somewhere in the interval $\frac{1}{20} \pm n^{-\log n}$ is overwhelming. Given any fixed particular circuit C of Alice, we have $\sum_{x\in\{0,1\}^n} \mathcal{D}(x)\tau_C(x) \leq \frac{1}{2} + n^{-\log n}$, where $\tau_C(x)$ is 1 if $C(x) = f(x)$ and 0 otherwise. Thus, $\sum_{x,\tau_C(x)=1} \mathcal{D}(x) \leq \frac{1}{2} + n^{-\log n}$. Therefore, the probability that the density of the set $H \cap \{x|\tau_C(x) = 1\}$ is at most $(\frac{1}{2} + n^{-\log n})(\frac{1}{20} + n^{-\log n})$ is overwhelming, certainly bigger than $1 - 2^{2^{-n/5}}$. Therefore the probability that the density of $H \cap \{x|\tau_C(x) = 1\}$ is smaller than this value for *all* circuits C of Alice is also overwhelming. In particular, we can fix a set H with these properties. This is the hardcore set. ■

7.2 THE GENERATOR

We now prove theorem 14. By Lemma 30, it is enough to show that the assumption of the theorem implies that there is a generator which on input r runs in time polynomial in r and outputs a set $W \subseteq \{0,1\}^r$ which is a $(1 - 2^{-r+r^{1/2}})$-hitting set for circuits of size r^q, for a certain constant q, which we assume, without loss of generality, to be bigger than 2.

The assumption of the theorem allows us, on input n, to generate the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ with no circuits of size $2^{\epsilon n}$, in polynomial time in the size of the truth table. By Lemma 38, we can convert this generator into a generator which on input n gives us the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ so that any circuit of size $2^{\epsilon' n}$ disagrees with f on at least a $\frac{1}{20}$ fraction of the input.

We now define the hitting set generator. Given the parameter r, the generator uses Lemma 27 to construct an $(r, c\log r, 4q\log r/\epsilon', \log r)$-Nisan-Wigderson design $S_1, S_2, \ldots, S_r$, where c is a constant depending on q and ϵ'. Letting $n = 4q\log r/\epsilon'$, the generator builds the truth table of a function $f : \{0,1\}^n \to \{0,1\}$ so that any circuit of size $2^{\epsilon' n}$ disagrees with f on at least a $\frac{1}{20}$ fraction of the inputs. The time used is polynomial in 2^n, i.e., polynomial in r. Now, the generator outputs the set

$$W = \{f(y_{S_1})f(y_{S_2})\ldots f(y_{S_r})|y \in \{0,1\}^{c\log r}\}.$$

We claim this is the desired hitting set.

Assume, to the contrary that $C : \{0,1\}^r \to \{0,1\}$ is a circuit of size r^q, so that if $Z = \{x \in \{0,1\}^r | C(x) = 0\}$, we have $|Z| \leq 2^{r^{1/2}}$, and $W \subseteq Z$.

By Lemma 41, f has a hardcore $H \subset \{0,1\}^n$ of density at least $\frac{1}{25}$, so that all circuits of size $2^{(\epsilon'/2)n} = r^{2q}$ fail to compute f on a $\frac{1}{2} - \frac{1}{s(n)}$ fraction of H, where s is a superpolynomial function.

Let $\mathcal{D}$ be the uniform distribution on W and consider the following derived distributions $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_r$, defined by the following sampling procedures.

A random sample from $\mathcal{D}_i$ is obtained by picking a random sample x from $\mathcal{D}$, together with the "history" of x, i.e., the seed $y \in \{0,1\}^{c \log r}$ so that $x = f(y_{S_1})f(y_{S_2}) \ldots f(y_{S_r})$. Looking at y, we identify those bits among $x_{i+1}, \ldots, x_r$, i.e., among the last $r - i$ bits of x, that are the results of applying the function f to elements of the hardcore H. *Each of these "hardcore bits" we replace with an unbiased, independent random bit.*

Note that $\mathcal{D}_r = \mathcal{D}$ and $\mathcal{D}_0$ is the distribution resulting from $\mathcal{D}$ when we replace every "hardcore bit" with a randomly chosen bit.

We immediately have that for a randomly chosen sample $x \in \mathcal{D}_r$, $\Pr[C(x) = 1] = 0$.

We also have that for a randomly chosen sample $x \in \mathcal{D}_0$, the probability of $\Pr[C(x) = 1] \geq \frac{1}{100}$. This can be seen as follows: The procedure for sampling $x \in \mathcal{D}_0$, first samples $x' \in \mathcal{D}$. As the hardcore H has density at least $\frac{1}{25}$ in $\{0,1\}^{m'}$, a randomly chosen bit in such a random sample x' will be a "hardcore bit" bit with probability at least $\frac{1}{25}$. Thus, with probability at least $\frac{1}{50}$, more than a $\frac{1}{50}$ fraction of the bits of the sample are hardcore bits. Call such a sample *good*. Then, if we fix our attention to a single good sample, the conditional distribution we get when we replace the hardcore bits of the sample with randomly chosen bits, is the uniform distribution on a subset of $\{0,1\}^r$ of size at least $2^{r/50}$. At most $2^{r^{1/2}}$ of these make C reject. Thus, the conditional probability that $C(x) = 1$, given that x comes from a good sample, is $1 - o(1)$, so the probability that $C(x) = 1$ for a randomly chosen sample $x \in \mathcal{D}_0$, is at least $(1 - o(1))\frac{1}{50} \geq \frac{1}{100}$.

As $\Pr[C(x) = 1] = 0$ if x is chosen at random from $\mathcal{D}_r$ and $\Pr[C(x) = 1] \geq 1/100$ if x is chosen at random from $\mathcal{D}_0$, the triangle inequality gives us an i, so that

$$\Pr_{x \in \mathcal{D}_i}[C(x) = 1] - \Pr_{x \in \mathcal{D}_{i+1}}[C(x) = 1] \geq \frac{1}{100(r+1)}$$

Note that a sample from $\mathcal{D}_i$ can be obtained from a sample from $\mathcal{D}_{i+1}$ by replacing the $i+1$'st bit with a random bit if it is a hardcore bit.

Samples from $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ are generated from a seed y. Let $\mathcal{D}'_i$ and $\mathcal{D}'_{i+1}$ be the distributions from $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ obtained by freezing all bits of y, *except* the bits of S_{i+1} to fixed values, while preserving

$$\Pr_{x \in \mathcal{D}'_i}[C(x) = 1] - \Pr_{x \in \mathcal{D}'_{i+1}}[C(x) = 1] \geq \frac{1}{100(r+1)}$$

As $\mathcal{D}'_i$ and $\mathcal{D}'_{i+1}$ only differ when $y_{S_{i+1}}$ is a hardcore bit, we also have,

$$\Pr_{x \in \mathcal{D}'_i}[C(x) = 1 | y_{S_{i+1}} \in H] - \Pr_{x \in \mathcal{D}'_{i+1}}[C(x) = 1 | y_{S_{i+1}} \in H] \geq \frac{1}{100(r+1)}.$$

Now, we describe a small circuit C' for f which will compute f correctly on significantly more than half its hardcore, and thus we shall reach a contradiction.

In the description and analysis of the circuit C', we shall assume that the input $z \in \{0,1\}^r$ given to the circuit is indeed a member of the hardcore. What happens on other inputs is irrelevant.

The circuit C' generates, on input $z \in H$, a random value x according to the distribution $\mathcal{D}'_{i+1}$, conditioned by $y_{S_{i+1}} = z$, where $y \in \{0,1\}^{c \log r}$ is the seed leading to x. We claim that this can be done by a circuit of size $O(r^3)$. This follows from the fact that each set S_j with $j \neq i$ intersects S_{i+1} in at most $\log n$ places and we can thus tabulate the value of $f(y_{S_j})$ as a function of $y_{S_j \cap S_{i+1}}$. We can also tabulate the Boolean predicate $y_{S_j} \in H$? as a function of $y_{S_j \cap S_{i+1}}$. The total size of these tables are $O(r^2)$ and a random value x according to $\mathcal{D}'_{i+1}$, conditioned by $y_{S_{i+1}} = z$, is easily generated from them. Now our circuit checks if $C(x) = 1$. If this is the case, it outputs $\neg x_{i+1}$. Otherwise it outputs x_{i+1}. If z is randomly chosen from the hardcore, we have that $C'(z) = f(z)$ with probability at least $\frac{1}{2} + \frac{1}{100(r+1)}$, as $\Pr_{x \in \mathcal{D}'_i}[C(x) = 1 | y_{S_{i+1}} \in H] - Pr_{x \in \mathcal{D}'_{i+1}}[C(x) = 1 | y_{S_{i+1}} \in H] \geq \frac{1}{100(r+1)}$. We can freeze the random choices made by C' to get a deterministic circuit with the same property. This contradicts H being a hardcore of f.

Thus, our assumption that W did not hit C was false, so W is a hitting set with the desired parameters. This completes the proof of Theorem 14.

8. THE COMPRESSION APPROACH TO GENERATOR CONSTRUCTION

In this section, we describe an alternative approach to hardness-based hitting set generators, developed in Andreev et al., 1997, and Miltersen and Vinodchandran, 1999. It is somewhat simpler than the approach using locally decodable codes and Nisan-Wigderson designs and yields comparable results.

