Computing with Very Weak Random Sources*

Aravind Srinivasan† David Zuckerman‡

Abstract. *We study how to run randomized algorithms when the source of randomness is very defective (weak). For any fixed $\epsilon > 0$, we show how to simulate RP algorithms in time $n^{O(\log n)}$ using the output of a δ-source with min-entropy R^ϵ. Such a weak random source is asked once for R bits; it outputs an R-bit string according to any probability distribution that places probability at most 2^{-R^ϵ} on each string. If $\epsilon > 1/2$, our simulation also works for BPP; for $\epsilon > 1 - 1/(k+1)$, our simulation takes time $n^{O(\log^{(k)} n)}$ ($\log^{(k)}$ is the logarithm iterated k times). We also give a polynomial-time BPP simulation using Chor-Goldreich sources of min-entropy $R^{\Omega(1)}$, which is optimal. We present applications to time-space tradeoffs, expander constructions, and to the hardness of approximation. Of independent interest is our randomness-efficient Leftover Hash Lemma. This also helps us build on the work of Nisan & Zuckerman in adding a small number t of truly random bits to a weak random source to extract quasi-random bits; for $\delta = o(1)$ we make t much smaller than does the work of Nisan & Zuckerman.*

Keywords. Derandomization, Expander Graphs, Hashing Lemmas, Hardness of Approximation, Imperfect Sources of Randomness, Measures of Information, Pseudo-randomness, Pseudo-random Generators, Randomized Computation, Time-Space Tradeoffs.

1 Introduction

Randomness plays a vital role in almost all areas of computer science, including simulations, algorithms, network constructions, cryptography, and distributed algorithms. In practice, programs get their "random" bits by using pseudo-random number generators. Yet even in

---

*Part of this work was done while the authors attended the workshop on "Probability and Algorithms" organized by Joel Spencer and Michael Steele, which was held at the Institute for Mathematics and its Applications at the University of Minnesota from September 20th-24th, 1993. A preliminary version of this work appears in *Proc. IEEE Symposium on Foundations of Computer Science*, pages 264-275, 1994.

†Dept. of Information Systems & Computer Science, National University of Singapore, Singapore 119260, Republic of Singapore. Work done in parts at the National University of Singapore, at the Institute for Advanced Study, Princeton, NJ, USA (supported in part by grant 93-6-6 of the Alfred P. Sloan Foundation), and at the DIMACS Center, Rutgers University, Piscataway, NJ, USA (supported in part by NSF-STC91-19999 and by support from the N.J. Commission on Science and Technology). E-mail: `aravind@iscs.nus.sg`.

‡Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712. Part of this work was supported by NSF NYI Grant No. CCR-9457799. Part of this work was done while this author was visiting The Hebrew University of Jerusalem, supported by a Lady Davis Postdoctoral Fellowship. E-mail: `diz@cs.utexas.edu`.

practice there are reports of algorithms giving quite different results under different pseudo-random generators; see, e.g., [FLW] for such reports on Monte-Carlo simulations, and [Hsu93, HRD94] for the deviant performance of some RNC algorithms for graph problems. Other approaches involve using a physical source of randomness, such as a Zener diode, or using the last digits of a real-time clock. Not only is it not clear that such bits will be random, but it is impossible to test them for "randomness". We can run certain statistical tests on the bits, but we cannot run all possible ones. It is therefore natural and important to ask whether randomness is just as helpful even if the source of randomness is defective, or weak. Thus we model a weak random source as outputting bits that are slightly random, but not perfectly random.

There have been two different directions in the study of weak random sources. The first is an attempt to describe a weak random source arising in practice. Thus Manuel Blum [Blu] looked at the model where bits are output by a Markov chain, and showed how to extract perfectly random bits from such a source. Santha and Vazirani [SV] then looked at a model where the only fact known about the source is that each bit has some randomness:

Definition 1.1 ([SV]) *A semi-random source with parameter α outputs bits $X_1, \ldots, X_R$, such that $\forall i \leq R\ \forall x_1, \ldots, x_i \in \{0,1\}$, $\alpha \leq Pr[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] \leq 1 - \alpha$.*

Simulations must work for all such sources; we do not want to assume any knowledge about the source, except that it is semi-random with the value of the parameter α being known. They proved that it is impossible to extract even a single almost-random bit from one such source (so Vazirani [Va1, Va3] used 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 [Va2] showed how to efficiently simulate all algorithms in RP and BPP with one semi-random source, for any constant $\alpha > 0$. Note that these simulations use $R = poly(n)$ bits from the semi-random source, where n is the length of the input to the RP or BPP language; this explains our notation.

Chor and Goldreich [CG1] generalized the Santha-Vazirani model by assuming that no sequence of l bits has too high a probability of being output. More precisely,

Definition 1.2 ([CG1]) *An (l, b) 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 $y_1, \ldots, y_{n/l}$, $Pr[Y_i = y_i | Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}] \leq 2^{-b}$.*

A semi-random source corresponds to $l = 1$. For $l = O(\log n)$ and constant $\delta > 0$, Chor and Goldreich showed how to efficiently simulate any BPP algorithm using one $(l, \delta l)$ source. They further showed how to obtain almost-random bits from four independent such sources at a constant rate.

The other direction researchers have taken is natural mathematically, although it appears to correspond less well to weak sources in practice. These models are called bit-fixing models: some of the bits are perfectly random, while others are controlled by an adversary. Cohen and Wigderson [CW] distinguish three models based on three different adversaries: oblivious bit-fixing sources [CG+], nonoblivious bit-fixing sources [BL, KKL], and adaptive bit-fixing sources [LLS]. Only for the first model could researchers do something better than they could for general weak random sources ([CW], see below).

The two directions in weak random sources were united by the model of δ-sources [Zu1, Zu2], which generalizes all the previous models:

Definition 1.3 *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.*

As usual, we associate a source with its distribution D. Let $D(x)$ denote the probability assigned to x under distribution D.

