Simulating BPP Using a General Weak Random Source*

David Zuckerman†
Dept. of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
diz@cs.utexas.edu

February 21, 1995

Abstract

We show how to simulate BPP and approximation algorithms in polynomial time using the output from a δ-source. A δ-source is a weak random source that is asked only once for R bits, and must output an R-bit string according to some distribution that places probability no more than $2^{-\delta R}$ on any particular string. We also give an application to the unapproximability of MAX CLIQUE.

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.

*This paper appeared in preliminary form in the *32nd Annual Symposium on Foundations of Computer Science*, 1991, pp. 79-89.

†Most of this research was done while the author was at U.C. Berkeley, and supported by an AT&T Graduate Fellowship, NSF PYI Grant No. CCR-8896202, and NSF Grant No. IRI-8902813. Part of this research was done while the author was at MIT, supported by an NSF Postdoctoral Fellowship, NSF Grant No. 92-12184 CCR, and DARPA Grant No. N00014-92-J-1799. Part of this research was done at UT Austin, where the author was supported by NSF NYI Grant No. CCR-9457799.

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:

Definition 1 *[SV] A semi-random source with parameter δ outputs bits $X_1X_2\ldots X_R$, such that for all $i \leq R$ and for all $x_1, \ldots, x_i \in \{0,1\}$,*

$$\delta \leq Pr[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 1 - \delta.$$

They proved that it is impossible to extract even a single almost-random bit from one such source (so Vazirani [Va2, Va3] showed how to extract almost-random bits from two independent sources).

In light of this result, one might give up hope for simulating randomized algorithms with one semi-random source. Nevertheless, [VV] and [Va1] showed how to simulate RP and BPP with one semi-random source.

Chor and Goldreich [CG1] generalized this model by assuming no sequence of l bits has too high a probability of being output. More precisely, [1]

Definition 2 *[CG1] An (l, δ) PRB-source outputs R bits as R/l blocks $Y_1, \ldots, Y_{R/l}$, each of length l, such that for all l-bit strings $y_1, \ldots, y_{R/l}$,*

$$Pr[Y_i = y_i | Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}] \leq 2^{-\delta l}.$$

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.

A semi-random source corresponds to $l = 1$. For $l = O(\log R)$, Chor and Goldreich showed how to simulate BPP using one such source.

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], [Va1], and [CG1] 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 be to 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 δR, for some constant δ. If we did this, however, the

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

source could output a useless string with constant probability, and we could never achieve a small error probability for our randomized algorithms.

Instead, following [Zu1], we propose upper bounding the probability that any particular string is output:

Definition 3 *For any number R of random bits requested, a δ-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 δ-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]: an algorithm that simulates RP or BPP using a δ-source also outputs the correct answer with constant probability using R bits from any source whose entropy is $\Omega(R)$, which as remarked earlier is best possible. As Cohen and Wigderson point out, the largest upper bound one could impose in the definition above and still possibly get polynomial-time (abstract) RP or BPP simulations is 2^{-R^ϵ} for an arbitrary constant $\epsilon > 0$.

The first results about these sources were obtained by Cohen and Wigderson [CWi], who showed how to simulate RP using a string from a δ-source for $\delta > 1/2$, and BPP for $\delta > 3/4$. Yet breaking the $\delta = 1/2$ barrier appeared difficult, because it involves doing more than can be done by bounding the second eigenvalue of ordinary graphs (see [WZ]).

Friedman and Wigderson [FW] showed how to break the barrier and improve this to all $\delta > 0$ in the RP case if one could construct hypergraphs with small second eigenvalue; yet, constructing such hypergraphs appears difficult. Then, using different techniques, the author showed how to simulate RP using a δ-source for all $\delta > 0$ in time $n^{O(\log n)}$, or in polynomial time under the Generalized Paley Graph Conjecture [Zu1]. Actually, Sahay and Sudan [SS] pointed out a mistake in that paper, which we correct in this paper.

In our main results, we show how to simulate RP and BPP in polynomial time using a δ-source without any unproven assumptions. These algorithms also yield solutions to more general problems. Our RP simulation implies that we can find an n-bit prime using a δ-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 include separate algorithms for RP and BPP, because the RP algorithm has the advantage of using few random bits from the δ-source. Namely, if r truly random bits are used by an RP algorithm to achieve probability of success $1/2$, then $O(r \log r)$ bits from a δ-source are used by our RP simulation. Hence this is also a quasi-perfect pseudo-random generator (see [San], [Sip]). Our BPP algorithm requires $r^{O((\log^2 \delta^{-1})/\delta)}$ bits.

Subsequent to this work, Noam Nisan and the author have extended and simplified the construction, building an "extractor" [NZ], although the construction there does not imply our result. However, the ideas there do help simplify our original construction, so we present the simplified construction here.

[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.

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]). It is easy to show that random graphs are dispersers, yet one cannot even use the eigenvalue techniques in [Tan] and [AM] to give a certificate that a random graph is a disperser (see [WZ] for more discussion on this point). This makes it especially interesting that our explicit construction works in time just polylogarithmic in the size of the graph.

It should not be surprising that our disperser constructions are useful in other areas. Here we give an application to showing the difficulty of approximating the size of the maximum clique. In the preliminary version [Zu2], we also gave an application to the problem of implicit $O(1)$ probe search. This is not included here because the author can use new dispersers to simplify and strengthen these results. The interested reader should see upcoming work by the author [Zu4].

Computing $\omega = \omega(G)$, the size of the maximum clique of the input graph G, is well known to be NP-complete [Kar]. Little was known about the difficulty of approximating ω until Feige, et.al. [FG+] showed that if approximating ω to within a factor of $2^{(\log n)^{1-\epsilon}}$ for some $\epsilon > 0$ is in $\tilde{P}$, then $N\tilde{P} = \tilde{P}$ ($\tilde{P}$ denotes quasi-polynomial time, i.e. $TIME(2^{polylog(n)})$). After the preliminary version of this paper appeared, the above hardness results have been improved: if approximating ω to within a factor of $n^{1/4-o(1)}$ is in $\tilde{P}$, then $N\tilde{P} = coR\tilde{P}$ [BS, AL+, AS].

Since it is infeasible to find close approximations, one can ask whether it is feasible to at least estimate the order of magnitude of ω. More precisely, is there an algorithm that for some constant t outputs a number between $\omega^{1/t}$ and ω^t? This is equivalent to approximating $\log \omega$ to within a constant factor. By applying our disperser construction to the proof of [FG+], we show that the existence of such an algorithm implies $N\tilde{P} = \tilde{P}$. The author has recently shown that the existence of an efficient algorithm to approximate any iterated logarithm of ω to within a constant factor implies that NP is recognized by slightly-superpolynomial randomized machines (see [Zu3] for the precise statement).

The extension of our results in [NZ] have had applications to showing that a randomized SPACE(S) machine using $poly(S)$ random bits can be simulated deterministically in SPACE(S) [NZ], and to constructing expander graphs that beat the eigenvalue bound [WZ]. The lemma about expander graphs used in the RP construction has been used to explicitly construct a hitting set for high-dimensional rectangles [LLSZ].

Our results have recently been extended by Srinivasan and the author [SZ] to the case of subconstant δ. In particular, they considered δ-sources outputting R bits such that any string has probability at most 2^{-R^ϵ}. For any fixed $\epsilon > 0$, they gave an $n^{O(\log n)}$ simulation of RP using the output from such a source. If $\epsilon > 1 - 1/(k+1)$ for a positive integer k, they gave $n^{O(\log^{(k)} n)}$ time simulations of BPP ($\log^{(k)}$ is the logarithm iterated k times). Even more recently, the above results for RP have been improved by Saks, Srinivasan, and Zhou [SSZ]. For any fixed $\epsilon > 0$, they give a polynomial-time simulation of RP.

Our construction has two basic parts: a simulation using a (particular) block-wise δ-source (given in Section 3), and a reduction to simulating from such a block-wise δ-source (given in Section 4 for BPP and Section 5 for RP). We give preliminary definitions and observations about δ-sources in Section 2, and the application to the unapproximability of MAX CLIQUE in Section 6.

