# Computing Efficiently Using General Weak Random Sources

David Zuckerman
University of California at Berkeley
Ph.D. Thesis
August, 1991

**Abstract**

In this thesis, we study how to efficiently simulate randomized algorithms while making only minimal assumptions on the quality of the randomness available to the simulation. We basically show that the randomness need not have a particular form; it suffices to have the randomness output at a constant rate. More specifically, when our model of a $\delta$-source outputs an $R$-bit string, the only constraint is that it must do so according to some distribution that places probability no more than $2^{-\delta R}$ on any particular $R$-bit string. We show how to simulate BPP and approximation algorithms using the output from such a source.

We also give two applications of these constructions: one to showing the difficulty of approximating the size of the maximum clique, and the other to the problem of implicit $O(1)$ probe search.

Furthermore, we show how to extract almost-random bits at a nearly optimal rate from a constant number of $\delta$-sources, under the Generalized Paley Graph Conjecture. With no unproven assumptions, we show how to extract almost-random bits at a nearly optimal rate from three PRB-sources of [CG2], improving the results there.

# Contents

**Acknowledgements**

First and foremost, I am grateful to my advisor, Umesh Vazirani, for helping me during all stages of my graduate career. In my first paper, when I had one last difficulty to overcome, he saw a brilliant way to overcome it. This helped give me the tenacity to push through ideas to the end. This, moreover, was only one of many great ideas he was to share with me. Another great quality he has is his enthusiasm – when he exclaimed "oo ba-ba," I knew I had done well.

I am also grateful to other professors who gave me both good technical and non-technical advice. David Aldous was especially helpful during my first two years, when he taught me a lot about random walks and probability theory. Mike Luby and I had many stimulating and informative conversations about randomness. Dick Karp taught two of my favorite classes and gave me many useful pointers. Manuel Blum showed me several great ideas he's had.

The students at Berkeley influenced me as much as the professors. I would like to thank my co-author Russell Impagliazzo for teaching me about the Leftover Hash Lemma and other aspects of randomness and cryptography. I had many interesting conversations with Noam Nisan about pseudo-random generators and, before that, random walks on graphs. Madhu Sudan made several important comments on my work, including pointing out, with Abhijit Sahay, the mistake in my original $n^{\log n}$ RP simulation. Among other things, Moni Naor pointed out the connection with his work. I spent many informative hours working with Sandy Irani. Ronitt Rubinfeld helped me a lot during my first year, and I had many interesting conversations with her. Milena Mihail taught me a lot about expanders.

I would also like to thank Pete Gemmell, Will Evans, David Wolfe, Sampath Kannan, Dana Randall, Nina Amenta, David Blackston, David Cohen, Eddie Grove, Mor Harchol, Lisa Hellerstein, Diane Hernek, Lee Newberg, Sridhar Rajagopalan, Ashu Rege, Steven Rudich, Jim Ruppert, Elizabeth Sweedyk, Debbie Weisser, and the rest of the theory group for making Berkeley a fun and interesting place to learn theoretical computer science.

# Chapter 1

# Introduction

Randomness plays a vital role in almost all areas of computer science, both in theory and in practice. Randomized algorithms are often faster or simpler than the deterministic algorithms for the same problem (see e.g. [Rab]).

To produce "random" bits, a computer might consult a physical source of randomness, such as a Zener diode, or use the last digits of a real time clock. In either case, it is not clear how random these "random" bits will be. Moreover, it is impossible to verify the quality of a random source. It is therefore of interest to see if weak, or imperfect, sources of randomness can be used in randomized algorithms. Indeed, the fewer assumptions we make about the quality of our random source, the greater the chances are that the source satisfies these assumptions and hence that our randomized algorithms work correctly. This emphasis on reliability seems even more important as computers become faster, because users should be willing to pay a greater price for greater accuracy. This ties in with the recent interest in checking.

The history of weak random sources reflects this ideal of using as weak a source as possible. As far back as 1951, Von Neumann [vN] gave a simple and practical algorithm to extract perfectly random bits from a source of independent coin flips of equal but unknown bias. Elias [Eli] improved this by showing how to extract bits at the optimal rate.

The problem of correlations, however, is more serious. Blum [Blu] was the first to tackle this by modeling a weak source as a Markov chain. He showed that a counter-intuitive generalization of von Neumann's algorithm converts the output from such a source into a truly random string.

Yet the requirement that a source be Markovian is very stringent, requiring the source to be very regular. This motivated Santha and Vazirani [SV] to ask: what if we only know that each bit that the source outputs is somewhat random? To answer this, they introduced the model of semi-random sources, where the probability of a given bit having a specific value, conditional on the value of previous bits, is not too large, but can otherwise be controlled by an adversary. They proved that it is impossible to extract even a single almost-random bit from one such source.

In light of this result, one might give up hope for simulating randomized algorithms with one semi-random source. Nevertheless, [VV] and [Va2] showed how to simulate RP and BPP with one semi-random source. Vazirani [Va1,Va3] also showed how to extract almost-random bits at a constant rate from two independent sources. Moreover, his algorithm extracts $n$ almost-random bits even if the sources are allowed to exchange $n/\log n$ communication bits.

Chor and Goldreich [CG2] generalized this model by assuming no sequence of $l$ bits has too high a probability of being output. More precisely, [1]

**Definition 1.1 (CG2)** *An $(l, \delta)$ PRB-source outputs $R$ bits as $R/l$ blocks $x_1, \ldots, x_{R/l}$, each of length $l$, such that for all $l$-bit strings $y_1, \ldots, y_{R/l}$,*

$$Pr[x_i = y_i | x_1 = y_1, \ldots, x_{i-1} = y_{i-1}] \leq 2^{-\delta l}.$$

A semi-random source corresponds to $l = 1$.

Note also that we do not know anything about the source except the restriction above; our simulations must work for all such sources. Equivalently, we may assume an adversary controls the output blocks, and need only abide by the restriction above.

For $l = O(\log R)$, Chor and Goldreich showed how to simulate BPP using one such source. They further showed how to obtain almost-random bits from four independent such sources at a constant rate.

Here we improve their construction by showing how to do it from three independent PRB-sources at a rate approaching the optimal one. We also show how to extract almost-random bits at a nearly optimal rate from one PRB-source using only $O(\log^2 n)$ bits from a second, independent, high quality (but not nearly uniformly random) PRB-source. We believe this simple algorithm to be of practical interest.

[1] We modify their definition into an equivalent form in order to correspond better with the rest of this thesis.

Various authors have also considered models for weak sources where an adversary chooses the values of certain bits, but the others are random (see [CG+], [BL], [LLS], [CWi]).

Given all of this previous work, it is natural to ask: what is the most general model of a weak source for which we can simulate randomized algorithms? Do we need the randomness in some particular form, or will any form suffice?

Of course, if BPP = P, then we don't need randomness at all. Yet we follow [VV], [Va2], and [CG] and deal with a more abstract "BPP" problem: let an adversary label strings in $\{0,1\}^r$ either "yes" or "no," provided that at least $3/4$ of the strings have the same label. We wish to find out whether the majority say "yes" or "no," with high probability. It is clear that randomness really is needed to answer this question quickly.

One's first guess at the most general source would probably impose a lower bound on the entropy. For example, if the source outputs $R$ bits, we might insist that the entropy of the distribution on output strings be at least $\delta R$, for some constant $\delta$. If we did this, however, the source could output a useless string with constant probability, and we could never achieve a small error probability for our randomized algorithms.

Instead, we propose upper bounding the probability that any particular string is output:

**Definition 1.2** *For any number $R$ of random bits requested, a $\delta$-source outputs an $R$-bit string such that no string has probability more than $2^{-\delta R}$ of being output, for some fixed $\delta > 0$.*

Again, we know nothing else about the source; our simulations must work for all such sources.

It is also important to note that if, say, we make two requests of 100 bits, we do not impose the $2^{-100\delta}$ upper bound on each group of 100 bits. Indeed, it is easy to imagine a source outputting good bits at the beginning but later outputting highly correlated bits. We therefore impose the $2^{-200\delta}$ upper bound on the total 200-bit string. Equivalently, we assume that only one request for random bits is made to the $\delta$-source.

This model essentially generalizes all of the above models,[2] making no structural assumptions about dependencies. The generality of our model is further substantiated by a lemma of Cohen and Wigderson [CWi]: if we can simulate RP or BPP using a $\delta$-source, then

[2]One of the bit-fixing sources in [CWi] has a weaker entropy bound than that which we impose. Our model can be modified to generalize this source, too, but then our simulations would fail.

we can achieve a constant probability of success with any source having constant entropy rate. As Cohen and Wigderson point out [CWi], the largest upper bound one could impose in the definition above and still possibly get (abstract) RP or BPP simulations is $2^{-R^\delta}$.

In our main results, we show how to simulate RP and BPP in polynomial time using the output of a $\delta$-source for all $\delta > 0$. These algorithms also yield solutions to more general problems. Our RP simulation implies that we can find an $n$-bit prime using a $\delta$-source. Our BPP simulation yields simulations for approximation algorithms, e.g. those used to approximate the volume of a convex body [DFK], or a statistical simulation to estimate some quantity or set of quantities.

We also show how to extract almost-random bits at a rate arbitrarily close to optimal from a constant number of $\delta$-sources, under the Generalized Paley Graph Conjecture. The Generalized Paley Graph Conjecture asserts, in a certain technical sense, the independence of the additive and multiplicative groups modulo a prime. Chor and Goldreich [CG2] have shown how to extract one almost-random bit from two $\delta$-sources under the Paley Graph Conjecture.

Simulating RP and BPP using the output of a $\delta$-source is related to deterministic amplification. In the deterministic amplification problem, the goal is to reduce the error probability of an RP or BPP algorithm from constant to exponential using few random bits. Building on work in [AKS], we show that if $r$ bits are needed to achieve constant error probability for a BPP algorithm, then only $O(r)$ bits are needed to achieve error $2^{-r}$ (this was done independently in [CWi]). This means that very few random strings give the wrong answer, and implies simulations using a $\delta$-source for RP when $\delta > 1/2$ and BPP when $\delta > 3/4$ (done previously in [CWi]).

Our simulations of RP for all $\delta > 0$ also yield interesting results for the deterministic amplification problem: in the above framework $O(r \log r)$ bits are needed to achieve error probability $2^{-r}$ (our BPP simulation is not so interesting in this regard, since it requires $r^{O(1/\delta)}$ bits). Furthermore, once we use at least $O(r \log r)$ bits, our RP simulation reduces the error faster than other known methods: using $cr \log r$ random bits yields error probability $2^{-(1-\delta)cr\log r}$ for any constant $\delta$.

The simulations of RP and BPP are equivalent to the explicit construction of a certain type of expander graph, called a disperser (see [San], [Sip], [CWi]). We believe our disperser constructions will be useful in many different areas. To illustrate this belief we give two applications: one to showing the difficulty of approximating the size of the maximum clique, and the other to the problem of implicit $O(1)$ probe search.

Computing $\alpha = \alpha(G)$, the size of the maximum clique, is well known to be NP-complete [Kar]. Little was known about the difficulty of approximating $\alpha$ until Feige, et.al. [FG+] showed that if approximating $\alpha$ to within a factor of $2^{(\log n)^{1-\epsilon}}$ is in $\tilde{P}$, then $N\tilde{P} = \tilde{P}$ ($\tilde{P}$ denotes quasi-polynomial time, i.e. $2^{polylog}$).

Since it is infeasible to find close approximations, one can ask whether it is feasible to at least estimate the order of magnitude of $\alpha$. More precisely, is there an algorithm that for some constant $t$ outputs a number between $\alpha^{1/t}$ and $\alpha^t$? By applying our disperser construction to the proof of [FG+], we show that the existence of such an algorithm implies $N\tilde{P} = \tilde{P}$.

Implicit $O(1)$ probe search is the problem of arranging $n$ elements from the domain $\{1, \ldots, m\}$ in a table of size $n$ so that searching for an element can be done with a constant number of probes. Fiat and Naor [FN], building on [FKS] and [FNSS], show how to do this when $m$ is polynomial in $n$. In addition, they show how to do it for the case $m = n^{\log n}$, assuming a certain type of disperser can be explicitly constructed, as was conjectured by Sipser [Sip].

We cannot construct this disperser, but by constructing three other dispersers, two of which come directly from the RP construction above and the third of which relies heavily on the theory we've developed, we give the first constructive solution for $m$ superpolynomial in $n$. The exact function is $m = n^{\frac{\sqrt{\log \log n}}{\log \log \log n}}$. Our solution is weaker than the earlier results, however, in that it does not allow search to be done in constant time, despite the constant number of probes.

Most of our main results appear in preliminary form in [Zu2].

# Chapter 2

# Hashing Lemmas

One of our key tools is a modification of the Leftover Hash Lemma. The Leftover Hash Lemma was introduced in [ILL] in order to construct cryptographically secure pseudo-random generators from one-way functions. It was then used extensively in [IZ] to construct pseudo-random generators for various tasks without any complexity assumptions. A similar lemma was also used in [BBR].

In order to state this lemma, we need some definitions.

**Definition 2.1** *A probability distribution $D$ on a set $S$ is* quasi-random *within $\epsilon$ if for all $X \subseteq S$, $|D(X) - |X|/|S|| \leq \epsilon$. Here $D(X)$ denotes the probability of the set $X$ according to distribution $D$.*

The maximum difference above is also called the variation distance between $D$ and the uniform distribution, and is also equal to $\frac{1}{2}\sum_{s\in S}|D(s) - 1/|S||$.

**Definition 2.2 (CWe)** *Let $A$ and $B$ be two sets, and $H$ a family of functions from $A$ to $B$. $H$ is called a* universal family of hash functions *if for every $x_1 \neq x_2 \in A$ and $y_1, y_2 \in B$,*

$$Pr_{h\in H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] = 1/|B|^2.$$

It is important for our purposes that there is a universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^n$ of size $2^{m+n}$.[1] To see this, let $F$ be the finite field on $2^m$ elements.

[1] If we only want $Pr[h(x_1) = h(x_2)] = 1/|B|$, which is all that is needed for the Modified Leftover Hash Lemma, then for certain lengths $m$ there are families of size $2^{m+1}$ (see [Va1]).