Definition 1.4 *The min-entropy of a distribution D is $\min_x\{-\log_2 D(x) \mid D(x) > 0\}$.*

Thus, a δ-source is equivalent to a source with min-entropy at least δR. We often go back and forth between these two terminologies. In [Zu2], Zuckerman showed how to efficiently simulate any BPP algorithm using a δ-source, for any fixed $\delta > 0$. Because a δ-source is the most general model, this implies that BPP algorithms can be simulated using any model of a source where randomness is output at a constant rate. In light of this, what is left to do? The answer is to extend the result for subconstant δ. Information-theoretically, it is necessary for the R bits to have min-entropy only R^ϵ, for an arbitrary but fixed $\epsilon > 0$ [CW]. (Of course, if $BPP = P$ then no random bits are necessary: this information-theoretic result holds for an abstract model of BPP where random bits are really necessary.) Can we match this lower bound? Previously, the only weak source where this information-theoretic lower bound could be achieved was the oblivious bit-fixing source: Cohen and Wigderson showed how to simulate BPP even if the adversary fixes all but R^ϵ bits and leaves the other bits unbiased and independent, matching the above lower bound [CW].

In this paper, we come close to our goal: we give a time $n^{O(\log n)}$ simulation for any RP algorithm using a δ-source with min-entropy R^ϵ. For $\epsilon > 1/2$, our simulations also work for BPP and approximation algorithms. Moreover, for $\epsilon > 1-1/(k+1)$, our simulations take time $n^{O(\log^{(k)} n)}$, giving a polynomial-time simulation for $\epsilon > 1 - 1/(2\log^* n)$. (Here $\log^{(k)}$ denotes the logarithm to the base 2, iterated k times; i.e., $\log^{(1)} x = \log_2 x$, $\log^{(2)} x = \log_2 \log_2 x$, etc.) Furthermore, we give a simple algorithm to simulate BPP and approximation algorithms using Chor-Goldreich's (l, b) PRB-source, as long as there are at least R^ϵ blocks (i.e., $R/l \geq R^\epsilon$) and the min-entropy of the R bits is at least R^ϵ (so $Rb/l \geq R^\epsilon$), for any fixed $\epsilon > 0$. (The second condition may not be implied by the first condition if $b < 1$.) Using $l = 1$, this gives a simulation of BPP and approximation algorithms for the Santha-Vazirani source with min-entropy R^ϵ. We also generalize Cohen and Wigderson's result on oblivious bit-fixing sources: it is not necessary for n^ϵ bits to be perfectly independent and uniform, but only "weakly independent" (see Section 4.1). Our BPP simulations also work for approximation algorithms, such as the one for approximating the volume of a convex body [DFK].

Just as simulations using δ-sources for constant δ had important applications [NZ, WZ], so too do our results for subconstant δ. Some of the applications are best seen by viewing our results in graph-theoretic terms. We describe how to do this in the last section.

The first application is to a relationship between the $RP = P$ question and time-space tradeoffs. Sipser [Sip] showed that if certain expander graphs can be constructed efficiently, then for some $\epsilon > 0$ and any time bound $t(n)$, either $RP = P$ or all unary languages in DTIME($t(n)$) are accepted infinitely often in SPACE($t(n)^{1-\epsilon}$). If we had a polynomial-time simulation of RP using a δ-source with min-entropy R^ϵ for some $\epsilon > 0$, we could construct his expanders as a corollary. Because our simulations take time $n^{O(\log^{(k)} n)}$, we instead show unconditionally that either $RP \subseteq \bigcap_k DTIME(n^{\log^{(k)} n})$ or all unary languages in DTIME$(t(n))$ are accepted infinitely often in $\bigcap_k SPACE(t(n)^{1-1/\log^{(k)} n})$.

Our second application is to improve the expanders constructed in [WZ], and hence all of the applications given there. In [WZ], graphs on n nodes were constructed such that for every pair of disjoint subsets S_1 and S_2 of the vertices with $|S_1| \geq n^\delta$ and $|S_2| \geq n^\delta$, there is an edge joining S_1 and S_2; the graphs so constructed had essentially optimal maximum degree $n^{1-\delta+o(1)}$. These expanders were used in [WZ] to explicitly construct: (i) a k-round sorting algorithm using $n^{1+1/k+o(1)}$ comparisons; (ii) a k-round selection algorithm using $n^{1+1/(2^k-1)+o(1)}$ comparisons; (iii) a depth-2 superconcentrator of size $n^{1+o(1)}$, and (iv) a depth-k wide-sense nonblocking generalized connector of size $n^{1+1/k+o(1)}$.

The reader is referred to [WZ] for the definitions and motivations for these constructions. All of these results are optimal to within factors of $n^{o(1)}$. In [WZ], these $n^{o(1)}$ factors were $2^{(\log n)^{4/5+o(1)}}$, improved to $2^{(\log n)^{2/3+o(1)}}$ in the final version of [WZ]. Our results further improve these $n^{o(1)}$ factors to $2^{(\log n)^{1/2+o(1)}}$. In addition, explicit linear-sized n-superconcentrators of depth $(\log n)^{2/3+o(1)}$ were presented in [WZ]; we improve this to $(\log n)^{1/2+o(1)}$ depth. These might seem like small improvements, given the major improvement of simulations using δ-sources. The reason for this is that our algorithms do not use a near-optimal number of random bits.

Our third application is to the hardness of approximating $\log\log\omega(G)$, where $\omega(G)$ is the maximum size of a clique in an input graph G. Let $\tilde{P}$ denote quasi-polynomial time, $\cup_{c>0} DTIME(2^{(\log n)^c})$. In [Zu2], it was shown that if $N\tilde{P} \neq \tilde{P}$, then approximating $\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. In [Zu3], a randomized reduction was given showing that any iterated log is hard to approximate; in particular if $N\tilde{P} \neq ZP\tilde{P}$, then approximating $\log\log\omega(G)$ to within a constant factor is not in $co-R\tilde{P}$. This used the fact that with high probability, certain graphs are highly expanding. Our work allows us to deterministically construct graphs that are almost as highly expanding as the non-explicit constructions, thus making this last reduction deterministic, with a slight loss of efficiency: if $N\tilde{P} \not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. Recent work of [SSZ] improves further on all of these applications.