2 Preliminaries

Throughout this paper, we use the convention that capital letters denote random variables, sets, distributions, and probability spaces; other variables will be in small letters. We often use a

correspondence where the small letter denotes an instantiation of the capital letter, e.g. $\vec{x}$ might be a particular input and $\vec{X}$ the random variable being uniformly distributed over all inputs.

For ease of reading, we also ignore round-off errors, assuming when needed that a number is an integer. It is not hard to see that these assumptions do not affect the validity of our arguments.

All logarithms are meant to the base 2.

2.1 Basic Definitions

Definition 4 *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*

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

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

Definition 5 *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*

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

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

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 (1), (2), and (3) to $1 - 2^{-poly(|a|)}$, $1 - 2^{-poly(|a|)}$, and $2^{-poly(|a|)}$, respectively.

Distance between Distributions

Let D_1 and D_2 be two distributions on the same space X. The variation distance between them is

$$\|D_1 - D_2\| = \max_{Y \subseteq X} |D_1(Y) - D_2(Y)| = \frac{1}{2} \sum_{x \in X} |D_1(x) - D_2(x)|.$$

A distribution D on X is called ϵ-quasi-random (on X) if the distance between D and the uniform distribution on X is at most ϵ.

A convenient fact to remember is that distance between distributions cannot be created out of nowhere. In particular if $f : X \to Y$ is any function and D_1, D_2 are distributions on X then $\|f(D_1) - f(D_2)\| \leq \|D_1 - D_2\|$. Also if E_1 and E_2 are distributions on Y then $\|D_1 \times E_1 - D_2 \times E_2\| \leq \|D_1 - D_2\| + \|E_1 - E_2\|$.

δ-sources

A distribution D on $\{0,1\}^n$ is called a δ-source if for all $x \in \{0,1\}^n$, $D(x) \leq 2^{-\delta n}$.

D is called a δ-source to within ϵ if there exists a δ-source D' such that $\|D - D'\| \leq \epsilon$.

A distribution D on the space $\{0,1\}^{l_1} \times \{0,1\}^{l_2} \times \cdots \times \{0,1\}^{l_k}$ is called a block-wise δ-source if for $1 \leq i \leq k$, and for all values $x_1 \in \{0,1\}^{l_1}, \ldots, x_i \in \{0,1\}^{l_i}$, we have that

$$Pr[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 2^{-\delta l_i},$$

where the vector of random variables $X_1 \ldots X_k$ is chosen according to distribution D. A block-wise δ-source is the same as the PRB-source of [CG1] except that here the block length is allowed to vary.

2.2 Simulations using δ-sources

The first way one would try to simulate randomized algorithms using a δ-source would be to do what von Neumann did with his sources: convert the "bad" random bits output by the δ-source into "good" random bits. This has been shown impossible for weaker models, such as the semi-random sources of [SV], but in our case the proof is particularly simple:

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

Proof. Suppose we had a deterministic 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 claim about f extracting a non-trivial bit. □

Thus we must resort to other methods. 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 δ-source we would need a different algorithm for each language in RP. Instead, we exhibit one simulation that works for all $W \subseteq \{0,1\}^r$ 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 6 *An algorithm simulates RP using a δ-source if it takes as input $R = poly(r)$ bits from the δ-source and outputs a polynomial number of r-bit strings z_i, such that for all $W \subset \{0,1\}^r, |W| \geq 2^{r-1}$, $Pr[(\exists i) z_i \in W] = 1 - 2^{-\Omega(R)}$.*

Our first theorem is a polynomial-time simulation of RP using a δ-source:

Theorem 1 *There is an algorithm that, given any $\delta > 0$, takes time $r^{O((\log \delta^{-1})/\delta^2)}$ to simulate RP using $R = O((r \log r)/\delta^2)$ bits from a δ-source.*

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

Lemma 2 *If there is a polynomial-time algorithm A simulating RP using a δ-source, then there is a polynomial-time algorithm that uses a δ-source and outputs an n-bit prime with probability $1 - 2^{-\Omega(n)}$.*

Proof. The difficulty will be that we need random bits both to pick the prime and to run the primality test. First we concentrate on picking the prime.

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 $1/2$. We represent such an n-tuple using $n(n-2)$ bits (the first and last bits of an odd n-bit number are always 1.) Let $R = n(n-2)$, and feed A an R-bit string from a δ-source such that A outputs polynomially many $n(n-2)$-bit strings z_i. Then the probability that at least one n-tuple among the z_i contains a prime is $1 - 2^{-\Omega(R)}$.

In order to run the primality tests we use the same strings z_i. 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. We claim that with high probability, each composite n-bit number will have some z_i as a witness to its compositeness. This is because

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

Thus it suffices to use the strings z_j as random bits when testing the n-bit numbers in z_i for primality. □

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 z_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 since we do so it makes it simpler to define it that way.

Definition 7 *A polynomial-time algorithm simulates BPP using a δ-source if it takes as input $R = poly(r)$ bits from the δ-source and outputs a polynomial number of r-bit strings z_i, such that for all $W \subset \{0,1\}^r, |W| \geq \frac{2}{3}2^r$, $Pr[$majority of z_i's lie in $W] = 1 - 2^{-\Omega(R)}$.*

We can now state our main theorem:

Theorem 2 *There is an algorithm that, given any $\delta > 0$, takes time $r^{O((\log^2 \delta^{-1})/\delta)}$ to simulate BPP using $R = r^{O((\log^2 \delta^{-1})/\delta)}$ bits from a δ-source.*

Note that such an algorithm 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 good approximation can be obtained with probability $1 - 2^{-\Omega(R)}$.

2.3 δ-sources and Dispersers

To give some intuition about δ-sources, we follow [CG1] and define flat δ-sources and show that we may assume without loss of generality that our δ-source is flat. We never need this explicitly, but it is useful to keep in mind.

Definition 8 *A flat δ-source is a source which places probability $2^{-\delta R}$ on $2^{\delta R}$ R-bit strings.*

Lemma 3 *Suppose an algorithm simulates RP (BPP) using a flat δ-source. Then it also simulates RP (BPP) using a δ-source.*

Proof. The proof in our case is much simpler than in [CG1]. Fix an algorithm A. On certain input strings, A outputs the correct answer, and on others it doesn't. The δ-source may as well place as much probability as possible on the strings for which A is incorrect, i.e. the δ-source may as well be a flat δ-source. □

Remark: This observation that some strings are good for A and others aren't implies that the error probability $1-2^{-\Omega(R)}$ in Definitions 6 and 7 are not important: we can make the error exponentially small simply by using a slightly higher quality source. More precisely, define simulations for RP (BPP) "with non-zero success probability" the same as simulations for RP (BPP) but with the success probability $1 - 2^{-\Omega(R)}$ replaced by an arbitrarily small but positive probability. Then an

algorithm simulates RP (BPP) using a δ-source if and only if it simulates RP (BPP) with non-zero probability from a δ'-source, for some $\delta' < \delta$. This is because if an algorithm simulates RP (BPP) with non-zero probability from a δ'-source, then there are at most $2^{\delta' R} - 1$ strings giving the wrong answer. Thus, the probability of error when the output is from a δ-source is at most $(2^{\delta' R} - 1)/2^{\delta R}$. The other direction is similar.

Simulations using a δ-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, it is useful to understand the equivalence, as it was used in e.g. [WZ]. Here we rephrase our results in terms of dispersers.

First, we define dispersers:

Definition 9 *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.*

Definition 10 *An (m, n, d, a, b)-majority 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 for any subset S of less than b nodes on the right, the number of vertices on the left side having a majority of neighbors in S is less than a.*

Note that a majority disperser is a disperser.