Then $H = \{(a,b) | a \in \{0,1\}^m, b \in \{0,1\}^n\}$ is universal, where

$$(a,b)(x) = (\text{last } n \text{ bits of } a \cdot x) \oplus b,$$

where the multiplication is in $F$ and $\oplus$ denotes bitwise exclusive-or. Using the results of [Sho], we can efficiently construct $F$ deterministically.

The Leftover Hash Lemma is best explained in the following context: suppose we have an element $x$ chosen uniformly at random from a set $A$, where $A$ is contained in a much larger set $\{0,1\}^n$. We don't know anything about $A$, except that $|A| \geq 2^l$. The Leftover Hash Lemma allows us to convert the randomness in $x$ into a more usable form, provided we have enough extra random bits. These extra bits are used to pick a uniformly random hash function $h$ mapping $n$ bits to $l-2k$ bits, where $k$ is a security parameter. The Leftover Hash Lemma guarantees that the ordered pair $(h, h(x))$ is almost-random.

**Leftover Hash Lemma [ILL]:** Let $X \subset \{0,1\}^n, |X| \geq 2^l$. Let $e > 0$, and let $H$ be a universal family of hash functions mapping $n$ bits to $l-2e$ bits. Then the distribution $(h, h(x))$ is quasi-random within $1/2^e$ (on the set $H \times \{0,1\}^{l-2e}$), where $h$ is chosen uniformly at random from $H$, and $x$ uniformly from $X$.

It is important that we have the ordered pair $(h, h(x))$, because it allows us to use the same hash function repeatedly: even after outputting $h(x)$, $h$ will appear close to random. When we do use the same hash function repeatedly, the following useful lemma will imply that the statistical error only grows additively.

**Lemma 2.1** *Suppose $f$ is a function satisfying the property that if $x$ is picked uniformly at random, then $f(x)$ is quasi-random within $\epsilon$. Then if $x$ is quasi-random within $\epsilon'$, then $f(x)$ is quasi-random within $\epsilon + \epsilon'$.*

We will need the Modified Leftover Hash Lemma to deal with the situation when the hash function is not close to uniformly random.

**Lemma 2.2 (Modified Leftover Hash Lemma)** *Let $H = \{h : I \rightarrow O\}$ be a universal family of hash functions, and let $A \subset I$ and $C \subset H$. Then the distribution $(h, h(x))$, where $h$ is chosen uniformly from $C$ and $x$ uniformly from $A$, is quasi-random within $\sqrt{\frac{|H||O|}{|C||A|}}$ on the set $C \times O$.*

**Remark 2.1** *For our purposes, it would suffice to derive the bound $\frac{|H|}{|C|}\sqrt{\frac{|O|}{|A|}}$ directly from the Leftover Hash Lemma: this follows because the probability space in the modified case is* *a subset of size $|C|/|H|$ of the original probability space. For completeness, however, we include the stronger bound.*

**Proof.** We use ideas of the proofs in [MNT],[BBR], [ILL], and [IZ]. Let $p_{h,z} = Pr[h(x) = z]$ where $x$ is chosen uniformly from $A$. Thus the probability that a particular $(h, z)$ occurs according to the distribution above is $\frac{p_{h,z}}{|C|}$. The variation distance between the two distributions above on $(h, h(x))$ is $1/2$ times

$$\sum_{h\in C, z\in O} \Big|\frac{p_{h,z}}{|C|} - \frac{1}{|O||C|}\Big| \leq \sqrt{|C||O| \sum_{h\in C, z\in O} \Big(\frac{p_{h,z}}{|C|} - \frac{1}{|C||O|}\Big)^2}$$

$$\leq \sqrt{|C||O| \sum_{h\in H, z\in O} \Big(\frac{p_{h,z}}{|C|} - \frac{1}{|C||O|}\Big)^2}$$

$$= \sqrt{|C||O| \sum_{h\in H, z\in O} \Big(\frac{p_{h,z}^2}{|C|^2} - \frac{2p_{h,z}}{|C|^2|O|} + \frac{1}{|C|^2|O|^2}\Big)}.$$

Now

$$\sum_{z\in O} p_{h,z} = Pr[h(x) \in O] = 1,$$

and

$$\sum_{h\in H, z\in O} p_{h,z}^2 = |H| Pr[h(x) = h(y)]$$

if $h$ is picked uniformly from $H$, and $x, y$ independently and uniformly from $A$. Using the above and

$$Pr[h(x) = h(y)] = Pr[x = y] + Pr[x \neq y]\frac{1}{|O|} < \frac{1}{|A|} + \frac{1}{|O|}$$

yields the lemma. □

When we analyze $\delta$-sources with $\delta > 1/2$, the following lemma found in [MNT] will be useful. At the expense of not bounding the distribution of the ordered pair $(h, h(x))$, it gives a somewhat better bound on the quasi-randomness of $h(x)$. Because it does not involve the ordered pair, however, it will not be as useful as the Modified Leftover Hash Lemma. [2]

[2] Another difference between this lemma and the Leftover Hash Lemma is that the Leftover Hash Lemma still holds when the hash family only has the property that $Pr[h(x_1) = h(x_2)] = 1/|B|$ but not the full pairwise independence.

**Lemma 2.3 (MNT)** *Let* $H = \{h : I \rightarrow O\}$ *be a universal family of hash functions, and let* $A \subset I$*,* $B \subset O$*, and* $C \subset H$*. Then*

$$|Pr_{s \in A, h \in C}[h(s) \in B] - |B|/|O|| < \sqrt{\frac{|H||B|}{|A||C||O|}}$$

# Chapter 3

# $\delta$-Sources and Deterministic Amplification

## 3.1 Definitions and Observations

**Definition 3.1** *RP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial time Turing machine $M_L(a,x)$ for which*

$$\begin{aligned} a \in L &\Rightarrow Pr[M_L(a,x) \ \textit{accepts}] \geq 1/2 \\ a \notin L &\Rightarrow Pr[M_L(a,x) \ \textit{accepts}] = 0 \end{aligned} \tag{3.1}$$

*where the probabilities are for an $x$ picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p$.*

**Definition 3.2** *BPP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial time Turing machine $M_L(a,x)$ for which*

$$a \in L \Rightarrow Pr[M_L(a,x) \ \textit{accepts}] \geq 2/3 \tag{3.2}$$

$$a \notin L \Rightarrow Pr[M_L(a,x) \ \textit{accepts}] \leq 1/3 \tag{3.3}$$

*where the probabilities are for an $x$ picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p$.*

As is well known, by running $M_L$ on independent random tapes, we can change the probabilities in (3.1), (3.2), and (3.3) to $1 - 2^{-poly(|a|)}$, $1 - 2^{-poly(|a|)}$, and $2^{-poly(|a|)}$, respectively.

To define what simulating RP means, say we wish to test whether a given element $a$ is in $L$. If $a \notin L$, then all random strings cause $M_L$ to reject, so there is nothing to do.

Suppose $a \in L$; then we wish to find with high probability a witness to this fact. Let $W$ be the set of witnesses, i.e. $W = \{x | M_L(a, x) \text{ accepts}\}$, and $N$ be the set of non-witnesses, i.e. the complement of $W$.

One might think that to simulate RP using a $\delta$-source we would need a different algorithm for each language in RP. Instead, we exhibit one simulation that works for all $W$ with $|W| \geq 2^{r-1}$. In particular, we don't make use of the fact that $W$ can be recognized in polynomial time.

**Definition 3.3** *A polynomial-time algorithm simulates RP using a $\delta$-source if it takes as input $R = poly(r)$ bits from the $\delta$-source and outputs a polynomial number of $r$-bit strings $y_i$, such that for all $W \subset \{0,1\}^r, |W| \geq 2^{r-1}$, $Pr[(\exists i) y_i \in W] = 1 - 2^{-\Omega(R)}$.*

As one might expect, the following lemma shows that such a universal algorithm can be used for other purposes.

**Lemma 3.1** *If $A$ simulates RP using a $\delta$-source, then in polynomial time we can find an $n$-bit prime, with probability $1 - 2^{-\Omega(n)}$.*

**Proof.** We use only the simpler co-RP algorithms for primality (see e.g. [Rab]). These algorithms require $O(n)$ random bits, but let us use $n(n-2)$ random bits. Run $A$ to produce polynomially many $n(n-2)$-bit strings $y_i$. We claim that with high probability, for each composite $n$-bit number there will be a $y_i$ being a witness to this fact. This is because

$$\begin{aligned}
& Pr[\exists n\text{-bit number with no witness among } y_i] \\
& \quad < \ 2^n Pr[\text{given } n\text{-bit number has no witness among } y_i] \\
& \quad \leq \ 2^n 2^{-\Omega(R)} \\
& \quad = \ 2^{-\Omega(R)}
\end{aligned}$$

By the Prime Number Theorem, the fraction of odd $n$-bit numbers that are prime is at least $1/n$. Thus, the probability that an $n$-tuple of odd $n$-bit numbers chosen uniformly at random contains an $n$-bit prime is at least $1/2$. (Note that this requires $n(n-2)$ bits, because the first and last bits of an odd $n$-bit number are always 1.) Thus, the probability that there is such an $n$-tuple among the $y_i$ is $1 - 2^{-\Omega(R)}$. Verifying the primality by the previous paragraph yields the lemma. □

For BPP, we have no "witnesses" to membership, but by an abuse of notation we use $W$ to denote the set of random strings producing the right answer, and $N$ as the complement of $W$. As before, a simulation of BPP will produce strings $y_i$ and use these to query whether $a \in L$. The simulation does not have to take the majority of these answers as its answer, but can instead use any function of the answers. In fact, in our BPP simulation we will build several ternary trees, associate the answers from the $y_i$'s with the leaves, take the majority of majorities in the trees, and then the majority of the answers given by the trees.

For the purpose of the following definition only, we define $S$ as the set of $r$-bit strings saying $a \in L$; we wish to know whether $S$ equals $W$ or $N$.

**Definition 3.4** *A polynomial-time algorithm $A$ simulates BPP using a $\delta$-source if it takes as input $R = poly(r)$ bits from the $\delta$-source and outputs a polynomial number of $r$-bit strings $y_i$; $A$ then takes some function of the answers to $y_i \in S$ to produce a single guess as to whether $|S|$ is larger or smaller than $2^{r-1}$, which for any $S \subset \{0,1\}^r$ with $|S| \geq \frac{2}{3}2^r$ or $|S| \leq \frac{1}{3}2^r$ must be correct with probability $1 - 2^{-\Omega(R)}$.*

Note that such an algorithm $A$ can be used to simulate approximation algorithms. This is because whenever a majority of numbers lie in a given range, their median also lies in that range. Thus, by taking medians instead of majorities, $A$ can output a good approximation with probability $1 - 2^{-\Omega(R)}$.

Simulations using a $\delta$-source are equivalent to the construction of a certain type of expander graph, called a disperser (see [San, Sip, CWi]). We never use this graph-theoretic approach, however, except in applying our construction to the problem of implicit O(1) probe search (see Section 7.2).

Following [CG2], we define flat $\delta$-sources and show that we may assume without loss of generality that our $\delta$-source is flat.

**Definition 3.5** *A flat $\delta$-source is a source which places probability $2^{-\delta R}$ on $2^{\delta R}$ $R$-bit strings.*

**Lemma 3.2** *Suppose an algorithm simulates RP (BPP) from a flat $\delta$-source. Then it also simulates RP (BPP) from a $\delta$-source.*

**Proof.** The proof in our case is much simpler than in [CG2]. Fix an algorithm $A$. On certain input strings, $A$ outputs the correct answer, and on others it doesn't. The $\delta$-source may as well place as much probability as possible on the strings for which $A$ is incorrect, i.e. the $\delta$-source may as well be a flat $\delta$-source. □

**Remark 3.1** *This observation, that some strings are good for $A$ and others aren't, implies that an algorithm simulates RP (BPP) with probability $1 - 2^{-\Omega(R)}$ from a $\delta$-source if and only if it simulates RP (BPP) with non-zero probability from a $\delta'$-source, for some $\delta' < \delta$. This is because the probability of outputting the wrong answer is at most the number of bad strings divided by $2^{\delta R}$ or $2^{\delta' R}$. Therefore, we need not worry about simulating RP or BPP with high probability, but only with non-zero probability.*

## 3.2 Deterministic Amplification

Let $L \in$ BPP, $a \in \{0,1\}^n$, and let $r = p(n)$ be the number of random bits required above. The usual way of improving the success probability to $1 - 2^{-r}$ is to generate $O(r)$ truly random strings of length $r$, query $a$ with these random strings, and take the majority vote. This requires $O(r^2)$ bits. The deterministic amplification problem is to do this with fewer random bits. Here we give a scheme, based on [AKS] and found independently by [CWi], where it suffices to use the optimal $O(r)$ bits. In fact, it suffices to use $O(r)$ bits from a $\delta$-source for large enough $\delta$.

Let us first observe the relationship of this problem to simulating BPP using a $\delta$-source. Say a pseudo-random generator needs $Cr$ bits to reduce the probability of error to $2^{-r}$. Then there are at most $2^{(C-1)r}$ "bad" strings of length $Cr$. Thus, if $\delta > (C-1)/C$, the probability of error will be $2^{(C-1)r}2^{-\delta r C} = 2^{-\Omega(r)}$. Our construction and a new expander construction [Mor] will imply RP simulations for $\delta > 1/2$ and BPP for $\delta > 3/4$. These bounds were achieved in [CWi] by combining this amplification construction with a different idea, as the above expander construction was not available then.

We give the construction only for the BPP case when $\delta = 3/4 + \epsilon$; the RP case is easier. For now, assume we have truly random bits available to us; we'll use the idea in the previous paragraph to do the final analysis at the end.

Our construction begins by using the usual technique to improve the success probability to $2^{-(t+1)}$, where $t$ is an integer greater than $4/\epsilon$. This requires $r' = O(r)$ bits, so for convenience replace $r'$ by $r$. With this new $r$, we show that to achieve error probability $2^{-k}$ we need only $r + O(k)$ bits. We do this by taking a random walk on an expander, an idea used in [AKS].