As in [Zu1, Zu2, NZ], we achieve our results using only elementary methods; in particular, we do not need expander graphs. Our main technical tool is a modification of the Leftover Hash Lemma. This very useful lemma was first proved in [ILL], and has been used extensively in simulations using δ-sources [Zu1, Zu2, NZ]. This lemma is a pseudo-random property of hash functions. However, a drawback of the lemma is that to hash from s bits to t bits, one needs at least s random bits. We show how a similar lemma can be achieved, using only $O(\log s + t)$ random bits. Because this modification was so useful to us here, we believe it will be useful elsewhere too. A similar lemma was proved independently in [GW]; however, our proof is somewhat simpler. One key consequence of our lemma is an improvement of the extractor of [NZ]. That is, we show how to add a small number t of truly random bits to a δ-source in order to extract almost-random bits; we make t much smaller than in [NZ]. This also uses some new techniques for using such extraction procedures recursively. Recently, by developing some new tools and by using some of our improved hashing lemmas, the above-mentioned work of [SSZ] has improved our RP simulation to $poly(n)$ time.

Section 2 sets up the required preliminary notions. Section 3 presents our first technical tool, the improved Leftover Hash Lemma; Section 4 shows how this new Leftover Hash Lemma can be used to run BPP algorithms using very weak Chor-Goldreich sources, which in turn proves useful in generalizing a result of [CW] on oblivious bit-fixing sources. Sections 5 and 6 contain some of our main results: simulating BPP and RP algorithms using general weak sources with very low min-entropy. Applications and open problems are presented in Section 7.

2 Preliminaries

We use capital letters to denote random variables, sets, distributions, and probability spaces; small letters denote other variables. 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. We ignore round-off errors, assuming when needed that a number is an integer; it can be seen that this does not affect the validity of our arguments. All logarithms are to the base 2, unless specified otherwise.

2.1 Basic definitions

Definition 2.1 *RP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial-time Turing machine $M_L(\cdot,\cdot)$ for which $a \in L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 1/2$, and $a \notin L \Rightarrow Pr[M_L(a,x) \text{ accepts}] = 0$, where the probabilities are for an x picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p = p_L$. BPP is the set of languages $L \subseteq \{0,1\}^*$ such that there is a deterministic polynomial-time Turing machine $M_L(\cdot,\cdot)$ for which $a \in L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \geq 2/3$ and $a \notin L \Rightarrow Pr[M_L(a,x) \text{ accepts}] \leq 1/3$, where the probabilities are for an x picked uniformly in $\{0,1\}^{p(|a|)}$ for some polynomial $p = p_L$.*

As is well-known, by running M_L on independent random tapes, we can change the probabilities 1/2, 2/3, and 1/3 above to $1-2^{-poly(|a|)}$, $1-2^{-poly(|a|)}$ and $2^{-poly(|a|)}$ respectively, while still retaining a polynomial running time.

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

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

A distribution D on X is called ϵ-quasi-random (on X) if the variation 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 $\|D_1 \times E_1 - D_2 \times E_2\| \leq \|D_1 - D_2\| + \|E_1 - E_2\|$. The triangle inequality is obvious: $\|D_1 - D_3\| \leq \|D_1 - D_2\| + \|D_2 - D_3\|$.

2.2 Simulations using weak random sources

A source that outputs R bits is a probability distribution on $\{0,1\}^R$; we often go back and forth between these two notions. By a simulation using, say, a Chor-Goldreich source, we really mean a simulation that will work for *all* Chor-Goldreich sources. Thus, we talk about a simulation for a family of sources.

To define what simulating RP means, say we wish to test whether a given string a is in an RP language 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., $W = \{x \in \{0,1\}^r | M_L(a,x) \text{ accepts}\}$, where r denotes the number of random bits used by M_L. One might think that to simulate RP using a 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 do not use the fact that W can be recognized in P Thus $r - O(\log r)$ bits of randomness are required. As we can ask for at most $r^{O(1)}$ bits from the source, this is what gives the R^ϵ min-entropy lower bound of [CW].

Definition 2.2 *A polynomial-time algorithm simulates RP using a source from a family of sources $\mathcal{S}$ if on input any constant $c > 0$ and $R = poly(r)$ bits from any $S \in \mathcal{S}$, it outputs a polynomial number of r-bit strings z_i, such that for all $W \subseteq \{0,1\}^r, |W| \geq 2^{r-1}$, $Pr[(\exists i) z_i \in W] \geq 1 - r^{-c}$.*

If we had a perfect random source, we could make the error exponentially small. Indeed, with sources like the Chor-Goldreich PRB-source, where we can request more bits with independence conditions from the first, we can always repeat the algorithm to achieve an exponentially small error. However, with arbitrary sources, it is not obvious that this can be done. Yet it seems reasonable to insist only on polynomially small error, as we want the error to fool polynomial-time machines.

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. 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 2.3 *A polynomial-time algorithm simulates BPP using a source from a family of sources $\mathcal{S}$ if on input any constant $c > 0$ and $R = poly(r)$ bits from any $S \in \mathcal{S}$, it outputs a polynomial number of r-bit strings z_i, such that for all $W \subseteq \{0,1\}^r, |W| \geq \frac{2}{3}2^r$, $Pr[\text{majority of } z_i\text{'s lie in } W] \geq 1 - r^{-c}$.*

Such an algorithm A can also be used to simulate approximation algorithms since if a majority of numbers lie in a given range, then their median also lies in it. Thus by taking medians instead of majorities, a good approximation can be obtained with probability at least $1 - r^{-c}$. Most of our simulations are actually constructions of something even stronger, an extractor [NZ]. An extractor produces a quasi-random output (here we modify the definition of [NZ] to general families of sources):

Definition 2.4 *Let $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$, and $\epsilon > 0$ be a parameter. E is called an extractor with quasi-randomness ϵ for a family of sources $\mathcal{S}$ on $\{0,1\}^n$ if for for any $S \in \mathcal{S}$,* *the distribution of $E(x,y)\circ y$ induced by choosing x according to S and y independently and uniformly from $\{0,1\}^t$ is quasi-random (on $\{0,1\}^m \times \{0,1\}^t$) to within ϵ. If, in particular, $\mathcal{S}$ is the class of δ-sources on $\{0,1\}^n$, then E is called an (n,δ,t,m,ϵ)-extractor.*

As in [NZ], an extractor construction yields a BPP simulation; the idea is to run the extractor using all possible 2^t strings y, and then produce an appropriate output:

Lemma 2.5 *If there is a polynomial-time extractor for $\mathcal{S}$ with parameters $\epsilon = n^{-\Omega(1)}$, $m = n^{\Omega(1)}$, and t, then there is a simulation of BPP using any source S from $\mathcal{S}$ and running in time $2^t n^{O(1)}$.* □

As observed in the final version of [Zu2] (or see the remark after Lemma 1 in the conference version of [Zu2]), for δ-sources we need only $\epsilon \le 1/3$, say, to achieve $1/poly(n)$ (even exponentially small) error. Also, though we focus on extractors that run in polynomial time, all of our extractors can actually be made to run in NC; see the remark at the end Section 3.