Definition 11 *An (m, n, d, a, b)-disperser or majority disperser is* efficiently constructible *if, for a given node on the left, its neighbors can be determined in time polynomial in d.*

Lemma 4 (CWi) *RP (BPP) can be simulated using $R = poly(r)$ bits from a δ'-source for all $\delta' > \delta$ iff a $(2^R, 2^r, poly(r), 2^{\delta R}, 2^{r-1})$-disperser ($(2^R, 2^r, poly(r), 2^{\delta R}, 2^{r-2})$-majority disperser) is efficiently constructible.*

Hence we can restate our results:

Theorem 3 *For any $\delta > 0$, a $(2^R, 2^r, r^{O((\log \delta^{-1})/\delta^2)}, 2^{\delta R}, 2^{r-1})$-disperser is efficiently constructible, where $R = O((r \log r)/\delta^2)$.*

Theorem 4 *For any $\delta > 0$, a $(2^R, 2^r, r^{O((\log^2 \delta^{-1})/\delta)} r^4 2^{O((\log^6 \delta^{-1})/\delta^3)}, 2^{\delta R}, 2^{r-2})$-majority disperser is efficiently constructible, where $R = r^{O((\log^2 \delta^{-1})/\delta)}$.*

2.4 Extractors

Our BPP simulations are even stronger than stated so far. We actually construct an extractor [NZ]. To understand an extractor, recall that we showed earlier that it is impossible to extract a stream of quasi-random bits from a δ-source. An extractor enables us to extract many quasi-random bits, if we add a small number of truly random bits (think of $t \ll m < n$ in the following definition):

Definition 12 *[NZ] $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is called a (δ, ϵ)-extractor if for any δ-source D on $\{0,1\}^n$, the distribution of $E(x, y) \circ y$ induced by choosing x according to D and y uniformly in $\{0,1\}^t$ is quasi-random (on $\{0,1\}^m \times \{0,1\}^t$) to within ϵ.*

It is not difficult to see how an extractor construction yields a BPP simulation:

Lemma 5 *Suppose that for all $\delta > 0$ there is a polynomial-time (δ, ϵ)-extractor, with $\epsilon < 1/3$, $t = O(\log n)$, and $m = n^{\Omega(1)}$. Then for all $\delta > 0$ there is a polynomial-time simulation of BPP using a δ-source.*

Proof. Let $\delta > 0$ be given. By using independent random strings, we may assume without loss of generality that the "witness" set $W \subseteq \{0,1\}^r$ has size $|W| = (1-\eta)2^r$ for some $\eta \leq 1/15$. Choose n a large enough polynomial in r so that $m \geq r$. The simulator simply requests $R = n$ bits from the source S, and outputs the first r bits of $E(x, y_i)$ for all t-bit strings y_i, where x is the string from S. This amounts to $2^t = poly(n)$ strings z_i. For a random x output by S, the expected fraction of z_i that lie outside W is at most $\eta + \epsilon \leq 2/5$. By Markov's inequality, the probability that this fraction exceeds .45 is at most $.4/.45 < 1$. Thus the simulator produces the correct answer with non-zero probability. By the remark in Subsection 2.3, this yields a simulator for BPP using a δ'-source for any $\delta' > \delta$. □

Therefore, instead of directly building a simulator for BPP, we build an extractor. (Our original construction was not an extractor, but using the simplifications in [NZ] we convert our construction to an extractor.) The following theorem immediately implies Theorem 2.

Theorem 5 *For any parameters $\delta = \delta(n)$ and $\epsilon = \epsilon(n)$ with $1/n \leq \delta \leq 1/2$, there exists an easily computable (and explicitly given) (δ, ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$, where $t = 4\log n + O((\log^6 \delta^{-1})/(\delta^3 \epsilon^2))$ and $m = n^{\Omega(\delta/\log^2 \delta^{-1})}$.*

Note that for constant δ and ϵ, our construction uses fewer truly random bits than the construction in [NZ], although it outputs fewer quasi-random bits:

Theorem 6 *[NZ] For any parameters $\delta = \delta(n)$ and $\epsilon = \epsilon(n)$ with $1/n \leq \delta \leq 1/2$ and $2^{-\delta n} \leq \epsilon \leq 1/n$, there exists an easily computable (and explicitly given) (δ, ϵ)-extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$, where $t = O((\log \epsilon^{-1})(\log^2 n)/\delta^2)$ and $m = \Omega(\delta^2 n/\log \delta^{-1})$.*

3 Simulating BPP Using a Block-Wise δ-source

3.1 A Hashing Lemma

One of our key tools is 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 define

Definition 13 (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 well known that there are universal families of hash functions from $\{0,1\}^m$ to $\{0,1\}^n$ of size 2^{2m} (in fact, of size 2^{m+n}, but we don't need this better bound).

The Leftover Hash Lemma shows that hash functions can be used as extractors, except that instead of adding a small number of truly random bits, we are adding a large number. The Leftover Hash Lemma was originally stated in terms of flat δ-sources, but it is a simple corollary of the proof that it holds for general δ-sources.

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

Corollary 1 *The conclusion above holds if x is chosen from any δ-source, $\delta = u/n$.*

Although the extractor requires far fewer auxiliary bits than the Leftover Hash Lemma, the Leftover Hash Lemma has some advantages over the extractor. The Leftover Hash Lemma gives a smaller error term, and extracts a larger number of random bits.

3.2 Simulating BPP

The most natural way to extract bits from a block-wise δ-source using the Leftover Hash Lemma requires many blocks from the block-wise δ-source. This method is described by the following lemma, which essentially strengthens related lemmas in [Va1] and [CG1]. This lemma is not necessary for the BPP simulation, but is helpful to better appreciate Lemma 7. Intuitively, because the Leftover Hash Lemma says that the ordered pair $(h, h(x))$ is close to uniform, we may use the same hash function repeatedly.

Lemma 6 *Let H be a universal family of hash functions mapping l bits to $m = \delta l - 2e$ bits, and let D be a block-wise δ-source on $(\{0,1\}^l)^k$. If $\vec{X} = X_1 \ldots X_k$ is chosen according to D and h is chosen uniformly at random from H, then the distribution of $(h, h(X_1), \ldots, h(X_k))$ is quasi-random to within $k2^{-e}$.*

Proof. Following the proof of Lemma 7, due to [NZ], it is easiest to prove this by working backwards. Namely, we proceed by induction from $i = k$ to $i = 1$ on the statement: for any sequence of values $x_1 \ldots x_i$, the distribution of $(h, h(X_{i+1}), \ldots, h(X_k))$ conditioned on $X_1 = x_1, \ldots, X_i = x_i$, is quasi-random to within $(k-i)2^{-e}$.

This is obvious for $i = k$. Suppose it is true for $i+1$. Fix the conditioning $X_1 = x_1, \ldots, X_i = x_i$, and let D_{i+1} denote the induced distribution on X_{i+1}. Since, by the induction hypothesis, for every x_{i+1}, the induced distribution on $(h, h(X_{i+2}), \ldots, h(X_k))$ is quasi-random to within $(k-i-1)2^{-e}$, we have that the distribution D' of $(X_{i+1}, h, h(X_{i+2}), \ldots, h(X_k))$ is within $(k-i-1)2^{-e}$ of the distribution $D'' = D_{i+1} \times U_{i+1}$, where U_{i+1} is the uniform distribution on $H \times \{0,1\}^{(k-i-1)m}$.

We now apply a deterministic function f to distributions D' and D'', namely using the second argument as a hash function to hash the first argument and then switching the order of the first two arguments. Thus, the distribution $f(D')$ of $(h, h(X_{i+1}), \ldots, h(X_k))$ is within $(k-i-1)2^{-e}$ of the distribution $f(D'')$ of $(h, h(X_{i+1}), Y_{i+2}, Y_{i+3}, \ldots, Y_k)$ obtained by picking X_{i+1} according to D_{i+1}, and $(h, Y_{i+2}, \ldots, Y_k)$ independently and uniformly at random in $H \times (\{0,1\}^m)^{k-i-1}$. Using Corollary 1 $f(D'')$ is quasi-random to within 2^{-e}, since we only added perfectly random bits at the end. Thus, by the triangle inequality, $f(D')$ is quasi-random to within $(k-i)2^{-e}$. This completes the induction and the proof of the lemma. □

Unfortunately, for the reductions from a general δ-source to a block-wise δ-source, we need to use a block-wise δ-source with fewer blocks. The following algorithm, taken from [NZ], allows us to do this. The idea is to view the initial $2l$ truly random bits as a hash function h mapping l bits to $\delta l/2$ bits, and to have the first block X_1 from the block-wise δ-source be l bits long. By the Leftover Hash Lemma, the ordered pair $(h, h(X_1))$ is almost random. At this point, we have $2l + \delta l/2$ almost-random bits, so we may view this as a new hash function which hashes $l + \delta l/4$ bits to $(\delta/2)(l + \delta l/4)$ bits. We therefore have the second block from the block-wise δ-source be

$l + \delta l/4$ bits. Continuing in this manner, we get geometrically increasing blocks, so the algorithm needs only $O((\log n)/\delta)$ blocks.