The first generator, due to Andreev et al., 1997, takes as input a table of a *multi-output* Boolean function with *maximum* circuit complexity and produces a $(1 - \frac{1}{r^3})$-hitting set for circuits of size r. Such a hitting set derandomizes **pRP** and hence **BPP**. Furthermore, we do not have to rely on explicit dispersers and Lemma 29 to see that: An error probability of $1/r^3$, using r random bits, can be obtained by simple repetition, so a simple modification of Lemma 2 will do.

Since the requirement for the hardness of the function that is given as input is so harsh, we see that it does *not* follow from the techniques of Section 5 that the generator is a disperser with non-trivial parameters. Indeed it isn't. A fortunate side-effect of this is that the correctness of the generator can be given a completely elementary one-page self-contained proof, unlike any of the other hitting set generators we have seen.

Theorem 18 *If there is an algorithm which on input n, m outputs the truth table of a Boolean function from $\{0,1\}^n$ to $\{0,1\}^m$ with maximum possible circuit complexity $M_{n,m}$, and runs in time polynomial in the length of its output, then there is a $(1 - 1/r^3)$-hitting set generator for circuits $C : \{0,1\}^r \to \{0,1\}$ of size r, running in time $r^{O(1)}$, and hence* **P=BPP**.

Proof Let $f_{n,m}$ be the function generated on input n, m. Fix r. Let I_i with $0 \leq i \leq r$ be the subset of $\{0,1\}^r$ obtained by discarding the last i bits of each of the elements of $\{f_{2\log r, r+i}(x) | x \in \{0,1\}^{2\log r}\}$. We shall argue that at least one of the sets I_i is a $(1 - 1/r^3)$-hitting set for circuits of size r. Thus, the union of these sets is a hitting set, and as their union can be generated in polynomial time, we are done.

Assume, to the contrary, that no set I_i is such a hitting set. In particular this is true for $i = 0$, i.e., for the image I of the function $f = f_{2\log r, r}$. Thus, there is a circuit of size r mapping r bits to 1 bits, so that if we let $Z = \{y \in \{0,1\}^r | C(y) = 0\}$, we have $|Z| \leq 2^r/r^3$ and $I \subseteq Z$.

Now let $\mathcal{H}$ be the universal class of hash functions of Lemma 7, mapping $\{0,1\}^r$ to $\{0,1\}^{r-1}$. Consider picking an $h \in \mathcal{H}$ at random. We have

$$\Pr_h[\exists x \in I, y \in Z : x \neq y \wedge h(x) = h(y)] \leq r^2 \cdot 2^r/r^3 \cdot 2^{r-1} < 1$$

Thus, there exists $h \in \mathcal{H}$, so that $\forall x \in I, y \in Z, x \neq y \Rightarrow h(x) \neq h(y)$. Fix such h. Let $f' : \{0,1\}^{2\log r} \to \{0,1\}^{r-1}$ be the map given by $f'(x) = h(f(x))$.

Now suppose we are given a circuit for f'. We can compute $f(x)$ by computing $f'(x)$, computing $\{y_1, y_2\} = h^{-1}(f'(x))$ and then returning whichever y_i has $C(y_i) = 0$. The right value (i.e., $f(x)$) has this property (as $I \subseteq Z$) and since h does not collide I and Z, the other value does not have this property.

As the circuit complexity of f is $M_{2\log r, r}$ by assumption, and the circuit complexity of f' is at most $M_{2\log r, r-1}$, we have

$$M_{2\log r,r} \leq M_{2\log r,r-1} + O(r)$$

Arguing similarly for each function $f_{2\log r, r+i}$, $i = 1..n$, we get

$$M_{2\log r,r+i} \leq M_{2\log r,r+i-1} + O(r),$$

i.e.

$$M_{2\log r,2r} \leq M_{2\log r,r} + O(r^2),$$

contradicting Theorem 1. ■

The assumption that a function with maximum circuit complexity can be generated efficiently is very strong. To see how we can get the proof to work with an assumption more like the assumption in the Impagliazzo-Wigderson theorem, it is useful to visualize the set I in the proof above as an $r^2 \times r$ 0-1 matrix. The proof works by *compressing* each row of the matrix by 1 bit in a way that makes it possible to retrieve each row again efficiently from the compressed image. This is inconsistent with the function corresponding to I having maximum circuit complexity.

Now, to get an assumption involving smaller circuits, we need to be able to compress each row by much more than one bit. The current compression is possible if the set I is not a $(1 - 1/r^3)$-hitting set. To do really significant compression, we need a stronger assumption about the non-hitting property of the set I: We need to assume that it is not a $(1-\epsilon(r))$-hitting set for $\epsilon(r)$ much closer to 0 than r^{-3}. But this is not a problem: Lemma 30 tells us that to derandomize **BPP**, it is enough to construct a hitting set which is $(1-2^{-r+r^\epsilon})$-hitting for circuits of size r^q for a constant q, depending on ϵ. This enables us to compress each row from r bits all the way down to r^ϵ bits. However, a more serious problem arises: If we compress by, say, t bits, the circuit we end up constructing for the function f needs to do an exhaustive search through 2^t different possibilities for the uncompressed value it is trying to compute. Thus, if we compress each row from r bits to r^ϵ bits, $t = r - r^\epsilon$, and we would have to search through 2^{r-r^ϵ} possibilities. This would make us lose all we had gained from the bigger compression.

A way out of this dilemma is to switch to a hardness assumption involving a more powerful kind a circuits, namely *nondeterministic* circuits. If the circuit we are trying to construct may be nondeterministic, we can easily perform the exhaustive search above by a simply *guessing* the decompressed value and verifying its correctness.

The above discussion leads to the following theorem, also proved by Andreev et al., 1997, though phrased somewhat differently there.

Theorem 19 *There is a $\delta > 0$ so that the following holds. If there is a language L in $\mathbf{E}$, so that for all n, the nondeterministic circuit complexity of $L \cap \{0,1\}^n$ is at least $2^{(1-\delta)n}$, then* $\mathbf{BPP} = \mathbf{P}$.

Proof By Lemma 30, it is enough to show that there is a generator which on input r runs in time polynomial in r and outputs a set $I \subseteq \{0,1\}^r$ which is a $(1 - 2^{-r+r^{1/3}})$-hitting set for circuit of size r^q, for a certain constant q. We can assume that $q \geq 2$. The value of δ for which we will show the theorem will be $\delta = \frac{1}{2q}$.

The assumption allows us, on input l, to generate the truth table of a Boolean function $f : \{0,1\}^l \to \{0,1\}$ with nondeterministic circuit complexity at least $2^{(1-\delta)l}$, in time polynomial in the size of the table.

Given the parameter r, we let $l = \log(r^{q+1})$ and generate the truth table of a Boolean function $f : \{0,1\}^l \to \{0,1\}$. We arrange the 2^l bits of the truth table in an $r^q \times r$ matrix. Let I be the set of rows of this matrix. We claim that I is the desired hitting set, if the parameter r is sufficiently large.

Suppose not. Then there is a circuit C of size r^q mapping r bits to 1 bit, so that if we let $Z = \{y \in \{0,1\}^r | C(y) = 0\}$, we have $|Z| \leq 2^{r^{1/3}}$ and $I \subseteq Z$.

Now let $\mathcal{H}$ be the universal class of hash functions of lemma 6, mapping $\{0,1\}^r$ to $\{0,1\}^{2r^{1/3}}$. Consider picking an $h \in H$ at random. The probability that h introduces any collisions on the set Z is smaller than 1, so we can pick an h so that $\forall x, y \in Z, x \neq y \Rightarrow h(x) \neq h(y)$.

Let $f' : \{1, 2, \ldots, r^q\} \to \{0,1\}^{2r^{1/3}}$ be the map which, given the index of a row in I, returns the hashed value of this row, using the hash function h.

Now suppose we are given a table for f'. We can construct a nondeterministic circuit for the original Boolean function f as follows. Given an input y, we first compute the index of the row of A where we can find the value $f(y)$. Then we look up the hashed value of the row in the table for f'. We then *guess* the value $z \in \{0,1\}^r$ of the original row and verify that our guess is correct by checking that $C(z) = 0$ and that $h(z)$ has the right value. We then read off the relevant bit of z.

The nondeterministic circuit for f we have just described contains C, a circuit for h, and a table for f'. Beyond that, only straightforward computation is performed. Thus, we can make its size $O(r^{q+o(1)} + r^q r^{1/5}) = O(r^{q+1/5})$. This is smaller than $2^{(1-\delta)l}$ for sufficiently large r. ∎

Note that the assumptions of Theorem 18 and Theorem 19 are incomparable - and both much stronger than the Impagliazzo-Wigderson assumption. However, the assumption of Theorem 19 seems somewhat closer in spirit to the Impagliazzo-Wigderson assumption than the assumption of Theorem 18. Next, we will show that we can make it even closer, by replacing $2^{(1-\delta)n}$, for *some* small constant δ, with $2^{\epsilon n}$, for *any* desired constant $\epsilon > 0$.