3 The Improved Leftover Hash Lemma

To understand the Leftover Hash Lemma intuitively, imagine that we have an element x chosen uniformly at random from an arbitrary set $A \subseteq \{0,1\}^s$ with $|A| = 2^t$, $t < s$. Thus we have t bits of randomness, but not in a usable form. If we use some additional random bits, the Leftover Hash Lemma allows us to convert the randomness in x into a more usable form. These extra bits are used to pick a uniformly random hash function h mapping s bits to $t-2k$ bits, where k is a security parameter. Recall that given finite sets A and B, a family H of functions mapping A to B is a *universal family of hash functions* if for any $a_1 \neq a_2 \in A$, and any $b_1, b_2 \in B$, $Pr[h(a_1) = b_1$ and $h(a_2) = b_2] = 1/|B|^2$, where the probability is over h picked uniformly at random from H. The Leftover Hash Lemma guarantees that the ordered pair $(h, h(x))$ is almost-random:

Leftover Hash Lemma ([ILL]): *Let $A \subseteq \{0,1\}^s, |A| \ge 2^t$. Let $k > 0$, and let H be a universal family of functions mapping $\{0,1\}^s$ to $\{0,1\}^{t-2k}$. Then the distribution of $(h, h(x))$ is quasi-random within 2^{-k} (on the set $H \times \{0,1\}^{t-2k}$), if (h,x) is chosen uniformly at random from $H \times A$.*

One drawback of this lemma is that to pick a universal hash function mapping s bits to $t-2k$ bits, one needs at least s bits. One attempt around this is to use the extractor of [NZ]; however that is only useful if t/s is large. Here we show how to use only $O(t + \log s)$ bits and achieve a good result. We first recall

Definition 3.1 ([NN]) *A "d-wise ρ-biased" sample space S of n-bit vectors has the property that if $\vec{X} = (X_1, \ldots, X_n)$ is sampled uniformly at random from S, then $\forall I \subseteq \{1,2,...,n\}, |I| \le d, \quad \forall b_1, b_2, \ldots, b_{|I|} \in \{0,1\}$,*

$$|Pr_{\vec{X}\in S}[\bigwedge_{i\in I} X_i = b_i] - 2^{-|I|}| \le \rho. \tag{1}$$

Simplifying the construction in [NN], d-wise ρ-biased spaces of cardinality $O((d\log n/\rho)^2)$ were constructed explicitly in [AG+]. In addition, given the random bits to sample from S, any bit of $\vec{X}$ can be computed in $poly(d, \log n, \log(\rho^{-1}))$ time.

Lemma 3.2 *Let $A \subseteq \{0,1\}^s, |A| \geq 2^t$, $k > 0$, and $\epsilon \geq 2^{1-k}$. There is an explicit construction of an efficiently computable family F of functions mapping s bits to $t-2k$ bits, such that the distribution of $(f, f(x))$ is quasi-random within ϵ (on the set $F \times \{0,1\}^{t-2k}$), where f is chosen uniformly at random from F, and x uniformly from A. Moreover, a random element from F can be specified using $4(t-k) + O(\log s)$ random bits.*

Proof. Any $g : \{0,1\}^s \to \{0,1\}^{t-2k}$ can be represented in the natural way by a vector in $\{0,1\}^\ell$, where $\ell = (t-2k)2^s$. Now let F be a $2(t-2k)$-wise ρ-biased sample space for ℓ-length bit vectors (i.e., a family of functions $\{\{0,1\}^s \to \{0,1\}^{t-2k}\}$), where $\rho = (\epsilon^2 2^{2k} - 1)2^{-2t+2k}$ (which is non-negative since $\epsilon \geq 2^{1-k}$). F can be sampled using $2(\log(t-2k) + \log\log\ell + \log\rho^{-1}) + O(1) \leq 4(t-k) + O(\log s)$ random bits. We now show that F has the desired property. We follow the proof of the Leftover Hash Lemma due to Rackoff (see [IZ]).

Definition 3.3 *The* collision probability *of a distribution D on a set S is $Pr[y_1 = y_2]$, where y_1 and y_2 are picked* independently *from S according to D.*

For the distribution D on $(f, f(x))$ described above, the collision probability is

$$p = Pr_{x_1,x_2 \in A, f_1, f_2 \in F}[f_1 = f_2, f_1(x_1) = f_2(x_2)],$$

where all the random choices are uniform and independent. We show that the collision probability using F is almost the same as it would be using a universal family of hash functions. Now,

$$\begin{aligned}
p &= \frac{1}{|F|^2} \sum_{f \in F} Pr_{x_1,x_2 \in A}[f(x_1) = f(x_2)] \\
&= \frac{1}{|F|^2} \sum_{f \in F} Pr_{x_1,x_2 \in A}[f(x_1) = f(x_2) | x_1 = x_2] \cdot Pr[x_1 = x_2] + \\
&\quad \frac{1}{|F|^2} \sum_{f \in F} Pr_{x_1,x_2 \in A}[f(x_1) = f(x_2) | x_1 \neq x_2] \cdot Pr[x_1 \neq x_2] \\
&= 2^{-t}/|F| + \frac{1}{|F|^2} \sum_{f \in F} Pr_{x_1,x_2 \in A}[f(x_1) = f(x_2) | x_1 \neq x_2] \cdot Pr[x_1 \neq x_2] \\
&\leq 2^{-t}/|F| + \frac{1}{|F|^2} \sum_{f \in F} Pr_{x_1,x_2 \in A}[f(x_1) = f(x_2) | x_1 \neq x_2]. \qquad (2)
\end{aligned}$$