The problem is that this doesn't quite work. The Leftover Hash Lemma fails if the δ-source can depend on the hash function. That is the situation here: the second block depends on the first block, but the first block helps define the hash function. Surprisingly, the algorithm works if the blocks are output in reverse order, with geometrically decreasing lengths. The intuition for this is that if an adversary could choose an arbitrary δ-source after seeing the hash function, we have no control. On the other hand, if an adversary could choose an arbitrary ϵ-quasi-random distribution for a hash function after seeing the output from a δ-source, the adversary can affect the error only by ϵ.

Function C:

The function has 3 parameters: δ, the quality of the source; l_1, l_s, the largest and smallest block sizes.

1. INPUT: $x_1 \in \{0,1\}^{l_1} \ldots x_s \in \{0,1\}^{l_s}$; $y \in \{0,1\}^{2l_s}$. Here $l_{i-1}/l_i = (1 + \delta/4)$ for $1 < i \leq s$.
2. We assume for each i a fixed universal family of hash functions $H_i = \{h : \{0,1\}^{l_i} \rightarrow \{0,1\}^{\delta l_i/2}\}$. Each function in H_i is described by $2l_i$ bits.
3. $h_s \leftarrow y$
4. For $i = s$ downto 1 do $h_{i-1} \leftarrow h_i \circ h_i(x_i)$
5. OUTPUT (a vector in $\{0,1\}^m$): h_0, excluding the bits of h_s.

Lemma 7 *[NZ] Let D be a block-wise δ-source on $\{0,1\}^{l_1} \times \ldots \{0,1\}^{l_s}$. If $\vec{X} = X_1 \ldots X_l$ is chosen according to D and Y is chosen uniformly at random in $\{0,1\}^{2l_s}$, then the distribution of $C(X,Y) \circ Y$ is quasi-random to within $2 \cdot 2^{-\delta l_s/4}$.*

Proof. As in Lemma 6, we will prove by induction from $i = s$ down to $i = 0$ the following claim, which clearly implies the lemma.

Claim: For any sequence of values $x_1 \ldots x_i$, the distribution of h_i conditioned on $X_1 = x_1, \ldots, X_i = x_i$, is quasi-random to within ϵ_i, where $\epsilon_i = \sum_{j=i+1}^{s} 2^{-\delta l_j/4}$.

This claim is clearly true for $i = s$. Now suppose it is true for $i + 1$. Fix the conditioning $X_1 = x_1, \ldots, X_i = x_i$, and let D_{i+1} denote the induced distribution on X_{i+1}. Since, by the induction hypothesis, for every x_{i+1}, the induced distribution on h_{i+1} is quasi-random, we have that the distribution (X_{i+1}, h_{i+1}) is within ϵ_{i+1} of the distribution $D_{i+1} \times U_{i+1}$, where U_{i+1} is the uniform distribution on H_{i+1}.

Thus, the distribution of h_i is within ϵ_{i+1} of the distribution D' of $(h, h(X_{i+1}))$ obtained by picking X_{i+1} according to D_{i+1}, and h_{i+1} independently and uniformly at random in H_{i+1}. Using Corollary 1 D' is quasi-random to within $2^{-\delta l_{i+1}/4}$. □

4 Reduction to Block-Wise Source with $O(\log n)$ Blocks: BPP

A δ-source is not necessarily a block-wise δ-source or close to one; all the randomness could be concentrated at the beginning, say. In order to create a distribution close to a block-wise δ-source, we need to add randomness. The idea for doing this is to take a pseudo-random permutation of a δ-source. Hopefully the randomness will spread out, and we will have a block-wise δ-source.

This is not quite true, however, as the randomness can get concentrated at the front. To illustrate this, imagine two bits which are always equal but otherwise uniformly distributed. Then whichever comes first in the permuted order will add randomness; the other will not. Elaborating on this idea, consider the 1/2-source where the first $n/2$ bits are perfectly random, and the next $n/2$ bits are equal to the first $n/2$ bits. In a pseudo-random permutation, we would expect almost all the last $\sqrt{n}$ bits to have already appeared among the first $n - \sqrt{n}$ bits, and therefore not to be very random conditional on the first $n - \sqrt{n}$ bits.

We get around this problem by using only the first $\delta n/4$ bits of the permutation. The pseudo-random generator for the permutations is based on pairwise independence, and is described below.

4.1 Extracting a block

In order to convert a δ-source into a distribution close to a block-wise δ-source, we must obtain smaller blocks which are close to δ-sources. In this section we show how to obtain one such block.

The idea to do this is as follows. Intuitively, a δ-source has many bits which are somewhat random. We wish to obtain l bits which contain a reasonable fraction of these somewhat random bits. We cannot do this deterministically, as an adversary can make any δn of these n bits somewhat random. We therefore pick the l bits at random using pairwise independence. We then use the following standard lemma:

Lemma 8 *Let $T \subseteq \{1, 2, \ldots, n\}$, $|T|/n \geq \alpha$. If the multi-set $S = \{s_1, \ldots s_l\}$ is chosen pairwise independently from $\{1, 2, \ldots, n\}$, then*

$$Pr[|S \cap T| \geq \alpha l/2] > 1 - 4/\alpha l.$$

Only $2 \log n$ random bits suffice to pick pairwise independent random variables: say there is a finite field F of size n (e.g. n is a power of 2), then if a, b are chosen uniformly at random then $s_i = ai + b$, $i = 0, \ldots, l-1$, are pairwise independent. This probability space is also nice in that if $a \neq 0$, then all the s_i are distinct. This yields the following corollary:

Corollary 2 *Lemma 8 also holds if S is viewed as a set (i.e. delete multiple elements), if the pairwise independent probability space above is used.*

Thus, we can describe our function to extract blocks:

The function B:

The function has 3 parameters: n, the size of the original input; l, the size of the output; and δ, the quality of randomness needed.

1. INPUT: $x \in \{0,1\}^n$; $y \in \{0,1\}^t$ (where $t = 2 \log n$).
2. Use y to choose a set $\{i_1 \ldots i_l\} \subset \{1 \ldots n\}$ of size l as described above.
3. OUTPUT (a vector in $\{0,1\}^l$): $x_{i_1} \ldots x_{i_l}$.

Lemma 9 *Suppose D is a δ-source on $\{0,1\}^n$ and $\vec{X}$ is chosen according to D. Then for all but an ϵ fraction of $y \in \{0,1\}^t$ the distribution of $B(\vec{X}, \vec{y})$ is within ϵ from a δ'-source, where $\delta' = c\delta / \log \delta^{-1}$ and $\epsilon = 3/\sqrt{\delta' l}$.*

We use the proof of this lemma given in [NZ], with a simplified proof of Lemma 11 due to Leonard Schulman. The intuition is best seen by considering a simple proof to a slightly weaker conclusion: for all but an ϵ fraction of the $\vec{y}$'s the distribution of $B(\vec{X}, \vec{y})$ has $\Omega(\delta l)$ entropy. The distribution on $\vec{X}$ clearly has entropy $H(\vec{X})$ of at least δn. Let q_i be the conditional entropy of X_i conditioned on $X_1 \ldots X_{i-1}$. From information theory, we know that $\sum_{i=1}^{n} q_i = H(\vec{X}) \geq \delta n$. Again from information theory we have that the entropy of the output is at least $\sum_{j=1}^{l} q_{i_j}$. All that is needed to complete the proof is that when $\{i_1 \ldots i_l\}$ are chosen pairwise independently, the above sum is, with high probability, close to its expected value δl.