To do this, we need to compress our matrix even more. We have already compressed each row more or less as much as we can, as the disperser we used was close to optimal. To compress it even more, we are going to compress it twice: *First we will compress each row, and then we will compress each column of the compressed matrix.* Now, there is no longer any reason for having the matrix having more columns than rows, so in the following discussion, we should have a square $r \times r$ matrix I in mind. To compress the matrix twice, the set we are going to assume is not hitting will be the set of rows *and* columns of this matrix. The proof should then proceed by first finding a hash function and use it to compress the rows of the matrix. This will be possible under the assumption that the set of rows is not a hitting set. Next, we will take our compressed matrix and now, under the assumption that the set of columns is not a hitting set, we will find a hash function and use it to compress the columns of the compressed matrix.

However, here a problem arises: *The columns of the row-compressed matrix have no relationship to the columns of the original matrix.* Indeed, applying a traditional hash function to a piece of data destroys all structure in the data completely. Thus, we *cannot* compress the columns of the row-compressed matrix under the assumption that the set of columns of the original matrix is not a hitting set, and we seem to be stuck.

To proceed, we need to be able hash the rows of a matrix, *without destroying the columns.* Fortunately, the family of hash function we use need only be weakly universal. Thus, Theorem 15 gives us a solution to the problem: If the rows of the matrix have been encoded in an error correcting code, we can use a projection to a small set of indices as our hash function. Applying such a hash function to all rows of a matrix yields a compressed matrix whose set of columns is just some subset of the columns of the original matrix. Thus, under the assumption that the set of columns of the original matrix is not a hitting set, we *can* compress the columns of the matrix, after having compressed the rows.

To get a hardness assumption involving circuits of size $2^{\epsilon n}$ for an arbitrary small constant $\epsilon > 0$, we need to carry out the above program, not on a square $n \times n$ matrix, but on a k-dimensional matrix of bits, for a constant value k, depending on ϵ and compress not twice, but k times, once along each dimension of the matrix. Thus, we need to encode our original data as a k-dimensional matrix of bits, so that each 1-dimensional slide of the encoding is a code word of an error correcting code e with good minimum distance. A simple way of doing this is to take the *tensor product* of our code e of Lemma 12 with itself k times. We omit the technical details of this and only state the following lemma.

Lemma 42 *There is an efficiently computable function taking as input a k-dimensional bit matrix $f : \{1, 2, \ldots, n\}^k \to \{0, 1\}$ and outputting a k-dimensional bit matrix $\tilde{f} : \{1, 2, \ldots, n^{10}\}^k \to \{0, 1\}$, so that*

- $\tilde{f}[i_1, \ldots, i_k] = f[i_1, \ldots, i_k]$, *for $i_1, \ldots, i_k \leq n$, and*
- *each one-dimensional* slice *of the matrix $\tilde{f}$, i.e. each vector $v = (\tilde{f}[i_1, i_2, \ldots, i_{j-1}, x, i_{j+1} \ldots, i_k])_{x=1..n^{10}}$, for any setting of j and $i_1, i_2, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k$, is a code word of e.*

Now we are ready for a more formal version of the proof sketch carried out above. The proof is due to Miltersen and Vinodchandran, 1999.

Theorem 20 *If there is an $\epsilon > 0$ and a language L in* **E***, so that for all n, the nondeterministic circuit complexity of $L \cap \{0, 1\}^n$ is at least $2^{\epsilon n}$, then* **BPP** $=$ **P**.

Proof The assumption allows us, on input n, to generate the truth table of a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ with nondeterministic circuit complexity at least $2^{\epsilon n}$, in time polynomial in the size of the table.

We show that such a procedure allows us to construct a $1 - 2^{-r+r^{\epsilon'}}$-hitting set generator for circuits $C : \{0, 1\}^r \to \{0, 1\}$ of size r^q, where $\epsilon' = \epsilon/100$ and where q is *any* fixed integer. By Lemma 30, we'll be done.

Given r, we use our procedure to generate the truth table of a function $f : \{0, 1\}^m \to \{0, 1\}$, where $m = \frac{k}{10} \log r$, and k is a constant to be decided later, and arrange the bits of the truth table in a k-dimensional matrix. More precisely, we identify f with a function $f : (\{0, 1\}^{\frac{\log r}{10}})^k \to \{0, 1\}$ in some canonical way. Using Lemma 42, we encode f as $\tilde{f} : \{1, 2, \ldots, r\}^k \to \{0, 1\}$.

Now let H be the set of all one-dimensional slices of $\tilde{f}$, i.e., the members of H are the vectors $\tilde{f}[i_1, i_2, \ldots, i_{j-1}, x, i_{j+1} \ldots, i_k])_{x=1..r}$ for any setting of j and $i_1, i_2, \ldots, i_{j-1}, i_{j+1}, \ldots, i_k$.

We now claim that H is a $(1 - 2^{-r+r^{\epsilon'}})$-hitting set for circuits of size r^q, if we set k appropriately.

Assume not. Let C be a circuit establishing this, i.e., C maps $\{0,1\}^r$ to $\{0,1\}$, it has size r^q, and if we by Z denote $\{x \in \{0,1\}^r | C(x) = 0\}$, then $|Z| \leq 2^{r^{\epsilon'}}$ and $H \subseteq Z$.

By Lemma 15, there is a projection $\pi_S : \{0,1\}^r \to \{0,1\}^{r^{2\epsilon'}}$ to a set of indices $S \subseteq \{1, 2, \ldots, r\}$, so that π_S is 1-1 on $Z \cap W$, where W is the set of code words of e.

We will now construct a small nondeterministic circuit for $\tilde{f}$ and hence f. Into the circuit we build as advice the circuit C, the set S, and a table of the restriction of $\tilde{f}$ to S^k.

To compute $\tilde{f}(a_1, a_2, \ldots, a_k)$ the procedure does as follows.

For every element $u \in S^{k-1}$ it *guesses* the vector $v = (\tilde{f}(j,u))_{j\in\{1,\ldots,r\}}$. It checks that its guess is correct by checking that v is a code word of e, checking that $C(v) = 0$, and checking that the entries of v corresponding to indices in S are correct, by consulting the table for $\tilde{f}$, restricted to S^k. By construction, the value $v = (\tilde{f}(j,u))_{j\in\{1,\ldots,r\}}$ is the only value with these properties. After having ensured that v is the correct value, it keeps the value $\tilde{f}(a_1, u)$ and throws away the rest of the vector v. Having done this for every possible u, the procedure has now built a table of $\tilde{f}$, restricted to $\{a_1\} \times S^{k-1}$.

Now, for every element $u \in S^{k-2}$ the procedure guesses the vector $v = (\tilde{f}(a_1, j, u))_{j\in\{1,\ldots,r\}}$. It checks that each guess is correct by checking that v is a code word, checking that $C(v) = 0$ and checking that the entries of v corresponding to indices in S are correct, by consulting the table for $\tilde{f}$, restricted to $\{a_1\} \times S^{k-1}$. After having ensured that v is the correct value, it keeps the value $\tilde{f}(a_1, a_2, u)$ and throws away the rest of the vector v. Having done this for every possible u, the procedure has now built a table of $\tilde{f}$, restricted to $\{a_1\} \times \{a_2\} \times S^{k-2}$.

Now it goes through a similar loop for every element $u \in S^{k-3}$, building a table of $\tilde{f}$, restricted to $\{a_1\} \times \{a_2\} \times \{a_3\} \times S^{k-3}$, and so on, until in the end it has the value of $\tilde{f}(a_1, a_2, \ldots, a_k)$ which was the value we wanted.

Converting the procedure into a nondeterministic circuit, building in the advice, we get a circuit of size at most $O(k(r^{3\epsilon'})^k r^q)$.

By fixing k to an appropriate constant, this is smaller than $2^{\epsilon m}$ for sufficiently large m. As a circuit for $\tilde{f}$ can also be used as a circuit for f, this contradicts the assumption on f. ■

Though the compression proof technique falls just short of proving the Impagliazzo-Wigderson theorem, it can be used to prove a derandomization result for the class **AM** we do not at present know how to prove using the techniques based on locally decodable codes and Nisan-Wigderson designs. This theorem is described in the next section.

9. OTHER HARDNESS BASED DETERMINISTIC SIMULATION RESULTS

In this section, we discuss hardness based deterministic simulation results for other randomized complexity classes, besides **BPP** and **RP**.

In general, the derandomization theorems we consider will have the form "If **A** contains a language not in **B** then **C** is contained in **D**", where **C** is the randomized class we are considering, **D** is the deterministic, uniform class in which we want to do the deterministic simulation, **A** is a uniform complexity class, and **B** is the class of problems infinitely often having a certain complexity in a non-uniform model.

The proofs we are considering proceed by adapting the proof techniques of Sections 6, 7 and 8. Thus, we first identify a appropriate non-uniform class of computational objects so that making pseudorandom or hitting set generators for objects of this class derandomizes **C**. This step is usually routine. For instance, for **RP** and **BPP**, the relevant objects are *circuits*, as shown in Lemma 2. For $\mathbf{BP_HL}$, they are *finite automata*, as shown in Lemma 3.

Now, given a class **A**, we consider constructing a pseudorandom or hitting set generator using a language of **A** by the same transformations as in Section 6, 7 or 8, or using one of alternative strategies indicated there. The complexity of a deterministic machine, going through all outputs of such a generator and using them as the random bits in a computation of type **C**, defines the class **D**. Getting the class **D** to be the desired one may require some fine-tuning of the construction of the generator.