For any $a_1, a_2 \in A$, $a_1 \neq a_2$,

$$\begin{aligned}
\frac{1}{|F|^2} |\{f \in F : f(a_1) = f(a_2)\}| &= \frac{1}{|F|} \sum_{b \in \{0,1\}^{t-2k}} Pr_{f \in F}[f(a_1) = f(a_2) = b] \\
&\leq \frac{1}{|F|} \sum_{b \in \{0,1\}^{t-2k}} (2^{-2(t-2k)} + \rho) \text{ (by (1))} \\
&= \frac{1}{|F| 2^{t-2k}} (1 + \epsilon^2 - 2^{-2k}).
\end{aligned}$$

Thus, from (2), $p \leq (1+\epsilon^2)/(|F| 2^{t-2k})$. We now invoke the rest of Rackoff's proof, which shows that such an upper bound on p implies the ϵ-quasi-randomness of D. □

Corollary 3.4 *The conclusion to Lemma 3.2 holds if x is chosen from any δ-source, $\delta = t/s$.*

Proof. The only place where the distribution of x is needed in the above proof is in showing that its collision probability is 2^{-t}; note that 2^{-t} is an upper bound on the collision probability if x is chosen from a δ-source with $\delta = t/s$. This concludes the proof. □

Remark: In order to use Lemma 3.2, we need an irreducible polynomial over $GF[2]$ for the d-wise ρ-biased spaces. For this we use an algebraic result that if an integer m is of the form $2 \cdot 3^t$, then $f(z) = z^m + z^{m/2} + 1$ is an explicit irreducible polynomial over $GF[2]$ of degree m (see exercise 3.96 on page 146 of [LN]). This allows our extractors to run in NC.

4 Simulating BPP using a Chor-Goldreich Source with min-entropy R^ϵ

We now show how to simulate BPP using a Chor-Goldreich source with min-entropy R^ϵ for any fixed $\epsilon > 0$; in fact, we build an extractor for Chor-Goldreich sources. Note that since we are talking about extractors, we use the more usual symbol n, rather than R, to denote the number of random bits requested from the source. Suppose we are given an (l, b) PRB-source with m $(= n/l) = n^{\Omega(1)}$ blocks and with the min-entropy mb of the n bits being at least $n^{\Omega(1)}$. First note that for any integral $a > 0$, an (l, b) PRB-source is also an (al, ab) PRB-source: simply group every a blocks together. Suppose we want the output of E to be quasi-random to within n^{-c}. By choosing a appropriately, we may assume that $b \geq 4(c+2)\log n$; we may also assume that $b = \Theta(\log n)$, since an (l, b_1) PRB-source is trivially an (l, b_2) PRB-source, if $b_2 < b_1$. We then pick a family F that satisfies Lemma 3.2 with parameters $k = (c+2)\log n$ and $\epsilon = n^{-(c+1)}$. Now we use the following modification of a lemma from the final version of [Zu2] which, using the Leftover Hash Lemma in the manner of [IZ], essentially strengthened related lemmas in [Va2] and [CG1].

Lemma 4.1 *Let F be a function family mapping l bits to $b - 2k$ bits, satisfying Lemma 3.2 with parameters $k = (c+2)\log n$ and $\epsilon = n^{-(c+1)}$. Let D be an (l, b) PRB-source on $\{0,1\}^{ml}$. If $\vec{Y} = Y_1 \ldots Y_m$ is chosen according to D and f is chosen uniformly at random from F, then the distribution of $(f, f(Y_1), \ldots, f(Y_m))$ is quasi-random to within $m\epsilon$.*

Proof. The proof is by backward induction, as in [NZ]. We proceed by induction from $i = m$ to $i = 0$ on the statement: for any sequence of values $y_1 \ldots y_i$, the distribution of $(f, f(Y_{i+1}), \ldots, f(Y_m))$ conditioned on $Y_1 = y_1, \ldots, Y_i = y_i$, is quasi-random to within $(m-i)\epsilon$. This is obvious for $i = m$. Suppose it is true for $i+1$. Fix the conditioning $Y_1 = y_1, \ldots, Y_i = y_i$ from now on, and let D_{i+1} denote the induced distribution on Y_{i+1}. We now use the obvious fact that if a statement is true for each element of a set, then it is also true for an element chosen randomly from the set, using any probability distribution. Since, by the induction hypothesis, for every y_{i+1}, the induced distribution on $(f, f(Y_{i+2}), \ldots, f(Y_m))$ is quasi-random to within $(m-i-1)\epsilon$, we have that the distribution $(Y_{i+1}, f, f(Y_{i+2}), \ldots, f(Y_m))$ is within $(m-i-1)\epsilon$ of the distribution $D_{i+1} \times U_{i+1}$, where U_{i+1} is the uniform distribution on $F \times \{0,1\}^{(m-i-1)(b-2k)}$. Thus, the distribution of $(f, f(Y_{i+1}), \ldots, f(Y_m))$ is within $(m-i-1)\epsilon$ of the distribution of $(f, f(Y_{i+1}), z_{i+2}, \ldots, z_m)$ obtained by picking Y_{i+1} according to D_{i+1}, and $(f, z_{i+2}, \ldots, z_m)$ independently and uniformly at random from $F \times \{0,1\}^{(m-i-1)(b-2k)}$ (since $\|g(D_1) - g(D_2)\| \leq \|D_1 - D_2\|$ for any two distributions D_1 and D_2 and any function g). Using Corollary 3.4, the distribution of $(f, f(Y_{i+1}), z_{i+2}, \ldots, z_m)$ is quasi-random to within ϵ, and the lemma follows from the triangle inequality for variation distance. $\Box$

Sampling from F requires $O(\log l + b + \log(1/\epsilon)) = O(\log n)$ random bits and thus, we have a good extractor for these sources. Using Lemma 2.5, we get

Theorem 4.2 *For any fixed $\epsilon > 0$, BPP can be simulated using a Chor-Goldreich (l, b) PRB-source, as long as there are at least n^ϵ blocks (i.e., $n/l \geq n^\epsilon$) and if the min-entropy of the n bits is at least n^ϵ (i.e., $nb/l \geq n^\epsilon$). In fact, an explicit extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^s$ that runs in NC can be built for this family of sources, with $t = O(\log n)$, $s = n^{\Omega(1)}$, and with the quasi-randomness of the output being $n^{-\Omega(1)}$.*

4.1 Sources with Many Weakly-Independent Bits

As an application of Theorem 4.2, we now show how to simulate BPP using a generalization of the oblivious bit-fixing source of [CW], using Lemma 4.1. Here again, we actually build an extractor for these sources. We need

Definition 4.3 *A bit X_i from a distribution on $X_1 X_2 \ldots X_n$ has* weak independence *α if α is the maximum value in $[0, 1/2]$ such that for every setting $X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n$ of all the other bits, $\alpha \leq Pr[X_i = 0 | \bigwedge_{j \neq i}(X_j = x_j)] \leq 1 - \alpha$.* Thus a bit of the oblivious bit-fixing source that is not fixed has weak independence 1/2. Note the difference between this definition and the semi-random source of [SV]: we look at a bit conditioned on all the other bits, not just the previous bits. It is not necessarily true that every bit of a semi-random source with parameter α has weak independence close to α.