A random walk on an undirected graph $G$ is the sequence of vertices $\{X_u\}$ visited by a particle that starts at a specified vertex and visits other vertices according to the following transition rule: if the particle is at vertex $i$ at time $u$, then at time $u+1$ it moves to a neighbor of $i$ picked uniformly at random. Associated with a random walk on $G$ is the transition matrix $A$ describing the probabilities of moving from one vertex to another. For convenience, we define $A$ as the transpose of the traditional transition matrix, i.e. $A_{ij} = 1/\deg(j)$ if $i$ and $j$ are neighbors, and 0 otherwise.

The matrix $A$ has eigenvalues $1 = \lambda_1 \geq |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_s|$. We use an expander construction by Morgenstern [Mor] (based on [LPS]) with the following properties: the graph $G$ is $d$-regular for $d = 2^t + 1$; $|G| = 2^r - 2^{r/3}$; $|\lambda_2| \leq 2\sqrt{d-1}/d$; and a random walk on $G$ can be simulated efficiently.

We can now state our algorithm.

**Algorithm:** Select a uniformly random start vertex $X_0$, and perform the random walk $\{X_u\}$ for $u \leq k$. For every $i \in \{1, 2, \ldots k\}$ query if $a \in L$ using the $r$-bit pseudo-random string represented by $X_i$. Output the majority vote of these queries.

**Proof of Correctness:** We use the techniques in [AKS]. First, some notation:

$C \subseteq G$ is the set of vertices representing strings for which queries are answered correctly. Note that $|C|/|G| \geq 1 - 2^{r-(t+1)}/|G| \geq 1 - 2^{-t}$.

$P$ is the vector space $R^s$, where $s = 2^r - 2^{r/3}$. $P$ represents probabilities.

$V$ is the subspace of multiples of $p_0 = (2^{-r}, 2^{-r}, \ldots, 2^{-r})$.

$W$ is the subspace orthogonal to $V$.

For $p = (p_1, p_2, \ldots, p_s) \in P$

$|p| = \sum_{i=1}^{s} |p_i|$, the $L_1$-norm.

$\|p\| = \sqrt{\sum_{i=1}^{s} p_i^2}$, the $L_2$-norm.

$N : P \to P$ is the linear transformation

$$N(e_i) = \begin{cases} e_i & \text{if } i \in G - C \\ 0 & \text{otherwise,} \end{cases}$$

where $e_i$ is the basis vector with a 1 in the $i$th place and 0's elsewhere.

$M = I - N$, where $I$ is the identity.

Note that if the vector $p = (p_1, \ldots, p_s)$ represents the probabilities of being at the different vertices, i.e. $p_i = \Pr[\text{particle is at vertex } i]$, then $Ap$ represents the probabilities after a step in our random walk. Furthermore, $NAp$ represents the probabilities of being at the different vertices and of not being in $C$. Extending this idea, we see that for a given sequence of the correctness of answers to queries, e.g. correct, correct, incorrect, ..., incorrect (denoted c,c,i,...,i)

$$\Pr[\text{c,c,i,...,i}] = |(NA)\dots(NA)\cdots(NA)(MA)(MA)p_0|$$

To bound the right-hand side, it is easier to use $L_2$ norms. We need the following lemma:

**Lemma 3.3** *For $p \in P$,*

*(i)* $\|MAp\| \le \|p\|$.

*(ii)* $\|NAp\| < 2^{2-t/2}\|p\|$.

**Proof.** (i) follows from the definition of $M$ and because the eigenvalues of $A$ are between -1 and 1. To see (ii), write $p = v + w$, $v \in V$, $w \in W$. Using $Av = v$ and $|G - C| \le 2^{-t}|G|$, we get $\|NAv\| \le 2^{-t/2}\|v\| \le 2^{-t/2}\|p\|$. Since $w$ is orthogonal to the eigenvector corresponding to $\lambda_1$, $\|NAw\| \le \|Aw\| \le |\lambda_2|\|w\| < 2^{1-t/2}\|w\| \le 2^{1-t/2}\|p\|$. By the triangle inequality, $\|NAp\| \le \|NAv\| + \|NAw\| < 2^{2-t/2}\|p\|$. □

Using $|p| \le 2^{s/2}\|p\|$, this lemma implies that for any sequence $s$ with at least $k/2$ incorrect answers,

$$\Pr[s] \le 2^{s/2}\|(NA)^{k/2}p_0\| < 2^{s/2}(2^{2-t/2})^{k/2}\|p_0\| = 2^{(1-t/4)k}$$

Because there are at most $2^k$ such sequences $s$,

$$\Pr[\exists s \text{ with} \ge \text{k/2 incorrect answers}] < 2^k 2^{(1-t/4)k} = 2^{(2-t/4)k}.$$

When implementing the random walk, the error probability could actually be twice that of the above, because the number of random strings is not a power of 2 and we have to take our real random string modulo $sd^k$.

We may use a $\delta$-source as long as we walk for at least $k \ge r$ steps. This is because

$$\begin{aligned}
Pr[\text{source outputs bad string}] &\le (\#\text{ bad strings})Pr[\text{source outputs particular string}] \\
&\le (sd^k \cdot 2^{(2-t/4)k})(2(sd^k)^{-\delta}) \\
&\le (2^r 2^{kt} e^{k/2^t})^{1-\delta} 2^{1+(2-t/4)k} \\
&\le 2^{(r+kt+k)(1/4-\epsilon)+(2-t/4)k+1} \\
&< 2^{-k}, \quad \text{using } t\epsilon \ge 4.
\end{aligned}$$

# Chapter 4

# Simulating RP

## 4.1 Case $\delta > 1/2$

Here we present another algorithm to simulate RP when $\delta > 1/2$. It has the advantage over the previous algorithm of not relying on the explicit construction of expanders, and hence being simpler to implement. More importantly, this algorithm sets the framework for our general algorithm. It is also of interest that it is the same algorithm that Nisan shows is a pseudo-random generator which fools all logspace machines [Nis]; his results, however, do not imply ours. [1]

Our algorithms to simulate RP divide the $R$-bit string $X$ into $r$-bit strings $x_1, \ldots, x_{k'}$ (where $R = O(r \log r)$) and combine these strings in various ways. In order to prove that our algorithms work, we need the following two lemmas, which show that it suffices to prove the correctness of our algorithm on a $(cr, \delta)$ PRB source as long as we only use $O(r \log r)$ bits from the source. Note that $cr$ is much larger than the block lengths used in [CG2], so our techniques are completely different.

**Lemma 4.1** *Fix $\alpha > 0$, $\delta' < \delta$, and an integer $k > 0$. Let $k' = \lceil \frac{k(1-\delta')+\alpha}{\delta-\delta'-l^{-1}} \rceil = O(k)$, and set $R = k'l$. Write the $R$-bit string $X$ from the $\delta$-source as $X = x_1 \circ x_2 \circ \ldots \circ x_{k'}$ (here $\circ$ denotes concatenation), where each $x_j$ is $l$ bits. For each initial string $X_i = x_1 \circ x_2 \circ \ldots \circ x_i$, label any $2^{\delta' l}$ of the $x_{i+1}$'s as "$\delta'$-bad"; the others we call "$\delta'$-good" (when it is not ambiguous,*

[1] When applied to the problem of deterministic amplification, his analysis yields an error probability of $2^{-r}$ using $O(r \log r)$ bits, while we show an error probability of $2^{-r \log r}$.

*we will simply say "bad" and "good"). Then*

$$Pr[\text{for} \geq k \text{ values of } i,\ x_i \text{ is good}] > 1 - 2^{-\alpha l},$$

*when $X$ is drawn from a $\delta$-source.*

**Proof.** Construct a tree corresponding to the outputs $X$ of the source as follows: let the nodes be all possible initial sequences $X_i$ for each $i$, $0 \leq i \leq k'$, and let the parent of $X_i$ be $X_{i-1}$. Define an edge $(X_{i-1}, X_i)$ to be bad (good) if the string $x_i$ is bad (good) with respect to $X_{i-1}$. We wish to show that few of the $2^{\delta R}$ leaves have root-leaf paths with less than $k$ good edges.

To bound this number, we observe that each parent has at most $2^{\delta' l}$ children connected by bad edges, and at most $2^l$ children. Thus, the total number of root-leaf paths with $k' - k$ specified bad edges (e.g. the edges at distances 2,3,6,7 from the root must be bad) is at most

$$2^{kl} 2^{(k'-k)\delta' l},$$

so the total number of root-leaf paths with at least $k' - k$ bad edges is

$$\binom{k'}{k} 2^{kl} 2^{(k'-k)\delta' l}.$$

Using $\binom{k'}{k} < 2^{k'}$ and substituting the definition of $k'$ in the above formula, we bound the number by $2^{-\alpha l} 2^{\delta R}$, as required. □

**Lemma 4.2** *If we have a polynomial-time algorithm that simulates RP using $O(r \log r)$ bits from a $(cr, \delta')$ PRB-source of length $O(r \log r)$, then we can construct a polynomial-time algorithm that simulates RP using the output of a $\delta$-source, for any $\delta > \delta'$.*

**Proof.** Suppose the simulation $A$ using a PRB-source needs $k$ blocks of length $cr$. Compute $k'$ as in Lemma 4.1, request $k'cr$ bits from the $\delta$-source, and divide the string into $k'$ blocks of length $cr$. We will run $A$ on all subsets of $k$ blocks, and show that at least one of them is likely to find a witness.

The idea of the proof is to build two trees, one corresponding to the PRB-source and the other to the $\delta$-source. We then put probabilities of finding witnesses at the nodes of the trees, and show that these probabilities are higher on the tree for the $\delta$-source.

As above, denote $x_1 \circ \ldots \circ x_i$ by $X_i$. Define

$$p(X_i, S) = Pr[A \text{ finds witness starting from } X_i],$$

where the probability is taken over the remaining $(k-i)r$ bits of the seed being output by a $(cr, \delta')$ PRB-source $S$. Now define

$$q(X_i) = \min_{(cr,\delta') \text{ PRB-sources } S} \{p(X_i, S)\}.$$

Next call $x_i$ bad* with respect to $X_{i-1}$ if $X_{i-1} \circ x_i$ has one of the $2^{\delta' cr}$ least $q(X_i)$'s, for $X_i$'s continuations of $X_{i-1}$. In other words, $x_i$ is bad* if $A$, when fed $X_{i-1} \circ x_i$ followed by the output of a worst case PRB-source for this $X_i$, has a low probability of finding a witness. Thus a worst case PRB-source would only place positive probability on $x_i$ which are bad*.

We use the above definitions of bad* to define good and bad on the tree for our original $\delta$-source, as defined in Lemma 4.1. We work from the root down, proceeding as follows: label a block $x_i$ by counting the number $j$ of good blocks in the string $X_{i-1}$, concatenating these good blocks together to form $X_j$, and labeling $x_i$ as bad iff it is bad* with respect to $X_j$. This only makes sense if $j \leq k$; if $j > k$ label $x_i$ bad if $x_i \leq 2^{\delta' cr}$. By Lemma 4.1, with high probability the string from the $\delta$-source is likely to have at least $k$ good blocks. Because there are only $O(\log r)$ blocks $x_i$, we can try all possible subsets of size $k$ in polynomial time and hence have our hands on the first k good blocks, which we call $x_1, \ldots, x_k$.

Now we claim that $A$ will do at least as well on input $x_1, \ldots, x_k$ as on a random output from an arbitrary $(cr, \delta')$ PRB-source, say source $S$. Since $A$ either fails or succeeds on a particular random string, this means $A$ *always* outputs the correct answer on a sequence of all good strings.

We prove our claim by induction on the statement:

$$q(X_i) \geq Eq(Y_i),$$

where $Y_i$ is a random $icr$-bit string from source $S$ and $X_i$ denotes $x_1 \circ \ldots \circ x_i$. It is true for $i = 0$, and assuming it for a particular $i$,

$$q(X_{i+1}) \geq E_{\text{bad strings } z_{i+1}} q(X_i \circ z_{i+1})$$

$$= q(X_i) \geq Eq(Y_i) = Eq(Y_{i+1}).$$

$\square$

Thus, from now on, we assume all blocks of $\Omega(r)$ bits are output from an $(\Omega(r), \delta)$ PRB-source, and we only request $O(\log r)$ blocks.

Suppose $a \in L$. The starting point of our algorithms is due to Vazirani and Vazirani [VV], which we state informally. Suppose we had a function $f : \{0,1\}^r \times \{0,1\}^w \to \{0,1\}^r$, such that for a random $w$-bit string $X$ from the source, the induced function $f(\cdot, X)$ (denoted $f_X(\cdot)$) maps many non-witnesses to witnesses. Then we could define a new witness set $W' = \{s | s \in W \text{ or } f_X(s) \in W\}$ which is much larger than $W$, with high probability.

We could then recurse on this idea, constructing

$$W^{(i)} = \{s | s \in W^{(i-1)} \text{ or } f_{X_i}(s) \in W^{(i-1)}\}.$$

The hope is that the $W^{(i)}$'s will grow quick enough so that with high probability, $W^{(k)} = \{0,1\}^r$ for $k = O(\log r)$. This would allow us to test whether $a \in L$ simply by testing whether $0 \in W^{(k)}$, because if $a \notin L$, then $W^{(k)} = \emptyset$. Now $0 \in W^{(k)}$ is equivalent to $0 \in W^{(k-1)}$ or $f_{X_k}(0) \in W^{(k-1)}$, which is further equivalent to $0 \in W^{(k-2)}$ or $f_{X_k}(0) \in W^{(k-2)}$ or $f_{X_{k-1}}(0) \in W^{(k-2)}$ or $f_{X_{k-1}}(f_{X_k}(0)) \in W^{(k-2)}$. Fleshing out the recursion completely, we are left with the algorithm that tests whether any combination $f_{X_{i_1}} \circ f_{X_{i_2}} \circ \ldots \circ f_{X_{i_l}}(0)$, $1 \le i_1 < i_2 < \ldots < i_l \le k$, is in $W$. Observe that $k = O(\log r)$ implies a polynomial number of queries.