To see which class **B** can be obtained, we assume that the pseudorandom generator does not work, i.e. that there is a non-uniform computational object of appropriate type on which it fails. Then we redo the sequence of lemmas of the last sections (or a similar sequence following one of the alternative strategies) that proves the correctness of the generator, with these objects replacing circuits, and get some kind of non-uniform simulation of **A**. This defines the class **B**.

We present a sample of results obtained this way. For several other variations, we refer to the papers cited below.

We have only worried about the time-efficiency of the pseudorandom generator. To derandomize space bounded computation, one needs to check that the computation of the pseudorandom generator can be done space-efficiently. This was done by Allender et al., 1999. This leads to the following theorem, a slightly weaker version of which is stated in Klivans and van Melkebeek, 1999.

Theorem 21 *If there is a language L in* **LINSPACE** *and an $\epsilon > 0$, so that for all n, no branching program of size $2^{\epsilon n}$ decides $L \cap \{0,1\}^n$, then* $\mathbf{BP_H L}$=**L**

For definition and discussion of the branching program model, see, e.g., Boppana and Sipser, 1990. One could perhaps think that since we can actually prove that certain explicit functions are not decided by small automata, it might be possible to construct a hardness based generator using no unproven assumption. However, the class of functions computed by small automata lacks nice closure properties, so if we try to let **B** be the class of functions requiring big automata, we find that we cannot make the necessary lemmas go through. In particular, the analogy of Lemma 33 would seem to fail. The class of functions having small branching programs is, essentially, the closure of the class of functions having small automata under the operations necessary to carry out the steps of the various constructions.

Since the computation of the pseudorandom generators can be done space-efficiently, they can also be done by shallow circuits. The class **BPNC** is the class of problems decided by a uniform family of two-sided error randomized circuits of polynomial size and polylogarithmic depth. Nisan, 1991, obtains the following result.

Theorem 22 *If there is a language L in* **PSPACE** *and an $\epsilon > 0$, so that for all n, the circuit depth of $L \cap \{0,1\}^n$ is at least n^ϵ, then* **BPNC** *is contained in* **SPACE(**polylog n**).**

Furthermore, the following theorem is implicit in Impagliazzo and Wigderson, 1997.

Theorem 23 *If there is a language L in* **LINSPACE** *and an $\epsilon > 0$, so that for all n, the circuit size of $L \cap \{0,1\}^n$ is at least $2^{\epsilon n}$, then* **BPNC** = **NC**.

Let $\mathbf{BPAC}^0$ be the class of languages decided by a uniform family of two-sided error randomized unbounded fan-in circuits of polynomial size and *constant* depth. To derandomize $\mathbf{BPAC}^0$ we need to construct pseudorandom generators for non-uniform $\mathbf{AC}^0$ circuits. This can be done using no unproven assumptions, as there are easily computed functions, such as the *parity* function, so for which all small constant depth circuits must err on almost half the inputs. Thus, a hardness based generator can be built along the lines of Section 6, and we can even skip the two first steps of the construction. Thus, Nisan, 1991, obtains the following *unconditional* derandomization of $\mathbf{BPAC}^0$.

Theorem 24 $\mathbf{BPAC}^0$ *has uniform constant depth circuits of quasipolynomial size (i.e.,* $\mathbf{BPAC}^0$ *is contained in quasi-*$\mathbf{AC}^0$*.)*

An interesting family of randomized complexity classes is the classes defined by *interactive proof systems*. In particular, **IP** is the class of languages decided by a randomized interactive proof system of polynomial complexity (we refer to Papadimitriou, 1994 for exact definitions). The natural goal would be to prove that randomized polynomial time interactive proof systems are not more powerful than deterministic polynomial time interactive proof systems, the latter systems capturing exactly **NP**. However, the classical result of Lund et al., 1992 and Shamir, 1992, shows that **IP**=**PSPACE**. Thus we cannot hope for such a derandomization of **IP** without proving a breakthrough result about **PSPACE**. More tractable are the classes **MA** and **AM** of *bounded round* interactive proof systems.

Along the lines of Impagliazzo and Wigderson, 1997, Goldreich and Zuckerman, 1997, proves[6]

Theorem 25 *If there is a language L in* $\mathbf{NE} \cap \mathbf{coNE}$ *and an $\epsilon > 0$, so that for all n, the circuit complexity of $L \cap \{0,1\}^n$ is at least $2^{\epsilon n}$, then* $\mathbf{MA} = \mathbf{NP}$.

It is interesting to note that the assumption of Theorem 25 is slightly weaker than the assumption of Theorem 14, as we only need to find L in $\mathbf{NE} \cap \mathbf{coNE}$ rather than **E**. Thus, perhaps **MA** is easier to derandomize than **BPP**.

A *strong nondeterministic circuit* for a function $f : \{0,1\}^n \to \{0,1\}$ is a pair of nondeterministic circuits: One for f and one for $\neg f$. Along the lines of Nisan and Wigderson, 1994, Arvind and Köbler proves the following theorem.

Theorem 26 *If there is a language L in* $\mathbf{NE} \cap \mathbf{coNE}$ *and an $\epsilon > 0$, so that for all n, no strong nondeterministic circuit of size $2^{\epsilon n}$ decides $L \cap \{0,1\}^n$ correctly on a $\frac{1}{2} + 2^{-\epsilon n}$ fraction of the inputs, then* $\mathbf{AM} = \mathbf{NP}$.

Klivans and van Melkebeek, 1999, observed that the steps outlined above leading to a derandomization result can sometimes be "automatized", using the fact that the Impagliazzo-Wigderson theorem *relativizes*. By plugging in the right oracle in the relativized version of the theorem(for instance, an oracle for SAT), a derandomization theorem pops out. They carry out this program for several randomized complexity classes and randomized constructions in complexity theory. In particular, for **AM** they prove

Theorem 27 *If there is a language L in* **NE** $\cap$ **coNE** *and an $\epsilon > 0$, so that for all n, no SAT-circuit of size $2^{\epsilon n}$ decides $L \cap \{0,1\}^n$, then* **AM** $=$ **NP**.

Miltersen and Vinodchandran, 1999, improved Theorem 26 as well as Theorem 27 by proving the following theorem.

Theorem 28 *If there is a language L in* **NE** $\cap$ **coNE** *and an $\epsilon > 0$, so that for all n, no strong nondeterministic circuit of size $2^{\epsilon n}$ decides $L \cap \{0,1\}^n$, then* **AM** $=$ **NP**.

Theorem 28 is proved using the compression technique along the lines of the proof of Theorem 20. It is not known if one can also obtain this result using locally decodable codes and Nisan-Wigderson designs. Thus, the proof of Theorem 28 is one of the few cases where the compression technique seems to be superior. The compression technique does not work very well for proving "low end results". Thus, it is at the time of writing an open problem if the statement "**NE** $\cap$ **coNE** requires superpolynomial strong nondeterministic circuits implies **AM** is in **SUBNEXP**" can be shown. The statement one obtains by replacing "strong nondeterministic circuits" with "SAT-circuits" was proved correct by Klivans and van Melkebeek, 1999.

All of the above results assume that a uniform class **A** is not in a small non-uniform class **B**. Impagliazzo and Wigderson, 1998 gave a highly interesting variation of the above scheme, where the non-uniform class **B** is replaced with a randomized uniform class. They prove the following theorem.

Theorem 29 *If* **BPP**$\neq$**EXP** *then* **BPP** $\subseteq$ *heur*-$\mathbf{SUBEXP}_{i.o.}$

Here, a language L is in *heur*-$\mathbf{SUBEXP}_{i.o.}$ if there, for any k, is a **SUBEXP**-machine M which, on infinitely many input lengths n, decides L correctly on at least a $1 - 1/n^k$ fraction of $\{0,1\}^n$. The assumption **BPP**$\neq$**EXP** is the weakest possible assumption about the power of **BPP** we can make. Indeed, if **BPP**$=$**EXP**, no nontrivial deterministic simulation is possible, by the time hierarchy theorem. Yet from this assumption, we can still get a non-trivial derandomization of **BPP**. Impagliazzo and Wigderson call their theorem a "gap"-theorem: Either **BPP** is everything or it is "small". It can't be "almost everything".

The proof is an ingenious modification of the proof of Theorem 12 we presented in Section 6 (or actually, the corresponding proof for Theorem 13). We outline it below and refer to the original paper for details.

The proof of Theorem 12 in Section 6 is, essentially, a combination of Lemma 34, Lemma 35, Lemma 33, and Lemma 2. To make the proof of Section 6 a proof of Theorem 29 rather than Theorem 12, we would have to redo the lemmas replacing "circuits" with "efficient randomized algorithms". A global view of the new proof is as follows: we use the same sequence of transformation of a truth table, through other truth tables, into a pseudorandom generator. The old proof would then, under the assumption that the generator does not work, prove the existence of a small circuit for the original truth table. In the new proof we shall also find such a circuit, but we want the proof to be constructive in the sense that it can be carried out by an efficient randomized algorithm. In a first attempt, such a proof should proceed by using a new version of Lemma 2, Lemma 33, Lemma 35 and Lemma 34, in that order.