Theorem 4.4 *For a source S outputting n bits $(X_1, X_2, \ldots, X_n)$, let α_i denote the weak independence α_i of bit X_i. For any fixed $\epsilon > 0$, there is an efficient (explicitly-given) extractor $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ for the class of sources with $\sum_{i=1}^n \alpha_i \geq n^\epsilon$, where $t = O(\log n)$ and $m = n^{\Omega(1)}$; the output of the extractor is $n^{-\Theta(1)}$-quasirandom. The extractor need only know the value ϵ, and not the quantities α_i.*

Note that this is best possible: the entropy of $(X_1, X_2, \ldots, X_n)$ is at least $k \doteq \sum_{i=1}^n H(\alpha_i)$ (and could be this low), where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the usual binary entropy function, with $H(0) = H(1) \doteq 0$. Given that $\beta \doteq \sum_{i=1}^n \alpha_i$ is a certain value t, k is maximized when each α_i equals t/n, by the concavity of H; so $k = O(\beta \log(n/\beta))$. Thus if $\beta = n^{o(1)}$, then the entropy of $(X_1, X_2, \ldots, X_n)$ can be $n^{o(1)}$ also; hence such an extractor construction would not be possible. Thus we indeed need $\sum_{i=1}^n \alpha_i \geq n^\epsilon$, for some fixed $\epsilon > 0$.

Proof. We proceed by showing how to use $O(\log n)$ purely random bits to obtain a source that is a Chor-Goldreich source with high probability; the theorem then follows from Lemma 4.1. The idea is to take a pairwise independent permutation of the bits as in [Zu2], and then divide our string into blocks. We then argue that many weakly independent bits fall in each block. The reason we need weak independence is because a bit's weak independence does not change if the bits are permuted. Thus, we do not need to pick the blocks independently, as in [NZ], or use more complicated methods, as in [Zu2].

Assume without loss of generality that n is prime and that $\sum_{i=1}^{n} \alpha_i = n^{\epsilon}$, and associate the finite field on n elements with $\{1, 2, \ldots, n\}$. Pick a to be a random *nonzero* element of the field, and b to be a random field element; the map π is then $\pi(i) = ai + b$. Since $a \neq 0$, π is a permutation. Divide the n bits into $m = n^{\epsilon/3}$ blocks $B_1, \ldots, B_m$ of length $l = n^{1-\epsilon/3}$, according to π; i.e., $B_i = (X_{\pi^{-1}((i-1)l+1)}, X_{\pi^{-1}((i-1)l+2)}, \ldots, X_{\pi^{-1}(il)})$. It suffices to show that with high probability, each of these blocks gets many weakly independent bits: this will give a Chor-Goldreich source. Fix $i \in \{1, 2, \ldots, m\}$ arbitrarily. Define the random variable W_i as the weak independence of block B_i, i.e., $\sum_{j=1}^{n} Y_j$, where $Y_j = \alpha_j$ if $\pi(j) \in \{(i-1)l+1, (i-1)l+2, il\}$, and 0 otherwise. Note that $E[Y_j] = \alpha_j/n^{\epsilon/3}$ and hence, $E[W_i] = n^{2\epsilon/3}$. Now since π is a pairwise independent permutation, $Pr[\pi(i_1) = j_1 \text{ and } \pi(i_2) = j_2] = 1/(n(n-1))$, for any distinct i_1 and i_2, and any distinct j_1 and j_2. Thus for any j, k, $j \neq k$,

$$E[Y_j Y_k] = |B_i||B_i - 1|\frac{\alpha_j \alpha_k}{n(n-1)} \leq \frac{\alpha_j \alpha_k}{n^{2\epsilon/3}} = E[Y_j]E[Y_k].$$

Thus, the variance $Var[W_i]$ of W_i is $\sum_j (E[Y_j^2] - (E[Y_j])^2) + 2\sum_{j<k}(E[Y_j Y_k] - E[Y_j]E[Y_k])$, which is at most

$$\sum_j (E[Y_j^2] - (E[Y_j])^2) \leq \sum_j E[Y_j^2] = \sum_j \frac{\alpha_j^2}{n^{\epsilon/3}} \leq n^{2\epsilon/3}/2.$$

So using Chebyshev's inequality, $Pr[W_i < n^{2\epsilon/3}/2] \leq 2n^{-2\epsilon/3}$ and hence, $Pr[(\exists i) W_i < n^{2\epsilon/3}/2] \leq 2n^{-\epsilon/3}$. Hence with probability at least $1 - 2n^{-\epsilon/3}$, we have the output of a Chor-Goldreich source with min-entropy $n^{\Omega(1)}$. Thus if we now run the extractor of Lemma 4.1 on this output, the quasi-randomness of the final output is at most $\epsilon' + 2n^{-\epsilon/3} = n^{-\Theta(1)}$, where $\epsilon' = n^{-\Theta(1)}$ is the amount of quasi-randomness introduced by the extractor of Lemma 4.1. $\Box$

5 Simulating BPP using a δ-source with min-entropy $R^{1/2+\epsilon}$

We now use our new Leftover Hash Lemma to modify the extractor developed in [NZ]. Part of the extractor there used the original Leftover Hash Lemma to show how to extract quasi-random bits from a certain kind of "block-wise δ-source" B (see definition below). However, since the original Leftover Hash Lemma needs a number of random bits proportional to the logarithm of the size of the domain of the hash functions, the block-sizes in B had to decrease at the rate of $(1 + \delta/4)$, thus requiring $O((\log n)/\delta)$ blocks overall. However, our new construction allows the block-sizes of B to decrease at a constant rate independent of δ, thus requiring only $O(\log n)$ blocks. This is because now if we need to hash from a δ-source on l bits to $\delta l/2$ bits, Lemma 3.2 and Corollary 3.4 guarantee us a hash function family having error $2^{1-\delta l/4}$ which can be described using $3\delta l + O(\log l)$ bits (this is at most $4\delta l$ bits, provided $1/\delta \leq cl/\log l$ for a sufficiently small constant c). To see this, just plug in $s = l$, $t = \delta l$, $k = \delta l/4$ and $\epsilon = 2^{1-k}$ in Corollary 3.4.