For $\delta > 1/2$, we have a simple function $f$, namely a hash function mapping $r$ bits to $r$ bits. This leads to the following simple algorithm: take $2kr$ bits from the source and forms the $2r$-bit numbers $h_1, h_2, \ldots, h_k$, where $k = O(\log r)$. Treating the $h_i$ as elements of a universal family of hash functions, it then forms the set $P$ of potential witnesses as follows:

$P \leftarrow \{0\}$

For $i = 1$ to $k$ do

$P \leftarrow P \cup h_{k-i}(P)$

Here $h(P)$ denotes $\{h(p) | p \in P\}$.

Defining the non-witness sets $N^{(i)}$ as the complements of the witness sets $W^{(i)}$ above, we see that

$$N^{(i)} = \{s \in N^{(i-1)} | h_i(s) \in N^{(i-1)}\}. \tag{4.1}$$

We can use Lemma 2.3 to analyze equation (4.1):

**Lemma 4.3** $E[|N^{(i)}|] \leq |N^{(i-1)}|^2 2^{-r} + |N^{(i-1)}| 2^{(1/2-\delta)r}$.

**Proof.** Apply Lemma 2.3 with $A = B = S_{i-1}$, $I = O = \{0,1\}^r$, and note that we are interested in $|S_{i-1}|$ times the given probability. □

**Theorem 4.1** *For all $\delta > 1/2$, the algorithm given above will simulate RP.*

**Proof.** By Remark 3.1, it suffices to show that with probability at least $1/2$, $N^{(k)} = \emptyset$ for $k = O(\log r)$. Let $\epsilon = \delta - 1/2$ and $q = 2^r$. We assume that our RP algorithm is incorrect with probability at most $1/128$, and that $q \geq (16/\epsilon)^{2/\epsilon}$, i.e. choose $r \geq (2/\epsilon)(4 - \log_2 \epsilon)$. In the beginning, the dominant term bounding $E[|N^{(i)}|]$ in Lemma 4.3 will be $|N^{(i)}|^2/q$. First we show that when this is the dominant term, the $N^{(i)}$'s shrink rapidly, and then we show the same for the other term.

While $|N^{(i)}| \geq q^{1-\epsilon}$, we show inductively that with probability at least $3/4+1/2^{i+2}$, $|N^{(i)}| \leq q/2^{2^i+i+5}$. It is true for $i = 0$, and suppose it is true for $i$. Then using $|N^{(i)}| \geq q^{1-\epsilon}$, $E[|N^{(i+1)}|] \leq 2|N^{(i)}|^2/q$. Then, using Markov, if

$$|N^{(i)}| \leq q/2^{2^i+i+5}, \tag{4.2}$$

then with probability at least $1 - 1/2^{i+3}$,

$$|N^{(i+1)}| \leq 2^{i+4}|N^{(i)}|^2/q \leq q/2^{2^i+i+6}. \tag{4.3}$$

Using the induction assumption on the probability of (4.2), we get that (4.3) holds with probability at least $3/4+1/2^{i+3}$, and the induction is complete. Therefore, this phase (when $|N^{(i)}| \geq q^{1-\epsilon}$) can only last $\log_2 r$ rounds.

Now we view the phase where $|N^{(i)}| \leq q^{1-\epsilon}$. Now $E[|N^{(i+1)}|] \leq 2|N^{(i)}|/q^{\epsilon}$. Using Markov again, with probability at least $1 - 2/q^{\epsilon/2}$, $|N^{(i+1)}| \leq |N^{(i)}|/q^{\epsilon/2}$. If these decreases in the $N^{(i)}$'s continue, then this phase can only last $(1-\epsilon)/(\epsilon/2) \leq 2/\epsilon$ rounds. The probability that the decreases continue is therefore bounded from below by $1 - (2/\epsilon)2/q^{\epsilon/2}$. Using $\epsilon q^{\epsilon/2} \geq 16$, we see that this probability is at least $3/4$.

Thus, the probability that both phases end as hoped is at least $1/2$, and we are done. □

## 4.2 General Case

For the general algorithm, note that the results in the previous section, or alternatively those of Section 3.2, imply that we may assume without loss of generality that

$|N| \le 2^{7r/8}$. This is because we can replace our original $r$ by $R$ in the above simulation and $W$ by the set of strings that produce at least one witness. The fact that the above simulation works means that our new $N$ is at most $2^{\delta r}$.

The key idea for thinking about constructing suitable functions $f : \{0,1\}^r \times \{0,1\}^w \to \{0,1\}^r$ in the general case was introduced in [Zu1]. Imagine that if the inputs to $f$ are chosen randomly, i.e. the $r$-bit string uniformly from the set of non-witnesses and the other bits according to a PRB-source, then the output is quasi-random within $2^{-\Omega(r)}$. This means that the output is a witness with reasonable probability. Substituting in the string from our source, we obtain a function $f_X$ which maps random non-witnesses to witnesses with reasonable probability. In other words, a large fraction of these non-witnesses will be mapped to witnesses, as we want.

Observe that a larger $N$ facilitates our construction of $f$, because this gives us implicit access to higher-quality random bits. We will therefore first show how to construct $f$ if $N$ is large, and then how to make progress when $N$ is small.

To get such a function $f$ when $N$ is large, we make use of the Modified Leftover Hash Lemma. Set $t = \lceil r/(1+\delta/3) \rceil$, and $u = r - t$, so that $u \le \delta t/3$. We now view our $r$-bit non-witness as a hash function $h$ mapping $t$-bit strings to $u$-bit strings. We then use $\lceil r/u \rceil$ $t$-bit blocks $y_1, \dots, y_{\lceil r/t \rceil}$ from the source, and the output of $f$ is the first $r$-bits of $h(y_1) \circ \cdots \circ h(y_{\lceil r/t \rceil})$ (here $\circ$ denotes concatenation). The Modified Leftover Hash Lemma then implies that the output of $f$ is quasi-random.

Thus we get a new set of non-witnesses $N'$, where

$$N' = \{s \in N | f_{X_1}(s) \in N\}. \tag{4.4}$$

We now show

**Lemma 4.4** *If* $|N| \le 2^{(1-\delta/8)r}$, *then* $E[|N'|] \le (1+o(1))2^{(1-\delta/8)^2 r}$.

**Proof.** We may assume without loss of generality that $|N| = \lfloor 2^{(1-\delta/8)r} \rfloor$. Imagine that $s \in N$ is picked uniformly at random, and view it as a hash function $h$. Using the Modified Leftover Hash Lemma with $I = \{0,1\}^t$, $A =$ the $2^{\delta t}$ possible values for $y_1$, $H = \{0,1\}^r$, and $C = N$, we see that $(h, h(y_1))$ is quasi-random within $\sqrt{2^{\delta r/8}/2^{2\delta t/3}} \le 2^{-3\delta/16}$ (using $\delta \le 1$). Repeating this and using Lemma 2.1 implies inductively that $h \circ h(y_1) \circ h(y_2) \circ \ldots \circ h(y_i)$ is quasi-random within $i2^{-3\delta/16}$. Hence the output of $f$ is quasi-random within $\lceil r/t \rceil 2^{-3\delta/16}$.

Multiplying the probability that this output lies in $N$ by $|N|$ yields

$$E[|N'|] \leq |N|^2 2^{-r} + O(|N| 2^{-3\delta r/16}) \leq (1+o(1)) 2^{(1-\delta/8)^2 r}.$$

$\square$

**Corollary 4.1**

$$Pr[|N'| \leq 2^{(1-\delta/8)(1-\delta/10)r}] \geq 1 - 2^{-\delta r/50}.$$

**Proof.** Use Lemma 4.4 and Markov's Inequality. $\square$

We can now prove:

**Theorem 4.2** *For all $\delta > 0$, there is an algorithm that simulates RP using a $\delta$-source.*

**Proof.** Since $N'$ is too small to give $f$ desirable properties, we make progress not by reducing $N'$, but by reducing $r$. Let $r' = \lceil (1-\delta/10)r \rceil$. We can map an $r'$-bit string to an $r$-bit string simply by padding with 0's on the front. This gives us new corresponding witness and non-witness sets $W''$ and $N''$ in $\{0,1\}^{r'}$, namely, an element is a witness in $\{0,1\}^{r'}$ iff it is mapped to a witness in $\{0,1\}^r$. Moreover,

$$|N''| \leq |N'| \leq 2^{(1-\delta/8)(1-\delta/10)} \leq 2^{(1-\delta/8)r'}.$$

Thus, replacing $r$ by $r'$ and $N$ by $N''$, we are left with the same problem but the size has decreased by a factor of $(1-\delta/10)$. Continuing in this manner, after $O(\log r/\delta)$ stages we will be working over logarithmic size sets, for which we can check every element (it will be more convenient to take the recursion only to logarithmic size sets, instead of to constant size sets as one might expect). Note that Corollary 4.1 ensures that all reductions in the size of $N$ occur with high probability, as we get a geometric progression.

We still have one problem left: each stage needs good blocks of different lengths. This was the main obstacle to improving the $n^{O(\log n)}$ algorithm of [Zu1] to polynomial. We get around this problem by introducing a new lemma about paths on expander graphs, which allows us to find the good blocks without using brute force.

It suffices to get $\delta/2$-good blocks, simply by replacing $\delta$ by $\delta/2$ in the above argument. Suppose at the $i$th stage we need a good block of length $l_i$. We will get this $\delta/2$-good block from the $i$th block of length $r$ output by the PRB-source. Actually, the proof will be easier if we assume we have $k$ $\delta$-good $r$-bit blocks available to us, as implied by Lemmas 4.1 and 4.2. We may assume without loss of generality that $l_i|r$, for if it does not, we may pad the $i$th $r$-bit block with 0's until it does, and we will still have a $\delta/2$-good block of length $r$ (so we will have to replace $\delta$ by $\delta/2$).

To get the good block for the $i$th stage, let an adversary label $2^{\delta l_i/2}$ of the $l_i$-bit strings as bad, so the rest are good. Subdivide a block of length $r$ into $b_i = r/l_i$ blocks of length $l_i$, and call a block of length $r$ good if at least $\delta/4$ fraction of the sub-blocks are good. Assuming $l_i \geq 4/\delta$, Lemma 4.1 implies that less than $2^{\delta r}$ $r$-bit strings of length $r$ are not good. Thus, this is a legitimate labelling, so by the techniques of Lemmas 4.1 and 4.2 we may assume we can get $k$ blocks, where for all $i$ the $i$th block has at least $\delta/4$ fraction of sub-blocks of length $l_i$ good.

To get one good block of each length, we could use brute force and try all possible combinations of sub-blocks, one of each length. This, however, yields an $r^{O(\log r)}$ algorithm. If there were only one good sub-block of each length, then we would have to use brute force; however, we know that a constant fraction of the sub-blocks are good. We exploit this fact by using the following lemma:

**Lemma 4.5** *Let $G = (V, E)$ be an expander graph (directed or undirected) on $n = |V|$ nodes, such that every set of size $n/2$ has at least $n/2 + \alpha n$ neighbors. For any $k$, let $S_1, \ldots, S_k$ be arbitrary subsets of $V$ such that $|S_i| \geq (1-\alpha)n$. Then there exists a path $v_1, \ldots, v_k$ in $G$ such that $v_i \in S_i$.*

**Proof.** By induction on $k$ in the statement: there are at least $n/2$ endpoints $v_k$ in such paths $v_1, \ldots, v_k$. This is true for $k = 1$; if it is true for a given $k$, then any neighbor of an endpoint that also lies in $S_{k+1}$ is a possibility for $v_{k+1}$. But the neighbor set is of size at least $n/2 + \alpha n$, so its intersection with $S_{k+1}$ is at least $n/2$. □

Viewing the degree 7 expanders in [GG] as directed graphs (instead of bipartite graphs), we see that these graphs have $\alpha = (2 - \sqrt{3})/4$, as well as being explicitly constructible.

We apply the lemma as follows: Using $n = r$, we'd like to set $S_i$ equal to the set of good sub-blocks of the $i$th good block $B_i$. However, we would then have a fraction $\delta/4$ of the sub-blocks in $S_i$, whereas we want a fraction $1 - \alpha$. We therefore set $n = r^m$, and let $S_i$ represent $m$-tuples of sub-blocks, of which at least one is good, where $m$ is chosen so that $(1-\delta/4)^m \leq \alpha$. We also must work with a modulus, so more formally:

$$S_i = \{(s_1, \ldots, s_m) | \text{for some } j, \text{ the } s_j (\mathrm{mod} r/l_i)\text{th sub-block of } B_i \text{ is good}\}.$$

We then don't have to run our algorithm on all combination of sub-blocks, but only on ones given by paths on the [GG] expanders. Since there are $7^{O(\log r)}$ paths of length $O(\log r)$ in a degree 7 expander, and each such path contains $m^{O(\log r)}$ combinations of sub-blocks, this yields a polynomial time algorithm.

$\square$

**Remark 4.1** *The time dependence on $\delta$ is $r^{O(\frac{1}{\delta^2}\log\frac{1}{\delta})}$, and $R = O(r\log r/\delta^3)$. This is because there are $O(\log r/\delta)$ stages, so $2^{O(\log r/\delta)}$ strings are produced for every sequence of possibly good blocks. The dominant factor is the number of sequences of possibly good blocks: each stage requires $\lceil r/t \rceil = O(1/\delta)$ good blocks, so $O(\log r/\delta^2)$ good blocks are needed in total. In order to get this many good blocks, we must request $O(\log r/\delta^3)$ blocks from the source (i.e. $R = O(r\log r/\delta^3)$, and try all sequences of length $O(\log r/\delta^2)$. This gives $\binom{\log r/\delta^3}{\log r/\delta^2} = 2^{O(\frac{\log r}{\delta^2}\log\frac{1}{\delta})}$. For each sequence of possibly good blocks, we must try all paths of $m$-tuples in an expander, which adds another factor of $m^{O(\log r/\delta^2)} = 2^{O(\frac{\log r}{\delta^2}\log\frac{1}{\delta})}$. Multiplying these factors together gives the result.*

# Chapter 5

# Simulating BPP

Our RP simulation consisted of two parts: Lemmas 4.1 and 4.2 reduced the problem to simulating RP using an $(r, \delta)$ PRB-source, and the rest of the proof showed how to simulate RP using an $(r, \delta)$ PRB-source. Most of the difficulty in extending the previous proof to BPP lies in this first reduction. For BPP, it is difficult to argue anything unless *all* the blocks are good (good meaning essentially having small conditional probability of being output). One bad block is enough to lose control over the majority of the output test strings.