The rest of this section is devoted to proving the slightly stronger conclusion, that the output is near a δ'-source. Our proof tries to retain the structure of the above proof but, since we do not have the powerful tools of information theory at our disposal, the proof is not very simple. The difficulty is perhaps best appreciated by observing that it is possible that for all $\vec{y}$, $B(\vec{X}, \vec{y})$ is not a δ'-source (for any δ'), but only statistically close to a δ'-source. To see this, consider the following $(1 - l/n)$-source: pick y' uniformly at random, use this to choose a set $\{i_1, \ldots, i_l\}$ as described earlier, set $X_{i_1} = \ldots = X_{i_l} = 0$, and choose the other X_j uniformly at random. Thus, for any $\vec{y}$, with probability $1/2^{|y|} = 1/n^2$, $B(\vec{X}, \vec{y})$ is the all 0 string.

Fix a δ-source D. We need the following definitions (which are relative to D).

Definition 14 *For a string $\vec{x} \in \{0,1\}^n$ and an index $1 \leq i \leq n$, let*

$$p_i(\vec{x}) = Pr_{\vec{X} \in D}[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}].$$

Index i is called good in $\vec{x}$ if $p_i(\vec{x}) < 1/2$ or $p_i(\vec{x}) = 1/2$ and $x_i = 0$.

The part of the definition with $p_i(\vec{x}) = 1/2$ is to ensure that exactly one of $x_i = 0$ and $x_i = 1$ is good, for a given prefix.

Definition 15 *$\vec{x}$ is α-good if there are at least αn indices which are good in x.*

Definition 16 *For $S \subseteq \{1, 2, \ldots, n\}$, $\vec{x}$ is α-good in S if there are at least $\alpha|S|$ indices in S which are good in $\vec{x}$.*

Definition 17 *S is α-informative to within β if*

$$Pr_{\vec{X} \in D}[\vec{X} \text{ is } \alpha - \text{good in } S] \geq 1 - \beta.$$

Denote by S_y the set of size l of indices chosen pairwise independently using the random bits $\vec{y}$. We will prove two lemmas which together clearly imply Lemma 9.

Lemma 10

$$Pr_{\vec{Y}}[S_Y \text{ is } \delta' - informative \text{ to within } \epsilon] \geq 1 - \epsilon.$$

Lemma 11 *Fix a set of indices $S = \{i_1 \ldots i_l\}$ that is δ'-informative to within ϵ. Then, the distribution of $X_{i_1} \ldots X_{i_l}$ induced by choosing $\vec{X}$ according to D is ϵ-near a δ'-source.*

Proof. Fix any string $x_{i_1} \ldots x_{i_l}$. Let

$$a_{t,j} = Pr[X_{i_1} = x_{i_1} \wedge \cdots \wedge X_{i_j} = x_{i_j} \wedge \text{exactly } t \text{ of the indices } i_1, \ldots, i_j \text{ are good}].$$

Claim: $\sum_t a_{t,j} 2^t \leq 1$ for all j, $0 \leq j \leq l$. This implies $\sum_{t \geq \delta' l} a_{t,l} \leq 2^{-\delta' l}$. Thus the contribution to the probability of any string $x_{i_1} \ldots x_{i_l}$ from x's which are δ'-good in S is at most $2^{-\delta' l}$. Since this accounts for $1 - \epsilon$ of the total probability, this will prove the lemma.
Proof of Claim: By induction on j. The base case of $j = 0$ is easy. Assume known for j; we prove for $j+1$. Consider any prefix r up to position $i_{j+1} - 1$ which so far agrees with $x_{i_1} \ldots x_{i_j}$. It has the form $r = w_1 x_{i_1} w_2 x_{i_2} w_3 \ldots w_j x_{i_j} w_{j+1}$ where the w's are strings.

Note that a particular prefix contributes to exactly one $a_{t,j}$; this contribution is the probability that this prefix occurs. Suppose r has t good indices among $i_1, \ldots, i_j$. If i_{j+1} is not a good index, then the contribution of $r \circ x_{i_{j+1}}$ to $a_{t,j+1}$ is at most the contribution of r to $a_{t,j}$. If i_{j+1} is a good index, then by the definition of good index, the contribution of $r \circ x_{i_{j+1}}$ to $a_{t+1,j+1}$ is at most half the contribution of r to $a_{t,j}$. In either case, $\sum_t a_{t,j+1} 2^t \leq \sum_t a_{t,j} 2^t \leq 1$.

□

4.1.1 Proof of Lemma 10

We first need the following lemma showing that most $\vec{x}$'s have many good indices.

Lemma 12

$$Pr_{\vec{X} \in D}[\vec{X} \textit{ is not } \alpha - \textit{good}] \leq 2^{-c_1 \delta n},$$

where $\alpha = c_1 \delta / \log \delta^{-1}$ *for some absolute positive constant* c_1.

Proof. Let us count the number of x's that are not α-good. There is a natural 1-1 correspondence between sequences in $\{good, bad\}^n$ and strings $\vec{x}$; namely one in which i is bad in $\vec{x}$ whenever the ith element of the sequence is "bad". Thus, the number of x's that are not α-good is at most the number of n-bit strings with less than αn "good" locations, i.e. $\sum_{i=0}^{\lceil \alpha n \rceil - 1} \binom{n}{i}$. Since D is a δ-source, the probability of each string is at most $2^{-\delta n}$, so

$$Pr_{\vec{X} \in D}[\vec{X} \textit{ is not } \alpha - \textit{good}] \leq 2^{-\delta n} \sum_{i=0}^{\lfloor \alpha n \rfloor} \binom{n}{i} \leq 2^{-c_1 \delta n}$$

for c_1 small enough. □

Proof of Lemma 10: For any fixed α-good string $\vec{x}$, we can apply Lemma 8 to the set of good indices and obtain

$$Pr_Y[\vec{x} \textit{ has } \alpha l/2 \textit{ good indices in } S_Y] > 1 - 4/\alpha l.$$

Using Lemma 12 it follows that

$$Pr_{\vec{X},Y}[\vec{X} \textit{ has } \alpha l/2 \textit{ good indices in } S_Y] > 1 - 4/\alpha l - 2^{-c_1 \delta n}.$$

Now set $\delta' = \alpha/2$ and $\epsilon = \sqrt{4/\alpha l + 2^{-c_1 \delta n}} \leq 3/\sqrt{\alpha l}$, and by Markov's inequality

$$Pr_Y[S_Y \textit{ is } \delta' - \textit{informative to within } \epsilon] \geq 1 - \epsilon.$$

□

4.2 Extracting a Block-Wise δ-source

The simplest way to use the previous section to get a distribution close to a block-wise δ-source would be to choose independently random y's and use these to extract different blocks. Indeed, this was done in [NZ]. However, this would require too many random bits.

Instead, we extract one large block, as described in the previous section. We first argue that initial segments of the block of geometrically growing size are close to δ'-sources. It then follows that the distribution of the large block is close to a block-wise δ-source, essentially because a δ-source of length l will still be a $\delta/2$-source even if the first $\delta l/2$ bits are removed. We want geometrically decreasing block sizes, however, in order to utilize Lemma 7; in the last step we show how to get the correct block sizes.

We will be interested in initial segments of length $m_i = l(4/\delta')^{i-1}$:

Lemma 13 *Suppose D is a δ-source on $\{0,1\}^n$ and $\vec{X}$ is chosen according to D. Then for all but an ϵ' fraction of $y \in \{0,1\}^t$, for all i with $m_i \leq n$ the distribution of the first m_i bits of $B(\vec{X}, \vec{y})$ is within ϵ' of a δ'-source, where $\delta' = c\delta/\log \delta^{-1}$ and $\epsilon' = 6/\sqrt{\delta' l}$.*

Proof. Applying Lemma 9 to each initial segment, we see that for all but $\epsilon_i = 3/\sqrt{\delta' m_i}$ of the $y \in \{0,1\}^t$, the first m_i bits of $B(\vec{X}, \vec{y})$ is within ϵ_i of a δ'-source. The lemma follows since $\sum \epsilon_i \leq \epsilon'$. □

Lemma 14 *Suppose D' is a distribution on $\{0,1\}^{n'}$ such that if Z is chosen according to D', then for all i with $m_i \leq n'$ the distribution of the first m_i bits of Z is a δ'-source. Then D' is within 2^{1-l} of a block-wise $\delta'/2$-source with block sizes $n_i = m_i - m_{i-1}$, with the convention that $m_0 = 0$.*

Proof. Write $Z = Z_1 \circ Z_2 \circ \ldots \circ Z_s$, where the length of Z_i is n_i.