The first obstacle is found in the application of Lemma 2. If the generator does not work on some machine and some input x, we used that lemma to build x into the machine, obtaining a non-uniform circuit. Now we want to construct this circuit, *without knowing* x. The solution is to pick an input x to be built into the circuit at random. Since we are only trying to prove by contradiction that $\mathbf{BPP} \subseteq heur\text{-}\mathbf{SUBEXP}_{i.o}$ and not $\mathbf{BPP} \subseteq \mathbf{SUBEXP}_{i.o}$, we can assume that such a random input will work with non-negligible probability. Indeed, this is the only reason why we shall only obtain this weaker conclusion.

The rest of the proof is more tricky. A crucial observation is this: If $\mathbf{EXP} \not\subseteq \mathbf{P/poly}$, we already have the desired conclusion, by Theorem 13. Thus, we can assume without loss of generality that $\mathbf{EXP}{=}\mathbf{P/poly}$. Then, a theorem by Karp and Lipton, 1980, states that $\mathbf{EXP}{=}\Sigma_p^2$. A theorem of Toda, 1991, states that the permanent of a matrix is hard for Σ_p^2 (indeed, it is hard for the polynomial hierarchy). As the permanent of a matrix can be computed in $\mathbf{EXP}$, we conclude that we can without loss of generality assume that the permanent is complete for $\mathbf{EXP}$. Thus, if there is a language L in $\mathbf{EXP}$-$\mathbf{BPP}$, we can assume that it is the permanent and use special properties of the permanent below.

The next observation is that if we use the permanent language to construct the pseudorandom generator, the use of Lemma 34 is unnecessary: The permanent is *random self-reducible*. This means that if the assumption about f of Lemma 34 holds for f being the permanent, then the conclusion about f' holds for f' being the permanent as well.

Thus, we only need to look at the transformations of Lemma 35 and Lemma 33 and in the circuits constructed in their proofs. Inspecting the constructions, we find that the only parts that cannot be carried out by an efficient randomized algorithm is the tabulation of $f(y_{S_j})$ as a function of $y_{S_i \cap S_j}$ in the proof of Lemma 33 and the use of a free random source of $(y, f(y))$-pairs in the proof of Lemma 35. In both cases the function f is an XOR of several copies of the permanent function. Thus, with an *oracle* for the permanent, the construction could be carried out efficiently.

Summing up, if the generator does not work, there is a small circuit for the permanent and we can construct it by an efficient randomized algorithm with oracle access to the permanent. In short, the permanent is *learnable*. Inspecting the proof further, we see that to construct the circuit on instances of length n, we would need to query the oracle on instances of length at most n.

Without any assumptions, we know that the permanent is *downward self-reducible*: There is an efficient procedure for computing the permanent on inputs of length n, given an oracle for the permanent on all smaller input lengths.

Combining the learnability and the downward self-reducibility of the permanent, we can now construct a sequence of small circuits for the permanent on inputs of length $1, 2, \ldots, n$: Having already constructed circuits for inputs of length $1, 2, \ldots, i$, we can learn the circuit for inputs of length $i+1$, as the circuit for inputs of length up to i, combined with the downward self-reducibility of the permanent, gives us an oracle for the permanent for all input lengths up to and including $i+1$. This completes the proof.

10. UNCONDITIONAL DERANDOMIZATION OF SPACE-BOUNDED COMPUTATION

In this section, we present unconditional deterministic simulation results for randomized space-bounded computation. It may, at first, be surprising that such results are possible at all, as this is not the case for randomized time-bounded computation. However, by Lemma 3, to derandomize space-bounded computation, we only have to construct pseudorandom generators for *automata*, rather than *circuits*, and as we understand how automata compute much better than we understand how circuits compute, this is a much easier task. However, the ultimate goal, to prove $\mathbf{L}=\mathbf{BP_H L}$, is still open (though we already saw a hardness based generator giving this statement under an reasonable unproven assumption in Theorem 21).

10.1 THE NISAN-ZUCKERMAN GENERATOR

We prove the following theorem, due to Nisan and Zuckerman, 1996, implying Statement 8 of the introduction. The theorem improves an earlier theorem of Ajtai et al., 1987.

Theorem 30 *Let $s(n) \geq \log n$ be a proper space bound. If a language L is decided by a two-sided error randomized Turing machine using space $O(s(n))$ and making $s(n)^{O(1)}$ random moves, then L is decided by a deterministic Turing machine using space $O(s(n))$.*

As we aim for a derandomization when the number of random moves made is limited, we cannot directly use Lemma 3 in a proof of Theorem 30. However, inspecting the proof of that lemma, we see that to prove Theorem 30, it is enough to consider the following promise problem, s-LAYERED AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION. As input, we are given a finite automaton A with n states. The automaton must be *layered*, meaning that for all states v in the automaton, all paths from the start state to v have the same length. In particular, the automaton must be acyclic. The set of nodes at distance i from the start state is called *layer i*. The maximum length of any path in the automaton (the *depth* of the automaton) must be at most $l = s(n)$. The automaton has a single accepting state. We are given as promise that *either* (a) $\Pr[A \text{ accepts } x] \leq \frac{1}{3}$ *or* (b) $\Pr[A \text{ accepts } x] \geq \frac{2}{3}$ for a randomly chosen $x \in \{0,1\}^l$, and we should tell which of (a) or (b) is the case.

To prove Theorem 30, it is enough to show that for all functions s with $s(n) = (\log n)^k$ for constant k, s-LAYERED AUTOMATON ACCEPTANCE PROBABILITY ESTIMATION is in $\mathbf{pBP_HL}$. Using the terminology of Section 5, to show that this problem is in $\mathbf{pBP_HL}$, it is enough to show that we can generate, on input n, using space $O(\log n)$, a pseudorandom set in $\{0,1\}^{(\log n)^k}$ for layered automata of size n and depth $(\log n)^k$.

In the proof, we shall actually make a generator using somewhat more space, namely $O((\log n)^{k-1/2})$ and then derive the stronger conclusion by "bootstrapping".

To define our generator G, we shall need an explicit extractor

$$E : \{0,1\}^{6\log n} \times \{0,1\}^{\sqrt{\log n}} \to \{0,1\}^{\log n}$$

with min-entropy threshold $3\log n$ and error $\epsilon < (\log n)^{-k-1}$. For instance, an extractor with the parameters of Theorem 6 or Theorem 7 (proved using either iterated universal hashing or codes and designs) will certainly do. The first extractor with the desired parameters was constructed for the application at hand, by Nisan and Zuckerman, 1996. For the application at hand, we need the map E to be computable in linear space (i.e., in space $O(\log n)$). This mild space constraint is satisfied by any of the constructions.

On input n, G goes through all strings $x \in \{0,1\}^{6\log n + (\log n)^{k-1/2}}$ and outputs a string in $\{0,1\}^{(\log n)^k}$ for each of them. For each x, it divides x into a a block $x' \in \{0,1\}^{\log n}$ and $l = (\log n)^{k-1}$ small blocks, $y_1, y_2, \ldots, y_l$, with $y_i \in \sqrt{\log n}$. Now, it outputs $E(x, y_1) \circ E(x, y_2) \circ \cdots \circ E(x, y_l)$.

Lemma 43 *G is a pseudorandom generator with error at most $\frac{1}{\log n}$ for layered automata of size n and depth $(\log n)^k$.*

Proof Given a layered automaton of size n and depth $(\log n)^k$, split its layers into $(\log n)^{k-1}$ blocks of depth $\log n$ each and collapse each block into a single layer, by changing the alphabet of the automaton from $\{0,1\}$ to $\{0,1\}^{\log n}$. Thus, we can view the automaton as a layered automaton of size n, alphabet size n and depth $l = (\log n)^{k-1}$.

Let U_i be the distribution on nodes in layer i induced by running the automaton on input uniformly chosen from $(\{0,1\}^{\log n})^i$ and let P_i be the distribution on nodes in layer i induced by running the automaton on a random member of the set generated by G, i.e., on input $E(x', y_1) \circ E(x', y_2) \cdots \circ E(x', y_i)$ for random uniformly chosen $x' \in \{0,1\}^{6\log n}, y_1, y_2, \ldots, y_i \in \{0,1\}^{\sqrt{\log n}}$. Note that $U_0 = P_0$. We show, by induction in i, that $\|U_i - P_i\|_1$ is small. The statement of the theorem will follow by taking $i = l$.

So suppose that $\|U_{i-1} - P_{i-1}\|_1$ is small. We shall bound $\|U_i - P_i\|_1$ by $\|U_i - M_i\|_1 + \|M_i - P_i\|_1$, where M_i is the distribution obtained by running the automaton on $E(x', y_1) \circ E(x', y_2) \circ \cdots \circ E(x', y_{i-1}) \circ v$ where $x', y_1, y_2, \ldots, y_{i-1}$ are as before, and $v \in \{0,1\}^{\log n}$ is uniformly and independently chosen. Thus, M_i is defined by running the automata on a pseudorandom input for $i-1$ steps and with a real random symbol in the i'th step. Note that by Lemma 19, $\|U_i - M_i\|_1 \leq \|U_{i-1} - P_{i-1}\|_1$, so we only have to bound $\|M_i - P_i\|_1$.

Let v be any state of the $i-1$'th layer of the automaton which receives probability at least $\frac{1}{n^2}$ according to distribution P_{i-1}.