We solve this problem by considering several permutations, and arguing that most of the permutations will have all "good" blocks. The argument requires more care than at first appears, as we must argue first about initial segments being good, and then individual blocks as being good. We use few permutations by using pairwise independence.

We first give an algorithm, Algorithm B, to simulate BPP, assuming we could always get good blocks. Our algorithm will be essentially the same as the RP algorithm, except that instead of using a function $f : \{0,1\}^r \times \{0,1\}^w \rightarrow \{0,1\}^r$, we use the same construction to give an $f : \{0,1\}^r \times \{0,1\}^w \rightarrow \{0,1\}^{2r}$. There is now a way of associating $x \in \{0,1\}^r$ with 2 other $r$-bit strings: namely, compute $f(x, X)$, where $X$ is a good block from the source, and split it into two $r$-bit strings $x_1$ and $x_2$. In our recursion, the answer we associate with $x$ is the majority of the answers associated previously with $x$, $x_1$, and $x_2$.

Let $N$ be the set of strings that give the wrong answer. Thus,

$$N' = \{x | \text{at least two of } x, x_1, x_2 \text{ are in } N\}.$$

Using the fact that $f(x, X)$ is quasi-random within $2^{-3\delta r/16}$ if $x$ is picked uniformly from $N$, (and surely if $x$ is picked uniformly from $\{0,1\}^r - N$), we get

**Lemma 5.1** *If* $|N| \leq 2^{(1-\delta/8)r}$, *then* $E[|N'|] < 2^{(1+o(1))(1-\delta/8)^2 r}$.

**Proof.** Similar to Lemma 4.4. Basically,

$$E[|N'|] < |N| Pr[x_1 \in N \text{ or } x_2 \in N | x \in N] + 2^r Pr[x_1 \in N \text{ and } x_2 \in N | x \in \{0,1\}^r - N]$$

$$\leq 2|N|^2 2^{-r} + O(|N| 2^{-3\delta r/16}) \leq 2^{(1+o(1))(1-\delta/8)^2 r}.$$

□

Given this lemma, the rest follows as in the RP case, once we can assume all our blocks are good. The rest of the proof is devoted to showing that we can make this assumption. The rest of this proof can also replace Lemmas 4.1 and 4.2 in our RP simulation, but then our RP simulation would use $r^{O(1/\delta)}$ random bits, instead of $O(r \log r)$.

We begin the reduction by reducing our problem to simulating BPP from the following kind of source:

**Lemma 5.2** *Let $S$ be a source outputting $b$ blocks of length $l = \lceil 3/\delta \rceil$ one at a time, controlled by an adversary with the following restrictions:*

*(i) the adversary must decide on-line whether to completely control the value of the output block, or only to select a set of size $\geq 2^{\delta l/3}$ from which the block is chosen uniformly at random;*

*(ii) the adversary must yield complete control on at least $\delta b/3$ blocks.*

*Then an algorithm that simulates BPP using $b$ blocks of length $l$ from any such source $S$ also simulates BPP using $bl$ bits from a $\delta$-source.*

**Proof.** Divide the $bl$ bits output from the $\delta$-source into $b$ blocks of length $l$. By Lemma 4.1, with high probability a fraction at least $\delta/3$ of the blocks will be $\delta/3$-good. Once this fraction occurs, we can build the two trees as in Lemma 4.2 to complete the proof. □

Because the blocks of length $l$ will serve as components of other, bigger blocks, we call them sub-blocks. We also call a sub-block that the adversary does not completely control a $\delta/3$-random sub-block. Note how this compares with $\delta/3$-good: intuitively, they correspond to the same idea, but the proofs will be easier using the $\delta/3$-random definition.

Assume without loss of generality that $b$ is a prime power, e.g. a power of 2. We will use $b^2 - b$ "permutations" (they will not really be permutations, but subsequences of sub-blocks), which except for a slight modification is a probability space where $\sqrt{b}$ of the sub-blocks are chosen pairwise independently. Namely, we work over the finite field $F$ with $b$ elements, and for all $(x, y) \in (F - \{0\}) \times F$, we let the $i$th sub-block in our sequence correspond to the element $ix + y$ in $F$. Note that because $x \neq 0$, all the sub-blocks in the subsequence are different.

We would like to argue that for most such subsequences, all of the larger blocks, each composed of many small sub-blocks, are good in the new, permuted order. The problem with this is that a sub-block which was $\delta/3$-random in the old order may not be $\delta/3$-random in the new order. For example, imagine two sub-blocks which are always equal, but otherwise uniformly random and independent of all the other sub-blocks. Then whichever sub-block appears first in the permuted string will be $\delta/3$-random, and the other will not.

We therefore take a different approach. We first argue that with high probability, all initial segments of length at least $m_0 = 24l \log r/\delta = O(\log r/\delta^2)$ in the permuted string will be good. We then show how to obtain good blocks from a source that outputs strings with all initial segments good.

To do this first part, we first argue that for most permutations, all initial segments contain many sub-blocks that were $\delta/3$-random in the unpermuted order; we then argue that this implies that most permutations contain only good initial segments (in the new permuted order). In fact, although it is true that all initial segments will be good, we will only need to deal with initial segments with lengths $m_i = \lceil m_0 d^i \rceil$ for some $i$, where $d \geq 2$ is a constant to be chosen later.

**Lemma 5.3** *For a fraction $1 - 1/\log r$ of the subsequences, all initial strings of length $m_i$ contain at least a fraction $\delta/6$ of $\delta/3$-random sub-blocks.*

**Proof.** When a subsequence is picked at random from a pairwise independent space, we can use Chebychev's inequality to bound the probability that a block of length $m_i$ contains at least $\delta/6$ fraction of $\delta/3$-random blocks (see e.g. [CG1]), given that a fraction at least $\delta/3$ of all sub-blocks are $\delta/3$-random. Using the value of $m_i$, this is at least $1 - 2^{-(i+1)}/\log r$. This probability only increases by removing the points from the space that we removed, so this bound holds for our probability space. Adding the geometric series, the probability that this holds for all non-negative integers $i$ is at least $1 - 1/\log r$. $\square$

**Lemma 5.4** *The probability that at least* $1-2/\log r$ *fraction of the subsequences have initial segments that are* $\delta^2/36$*-good for each length* $m_i$ *is at least* $1 - 1/r$.

**Proof.** The idea is that if an initial segment has many $\delta/3$-random sub-blocks, it will be good. It is not this simple, however, because the adversary can choose which subsequences to give few $\delta/3$-random sub-blocks after seeing some of the output blocks, i.e. if some subsequences look like they will be bad anyway, she need not focus on these. To get around this, we will give the adversary even more power: namely, she can decide which subsequences to give few $\delta/3$-random sub-blocks after seeing how all subsequences would have turned out with many $\delta/3$-random sub-blocks. Since the adversary can give at most $1/\log r$ subsequences few $\delta/3$-random sub-blocks, we need only bound the probability that, if every subsequence contained many $\delta/3$-random sub-blocks, that at least $1/\log r$ of them would have a $\delta^2/36$-bad initial segment of length $m_i$.

If an initial segment of length $m_i$ has at least $\delta/6$ fraction of $\delta/3$-random sub-blocks, then the probability of this initial segment taking on a specific value is at most $2^{-\delta^2 m_i/18}$. Thus, the probability it is $\delta^2/36$-bad is at most $2^{\delta^2 m_i/36}$ times this value, or $2^{-\delta^2 m_i/36}$. The probability that a subsequence has a bad initial segment for some $i$ is at most the sum of these, which is at most $2 \cdot 2^{-\delta^2 m_0/36} \leq 2/r^2$, using the value of $m_0$ in terms of $l$ and $l \geq 3/\delta$. Thus, the expected fraction of bad subsequences is at most $2/r^2$, so by Markov the probability this fraction exceeds $1/\log r$ is at most $2\log r/r^2 \leq 1/r$. $\square$

For ease of reading, redefine $\delta$ as $\delta^2/36$, so now we know that for most subsequences, all initial segments of length $m_i$ are $\delta$-good. Let us focus now on these subsequences. We now explicitly set the $d$ in the definition of $m_i$; namely $d = 2/\delta$ so $m_i = \lceil m_0(2/\delta)^i \rceil$. We next show that the initial segments of length $m_i$ being $\delta$-good implies that the block containing bit positions $m_i + 1$ through $m_{i+1}$ is $\delta/2$-good. For fix $2^{\delta(m_{i+1}-m_i)/2}$ bad strings for such a block; even allowing all $2^{m_i}$ strings to be bad for the initial segment causes $2^{m_i+\delta(m_{i+1}-m_i)/2} \leq 2^{\delta m_{i+1}}$ possibilities for bad strings in the initial segment of length $m_{i+1}$. Thus, we can force a $\delta$-good initial segment of length $m_{i+1}$ to lie outside such a set.

Again to ease the comparison to previous lemmas, redefine $\delta$ as $\delta/2$. Once we have $\delta$-good, disjoint blocks, we can select the smaller size blocks that we will need by the same process of picking sub-blocks in a pairwise independent manner. Our algorithm needs good blocks of sizes $cr, cr(1-\delta/8), cr(1-\delta/8)^2, \ldots, m_0 = O(\log r/\delta^2)$; we rewrite these block lengths as approximately $l_i = m_0(1-\delta/8)^{-i}$. Our algorithm needs $O(1/\delta)$ good blocks of each size. Picking in the pairwise independent manner, the probability that a fraction $1-(1-\delta/8)^i/k\log r$ of the blocks of size $l_i$ are $\delta^2/36$-good is at least $1-1/r$. Thus, with high probability a fraction $1-O(1/\delta)\sum(1-\delta/8)^i/\log r = 1-O(1/\delta^4\log r)$ of the sequences of different size blocks contain all good blocks. Because Algorithm B using only good blocks always outputs the right answer, we have:

**Theorem 5.1** *For all $\delta > 0$, there is an algorithm that simulates BPP using a $\delta$-source.*

# Chapter 6

# Extracting Almost-Random Bits

## 6.1 PRB-Sources

The first source we consider is the PRB-source of Chor-Goldreich. We show how to extract almost-random bits at close to optimal rates in two situations. We first define almost-random and rate:

**Definition 6.1** *A sequence of bits $b_1 \dots b_k$ is almost-random if the probability distribution on $b_1 \dots b_k$ is quasi-random within $n^{-\omega(1)}$.* Thus almost-random bits are indistinguishable from perfectly random bits by polynomial-time machines.

**Definition 6.2** *The* rate *at which an algorithm outputs almost-random bits is*

$$\frac{\textit{\# of almost-random bits output}}{\textit{total \# of bits used from sources}}$$

Thus, the optimal rate from several $(l, \delta)$ PRB-sources is $\delta$.

The first situation where we can extract bits is when we have two sources, one low quality (i.e. small $\delta$) and the other high quality (i.e. large $\delta$), but we can use very few bits, say $O(\log^2 n)$, from this second source. The second is when we have three low-quality sources.

This first situation is useful if we believe that we have a small number of reasonable quality bits, such as some of those from the RAND table. We could use a small number of the bits from the RAND table for the lifetime of our computer, and the algorithm is so simple that it should be practical.

Before beginning our proofs, we follow Chor and Goldreich by considering the worst-case sources, flat sources:

**Definition 6.3 (CG2)** *An $(l, \delta)$ PRB-source is* flat *if for all $i$ and all $l$-bit strings $y_1, \ldots, y_i$,*

$$Pr[x_i = y_i | x_1 = y_1, \ldots, x_{i-1} = y_{i-1}] = 2^{-\delta l} \text{ or } 0.$$

**Theorem 6.1 (CG2)** *An algorithm $A$ extracts almost-random bits from any $t$ $(l, \delta)$ PRB-sources if and only if $A$ does so from any $t$ flat $(l, \delta)$ PRB-sources.*

We can now prove the following theorem, which applies to the first situation above by taking $k = \log^2 n$ and $\epsilon(n)$ equal to a small constant.

**Theorem 6.2** *Suppose we wish to output $n$ bits quasi-random within $2^{-k}$ from an $(l, \delta)$-PRB source at a rate $\delta - \epsilon(n)$. Let $s = \frac{6}{\epsilon(n)}(k + \log n)$, and assume $l \leq s$. Then we need only $2s$ bits from an independent $(l', 1 - \epsilon(n)/3)$-PRB source, where $l' \leq 2s$.*

**Proof.** Assume w.l.o.g. that both sources are flat and that $l = sl' = 2s$. View the bits from the high quality source as a hash function $h$ mapping $s$ bits to $t = (\delta - \epsilon(n))s$ bits; say the source picks $h$ uniformly at random from some subset $C$ of the hash functions. Let the blocks from the low quality source be $x_1, \ldots, x_m$, where $m = n/t$. We claim that $h(x_1), \ldots, h(x_m)$ is quasi-random within $2^{-k}$. This follows from the following lemma.

This lemma is a generalization of a similar lemma in [IZ], and essentially strengthens related lemmas in [Va2] and [CG2]:

**Lemma 6.1** *Let $x_1, \ldots, x_k$ be $l$-bit block outputs from an $(l, \delta)$ PRB-source. Pick $h$ uniformly from a universal family of hash functions mapping $l$ bits to $\delta l - 2e$ bits. Then the distribution $(h, h(x_1), \ldots, h(x_k))$ is quasi-random to within $k2^{-e}$.*

**Proof.** Let $s = \delta l - 2e$, so $|H| = 2^{l+s}$. We use boldface letters to denote random variables, and denote

$$p(h, z_1, \ldots, z_t) = Pr[\mathbf{h} = h, h(\mathbf{x_1}) = z_1, \ldots, h(\mathbf{x_t}) = z_t],$$

$$q(x_1, \ldots, x_t) = Pr[\mathbf{x_1} = x_1, \ldots, \mathbf{x_t} = x_t] = 2^{-\delta tl}$$

if $x_1, \ldots, x_t$ is a possible output from our flat PRB-source.