The following definition generalizes the Chor-Goldreich class of sources by allowing different block-lengths l_i.

Definition 5.1 *A block-wise δ-source outputs bits as blocks $Y_1, \ldots, Y_s$, where Y_i has length l_i, such that for all $i, y_1, \ldots, y_i$, $Pr[Y_i = y_i | Y_1 = y_1, \ldots, Y_{i-1} = y_{i-1}] \leq 2^{-\delta l_i}$.*
Note that δ and the block-lengths l_i are assumed to be known parameters of these sources.

As in [NZ], our extractor is composed of a *block-wise converter* and a *block-wise extractor*. Unlike [NZ], we define them explicitly here.

Definition 5.2 **(i)** *$E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^{l_1+\cdots+l_s}$ is an $(n, \delta, t, (l_1, \ldots, l_s), \delta', \epsilon)$ block-wise converter if, for x chosen from a δ-source on $\{0,1\}^n$ and y independently and uniformly at random from $\{0,1\}^t$, $E(x, y) \circ y$ is within ϵ of a distribution $D \times U$, where D is some block-wise δ'-source with block lengths $l_1, \ldots, l_s$ and U is the uniform distribution on $\{0,1\}^t$.*
(ii) *$E : \{0,1\}^{l_1+\cdots+l_s} \times \{0,1\}^t \to \{0,1\}^m$ is an $((l_1, \ldots, l_s), \delta, t, m, \epsilon)$ block-wise extractor if, for x chosen from a block-wise δ-source on $\{0,1\}^n$ and y independently and uniformly at random from $\{0,1\}^t$, $E(x, y) \circ y$ is ϵ-quasi-random on $\{0,1\}^m$.*

Lemma 5.3, implicit in [NZ], shows how to combine a block-wise converter and a block-wise extractor:

Lemma 5.3 *Suppose we are given an efficient $(n, \delta, t_1, (l_1, \ldots, l_s), \delta', \epsilon_1)$ block-wise converter and an efficient $((l_1, \ldots, l_s), \delta', t_2, m, \epsilon_2)$ block-wise extractor. Then we can construct an efficient $(n, \delta, t_1 + t_2, m, \epsilon_1 + \epsilon_2)$-extractor.*

Proof. Just run the converter on the output of the δ-source, and then run the extractor on the output of the converter. □

Given Lemma 5.3, we now focus on constructing an appropriate block-wise converter and a block-wise extractor; these are described in Sections 5.1 and 5.2 respectively.

5.1 A Block-Wise Extractor

Lemma 4.1 gives a block-wise extractor. However, our block-wise converter will only be useful if the number of blocks s in the block-wise δ-source is small; Lemma 4.1 results in $s = n^{\Theta(1)}$, which is too high for our purposes. We therefore use the following block-wise extractor C, which is similar to the one in [NZ] except that we use our improved version of the Leftover Hash Lemma.

Function C: The function C has 4 parameters: r, the number of bits used to describe a member of the hash family, s, the number of blocks, l_s, the smallest block size, and δ, the quality of the source. C only works if r bits suffice to hash from l_s bits to $\delta l_s/2$ bits to get a distribution that is quasi-random to within $2^{1-r/16}$, as prescribed by Corollary 3.4. Thus, $c \log l_s \leq r \leq 3\delta l_s + O(\log l_s) \leq 4\delta l_s$ for a suitably large constant c. (If the upper bound on r does not hold then the error cannot be made small enough: the error will be $O(2^{-\delta l_s/4})$, rather than $O(2^{-r/16})$. If the lower bound does not hold then the domain is too large for our hash family to work.)

We define $r_s = r$ and $r_{i-1}/r_i = 9/8$, and then the block lengths $l_i = \max(r_i/(4\delta), l_s)$. (The reader should first think of $r = 4\delta l_s$, and $l_{i-1}/l_i = 9/8$. The reason we choose l_s larger is to reduce the error ϵ of the block-wise converter in Lemma 5.12. Note that Lemma 5.5 allows smaller errors for larger values of l.) Then Lemma 3.2 assures us for each i a fixed family of hash functions $H_i = \{h : \{0,1\}^{l_i} \to \{0,1\}^{\delta l_i/2}\}$ with $|H_i| \le 2^{r_i}$, so we assume $|H_i| = 2^{r_i}$.

1. INPUT: $x_1 \in \{0,1\}^{l_1}, \ldots, x_s \in \{0,1\}^{l_s}$; $y \in \{0,1\}^r$.
2. $h_s \leftarrow y$.
3. For $i = s$ downto 1 do $h_{i-1} \leftarrow h_i \circ h_i(x_i)$.
4. OUTPUT (a vector in $\{0,1\}^{r_0 - r}$): h_0, excluding the bits of h_s.

By choosing s large enough, we can ensure that $l_0 = r_0/(4\delta)$. Specifically, suppose $s \ge \log_{9/8}(4\delta l_s/r)$. Then, $r_0/(4\delta) = r(9/8)^s/(4\delta) \ge l_s$ and hence, by the definition of l_i, $r_0 = 4\delta l_0$.

Lemma 5.4 *C is an $((l_1, \ldots, l_s), \delta, r, r_0 - r, 4 \cdot 2^{-r/16})$ block-wise extractor.*

Proof. This proof is very similar to a corresponding proof in [NZ]. Let $\vec{X} = X_1 \ldots X_l$ be chosen according to D, a block-wise δ-source on $\{0,1\}^{l_1 + \cdots + l_s}$, and Y be chosen uniformly from $\{0,1\}^r$. Let $r_s = r$ and $r_{i-1}/r_i = 9/8$. Then $r_i \le 4\delta l_i$. 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 $\epsilon_i = \sum_{j=i+1}^{s} 2^{1-r_j/16}$.