Consider the distribution of x, conditioned by the event V that state v is the state visited in the $i-1$'st layer in the corresponding computation. Any outcome of x has conditional probability at most $n^2 2^{-6\log n}$, so x has min-entropy is at least $(6-2)\log n > 3\log n$, conditioned by V. Thus, conditioned by V, $E(x', y_i)$ is ϵ-close to uniform, where $\epsilon < \frac{1}{(\log n)^{k+1}}$. This means that the distributions P_i and M_i, conditioned by V are ϵ-close. Now let G be the event that *some* state v of P_{i-1}-probability more than $\frac{1}{n^2}$ is visited. The conditional distributions of P_i and M_i are ϵ-close, given G. The probability of $\neg G$, under P_{i-1}, is at most $\frac{n}{n^2} = \frac{1}{n}$. By Lemma 18, this means that P_i and M_i are $\epsilon + 2 \cdot \frac{1}{n}$ close.

Thus, U_l and P_l are $l\epsilon + 2l\frac{1}{n}$-close, in particular, they are $\frac{1}{\log n}$-close. ■

As an immediate corollary of Lemma 43, we get that a space $s(n)$ computation using $s(n)^{\frac{3}{2}}$ random binary moves can be simulated deterministically in space $O(s(n))$. For a space $s(n)$ computation using $s(n)^k$ random bits for a bigger constant k, it seems we only get a deterministic simulation in space $O(s(n)^{k-1/2})$. However, using the pseudorandom generator to do deterministic simulation, by cycling through all the strings in the pseudorandom set is not the only way we can use it. Looking at the way the pseudorandom set is defined, we see that we can *sample* a random element from the set using $s(n)^{k-1/2}$ random moves and using space only $s(n)$. Thus, we get the following lemma.

Lemma 44 *Let $k \geq \frac{3}{2}$ be a rational number and let $s(n) \geq \log n$ be a proper space bound. If a language L is decided by a two-sided error randomized Turing machine using space $O(s(n))$ and making $s(n)^k$ random moves, then L is decided by a deterministic Turing machine using space $O(s(n))$ and making $O(s(n)^{k-1/2})$ random moves.*

Now, Theorem 30 follows, by applying Lemma 44 a sufficient number of times.

10.2 THE IMPAGLIAZZO-NISAN-WIGDERSON GENERATOR

The generator of Impagliazzo et al., 1994 uses space $O((\log n)^2)$ and produces, on input n, a pseudorandom set in $\{0,1\}^n$ for automata of size n. Thus, it can be used to derandomize space-bounded computation without a bound on the number of random bits used, by Lemma 3.

The following explanation of the generator is due to Raz and Reingold, 1999.

Recall that the Nisan-Zuckerman generator generates a set $\{G(x, y_1, \ldots, y_k) | x, y_1, y_2, \ldots, y_k\}$ where $G(x) = E(x, y_1) \circ E(x, y_2) \circ \cdots \circ E(x, y_l)$ and E is an extractor of appropriate parameters. Similarly, the Impagliazzo-Nisan-Wigderson generator generates a set $\{G_l(x) | x \in \{0,1\}^{O(\log^2 n)}\}$, where $l = O(\log n)$ and G_l is defined by the following recursion:

$G_0(x) = x$

$G_i(xy) = G_{i-1}(x) \circ G_{i-1}(E_i(x, y))$,

where in the last equation, $x \in \{0,1\}^{O(i \log n)}$, $y \in \{0,1\}^{O(\log n)}$ and E_i is an extractor of appropriate parameters. G_l expands $O((\log n)^2)$ bits to

n bits. By a proof similar to the proof of the correctness of the Nisan-Zuckerman generator, the set of outputs of G_l is a pseudorandom set for automata of size n.

Note that we want the input and output domain of E_i to be identical. Appropriate extractors with this property are the ones directly based on expanders, described in Section 4.1. Thus in the construction of Impagliazzo et al., 1994, $E_i(x, y)$ is the y'th neighbor of x in (a power of) an explicit expander.

Using the pseudorandom generator, we can deterministically simulate any randomized space $s(n)$ computation in deterministic space $O(s(n)^2)$. As we mentioned in Section 2.1, that this is possible was known since the early 80's and without using pseudorandom generators. To get new simulation results, one has to use a modified generator and explore its structure carefully. This is done in the next subsection.

10.3 NISAN'S GENERATOR

Nisan's pseudorandom generator of Nisan, 1990 is very similar to the Impagliazzo-Nisan-Wigderson generator, and though it historically precedes it, it is at this point of our presentation best explained in terms of that generator.

Given a parameter n, let $m = c \log n$ for a suitable constant c, and let $\mathcal{H}$ be a strongly universal family of hash functions mapping $\{0,1\}^m$ to $\{0,1\}^m$.

We now define maps $G_i : \{0,1\}^m \times \mathcal{H}^i \to \{0,1\}^{2^i m}$, $i = 1, \ldots, m$, by

$G_0(x) = x$,

$G_i(x, h_1, \ldots, h_i) = G_{i-1}(x, h_1, \ldots, h_{i-1}) \circ G_{i-1}(h_i(x), h_1, \ldots, h_{i-1})$.

The claim is that $\{G_i(x)|x\}$ for $i \leq \log n$ are pseudorandom sets with small error for automata of size n. If we think of the value $h_i(x)$ as $E(x, h_i)$ where E is an extractor, and compare the recursion with the recursion in the definition of the Impagliazzo-Nisan-Wigderson generator, we see that we have essentially replaced $E(xh_1h_2 \ldots h_i)$ with $E(x, h_i)h_1h_2 \ldots h_{i-1}$, i.e., we do not feed the values $h_1, h_2, \ldots, h_{i-1}$ through the extractor, but just reuse them. Such a change would not work for any extractor, but it does work for *strongly* universal hash functions. The reason is that a stronger statement that just pseudorandomness of the resulting set can be shown by induction.

Recall that we use the term "pseudorandom set in $\{0,1\}^i$ for automata of size n" as a short hand for a pseudorandom set for sets in $\{0,1\}^i$ decided by a finite automaton with n states. Now consider a *fixed* finite automaton A with n states, and consider all n^2 variations of A obtained by moving the start state and single accepting state. These n^2 automata define n^2 languages over $\{0,1\}^i$. Now use the term "pseudorandom set in $\{0,1\}^i$ for A" as a short hand for a pseudorandom set for these n^2 languages.

The extra property that Nisan's generator enjoys is this: Given a *fixed* automaton A with n states, if we pick $h_1, h_2, \ldots, h_k$ at random and freeze them, then, with high probability, the distribution resulting from *then* picking just x at random, is pseudorandom for A.

In particular, in the bottom of the recursion, if h_1 is picked at random, then with high probability, the distribution $x \circ h_1(x)$ induced by random x, is pseudorandom.

The truth of this statement at the bottom of the recursion is the *hash mixing lemma* of Nisan, 1990, which we state without proof (the proof is a straightforward application of Chebyshev's inequality).

Lemma 45 *Let $\mathcal{H}$ be a strongly universal family of hash functions mapping $\{0,1\}^m$ to $\{0,1\}^m$. Let A and B be subsets $\{0,1\}^m$. Then*

$$\Pr_{h \in \mathcal{H}}[|\Pr_{x \in A}[h(x) \in B] - \mu(B)| \geq \epsilon] \leq \frac{\mu(B)(1-\mu(B))}{\mu(A)2^m\epsilon^2}$$

Note that it follows from $\Pr_{h\in\mathcal{H}}[|\Pr_{x\in A}[h(x) \in B] - \mu(B)| \geq \epsilon] \leq \frac{\mu(B)(1-\mu(B))}{\mu(A)2^m\epsilon^2}$ that $|\Pr_{h\in\mathcal{H},x\in A}[h(x) \in B] - \mu(B)| \leq \epsilon + \frac{\mu(B)(1-\mu(B))}{\mu(A)2^m\epsilon^2}$. Thus, thinking of B as a statistical test, we see the hash mixing lemma implies an extraction property of the family $\mathcal{H}$, but it is the stronger mixing property that is relevant to us.

Using an induction based on Lemma 45, Nisan now shows that for a fixed automaton A of size n and random $h_1, h_2, \ldots, h_i$, with very high probability the set $\{G_i(x, h_1, h_2, \ldots, h_i) \; |x\}$ is a pseudorandom set in $\{0,1\}^{2^i m}$ for A.

In particular, the set $\{G_m(x, h_1, h_2, \ldots, h_i)|x, h_1, h_2, \ldots, h_i\}$ is a pseudorandom set for all automata of size n, which we can use by Lemma 3 to derandomize $\mathbf{BP_H L}$, again obtaining a simulation in deterministic space $O((\log n)^2)$, as we did using the Impagliazzo-Nisan-Wigderson generator.

However, *more can be said*. Suppose we have already found h_1, h_2, $\ldots$, h_{i-1} so that $\{G_i(x, h_1, h_2, \ldots, h_{i-1})|x\}$ is a pseudorandom set in $\{0,1\}^{2^i m}$ for A. Now we can do an *exhaustive search* for an appropriate setting of h_{i+1} so that $\{G_{i+1}(x, h_1, h_2, \ldots, h_{i+1})|x\}$ is a pseudorandom set in $\{0,1\}^{2^{i+1} m}$ for A: We can check if a particular setting of h_{i+1} is okay by checking, for each pair of states a and b in the automaton, that a randomly chosen pseudorandom input from $\{G_{i+1}(x, h_1, h_2, \ldots, h_{i+1})|x\}$ takes us from state a to state b with approximately the same probability as the concatenation of two randomly chosen inputs from

$\{G_i(x, h_1, h_2, \ldots, h_i)|x\}$ takes us from a to b. With a bit of care, this procedure can be made space-efficient. Thus, we can iteratively find appropriate values of $h_1, h_2, \ldots, h_{O(\log n)}$, thus obtaining a pseudorandom set in $\{0,1\}^n$ for A. This implies the following theorem of Nisan, 1991, appearing as Statement 9 of the introduction.