Our proof is by induction on $k$. The case $k = 1$ follows immediately from the Leftover Hash Lemma. Now suppose it is true for a given $k$, i.e. that

$$\sum_{h,z_1,\ldots,z_k} |p(h, z_1, \ldots, z_k) - 2^{-(l+(k+1)s)}| < 2k2^{-e}.$$

Then using the triangle inequality,

$$\sum_{h,z_1,\ldots,z_{k+1}} |p(h, z_1, \ldots, z_{k+1}) - 2^{-(l+(k+2)s)}|$$

$$\leq \sum_{h,z_1,\ldots,z_{k+1}} |p(h, z_1, \ldots, z_{k+1}) - p(h, z_1, \ldots, z_k)2^{-s}| + \sum_{h,z_1,\ldots,z_{k+1}} |p(h, z_1, \ldots, z_k)2^{-s} - 2^{-(l+(k+2)s)}|.$$

Noting that there are $2^s$ possible $z_{k+1}$'s bounds the second term by $2k2^{-e}$, using the inductive assumption. To bound the first term, we use that

$$p(h, z_1, \ldots, z_k) = \sum_{\substack{x_1,\ldots,x_k \\ (\forall i)h(x_i)=z_i}} Pr[\mathbf{h} = h]q(x_1, \ldots, x_k) = \sum_{\substack{x_1,\ldots,x_k \\ (\forall i)h(x_i)=z_i}} q(x_1, \ldots, x_k)2^{-(l+s)}$$

and

$$p(h, z_1, \ldots, z_{k+1}) = \sum_{\substack{x_1,\ldots,x_k \\ (\forall i)h(x_i)=z_i}} Pr[\mathbf{h} = h]q(x_1, \ldots, x_k)Pr[h(\mathbf{x_{k+1}}) = z_{k+1}|\mathbf{x_1} = x_1, \ldots, \mathbf{x_k} = x_k].$$

Using these two identities and the triangle inequality, we bound the first term by

$$\sum_{h,z_1,\ldots,z_{k+1}} \sum_{x_1,\ldots,x_k:h(x_i)=z_i} |q(x_1, \ldots, x_k)Pr[\mathbf{h} = h]Pr[h(\mathbf{x_{k+1}}) = z_{k+1}|\mathbf{x_1} = x_1, \ldots, \mathbf{x_k} = x_k] - q(x_1, \ldots, x_k)2^{-(\cdots}$$

$$= \sum_{x_1,\ldots,x_k} q(x_1, \ldots, x_k) \sum_{h,z_{k+1}} |Pr[\mathbf{h} = h]Pr[h(\mathbf{x_{k+1}}) = z_{k+1}|\mathbf{x_1} = x_1, \ldots, \mathbf{x_k} = x_k] - 2^{-(l+2s)}|.$$

But the second summation is bounded above by $2 \cdot 2^{-e}$, since it is twice the quasi-randomness of $(\mathbf{h}, \mathbf{h}(\mathbf{x_{k+1}}))$ conditional on $\mathbf{x_1}, \ldots, \mathbf{x_k}$. Using $\sum_{x_1,\ldots,x_k} q(x_1, \ldots, x_k) = 1$ bounds the first term by $2 \cdot 2^{-e}$ and proves the inductive claim.

□

In order to generate almost-random bits from three low-quality sources, we proceed as in [CG2]. We use their lemma:

**Lemma 6.2 (CG2)** *For $l = O(\log \log n)$ and all $\delta > 0$, in polynomial time we can transform $n$ bits from each of two independent $(l, \delta)$ PRB-sources into $\Omega(n)$ bits from an $(O(\log \log n), 1 - 1/\log n)$ PRB-source.*

To prove this, they show by a counting argument that there exists such a transformation, and because $l$ is so small this function can be found. We are now ready to prove:

**Theorem 6.3** *For $l = O(\log \log n)$ and all $\delta > 0$, in polynomial time we can extract $n$ almost-random bits from three independent $(l, \delta)$ PRB-sources at a rate $\delta - o(1)$.*

**Proof.** By Lemma 6.2, we can use $O(\log^3 n)$ bits from two sources to get $O(\log^3 n)$ bits from an $(O(\log \log n), 1 - 1/\log n)$ PRB-source. Then use Theorem 6.2 with these $O(\log^3 n)$ bits and the third source to output almost random bits at a rate $\delta - \epsilon(n)$, where $\epsilon(n) = 3/\log n$. $\square$

## 6.2 $\delta$-Sources: Case $\delta > 1/2$

First we show that it is impossible to extract an almost-random bit from one $\delta$-source. This has been shown for weaker models, such as the semi-random sources of [SV], but in our case the proof is particularly simple:

**Lemma 6.3** *For any $\delta < 1 - 1/n$, it is impossible to extract a bit from $n$ bits of a $\delta$-source that takes on both values 0 and 1 with non-zero probability.*

**Proof.** Suppose we had a function $f : \{0,1\}^n \to \{0,1\}$ that claimed to do the above. Suppose without loss of generality that $f$ takes on the value 0 at least as often as the value 1. Then any source with $\delta < 1 - 1/n$ could output only values in $f^{-1}(0)$, contradicting the claims about $f$ extracting a non-trivial bit. $\square$

For $\delta > 1/2$, from two sources, we can extract almost-random bits at a constant rate. From a constant number of sources, using the techniques of the previous section we can extract almost-random bits at nearly the optimal rate.

**Theorem 6.4** *Suppose we have two independent $\delta$-sources with parameters $\delta_1$ and $\delta_2$ with $\delta_1 + \delta_2 > 1$. Then using $O(n)$ bits from the two sources, there is an algorithm that extracts $n$ bits that are quasi-random within $2^{-\Omega(n)}$.*

**Proof.** We view the first source as a hash function $h$ from $m$ bits to $n$ bits, which requires $n + m$ bits, and the second source as an $m$-bit string $s$. The output of the algorithm is $h(s)$. By Lemma 2.3, $h(s)$ is quasi-random within $\sqrt{2^{m+n-\delta_2 m - \delta_1(m+n)}}$. Choosing $m = \left\lceil \frac{n}{\delta_1 + \delta_2 - 1} \right\rceil$ gives the result. $\square$

To achieve nearly the optimal rate, we apply the techniques of the previous section.

**Theorem 6.5** *For all $\delta > 1/2$, for all $\epsilon > 0$, there is an algorithm that produces almost-random bits at rate $\delta - \epsilon$ using strings from a constant number of independent $\delta$-sources.*

**Proof.** (Sketch.) We use the techniques of Lemma 6.1 to get approximately $(1 + 1/\delta)n$ almost-random bits. View this as a hash function $h$ from $n/\delta$ bits to $(1 - \epsilon/2)n$ bits. Request $n/\delta$ bits from many $\delta$-sources, and apply $h$ to each of these strings. The Leftover Hash Lemma implies that these new bits will be quasi-random within $2^{-\epsilon n/4}$. □

## 6.3 $\delta$-Sources: Case $\delta > 0$

Chor and Goldreich [CG2] have shown how to extract one almost-random bit from two independent $\delta$-sources for all $\delta > 0$ under the Paley Graph Conjecture. In this section, we show how to generate almost-random bits at nearly the optimal rate from a constant number of independent $\delta$-sources for all $\delta > 0$ under the Generalized Paley Graph Conjecture.

The Generalized Paley Graph Conjecture asserts the "independence" of the additive and multiplicative groups modulo a prime in the following sense: if $A$ and $B$ are large enough sets, then the discrete logs of $A + B$ are "well distributed." In order to define well distributed, we must define characters. A multiplicative character $\chi$ is a homomorphism from $Z_p^*$ to the complex numbers, i.e. $\chi(ab) = \chi(a)\chi(b)$ and $\chi(1) = 1$. We also define $\chi(0) = 0$. There are $p - 1$ multiplicative characters.

The well-known Paley Graph Conjecture asserts that the elements of a large rectangle in a specific matrix have small sum:

**Paley Graph Conjecture:** Let $\chi_2$ be the quadratic character $\chi_2(a) = (\frac{a}{p})$. Then for any $\delta > 0$, there exists an $\epsilon > 0$ such that for sufficiently large primes $p$, if $|S|, |T| \geq p^\delta$, then

$$\Big| \sum_{s \in S, t \in T} \chi_2(s + t) \Big| \leq |S||T|/p^\epsilon.$$

We conjecture that something more general is true:

**Generalized Paley Graph Conjecture:** The Paley Graph Conjecture remains true if $\chi_2$ is replaced by any non-trivial character $\chi$.

There are good reasons for believing that this more general conjecture is true if the Paley Graph Conjecture is true. First, techniques used to evaluate character sums involving $\chi_2$ work for any non-trivial character; for example, both the Paley Graph Conjecture and its general version are proved using the same methods for $\delta > 1/2$. Second, as pointed out by Lenstra [Len], the case with $\chi_2$ is probably the worst case, since $\chi_2(a)$ has a 1/2 "probability" of being a specific value, so it should be easier to find a counter-example in this case.

The reason we need the Generalized Paley Graph Conjecture is that it allows us to improve Lemma 2.3 for the linear congruential hash function modulo an $n$-bit prime. Note that we can find an $n$-bit prime using one $\delta$-source by using Lemma 3.1.

We write our hash function in the form $h(s) = (s+t)t'$, where $h = (t, t')$. We will also have the slightly stronger assumption that $t$ and $t'$ are chosen from large sets $T$ and $T'$, and $s$ from a large set $S$; we do allow $T'$ to depend arbitrarily (as long as it is of the required size) on the element $t$ chosen from $T$.

In order to compute the probability that $h(s)$ is in a specific set, we need to analyze the expected number of solutions to

$$b = (s+t)t', b \in B, s \in S, t' \in T', \tag{6.1}$$

where the expectation is over $t$ picked uniformly from $T$, and $T'$ can depend on the element $t$ picked from $T$.

To do this, we outline some work from [AB+]. First, we need some basic results about multiplicative characters from e.g. [Hua]. For $a \neq 1$,

$$\sum_{\chi} \chi(a) = 0,$$

and if $\chi$ is not the trivial character $\chi_0$ which sends everything in $Z_p^*$ to 1, then

$$\sum_{a \in Z_p} \chi(a) = 0.$$

For $A \subseteq Z_p^*$, $\chi$ a multiplicative character, define

$$\psi_A(\chi) = \sum_{a \in A} \chi(a),$$

and

$$\Psi_A = \max_{\chi \neq \chi_0} \{|\psi_A(\chi)|\}.$$

Then

$$\sum_{\chi} |\psi_A(\chi)|^2 = (p-1)|A|.$$

**Lemma 6.4 (AB+)** *The number of solutions to $ab = c$, where we restrict $a \in A$, $b \in B$, and $c \in C$, $A, B, C \subseteq Z_p^*$, is*

$$\frac{1}{p-1} \sum_{\chi} \psi_A(\chi)\psi_B(\chi)\psi_C(\chi^{-1}), \tag{6.2}$$

*which is at most*

$$\frac{|A||B||C|}{p-1} + \Psi_A \sqrt{|B||C|}. \tag{6.3}$$

From the bound (6.3), it would seem like a good idea to bound the expectation of $\Psi_{S+t}$. We can't do this, but the Generalized Paley Graph Conjecture will help us obtain a bound on the expectation of $|\psi_{S+t}(\chi)|^2$, which we will show is good enough.

**Lemma 6.5** *Suppose the Generalized Paley Graph Conjecture is true. Then for any $\delta > 0$, there exists an $\epsilon > 0$ such that for large primes $p$, if $|S|, |T| \geq p^\delta$, then*

$$\sum_{t \in T} |\psi_{S+t}(\chi)| \leq |S||T|/p^\epsilon.$$

**Proof.** (Sketch) Let $z_t = \psi_{S+t}(\chi)$, and set $z_t = a_t + b_t i$, $a_t, b_t$ real. Let $C = \sum_{t \in T} |z_t|$. It suffices to find a $W \subseteq T$ such that $|W| \geq |T|/2$ and $\sum_{w \in W} |z_w| \geq C/6$.