We call the vector of values $z_1, \ldots, z_i$ *tiny* if

$$Pr[Z_1 = z_1, \ldots, Z_i = z_i] \leq 2^{-2m_i}.$$

Since there are at most 2^{m_i} possible values for $z_1 \ldots z_i$, $Pr[Z_1 \ldots Z_i \text{ is tiny}] \leq 2^{-m_i}$.

For any $z_1 \ldots z_i$ consider the distribution $D'_{z_1 \ldots z_i}$ defined to be the distribution on Z_{i+1} conditioned on $Z_1 = z_1 \ldots Z_{i-1} = z_{i-1}$. If $z_1 \ldots z_i$ is not tiny, then for all $x \in \{0,1\}^{n_{i+1}}$:

$$D'_{z_1 \ldots z_i}(x) \leq 2^{2m_i} 2^{-\delta' m_{i+1}} = 2^{-(\delta'/2)m_{i+1}} \leq 2^{-(\delta'/2)n_{i+1}},$$

and thus $D'_{z_1 \ldots z_i}$ is a $\delta'/2$-source.

Thus the distance from D' to a block-wise $\delta'/2$-source is at most

$$Pr[(\exists i) Z_1 Z_2 \ldots Z_i \text{ is tiny}] \leq \sum_{i=1}^{s-1} 2^{-m_i} < 2^{1-l}$$

. □

Corollary 3 *Suppose D is a δ-source on $\{0,1\}^n$ and $\vec{X}$ is chosen according to D. Then for all but an ϵ' fraction of $y \in \{0,1\}^t$, $B(\vec{X}, \vec{y})$ is within $\epsilon' + 2^{1-l}$ of a block-wise $\delta'/2$-source with block sizes $n_i = m_i - m_{i-1}$, where δ' and ϵ' are as above.*

Now that we have a block-wise source (although the block sizes n_i are not what we want), we can use the same random bits on each block to extract blocks of the correct size l_i, as in the following algorithm for the extractor.

Description of Extractor E:

1. INPUT: $x \in \{0,1\}^n$; $y_1, y_2 \in \{0,1\}^{t_1}$; $y_0 \in \{0,1\}^{t_2}$ (where $t_1 = 2\log n$, $t_2 = 2l_s$).
2. $z \leftarrow B(x, y_1)$. (We use B with parameters n, m_s, δ.)
3. Let $z = z_1 \circ z_2 \circ \ldots \circ z_s$, where the length of z_i is n_i.
4. For $i = 1 \ldots s$ do $w_i \leftarrow B(z_i, y_2)$. (We use B with parameters $n_i, l_i, \delta'/2$.)
5. OUTPUT (a vector in $\{0,1\}^m$): $C(w_1 \ldots w_s, y_0)$. (We use C with the parameters δ'', l_1, l_s.)

Lemma 15 *Suppose $Z = Z_1 \circ Z_2 \circ \ldots \circ Z_s$ is the output of a block-wise $\delta'/2$-source on $\{0,1\}^n$, with block lengths $|Z_i| = n_i$. Let $W_i = B(Z_i, y_2)$ with parameters as above. Then for all but an ϵ'' fraction of the y_2's, the distribution of $W = W_1 \circ W_2 \circ \ldots \circ W_s$ is within ϵ'' of a block-wise δ''-source, for $\delta'' = c(\delta'/2)/\log(2/\delta')$ and $\epsilon'' = 15(\delta'')^{-3/2} l_s^{-1/2}$.*

Proof. By Lemma 9, if Z_i is a δ'-source conditional on $Z_1, \ldots, Z_{i-1}$, then for all but $\epsilon_i'' = 3/\sqrt{\delta'' l_i}$ of the y_2's, W_i is a δ''-source conditional on $Z_1, \ldots, Z_{i-1}$, and hence also conditional on $W_1, \ldots, W_{i-1}$. Let $\sigma_i = \sum_{j=1}^{i} \epsilon_i''$. Then it follows inductively that for all but σ_i of the y_2's, $W_1 \circ \ldots \circ W_i$ is within σ_i of a block-wise δ''-source. □

Corollary 4 *Suppose D is a δ-source on $\{0,1\}^n$ and $\vec{X}$ is chosen according to D. Then for all but at most an $\epsilon/3$ fraction of y_1, y_2 pairs, the output at Step 4 is within $\epsilon/3$ of a block-wise δ''-source.*

Proof. Use Corollary 3 and Lemma 15, and note that $\delta'' \leq \delta'/2$ and $2^{1-l} \leq 2/\sqrt{\delta' l}$. Finally, recall that $\epsilon/3 = 20(\delta'')^{-3/2} l_s^{-1/2}$. □

We can now prove that our extractor construction works:

Theorem 5 For any δ-source D the distribution of $E(X,Y) \circ Y$ induced by choosing X according to D and Y uniformly in $\{0,1\}^t$ is ϵ-quasi-random on $\{0,1\}^m \times \{0,1\}^t$. Here $t = 4\log n + O((\log^6 \delta^{-1})/(\delta^3 \epsilon^2))$ and $m = n^{\Omega((\log^2 \delta^{-1})/\delta)}$.

Proof. By Corollary 4 for all but $\epsilon/3$ fraction of pairs y_1, y_2 the distribution on the w's is within $\epsilon/3$ of a block-wise δ''-source. For each such value of the y's, by Lemma 7, the output concatenated with y_0 is quasi-random within $2\epsilon/3$. Add the $\epsilon/3$ "bad" y's and the lemma follows. □

We can now conclude

Theorem 2: There is an algorithm that, given any $\delta > 0$, takes time $r^{O((\log^2 \delta^{-1})/\delta)}$ to simulate BPP using $R = r^{O((\log^2 \delta^{-1})/\delta)}$ bits from a δ-source.

For the reader's reference, we attach a summary of the way the parameters are chosen below. The reader is advised to skip this on the first reading.

Parameters:

1. The parameters n, ϵ and δ are given. We assume $1/n \leq \delta \leq 1/2$.
2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 9.
3. $\delta'' = c(\delta'/2)/\log(2/\delta')$, where c is from Lemma 9.
4. s is chosen to be the largest integer such that $((4+\delta')/\delta'')^s \leq n$. Thus $s = O((\log n)\log \delta^{-1})$.
5. l_s is chosen to be the smallest integer such that $60(\delta'')^{3/2} l_s^{-1/2} \leq \epsilon$. Note that this also implies $2^{-\delta'' l_s/4} \leq \epsilon/3$. Thus $l_s = O(\epsilon^{-2}(\delta/\log^2 \delta^{-1})^{-3})$.

6. For each $i \geq 0$, set $l_{i-1} = l_i(1 + \delta'/4)$. Using this and the definition of s, $l_0 = n^{\Omega(\delta/\log^2 \delta^{-1})}$.

7. Set $m_0 = l_0$, and for each $i > 0$ set $m_i = (4/\delta')m_{i-1}$.

8. $t_1 = 2 \log n$.

9. $t_2 = 2l_s$. Thus $t_2 = O(\epsilon^{-2}(\delta/\log^2 \delta^{-1})^{-3})$.

10. The length of the second parameter to E is given by $t = 2t_1 + t_2$. Thus $t = 4 \log n + O((\log^6 \delta^{-1})/(\delta^3 \epsilon^2))$.

11. The length of the output of E is given by $m = 2l_0 - l_s$. Thus $m = n^{\Omega(\delta/\log^2 \delta^{-1})}$.

5 Reduction to Block-Wise Source with $O(\log n)$ Blocks: RP

In this section, we show how to simulate RP using a δ-source, given a simulation A using a block-wise δ'-source with geometrically decreasing length blocks, where $\delta' = \delta/18$. The intuition is best described for reducing the problem to a simulation A' using a block-wise δ'-source with $k = O(\log r)$ equal length blocks (recall that r is the number of random bits required by the RP algorithm). We divide the output of the δ-source into k' blocks. At least $k \geq \delta k'/3$ (say) of the blocks will be close to $\delta/3$-sources, conditional on the previous blocks.[3] We therefore run A' on all $\binom{k'}{k}$ k-subsets of blocks.