Theorem 31 *Any language in* $\mathbf{BP_H L}$ *can be decided by a polynomial time algorithm using space* $O((\log n)^2)$.

Before Nisan's result, it was known that any language in $\mathbf{BP_H L}$ could be simulated in polynomial time by one algorithm and in space $O((\log n)^2)$ by another, but not that these time and space bounds could be obtained simultaneously.

10.4 SAKS AND ZHOU'S MATRIX EXPONENTIATION ALGORITHM

In this section, we sketch the proof of the following result of Saks and Zhou, 1999, yielding Statement 10 of the introduction.

Theorem 32 $\mathbf{BP_H L}$ *is a subset of* $\mathbf{SPACE}((\log n)^{3/2})$

By lemma 4, we only have to show that APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION can be solved using deterministic space $O((\log n)^{3/2})$.

APPROXIMATE STOCHASTIC MATRIX EXPONENTIATION can easily be solved using space $O((\log n)^2)$ by $O(\log n)$ repeated squarings, rounding off all entries in intermediate results to $O(\log n)$ bits. This yields Simon's result that $\mathbf{BP_H L} \subseteq \mathbf{SPACE}((\log n)^2)$.

Alternatively, we can obtain the same space bound by using Lemma 4 to convert the problem into a question about the acceptance probability of a finite automaton, and then use Nisan's generator to generate, using space $O((\log n)^2)$, a pseudorandom set for the automaton and simulate the automaton on all elements of the set to answer the question.

Now consider the following *hybrid* between these two extremes. Let M be the original matrix. Then, $\sqrt{\log n}$ times, replace M with an approximation of M^l with $l = 2^{\sqrt{\log n}}$, the latter matrix obtained by converting all questions about entries in M^l to questions about the acceptance probability of a finite automaton with n states on an input of length $2^{\sqrt{\log n}}$, using the technique of Lemma 4, and then using Nisan's generator G_l to generate a pseudorandom set for this automaton and simulate the automaton on the set to answer the question.

If the latter hybrid technique is done as described, we would again use space $O((\log n)^2)$. However, recall that the pseudorandom set is indexed by $x, h_1, h_2, \ldots, h_l$, and for a fixed automaton A, if $h_1, h_2, \ldots, h_l$ are fixed to random values, with very high probability the distribution induced by picking x at random will be pseudorandom for A.

Now, if $h_1, h_2, \ldots, h_l$ are set to random values at the beginning of the computation and the above algorithm is carried out, *assuming* that the distribution induced by picking x at random will be a pseudorandom set for each of the automata occurring during the computation, we have a randomized algorithm using space $O((\log n)^{3/2})$ and using $O((\log n)^{3/2})$ random bits - and thus a deterministic algorithm using space $O((\log n)^{3/2})$, as desired.

However it is not clear that the assumption made will be correct with high probability, so the simulation may be incorrect. The reason is this. Picking random $h_1, h_2, \ldots, h_l$ at random works with very high probability for any *fixed* automaton. However, the automata occurring in the above simulation after the first approximate exponentiation has been done *depend on the* $h_1, h_2, \ldots, h_l$ *chosen*, as they depend on the first approximate matrix power obtained. If there was no such dependence, the above simulation would work.

Saks and Zhou solve this problem by modifying the simulation so that with very high probability over the choices of $h_1, h_2, \ldots, h_l$, the matrices (and hence the automata) occurring as subresults are actually a fixed sequence of matrices. That this is possible follows from the fact that though the matrices occurring as subresults of the computation depend on $h_1, h_2, \ldots, h_l$, they are, with high probability, a very good approximation of the original matrix to a certain power, say M^r. Thus, if we do a *rounding* of all entries in the matrix to a certain number of binary digits and do the same rounding of the entries in M^r, we are very likely to end up at the same matrix. Note that this may *not* hold if the entry in one matrix, is, say, 0.1010001, and the corresponding entry in the other is 0.1001111. Then, rounding down both entries to three digits would make the matrices different. Similar cases could occur for all other deterministic rounding schemes. Thus, Saks and Zhou use a *randomized rounding* scheme, essentially by picking a random offset δ, subtracting δ from each matrix entry, rounding down the entry, and adding δ again. Doing this for every matrix obtained as a subresult in the simulation ensures that, with high probability, the sequence of matrices occurring depends on δ and the original matrix M only. Thus, the simulation gives the right result with high probability. The extra randomness needed for the offset δ is negligible, so we can derandomize the choice of δ by running through all possible values.

10.5 FURTHER RESULTS

Armoni, 1998, presents a hybrid between the generators of Nisan and Zuckerman, Impagliazzo, Nisan and Wigderson, and Nisan. For certain setting of the parameters (number of random bits vs. space) the generator obtained gives a better deterministic simulation result than is obtainable using either of the 3 generators.

The Impagliazzo-Nisan-Wigderson generator extracts remaining randomness in the seed x after having finished half the computation. Raz and Reingold, 1999, observes that some randomness may be present in the *state* of the computation at this point and that this randomness could be extracted, making certain assumptions about the computation. Thus, they obtain improved derandomization results for restricted classes of space bounded computation.

For the special case of UNDIRECTED GRAPH CONNECTIVITY, probably the most natural example of a problem in $\mathbf{BP_H L}$ not known to be in $\mathbf{L}$, Armoni et al., 1997, present an $O((\log n)^{4/3})$ space algorithm, based on refining the Saks-Zhou technique described above.

Notes

1. The fact that this answer is either "yes" or "no" is less important. Thus, most of the results can be adapted to deal with algorithms seeking a particular numerical value, such as Monte Carlo integration, but we shall not do so.
2. One can try, however, to get the expected number of random bits used to be close to the entropy of the desired output distribution. This problem was considered by Impagliazzo and Zuckerman, 1989.
3. Though they shall not be covered explicitly, both the technique of conditional probabilities and the technique of small sample spaces play a role for generic derandomization as well, as will be apparent.
4. The notation is not quite standard. Buhrman and Fortnow, 1999, use **promiseC**, but this is a bit too awkward if one really wants to use the notation as we do.
5. Polishchuk and Spielman, 1994, improve the decoding efficiency of the construction somewhat.
6. Goldreich and Zuckerman, 1997 only states the result for $L \in \mathbf{E}$, but the proof is easily seen to generalize.

References

Adleman, L. (1978). Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83.

Ajtai, M., Komlós, J., and Szemerédi, E. (1987). Deterministic simulation in LOGSPACE. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 132–140.

Aleliunas, R., Karp, R. M., Lipton, R. J., Lovász, L., and Rackoff, C. (1979). Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science*, pages 218–223.

Allender, E., Reinhardt, K., and Zhou, S. (1999). Isolation, matching, and counting uniform and nonuniform upper bounds. *J. Comput. System Sci.*, 59(2):164–181.

Alon, N. (1986). Eigenvalues and expanders. *Combinatorica*, 6(2):83–96.

Alon, N. and Milman, V. D. (1985). λ_1, isoperimetric inequalities for graphs, and superconcentrators. *J. Combin. Theory Ser. B*, 38(1):73–88.

Alon, N. and Spencer, J. H. (1992). *The probabilistic method*. John Wiley & Sons Inc., New York.

Andreev, A. E., Clementi, A. E. F., and Rolim, J. D. P. (1997). Worst-case hardness suffices for derandomization: A new method for hardness-randomness trade-offs. In *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 177–187. Springer-Verlag.

Andreev, A. E., Clementi, A. E. F., and Rolim, J. D. P. (1998). A new general derandomization method. *Journal of the Association for Computing Machinery*, 45(1):179–213.

Andreev, A. E., Clementi, A. E. F., Rolim, J. D. P., and Trevisan, L. (1999). Weak random sources, hitting sets, and BPP simulations. *SIAM J. Comput.*, 28(6):2103–2116.

Armoni, R. (1998). On the de-randomization of space-bounded computations. In *Randomization and Approximation Techniques in Computer Science, Second International Workshop*, volume 1518 of *Lecture Notes in Computer Science*, pages 47–59.

Armoni, R., Ta-Shma, A., Wigderson, A., and Zhou, S. (1997). $SL \subseteq L^{\frac{4}{3}}$. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 230–239.

Babai, L., Fortnow, L., Levin, L. A., and Szegedy, M. (1991a). Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 21–31.

Babai, L., Fortnow, L., and Lund, C. (1991b). Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40.

Bellare, M., Goldreich, O., and Sudan, M. (1998). Free bits, PCPs, and non-approximability - towards tight results. *SIAM J. Comput.*, 27:804–915.

Berlekamp, E. and Welch, L. (1986). Error correction of algebraic block codes. US Patent Number 4,633,470.

Blum, M. and Micali, S. (1984). How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864.

Boppana, R. B. and Sipser, M. (1990). The complexity of finite functions. In *Handbook of theoretical computer science, Vol. A*, pages 757–804. Elsevier, Amsterdam.