Now either $\sum_{t \in T} |a_t| \geq C/2$ or $\sum_{t \in T} |b_t| \geq C/2$; w.l.o.g. suppose $\sum_{t \in T} |a_t| \geq C/2$. Let $T_1 = \{t \in T | a_t \geq 0\}$, and $T_2 = T \setminus T_1$. Then either $|T_1| \geq |T|/2$ or $|T_2| \geq |T|/2$; say it is $T_2$. If $|\sum_{t \in T_1} z_t| \geq C/6$, then we can take $W = T_2$. If not, $|\sum_{t \in T_1} z_t| \geq \sum_{t \in T_1} a_t \geq C/2 - C/6 = C/3$. Then, by the triangle inequality, $\sum_{t \in T} z_t \geq C/3 - C/6 = C/6$, so we can take $W = T$. □

**Corollary 6.1** *Suppose the Generalized Paley Graph Conjecture is true. Then for any $\delta > 0$, there exists an $\epsilon > 0$ such that for large primes $p$, if $|S|, |T| \geq p^\delta$, then*

$$\sum_{t \in T} |\psi_{S+t}(\chi)|^2 \leq |S|^2|T|/p^\epsilon.$$

**Proof.** Follows from Lemma 6.5 and $|\psi_{S+t}(\chi)| \leq |S|$. □

We are now ready to prove:

**Lemma 6.6** *Under the Generalized Paley Graph Conjecture, for all $\delta > 0$ and large $n$, there exists an $\epsilon > 0$ such that for all $\delta' \geq \delta$ and all $S, T, T', B \subseteq \{0,1\}^n$ with $|S|, |T| \geq 2^{\delta n}$, $|T'(t)| \geq 2^{\delta' n}$ but otherwise can depend on the element $t$ chosen from $T$,*

$$\left| Pr_{s \in S, t \in T, t' \in T'}[(s+t)t' \in B] - \frac{|B|}{p} \right| \leq \sqrt{|B| 2^{-(\delta'+\epsilon)n}}$$

**Proof.** All expectations will be for $t$ picked uniformly at random from $T$. The probability above is equal to the expected number $e$ of solutions to (6.1) divided by $|S||T'|$. Using (6.2) and Cauchy-Schwartz,

$$\begin{aligned}
&|(p-1)e - |B||S||T'||\\
&= E\Big[\sum_{\chi \neq \chi_0} \psi_B(\chi^{-1}) \psi_{S+t}(\chi) \psi_{T'}(\chi)\Big]\\
&\leq E\Big[\sqrt{\Big(\sum_{\chi \neq \chi_0} |\psi_B(\chi^{-1})|^2 |\psi_{S+t}(\chi)|^2\Big)\Big(\sum_{\chi \neq \chi_0} |\psi_{T'}(\chi)|^2\Big)}\Big]\\
&\leq \sqrt{\Big(\sum_{\chi \neq \chi_0} |\psi_B(\chi^{-1})|^2 E\left[|\psi_{S+t}(\chi)|^2\right]\Big)(p-1)|T'|}.
\end{aligned}$$

Using the Generalized Paley Graph Conjecture, we get an $\epsilon > 0$ such that

$$\begin{aligned}
&\left| e - \frac{|B||S||T'|}{p-1} \right|\\
&\leq \frac{1}{p-1} \sqrt{\sum_{\chi \neq \chi_0} |\psi_B(\chi^{-1})|^2 \frac{|S|^2}{p^\epsilon}(p-1)|T'|}\\
&\leq \frac{1}{p-1} \sqrt{(p-1)|B| \frac{|S|^2}{p^\epsilon}(p-1)|T'|}\\
&= |S| \sqrt{|B||T'|/p^\epsilon}.
\end{aligned}$$

Modifying the $\epsilon$ to take into account $2^{n-1} < p < 2^n$ yields the lemma. □

**Lemma 6.7** *Suppose that $s$ and $t$ are taken as the outputs of two $\delta$-sources, and $t'$ from a $\delta'$-source. Then under the Generalized Paley Graph Conjecture, for large $n$, there exists an $\epsilon > 0$ such that the distribution of $(s+t)t'$ is statistically indistinguishable from the output of a $(\delta' + \epsilon)$-source.*

**Proof.** We may assume the sources are flat. Our question therefore reduces to showing that the distribution indicated in Lemma 6.6 is quasi-random within an exponential amount to a source which doesn't put probability more than $2^{-\delta'+\epsilon/2}$ on any point, where the $\epsilon$ is from Lemma 6.6.

Order the points in decreasing order of the probabilities assigned to them. The worst case is when as much probability is assigned to the first few points as possible. For $i \geq 2^{\delta' n}$, the probability assigned to the $i$th point is at most

$$\sqrt{i 2^{-(\delta'+\epsilon)n}} - \sqrt{(i-1)2^{-(\delta'+\epsilon)n}}$$

$$\leq \frac{1}{2}\sqrt{2^{-(\delta'+\epsilon)n}/(i-1)} \leq 2^{-(\delta'+\epsilon/2)n}.$$

Because the first $2^{\delta' n}$ points have exponentially small probability, we are done. □

We can now show

**Lemma 6.8** *Under the Generalized Paley Graph Conjecture, for all $\delta > 0$, there is an algorithm that produces almost-random bits at a constant rate using strings from a constant number of independent $\delta$-sources.*

**Proof.** Use the above corollary to produce bits like those output from a $\delta' + \epsilon$-source. Using these bits as the new $t'$ and getting new $s$ and $t$ from two new $\delta$-sources, we get bits like those output from a $(\delta + 2\epsilon)$-source. Continuing in this manner, we finally get $n$ almost-random bits.

□

**Theorem 6.6** *Under the Generalized Paley Graph Conjecture, for all $\delta > 0$, for all $\epsilon > 0$, there is an algorithm that produces almost-random bits at rate $\delta - \epsilon$ using strings from a constant number of independent $\delta$-sources.*

**Proof.** The proof of Theorem 6.5 applies, with Lemma 6.8 taking the place of Lemma 6.1.
□

# Chapter 7

# Applications

## 7.1 The Difficulty of Computing MAX CLIQUE

Let $\alpha$ denote the size of the maximum clique. Feige, et.al. [FG+] showed

**Theorem 7.1** *If for some $\epsilon$ approximating $\alpha$ to within a factor of $2^{(\log n)^{1-\epsilon}}$ is in $\tilde{P}$, then $N\tilde{P} = \tilde{P}$ ($\tilde{P}$ denotes quasi-polynomial time, i.e. $2^{polylog}$).*

Yet it seems like a more balanced question to ask for an approximation factor as a function of $\alpha$, rather than the graph size. For example, there is a simple polynomial-time algorithm to distinguish between graphs having $\alpha$ equal to $O(1)$ or $2^{(\log n)^{1-\epsilon}}$, whereas the proof in [FG+] shows that it is difficult to distinguish between graphs having $\alpha$ equal to $n^\delta$ or $n^\delta 2^{(\log n)^{1-\epsilon}}$. In light of this, we show:

**Theorem 7.2** *If for any constant $t$ there is a quasi-polynomial time algorithm that always outputs a number between $\alpha^{1/t}$ and $\alpha^t$, then $N\tilde{P} = \tilde{P}$.*

Note that such an algorithm has to decide among only $O(\log\log n)$ values (some approximation of the form $2^{t^i}$ will do). Thus one might have thought it would be easier to construct such an approximation algorithm.

Our proof closely follows the proof of [FG+], making great use of the proof in [BFL] that NEXP = MIP. Actually, we really use the equivalent theorem (see [FRS]) that any language in NEXP is accepted by a polynomial time probabilistic oracle machine.

**Definition 7.1** *A language $L$ is accepted by a probabilistic oracle machine $M$ iff*

$$\begin{aligned} x \in L &\Rightarrow (\exists oracle\ O) Pr_r[M^O(x,r)] = 1 \\ x \notin L &\Rightarrow (\forall oracles\ O) Pr_r[M^O(x,r)] < 1/4. \end{aligned}$$

Denoting the maximum number of random and communication bits used on inputs of size $n$ as $r(n)$ and $c(n)$, respectively, Feige et.al. give the following improvement of [BFL]:

**Theorem 7.3 (FG+)** *Any language $L \in NTIME(T(n))$ (for $n \leq T(n) \leq 2^{poly(n)}$) is accepted by a probabilistic oracle machine running in time $T(n)^{O(\log\log T(n))}$ and having $r(n) + c(n) = O(\log T(n) \log\log T(n))$.*

Using this, they construct a graph $G_x$ which has a large clique iff $x \in L$. In order to do this, they define transcripts and a notion of consistency among them. A transcript is basically a set of questions to and answers from the oracle; two transcripts are consistent if the oracle is consistent:

**Definition 7.2 (FG+)** *A string $t = r, q_1, a_1, \ldots, q_l, a_l$ is a transcript of a probabilistic oracle machine $M$ on input $x$ if $|r| = r(n), |q_1, a_1, \ldots, q_l, a_l| \leq c(n)$, and for every $i$, $q_i = M(x, r, < q_1, a_1, \ldots, q_{i-1}, a_{i-1} >)$. A transcript is accepting if $M$ on input $x$, random string $r$, and history of communiction (questions and answers) $< q_1, a_1, \ldots, q_l, a_l >$ accepts $x$.*

**Definition 7.3 (FG+)** *Two transcripts $t = r, q_1, a_1, \ldots, q_l, a_l$ and $\hat{t} = \hat{r}, \hat{q}_1, \hat{a}_1, \ldots, \hat{q}_l, \hat{a}_l$ are consistent if for every $i$, $q_i = \hat{q}_i$ implies $a_i = \hat{a}_i$.*

We can now define $G_x$: the vertices are all accepting transcripts, and two nodes are connected iff the corresponding transcripts are consistent. It is then not hard to see:

**Lemma 7.1 (FG+)** $\max_O Pr_r[M^O(x,r)$ *accepts*$] \cdot 2^{r(n)} = \alpha(G_x)$.

Thus, if $x \in L$ then $\alpha(G_x) = 2^{r(n)}$ and if $x \notin L$ then $\alpha(G_x) < 2^{r(n)}/4$.

To get the second part of their theorem, Feige et.al. construct the graph $G'_x$ corresponding to a protocol $M'$. $M'$ runs $\log^{O(1)} T(n)$ independent iterations of $M$ on $x$. This reduces the error probability if $x \notin L$ and therefore produces a wider separation in the clique sizes.

Yet once we fix an oracle $O$, $M^O$ basically corresponds to a co-RP machine: always accepting when $x \in L$ and usually rejecting if $x \notin L$. Thus we can apply Theorem 4.2, which we rephrase here:

**Theorem 7.4** *Let $\epsilon > 0$ be given. Then we can choose a constant $c$ such that the following holds. Let $A$ be any co-RP algorithm that uses $r$ bits to reduce the error probability to $1/4$ if $x \notin L$. Then for $s = cr\log r$ there is an algorithm that uses $s$ random bits, produces*

*$m = poly(r)$ $r$-bit strings $y_1, \ldots, y_m$ such that if $x \notin L$, the probability that $A$ accepts all $y_i$ is at most $2^{-(1-\epsilon)s}$.*

**Proof of Theorem 7.2:** Take $\epsilon = 1/t^2$ and form $M''$ by running $M$ on $x$ with the random strings $y_1, \ldots, y_m$. Construct $G''_x$ corresponding to $M''$. Observe that if $x \in L$ then $\alpha(G''_x) = 2^s$, and if $x \notin L$ then $\alpha(G''_x) < 2^{s/t^2}$. Thus any algorithm that always outputs a number between $\alpha(G''_x)^{1/t}$ and $\alpha(G''_x)^t$ can always tell whether or not $x \in L$. Since $G''_x$ has at most $2^{m(r(n)+c(n))} = 2^{polylog(T(n))}$ vertices, this yields the theorem.

## 7.2 Implicit $O(1)$ Probe Search

Recall that implicit $O(1)$ probe search is the problem of arranging $n$ elements from the domain $\{1, \ldots, m\}$ in a table of size $n$ so that searching for an element can be done with a constant number of probes. In this section we show how to do this for $m$ superpolynomial in $n$. We rely heavily on the work of Fiat and Naor [FN], who showed a correspondence between the existence of an $O(1)$ probe search scheme and the constructibility of a certain combinatorial structure called a rainbow:

**Definition 7.4** *A $(c, m, n, t)$-rainbow is a coloring of all $t$-tuples (without repetitions) of elements in $\{1, \ldots, m\}$ with $c$ colors, so that for any subset $S \subset \{1, \ldots, m\}$, $|S| = n$, all $c$ colors appear in the $t$-tuples over $S$.*

**Theorem 7.5 (FN)** *Let $c = \max(n, \log m)$. The existence of a $(c, m, n, t = O(1))$-rainbow yields an implicit $O(1)$ probe search scheme for $n$ elements from the domain $\{1, \ldots m\}$. Conversely, given an implicit $O(1)$ probe search scheme for $n$ elements chosen from the domain $\{1, \ldots, m\}$, an $(n, m, n, t = O(1))$-rainbow can be constructed.*

Thus Fiat and Naor prove their main theorem by constructing an $(n, n^k, n, O(1))$-rainbow for any constant $k$. Moreover, their proof implies the following useful lemma:

**Lemma 7.2 (FN)** *If $p_1, p_2, p_3$ are polynomials and $m$ is a function of $n$ such that a $(c, m, n, O(1))$-rainbow can be constructed, then a $(p_1(c), p_2(m), p_3(n), O(1))$-rainbow can be constructed.*

Fiat and Naor [FN] further show how to use a certain type of expander graph, called a disperser, to construct rainbows:

**Definition 7.5** *An $(m, n, d, a, b)$-disperser is a bipartite graph with $m$ nodes on the left side, each with degree $d$, and $n$ nodes on the right side, such that every subset of $a$ nodes on the left side is connected to at least $b$ nodes on the right.*

Sipser [Sip] showed that $(m, n, \log^2 n, n, n/2)$-dispersers exist, and conjectured that they could be explicitly constructed. Fiat and Naor [FN] show that under this conjecture, an $(n, m = n^{\log n}, n, O(1))$-rainbow can be constructed. Indeed, this construction is an immediate corollary of the following two lemmas:

**Lemma 7.3 (FN)** *Given a $(d, m, n, O(1))$-rainbow and an $(m, c, d, n, c/2)$-disperser, a $(c, m, n, O(1))$-rainbow can be constructed.*

**Lemma 7.4 (FN)** *For $m = n^{polylog(n)}$, there exists an explicit construction for a $(\log n, m, n, O(1))$-rainbow.*

We show how to use these two lemmas with our own disperser constructions to construct a rainbow with $m$ superpolynomial in $n$. First we note the connection between the efficient construction of dispersers and the simulation of RP using $\delta$-sources:

**Lemma 7.5 (CWi)** *For $\delta' > \delta$, RP can be simulated using $s$ bits from a $\delta'$-source iff a $(2^s, 2^r, poly(r), 2^{\delta s}, 2^{r-1})$-disperser can be efficiently constructed.*

Rewriting our results on simulating RP:

**Theorem 7.6** $(\forall n, \delta = \delta(n))$, *$(m(n, \delta) = n^{O(\log\log n/\delta^3)}, n, (\log n)^{O(\frac{1}{\delta^2}\log\frac{1}{\delta})}, m(n, \delta)^\delta, n/2)$-dispersers can be explicitly constructed.*

We also need that such dispersers exist for larger $m$:

**Corollary 7.1** $(\forall n, \delta = \delta(n), m \geq m(n, \delta))$ *$(m, n, (m/m(n, \delta))(\log n)^{O(\frac{1}{\delta^2}\log\frac{1}{\delta})}, m^\delta, n/2)$-dispersers can be explicitly constructed.*

**Proof.** Say $m = m(n, \delta)^k$. Express elements in $\{1, \ldots, m\}$ as elements of $\{1, \ldots, m(n, \delta)\}^k$. Connect $(v_1, \ldots, v_k)$ to all of the neighbors of each of $v_1, \ldots, v_k$ in the original disperser given by Theorem 7.6. Because for any set $S$ of size $m^\delta$, there must exist some component $i$ such that $S$ takes on at least $(m^\delta)^{1/k} = m(n, \delta)^\delta$, the corollary follows. $\square$

Nevertheless, using these dispersers and applying Lemma 7.3 to Lemma 7.4 does not yield any improvements immediately. Rather, we must apply Lemma 7.3 twice, construct a new disperser, and apply Lemma 7.3 again. First, we apply Lemma 7.3 twice:

**Lemma 7.6** *There is an explicit construction for an* $(n, m = n^{\log\log n/\delta^3}, m^\delta, O(1))$*-rainbow, for* $\delta = \log\log\log n/\sqrt{\log\log n}$.