Fix a k-subset S of the k' blocks. To get the geometrically decreasing block lengths l_i that we need, we divide the ith block of S into sub-blocks of size l_i. For the same reason as above, we expect a fraction at least $\delta/9$ of these sub-blocks to be close to $\delta/9$-sources. If we tried all possible sequences of sub-blocks, we would arrive at the $n^{O(\log n)}$ algorithm of [Zu1]. To get down to polynomial time, we need to consider walks on expander graphs, and must introduce a new lemma about expanders.

We now need to modify, elaborate, and formalize the above intuition. Fix a simulation A using k blocks $x_1, \ldots, x_k$ from a block-wise δ'-source, and fix an input to an RP algorithm. Note that our simulation (Function C of Subsection 3.2) requires $k = O((\log r)/\delta')$. Rather than showing that many blocks of a δ-source will be close to $\delta/3$-sources, we show, in Lemma 16, that there are many sequences of blocks on which A outputs the correct answer. We then show that the output of a δ-source is likely to have such a sequence as a subsequence.

Denote $x_1 \circ \ldots \circ x_i$ by y_i, where $\circ$ denotes concatenation. Define

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

where the probability is taken over the remaining $k - i$ blocks from the block-wise δ'-source S. Now define

$$q(y_i) = \inf_{\text{block-wise } \delta'\text{-sources } S} \{p(y_i, S)\}.$$

We now define good* and bad* according to the worst-case block-wise δ'-source. Namely, call x_i bad* with respect to y_{i-1} if $y_{i-1} \circ x_i$ has one of the $2^{\delta' l_i}$ least $q(y_i)$'s, for y_i's continuations of y_{i-1} (break ties arbitrarily); x_i is good* if it is not bad*. In other words, x_i is bad* if A, when fed $y_{i-1} \circ x_i$ followed by the output of a worst case block-wise δ'-source, has a low probability of finding a witness. Thus a worst-case block-wise δ'-source would place positive probability only on x_i which are bad*.

[3] This is not quite true, but gives reasonable intuition.

Lemma 16 *On input any k good* strings $x_1, \ldots x_k$, A outputs the correct answer.*

Proof. We prove inductively that $q(y_k) > 0$; this suffices since $q(y_k)$ is either 1 or 0 depending on whether A outputs the correct answer or not. Now $q(y_0) > 0$, since A has positive probability of outputting the correct answer for any block-wise δ'-source. Moreover,

$$q(y_{i+1}) \geq E_{\text{bad* strings } z_{i+1}} q(y_i \circ z_{i+1}) = q(y_i).$$

□

Thus, we need to get good* strings from the output of a δ-source. The following lemma would enable us to do this if all the blocks had equal length $l_i = l$: we could request $k'l$ bits from the δ-source, run A on all k-subsets of l-bit blocks, and with high probability one of them would contain all good* strings. To get intuition, think of $\delta = \Theta(1)$, $l = r$ and $k' = \Theta(k) = \Theta(\log r)$.

Lemma 17 *[Zu1] Let δ, $l \geq 3/\delta$, and k be given. Let $k' = 3k/\delta$, and set $R = k'l$. Let z be the R-bit output from the δ-source. Write $z = z_1 \circ z_2 \circ \ldots \circ z_{k'}$, where each z_i has length l. For each initial string $w_i = z_1 \circ z_2 \circ \ldots \circ z_i$, label any $2^{\delta l/3}$ of the z_{i+1}'s as bad; the others we call good. Then*

$$Pr[\text{for} \geq k \text{ values of } i,\ z_i \text{ is good}] > 1 - 2^{-\delta kl/3},$$

where the probability is taken over any δ-source. In particular, the number of z with fewer than k values of good i's is less than $2^{\delta R}$.

The plan will be to label z_i inductively as follows (although we will actually do something more complicated later). Let $z_{i_1}, z_{i_2}, \ldots, z_{i_p}$ be the good blocks seen so far, where $i_1 < i_2 < \ldots < i_p < i$ ($p = 0$ is possible). If $p \geq k$, then our labeling of z_i is irrelevant. Otherwise, label z_i good iff z_i is good* with respect to the prefix $z_{i_1} \circ z_{i_2} \ldots \circ z_{i_p}$.

Proof. Construct a tree corresponding to the outputs z of the source as follows: let the nodes be all possible initial sequences w_i for each i, $0 \leq i \leq k'$, and let the parent of w_i be w_{i-1}. Define an edge (w_{i-1}, w_i) to be bad (good) if the string z_i is bad (good) with respect to w_{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 $2^{\delta l/3}$ 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

$$2^{kl} 2^{(k'-k)\delta l/3},$$

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

$$\binom{k'}{k} 2^{kl} 2^{(k'-k)\delta l/3}.$$

Using $\binom{k'}{k} < 2^{k'}$ and substituting the definition of k' in the above formula, we bound the number by $2^{-\delta kl/3} 2^{\delta R}$, as required. □

We need to get good* blocks of different lengths (namely, geometrically decreasing) in order to apply Lemma 7. We do this by defining good r-bit blocks as ones with many good* l_i-bit blocks; then we get good r-bit blocks and search these for good* l_i-bit blocks. The problem with this is that if we use a brute-force search for the good* blocks, then we end up with an algorithm taking time roughly $\prod(r/l_i) = r^{O(\log r)}$. Indeed, this was the main obstacle to improving the $r^{O(\log r)}$ algorithm of [Zu1] to polynomial.

To get around this problem, note that the brute force approach does not exploit the fact that good r-bit blocks have many (an $\Omega(\delta)$ fraction) good* l_i-bit blocks. We exploit this by introducing a new lemma about paths on expander graphs.

We first give a way to define "good" for use in Lemma 17. We will use Lemma 17 with $l = r$. We may assume without loss of generality that $l_i | r$ (by using a padding argument). Then we inductively define whether z_i is good with respect to the initial sequence $w_{i-1} = z_1 \circ \ldots \circ z_{i-1}$ as follows: count the number p of good blocks in the string w_{i-1}, and assume $p < k$ (if $p \geq k$, our labeling won't matter). Write the jth good block $z_j = x_{j,1} \circ \ldots \circ x_{j,r/l_j}$ as a concatenation of l_j-bit sub-blocks. An initial sequence $y_p = x_{1,v_1} \circ x_{2,v_2} \circ \ldots \circ x_{p,v_p}$ is called *all-good* if each x_{j,v_j} is good* with respect to y_{j-1}. We will choose a subset T_p of possible initial sequences y_p, and we will ensure $|T_p| \leq 2^{\delta' l_{p+1}}$ (recall that $\delta' = \delta/18$). We say $x_{p+1} \in \{0,1\}^{l_{p+1}}$ is good** with respect to T_p if it is good* with respect to each all-good $y_p \in T_p$, and bad** otherwise. Then the number of strings in $\{0,1\}^{l_{p+1}}$ which are bad** is at most $|T_p| 2^{\delta' l_{p+1}} \leq 2^{2\delta' l_{p+1}}$. Call z_i good if at least a fraction $\delta/9$ of its l_{p+1}-bit sub-blocks are good** with respect to T_p. Replacing δ, l, and "good" in Lemma 17 with $\delta/3$, l_{p+1}, and "good**", we see that at most $2^{\delta r/3}$ of r-bit strings are bad. Thus, our definition of good is a valid labeling, and Lemma 17 applies: if we ask for $R = k'l$ bits from a δ-source, with high probability at least k of these blocks will be good. We find these good blocks by running our algorithm on each k-subset, multiplying our time by $\binom{k'}{k} = r^{O((\log \delta^{-1})/\delta)}$. Thus, we assume we have k good r-bit blocks $z_1, \ldots, z_k$.