Buhrman, H. and Fortnow, L. (1999). One-sided versus two-sided error in probabilistic computation. In *Theoretical Aspects of Computer Science, 16th Annual Symposium*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109. Springer-Verlag.

Carter, J. L. and Wegman, M. N. (1979). Universal classes of hash functions. *J. Comput. System Sci.*, 18(2):143–154.

Chor, B. and Goldreich, O. (1989). On the power of two-point based sampling. *Journal of Complexity*, 5:96–106.

Clementi, A. E., Rolim, J. D., and Trevisan, L. (1998). Recent advances towards proving P = BPP. *Bulletin of the European Association for Theoretical Computer Science*, 64:96–103.

Cohen, A. and Wigderson, A. (1989). Dispersers, deterministic amplification, and weak random sources (extended abstract). In *30th Annual Symposium on Foundations of Computer Science*, pages 14–19.

Dietzfelbinger, M. (1996). Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *13th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1046 of *lncs*, pages 569–580. Springer.

Gabber, O. and Galil, Z. (1981). Explicit construction of linear sized superconcentrators. *Journal of Computer and System Sciences*, 22:407.

Goldreich, O. (1999). *Modern cryptography, probabilistic proofs and pseudorandomness.* Springer-Verlag, Berlin.

Goldreich, O. and Levin, L. A. (1989). A hard-core predicate for all one-way functions. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 25–32.

Goldreich, O., Nisan, N., and Wigderson, A. (1995). On Yao's XOR-lemma. Technical Report TR95-050, Electronic Colloquium on Computational Complexity.

Goldreich, O. and Wigderson, A. (1994). Tiny families of functions with random properties: A quality-size trade-off for hashing (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 574–583.

Goldreich, O. and Wigderson, A. (1999). Improved derandomization of BPP using a hitting set generator. In *Randomization and Approximation Techniques in Computer Science, Third International Workshop.*, volume 1671 of *Lecture Notes in Computer Science*, pages 131–139.

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

Impagliazzo, R. (1995). Hard-core distributions for somewhat hard problems. In *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 538–547.

Impagliazzo, R., Levin, L. A., and Luby, M. (1989). Pseudo-random generation from one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 12–24.

Impagliazzo, R., Nisan, N., and Wigderson, A. (1994). Pseudorandomness for network algorithms. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 356–364.

Impagliazzo, R., Shaltiel, R., and Wigderson, A. (1999). Near-optimal conversion of hardness into pseudo-randomness. In *40th Annual Symposium on Foundations of Computer Science*, pages 181–190.

Impagliazzo, R., Shaltiel, R., and Wigderson, A. (2000). Extractors and pseudo-random generators with optimal seed length. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 1–10.

Impagliazzo, R. and Wigderson, A. (1997). P=BPP if E requires exponential circuits: Derandomizing the XOR lemma. In *Proc. 29th Annual ACM Symposium on Theory of Computing*, pages 220–229.

Impagliazzo, R. and Wigderson, A. (1998). Randomness vs. time: Derandomization under a uniform assumption. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 734–743.

Impagliazzo, R. and Zuckerman, D. (1989). How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253.

Karp, R., Pippenger, N., and Sipser, M. (1986). Expanders, randomness, or time versus space. In *First Annual Conference on Structure in Complexity Theory*, pages 325–329.

Karp, R. M. and Lipton, R. J. (1980). Some connections between nonuniform and uniform complexity classes. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 302–309, Los Angeles, California.

Klivans, A. and Servedio, R. (1999). Boosting and hard-core sets. In *40th Annual Symposium on Foundations of Computer Science*, pages 624–633.

Klivans, A. R. and van Melkebeek, D. (1999). Graph nonisomorhism has subexponential size proofs unless the polynomial-time hierarchy collapses. In *Proc. 31st ACM Symposium on Theory of Computing*, pages 659–667.

Knuth, D. E. (1981). *The art of computer programming. Vol. 2*. Addison-Wesley Publishing Co., Reading, Mass., second edition. Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing.

Lautemann, C. (1983). BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217.

Levin, L. A. (1987). One way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363.

Lubotzky, A., Phillips, R., and Sarnak, P. (1986). Explicit expanders and the Ramanujan conjectures. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 240–246, Berkeley, California.

Lund, C., Fortnow, L., Karloff, H., and Nisan, N. (1992). Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868.

Lupanov, O. (1965). A method of synthesis of control system - the principle of local coding. *Problemy Kibernet.*, 10:31–110.

Mansour, Y., Nisan, N., and Tiwari, P. (1993). The computational complexity of universal hashing. *Theoretical Computer Science*, 107(1):121–133.

Margulis, G. A. (1973). Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80.

Miltersen, P. B. (1998). Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In *Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 556–563.

Miltersen, P. B. and Vinodchandran, N. (1999). Derandomizing Arthur-Merlin games using hitting sets. In *40th Annual Symposium on Foundations of Computer Science*, pages 71–80.

Motwani, R. and Raghavan, P. (1995). *Randomized algorithms*. Cambridge University Press, Cambridge.

Nisan, N. (1990). Psuedorandom generators for space-bounded computation. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 204–212.

Nisan, N. (1991). Pseudorandom bits for constant depth circuits. *Combinatorica*, 1:63–70.

Nisan, N. (1994). RL $\subseteq$ SC. *Computational Complexity*, 4(1):1–11.

Nisan, N. and Ta-Shma, A. (1999). Extracting randomness: a survey and new constructions. *J. Comput. System Sci.*, 58(1, part 2):148–173.

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

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

Papadimitriou, C. (1994). *Computational Complexity*. Addison-Wesley.

Polishchuk, A. and Spielman, D. A. (1994). Nearly-linear size holographic proofs. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 194–203.

Rabin, M. O. (1976). Probabilistic algorithms. In *Algorithms and complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1976)*, pages 21–39. Academic Press, New York.

Radhakrishnan, J. and Ta-Shma, A. (1997). Tight bounds for depth-two superconcentrators. In *38th Annual Symposium on Foundations of Computer Science*, pages 585–594.

Raz, R. and Reingold, O. (1999). On recycling the randomness of states in space bounded computation. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 168–178.

Raz, R., Reingold, O., and Vadhan, S. (1999a). Error reduction for extractors. In *40th Annual Symposium on Foundations of Computer Science*.

Raz, R., Reingold, O., and Vadhan, S. (1999b). Extracting all the randomness and reducing the error in Trevisan's extractor. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 149–158.

Saks, M. (1996). Randomization and derandomization in space-bounded computation. In *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, pages 128–149, Philadelphia, Pennsylvania. IEEE Computer Society Press.

Saks, M., Srinivasan, A., and Zhou, S. (1998). Explicit OR-dispersers with polylogarithmic degree. *Journal of the Association for Computing Machinery*, 41(1):123–154.

Saks, M. and Zhou, S. (1999). $\mathrm{BP}_H\mathrm{SPACE}(S) \subseteq \mathrm{DSPACE}(S^{3/2})$. *J. Comput. System Sci.*, 58(2):376–403.

Santha, M. and Vazirani, U. V. (1986). Generating quasirandom sequences from semirandom sources. *J. Comput. System Sci.*, 33(1):75–87.

Schönhage, A. and Strassen, V. (1971). Schnelle Multiplikation grosser Zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292.

Shamir, A. (1992). IP=PSPACE. *Journal of the ACM*, 39:869–877.

Shannon, C. E. (1949). The synthesis of two-terminal switching circuits. *Bell. Syst. Tech. J.*, 28:59–98.

Shoup, V. (1994). Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391.

Simon, J. (1981). On tape-bounded probabilistic Turing machine acceptors. *Theoretical Computer Science*, 16(1):75–91.

Sipser, M. (1983). A complexity theoretic approach to randomness. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 330–335.

Sipser, M. (1988). Expanders, randomness, or time versus space. *Journal of Computer and System Sciences*, 36:379–383.

Srinivasan, A. and Zuckerman, D. (1999). Computing with very weak random sources. *SIAM J. Comput.*, 28(4):1433–1459.

Sudan, M., Trevisan, L., and Vadhan, S. (1999). Pseudorandom generators without the XOR lemma. In *Proc. 31st Annual ACM Symposium on Theory of Computing*, pages 537–546.

Ta-Shma, A. (1996). On extracting randomness from weak random sources (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 276–285.

Ta-Shma, A. (1998). Almost optimal dispersers. In *Proc. 30th Annual ACM Symposium on the Theorem of Computing*, pages 196–202.

Tanner, R. M. (1984). Explicit concentrators from generalized N-gons. *SIAM J. Algebraic Discrete Methods*, 5(3):287–293.

Toda, S. (1991). PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877.

Trevisan, L. (1999). Constructions of near-optimal extractros using pseudo-random generators. In *Proc. 31st ACM Symposium on Theory of Computing*, pages 141–148.

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

Yao, A. C. (1982). Theory and application of trapdoor functions. In *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91.

Zuckerman, D. (1990). General weak random sources. In *31st Annual Symposium on Foundations of Computer Science*, volume II, pages 534–543.

Zuckerman, D. (1991). Simulating BPP using a general weak random source. In *32nd Annual Symposium on Foundations of Computer Science*, pages 79–89.

Zuckerman, D. (1996). Randomness-optimal sampling, extractors, and constructive leader election. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 286–295.