**Proof.** Construct two dispersers using Corollary 7.1 and Theorem 7.6, respectively, both with the same $m$ and $\delta = \log\log\log n/\sqrt{\log\log n}$. For the first, use an $n$ in Corollary 7.1 (which we will call $n_1$ so as not to confuse with the $n$ in this lemma) equal to $n_1 = 2^{(\log\log n)^2/\log\log\log n}$. Then apply Lemmas 7.3 and 7.2 to construct an $(n_1, m, m^\delta, O(1))$-rainbow. Next construct a disperser from Theorem 7.6 with $n$ as in the statement of this lemma. Applying Lemmas 7.3 and 7.2 completes the proof.

□

The new disperser construction we will need is:

**Lemma 7.7** *For all* $n, \delta$, $(m = n^{O(1/\delta)}, n, 2^{O(\sqrt{\frac{\log n}{\delta}\log\log\frac{1}{\delta}})}, n/2)$*-dispersers can be explicitly constructed.*

**Proof.** Let $r = \lg n$. We wish to find an $r$-bit witness using the optimal number of random bits from a $\delta$-source, namely $O(r/\delta)$, but we are allowed a lot of time, i.e. we can try $2^{O(s)}$ possibilities, where $s = \sqrt{\frac{r}{\delta}\log\log\frac{1}{\delta}}$.

The key idea is to imagine we have an additional $O(s)$ independent and uniformly random bits – in reality we will try all $2^{O(s)}$ possibilities – and use the Leftover Hash Lemma. A first try would be to say a fraction $\delta/2$ of the $s$-bit blocks from the $\delta$-source are $\delta/2$-good; if we hash these good blocks down to $\delta s/2$ bits each and concatenate them together, we will have a quasi-random string. The problem with this is that it loses two factors of $\delta/2$, one because only a fraction $\delta/2$ of the blocks are good, the other because we have to hash down by a factor of $\delta/2$. This would require $O(r/\delta^2)$ bits from the $\delta$-source; we are allowed only $O(r/\delta)$.

Intuitively, we were throwing away a lot of our randomness before, because either many blocks which were, say, 1/2-good, and we assumed they were only $\delta/2$-good, or, say, 1/2 the blocks were $\delta/2$-good, and we assumed only a fraction $\delta/2$ were. We could be more efficient by assigning guesses about the goodness of each block, say among the values $0, \delta/4, \delta/2, \delta, \ldots, 2^k\delta$, where $k = \lfloor \lg 1/\delta \rfloor$. For one such set of guesses, each guess will either be between 1/2 and 1 times the actual "goodness," or else only $\delta/4$ below the actual goodness (because the goodness is in $(0, \delta/4)$ and we guessed 0). The second part causes us to lose at most $\delta/4$ of the goodness, and the first at most another constant factor, so this should keep us from losing much randomness. This is formalized in the following lemma:

**Lemma 7.8** *Suppose we ask for bl bits from a $\delta$-source, viewed as b blocks of length l. Assume $2^{\delta l/2} > k+4$, where $k = \lfloor \lg 1/\delta \rfloor$. Then there exists a sequence $\{\delta_i\}_{i=1}^{b}$, $\delta_i \in \{0, \delta/4, \delta/2, \ldots, 2^k\delta\}$, such that the ith block is $\delta_i$-good and $\sum_{i=1}^{m} \delta_i \geq \delta b/4$.*

**Proof.** Construct a tree of initial sequences $X_i$, as in Lemma 4.1, and let $L(X_i)$ denote the number of leaves underneath $X_i$. We prove our lemma by induction on the statement: there exists an initial sequence $X_i$ and values $\{\delta_j\}_{j=1}^{i}$ so that for all $j \leq i$, the $j$th block is $\delta_j$-good, and

$$L(X_i) \geq 2^{(\delta b - 2(\sum_{j=1}^{i}(\delta_j + \delta/4)))l}.$$

This will prove the lemma because $L(X_b) = 1$ so

$$\delta b - 2(\sum_{j=1}^{b}(\delta_j + \delta/4)) \leq 0,$$

and therefore $\sum_{j=1}^{b} \delta_j \geq \delta b/4$.

The base case of the induction is clear. For the inductive step, assume it holds for a given $i$, but not for $i+1$. Let $\alpha = \delta b - 2(\sum_{j=1}^{i}(\delta_j + \delta/4))$. Then all strings $x_{i+1}$ must correspond to initial sequences $X_{i+1}$ with $L(X_{i+1}) < 2^{(\alpha - \delta/2)l}$; all $\delta/4$-good strings $x_{i+1}$ correspond to initial sequences $X_{i+1}$ with $L(X_{i+1}) < 2^{(\alpha-\delta)l}$; in general, all $2^t\delta$-good strings correspond to initial sequences with $L(X_{i+1}) < 2^{(\alpha - 2^{t+1}\delta - \delta/2)l}$. Thus, the maximum number of leaves we can have is at most

$$\sum_{t=-2}^{k+1} (\text{\# of } 2^t\delta\text{-bad strings})(\text{maximum } L(X_{i+1}) \text{ for } 2^t\delta\text{-bad string})$$

$$\leq \sum_{t=-2}^{k+1} 2^{2^t\delta l} 2^{(\alpha - 2^t\delta - \delta/2)l} = (k+4)2^{(\alpha-\delta/2)l} < 2^{\alpha l},$$

contradicting our inductive assumption.

□

Thus, we view our $2s$ random bits as a hash function $h$ mapping $s$ bits to $s$ bits (we will always map to fewer than $s$ bits, but we ensure there are enough bits in the description of $h$). We divide our $8r/\delta$-bit string from the $\delta$-source into $8r/\delta s$ blocks of $s$ bits each. For each sequence $\{\delta_i\}$, $\sum \delta_i \geq 2r$, $\delta_i \in \{0, \delta/4, \delta/2, \ldots, 2^k\delta\}$, we assume the $i$th block $x_i$ is $\delta_i$-good. Lemma 7.8 implies that for one such choice of the $\delta_i$ the assumption will be true. We then use $h$ to hash $x_i$ to $\delta_i s/2$ bits. The Leftover Hash Lemma, along with Lemma 2.1, inductively implies that the sequence $h(x_1) \circ h(x_2) \circ \ldots \circ h(x_{8r/\delta s})$ is quasi-random within

$\sum 2^{-\delta_i s/4} << 1$. Thus, with non-zero probability this string lies in any witness set of size $n/2$, i.e. it does so for at least one choice of $h$ and $\delta_i$-good $x_i$.

The total number of possibilities we have to try is

$$(\#\text{ possible } h\text{'s})(\#\text{ possible sequences } \{\delta_i\}) = 2^{O(s)}(k+4)^{O(r/\delta s)} = 2^{O(s)},$$

as required.

□

We can now prove our main result:

**Theorem 7.7** *There is an explicit construction for an* $(n, m = n^{\sqrt{\log\log n}/\log\log\log n}, n, O(1)$*-rainbow, and hence implicit* $O(1)$*-probe search can be done for this value of* $m$.

**Proof.** Using Lemma 7.6, build a $(c = 2^{O(\sqrt{\log m \log\log\frac{1}{\delta}})}, m, n = m^{\delta}, O(1))$-rainbow, for $\delta = \log\log\log n/\sqrt{\log\log n}$. Note that Lemma 7.6 allows for a larger value of $c$, but we can always use a smaller one. Using Lemma 7.2 to show that we can ignore the big $O$'s in the exponents, and Lemma 7.3 to combine this rainbow with the disperser of Lemma 7.7 yields the theorem. □

# Bibliography

[AB+] M. Ajtai, L. Babai, P. Hajnal, J. Komlos, P. Pudlak, V. Rodl, E. Szemeredi, and G. Turan, "Two Lower Bounds for Branching Programs," 18th STOC, 1986, pp. 30-38.

[Blu] M. Blum, "Independent Unbiased Coin Flips from a Correlated Biased Source: a Finite Markov Chain," *Combinatorica*, 6 (2): 97-108, 1986.

[BBR] C.H. Bennett, G. Brassard, and J.-M. Robert, "Privacy Amplification by Public Discussion," *SIAM Journal on Computing*, 17 (2): 210-229, 1988.

[BL] M. Ben-Or and N. Linial, "Collective Coin Flipping Robust Voting Schemes and Minimal Banzhaf Values," 26th FOCS, 1985, pp. 408-416.

[CWe] L. Carter and M. Wegman, "Universal Hash Functions," *J. Comp. and Syst. Sci.*, 18(2): 143-154, 1979.

[CG1] B. Chor and O. Goldreich, "On the Power of Two-Point Based Sampling," *Journal of Complexity*, 5:96-106,1989.

[CG2] B. Chor and O. Goldreich, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity," *SIAM J. Comput.*, 17(2):230-261, 1988.

[CG+] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolenski, "The Bit Extraction Problem or t-Resilient Functions," 26th FOCS, 1985, pp. 396-407.

[CWi] A. Cohen and A. Wigderson, "Dispersers, Deterministic Amplification, and Weak Random Sources," 30th FOCS, 1989, pp. 14-19.

[DFK] M. Dyer, A. Frieze, and R. Kannan, " A Random Polynomial Time Algorithm for Approximating the Volume of Convex Bodies," 21st STOC, 1989, pp. 375-381.

[Eli] P. Elias, "The Efficient Construction of an Unbiased Random Sequence," *Ann. Math. Stat.*, 43(3):865-870, 1972.

[FG+] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, M. Szegedy, "Approximating Clique is Almost NP-Complete." 32nd FOCS, 1991.

[FN] A. Fiat and M. Naor, "Implicit $O(1)$ Probe Search," 21st STOC, 1989, pp. 336-344.

[FNSS] A. Fiat, M. Naor., J.P. Schmidt, and A. Siegel, "Non-Oblivious Hashing," 20th STOC, 1988, pp. 367-376.

[FRS] L. Fortnow, J. Rompel, and M. Sipser, "On the Power of Multi-Prover Interactive Protocols," *Proc. 3rd Structure and Complexity Theory Conf.*, 1988, pp. 156-161.

[FKS] M.L. Fredman, J. Komlos, and E. Szemeredi, "Storing a Sparse Table with $O(1)$ Worst Case Access Time," *Journal of the Association for Computing Machinery*, 31: 538-544, 1984.

[Hua] H. Hua, *Introduction to Number Theory*, Springer-Verlag, 1982.

[ILL] R. Impagliazzo, L. Levin, and M. Luby, "Pseudo-Random Generation from One-Way Functions," 21st STOC, 1989, pp. 12-24.

[IZ] R. Impagliazzo and D. Zuckerman, "How to Recycle Random Bits," 30th FOCS, 1989, pp. 248-253.

[Kar] R.M. Karp, "Reducibility Among Combinatorial Problems. In R.E. Miller and J.W. Thatcher, eds., *Complexity of Computer Computations*, 1972, pp. 85-103.

[Len] H.W. Lenstra, personal communication.

[LLS] D. Lichtenstein, N. Linial, and M. Saks, "Imperfect Sources of Randomness and Discrete Controlled Processes," 19th STOC, 1987, pp. 169-177.

[LPS] A. Lubotzky, R. Philips, P. Sarnak, "Ramanujan Graphs," *Combinatorica*, 8(3): 261-277, 1988.

[MNT] Y. Mansour, N. Nisan, and P. Tiwari, "The Computational Complexity of Universal Hashing," 22nd STOC, 1990, pp. 235-243.

[Mor] M. Morgenstern, "Existence and Explicit Construction of $q+1$ Regular Ramanujan Graphs for Every Prime Power $q$," manuscript.

[Nis] N. Nisan, "Pseudorandom Generators for Space-Bounded Computation," 22nd STOC, 1990, pp. 204-212.

[Rab] M. O. Rabin, "Probabilistic Algorithm for Testing Primality," *Journal of Number Theory*, 12: 128-138, 1980.

[SS] A. Sahay and M. Sudan, personal communication.

[San] M. Santha, "On Using Deterministic Functions in Probabilistic Algorithms," *Information and Computation*, 74(3): 241-249, 1987.

[SV] M. Santha and U. Vazirani, "Generating Quasi-Random Sequences from Slightly Random Sources," 25th FOCS, 1984, pp. 434-440.

[Sho] V. Shoup, "New Algorithms for Finding Irreducible Polynomials over Finite Fields," 29th FOCS, 1988, pp. 283-290.

[Sip] M. Sipser, "Expanders, Randomness, or Time versus Space," *Journal of Computer and System Sciences*, 36: 379-383, 1988.

[Va1] U. Vazirani, "Efficiency Considerations in Using Slightly-Random Sources," 19th STOC, 1987.

[Va2] U. Vazirani, "Randomness, Adversaries and Computation," PhD Thesis, University of California, Berkeley, 1986.

[Va3] U. Vazirani, "Strong Communication Complexity or Generating Quasi-Random Sequences from Two Communicating Slightly-Random Sources," *Combinatorica*, 7 (4): 375-392, 1987.

[VV] U. Vazirani and V. Vazirani, "Random Polynomial Time is Equal to Slightly-Random Polynomial Time," 26th FOCS, 1985, pp. 417-428.

[vN] J. von Neumann, "Various Techniques Used in Connection with Random Digits," Notes by G.E. Forsythe, National Bureau of Standards, *Applied Math Series*, 12:36-38, 1951. Reprinted in *Von Neumann's Collected Works*, 5:768-770, 1963.

[Zu1] D. Zuckerman, "General Weak Random Sources," 31st FOCS, 1990, pp. 534-543.

[Zu2] D. Zuckerman, "Simulating BPP Using a General Weak Random Source," 32nd FOCS, 1991, to appear.