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 18 *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 ith good block z_i. However, we would then have a fraction $\delta/9$ of the sub-blocks in S_i, whereas we want a fraction $1 - \alpha$. We therefore set $n = r^t$, and let S_i represent t-tuples of sub-blocks, of which at least one is good, where t is chosen so that $(1 - \delta/9)^t \leq \alpha$, so $t = O(1/\delta)$. We also must work with a modulus, so more formally:

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

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. Thus T_p consists of, for each of $n7^{p-1}$ paths of length p in a degree 7 expander, all $t^p = r^{O((\log \delta^{-1})/\delta)}$ possible combinations of sub-blocks. To ensure $|T_p| \leq 2^{\delta' l_{p+1}}$, it suffices to ensure $|T_{k-1}| \leq 2^{\delta' l_k}$, since $|T_p|$ grows with p but l_{p+1} shrinks with p. Thus $l_k = O((\log \delta^{-1})/\delta^2)$, and we have proved

Theorem 1: There is an algorithm that, given any $\delta > 0$, takes time $r^{O((\log \delta^{-1})/\delta^2)}$ to simulate RP using $R = O((r \log r)/\delta^2)$ bits from a δ-source.

6 The Unapproximability of MAX CLIQUE

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

Theorem 7 *If for some ϵ the task of approximating ω 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. $TIME(2^{polylog})$).*

Instead of considering the approximation factor as a function of n, it seems like a more balanced question to ask for a function of ω. For example, there is a simple polynomial-time algorithm to distinguish between graphs having ω 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 ω equal to n^{δ} or $n^{\delta}2^{(\log n)^{1-\epsilon}}$. In light of this, we show:

Theorem 8 *If for any constant t there is a quasi-polynomial time algorithm that always outputs a number between $\omega^{1/t}$ and ω^{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 18 *A language L is accepted by a probabilistic oracle machine M iff*

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

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 9 (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 19 (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 communication (questions and answers) $< q_1, a_1, \ldots, q_l, a_l >$ accepts x.*

Definition 20 (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 19 (FG+) $\max_O Pr_r[M^O(x,r)\ accepts] \cdot 2^{r(n)} = \omega(G_x)$.

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

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 1, which has the following corollary:

Corollary 5 *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 8: 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 $\omega(G''_x) = 2^s$, and if $x \notin L$ then $\omega(G''_x) < 2^{s/t^2}$. Thus any algorithm that always outputs a number between $\omega(G''_x)^{1/t}$ and $\omega(G''_x)^t$ 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.

Acknowledgements

I am grateful to Noam Nisan, Umesh Vazirani, Madhu Sudan, Moni Naor, Abhijit Sahay, Aravind Srinivasan, and Mike Saks for valuable discussions and comments. I also thank Leonard Schulman for a careful reading and for simplifying the proof of Lemma 11, and the anonymous referee for a careful reading and helpful suggestions.

References

[AM] N. Alon and V.D. Milman, "λ_1, Isoperimetric Inequalities for Graphs and Superconcentrators," *J. Combinatorial Theory Ser. B*, 38:73-88, 1985.

[AL+] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, "Proof Verification and Intractibility of Approximation Problems," *33rd Annual Symposium on Foundations of Computer Science*, 1992, pp. 14-23.

[AS] S. Arora and S. Safra, "Approximating Clique is NP-Complete," *33rd Annual Symposium on Foundations of Computer Science*, 1992, pp. 2-13.

[BFL] L. Babai, L. Fortnow, and C. Lund, Non-Deterministic Exponential Time Has Two-Prover Interactive Protocols, *Computational Complexity*, 1:16-25, 1991.

[BS] M. Bellare and M. Sudan, "Improved Non-Approximability Results," *26th ACM Symposium on Theory of Computing*, 1994, pp. 184-193.

[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," in *Advances in Computing Research 5: Randomness and Computation*, S. Micali, ed., JAI Press, 1989.

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

[CG1] B. Chor and O. Goldreich, "Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity," *SIAM J. Comput.*, 17(2):230-261, 1988.

[CG2] B. Chor and O. Goldreich, "On the Power of Two-Point Based Sampling," *Journal of Complexity*, 5:96-106, 1989.

[CG+] B. Chor, O. Goldreich, J. Hastad, J. Friedman, S. Rudich, and R. Smolensky, "The Bit Extraction Problem or t-Resilient Functions," *26th Annual Symposium on Foundations of Computer Science*, 1985, pp. 396-407.

[CWi] A. Cohen and A. Wigderson, "Dispersers, Deterministic Amplification, and Weak Random Sources," *30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 14-19.

[DFK] M. Dyer, A. Frieze, and R. Kannan, " A Random Polynomial Time Algorithm For Approximating the Volume of a Convex Body," *J. ACM*, 38:1-17, 1991.

[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, and M. Szegedy, "Approximating Clique is Almost NP-Complete," *32nd Annual Symposium on Foundations of Computer Science*, 1991.

[FW] J. Friedman, A. Wigderson, "On the Second Eigenvalue of Hypergraphs," Princeton University Technical Report CS-TR-232-89, 1989.

[ILL] R. Impagliazzo, L. Levin, and M. Luby, "Pseudo-Random Generation from One-Way Functions," *21st ACM Symposium on Theory of Computing*, 1989, pp. 12-24.

[IZ] R. Impagliazzo and D. Zuckerman, "How to Recycle Random Bits," *30th Annual Symposium on Foundations of Computer Science*, 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.

[LLS] D. Lichtenstein, N. Linial, and M. Saks, "Some Extremal Problems Arising from Discrete Control Processes," *Combinatorica*, 9:269-287, 1989.

[LLSZ] N. Linial, M. Luby, M. Saks, and D. Zuckerman, "Efficient Construction of a Small Hitting Set for Combinatorial Rectangles in High Dimension," *25th ACM Symposium on Theory of Computing*, 1993, pp. 258-267.

[MNT] Y. Mansour, N. Nisan, and P. Tiwari, "The Computational Complexity of Universal Hashing," *22nd ACM Symposium on Theory of Computing*, 1990, pp. 235-243.

[NZ] N. Nisan and D. Zuckerman, "Randomness is Linear in Space," *J. Comput. Sys. Sci*, to appear. Preliminary version, entitled "More Deterministic Simulation in Logspace," appeared in the *25th ACM Symposium on Theory of Computing*, 1993, pp. 235-244.

[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.

[SSZ] M. Saks, A. Srinivasan, and S. Zhou, "Explicit Dispersers with Polylog Degree," *27th ACM Symposium on Theory of Computing*, 1995, to appear.

[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," *Journal of Computer and System Sciences*, 33:75–87, 1986.

[Sho] V. Shoup, "New Algorithms for Finding Irreducible Polynomials Over Finite Fields," *Mathematics of Computation*, 54(189):435-447, 1990.

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

[SZ] A. Srinivasan and D. Zuckerman, "Computing With Very Weak Random Sources," *35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 264-275.

[Tan] R.M. Tanner, "Explicit Construction of Concentrators from Generalized N-gons," *SIAM J. Alg. Discr. Meth.*, 5:287-293, 1984.

[Va1] U. Vazirani, "Randomness, Adversaries and Computation," Ph.D. Thesis, University of California, Berkeley, 1986.

[Va2] U. Vazirani, "Efficiency Considerations in Using Semi-Random Sources," *19th ACM Symposium on Theory of Computing*, 1987.

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

[VV] U. Vazirani and V. Vazirani, "Random Polynomial Time is Equal to Semi-Random Polynomial Time," *26th Annual Symposium on Foundations of Computer Science*, 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.

[WZ] A. Wigderson and D. Zuckerman, "Expanders that Beat the Eigenvalue Bound: Explicit Construction and Applications," *25th ACM Symposium on Theory of Computing*, 1993, pp. 245-251.

[Zu1] D. Zuckerman, "General Weak Random Sources," *31st Annual Symposium on Foundations of Computer Science*, 1990, pp. 534-543.

[Zu2] D. Zuckerman, "Simulating BPP Using a General Weak Random Source," *32nd Annual Symposium on Foundations of Computer Science*, 1991, pp. 79-89.

[Zu3] D. Zuckerman, "NP-Complete Problems Have a Version That's Hard to Approximate," *8th IEEE Conference on Structure in Complexity Theory*, 1993, pp. 305-312.

[Zu4] D. Zuckerman, "An Improved Extractor and Applications." Unpublished manuscript.