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 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 3.4 this second distribution is quasi-random to within $2^{1-r_{i+1}/16}$. $\square$

5.2 A Block-Wise Converter

Our block-wise converter is a small modification of that in [NZ]. For our simulations of RP and BPP, it would suffice to change the k-wise independence in [NZ] to pairwise independence, as was done in [Zu2]. However, by using an improved analysis of k-wise independence from [BR94], we can give a good extractor for a wider range of parameters. This subsection is essentially taken from [NZ], with the only changes being this improved analysis of [BR94]. In order to define our block-wise converter, we first show how to extract one block from a δ-source. The idea to do this is as follows. Intuitively, a δ-source has many bits which are somewhat random. We wish to obtain l of these somewhat random bits. This is not straightforward, as we do not know which of the n bits are somewhat random. We therefore pick the l bits at random using k-wise independence.

5.2.1 Choosing l out of n elements

We divide the n elements into disjoint sets $A_1, \ldots, A_l$ of size $m = n/l$, i.e., $A_i = \{(i-1)m+1, (i-1)m+2, \ldots, im\}$. We then use $k \log n$ random bits to choose $X_1, \ldots, X_l$ k-wise independently, where the range of X_i is A_i, and define our (random) output to be $S = \{X_1, \ldots, X_l\}$. The property we will require is:

Lemma 5.5 *Let $T \subseteq \{1, 2, \ldots, n\}$, $|T|/n \geq \delta$. Suppose k is even, $4 \leq k \leq (\delta l)^{1-\beta}/8$ for some $\beta > 0$. If S is chosen at random as described above, then*

$$Pr[|S \cap T| \leq \delta l/2] \leq 8(\delta l)^{-\beta k/2}.$$

We use the following lemma from [BR94]:

Lemma 5.6 *For $k \geq 4$ an even integer, let $Y_1, \ldots, Y_l$ be k-wise independent 0-1 random variables, $Y = \sum_{i=1}^{l} Y_i$, and $\mu = E[Y]$. Then for $\gamma > 0$, $Pr[|Y - \mu| \geq \gamma] \leq 8((k\mu + k^2)/\gamma^2)^{k/2}$.*

Proof of Lemma 5.5: Define the random variables Y_i to be 1 iff $X_i \in T$, and 0 otherwise. Let $\delta_i = E[Y_i] = |T \cap A_i|/m$. Then for $Y = \sum_{i=1}^{l} Y_i$, $E[Y] = \sum_{i=1}^{l} \delta_i \geq \delta l$. Setting $\gamma = \delta l/2$ in Lemma 5.6 concludes the proof. □

Lemma 5.7 *Suppose $ck \leq \delta^2 l$. Then we can use $O(k/\delta + \log n)$ random bits to pick l random variables $X_1, \ldots, X_l$ in $\{1, 2, \ldots, n\}$ such that $Pr[\geq \delta^2 l/16$ of the X_i's lie in $T] \geq 1 - 2^{-k}$.*

5.2.2 Extracting One Block

The function B:

B has 4 parameters: n, the size of the original input; l, the size of the output; k, the amount of independence used; and δ, the quality of randomness needed.

1. INPUT: $x \in \{0,1\}^n$; $y \in \{0,1\}^t$ (where $t = k \log n$).
2. Use y to choose a set $\{i_1 \ldots i_l\} \subset \{1 \ldots n\}$ of size l as described in Section 5.2.1.
3. OUTPUT (a vector in $\{0,1\}^l$): $x_{i_1} \ldots x_{i_l}$ (here x_j is the jth bit of x).

Lemma 5.8 *If 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. Here $\delta' = c\delta/\log \delta^{-1}$, and for $k \leq (\delta' l)^{1-\beta}$, $\epsilon = (\delta' l)^{-c\beta k}$ for some sufficiently small positive constant c.*

To prove Lemma 5.8, we proceed as in [NZ]. Fix a δ-source D. We need the following definitions that are relative to D.

Definition 5.9 *For $\vec{x} \in \{0,1\}^n$ and $1 \leq i \leq n$, 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 if $p_i(\vec{x}) = 1/2$ 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 5.10 *$\vec{x}$ is α-good if there are at least αn indices which are good in x. 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 $\vec{x}$; S is α-informative to within β if $Pr_{\vec{X}\in D}[\vec{X}\ is\ \alpha - good\ in\ S] \geq 1 - \beta$.*

Denote by S_y the set of l indices chosen using the random bits $\vec{y}$, as described in Section 5.2.1. A useful result shown in [NZ] is that for any set of indices $\{i_1 \ldots i_l\}$ that is δ'-informative to within ϵ, the distribution of $X_{i_1} \ldots X_{i_l}$ induced by choosing $\vec{X}$ according to D is ϵ-near a δ'-source. This result, together with 5.11, will clearly prove Lemma 5.8.

Lemma 5.11 $Pr_{\vec{Y}}[S_Y\ is\ \delta' - informative\ to\ within\ \epsilon] \geq 1 - \epsilon$.

Proof. We first need the following result from [NZ]:

$$Pr_{\vec{X}\in D}[\vec{X}\ is\ not\ \alpha - good] \leq 2^{-c_1 \delta n}, \tag{3}$$

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

For any fixed α-good string $\vec{x}$, we can apply Lemma 5.5 to the set of good indices and obtain $Pr_Y[\vec{x}\ has\ \leq \alpha l/2\ good\ indices\ in\ S_Y] \leq 8(\alpha l)^{-\beta k/2}$. Using (3), it follows that

$$Pr_{\vec{X},Y}[\vec{X}\ has\ \leq \alpha l/2\ good\ indices\ in\ S_Y] \leq 8(\alpha l)^{-\beta k/2} + 2^{-c_1\delta n}.$$

Set $\delta' = \alpha/2$ and $\epsilon = \sqrt{8(\alpha l)^{-\beta k/2} + 2^{-c_1 \delta n}}$. We will now use Markov's inequality in the following way. Let $A_y = Pr_{\vec{X}\in D}[\vec{X}$ is not δ'-good in $S_y]$. Thus A_Y is a random variable determined by Y. From the above analysis, $E_Y[A_Y] \leq \epsilon^2$. Therefore, by Markov's inequality, $Pr_Y[A_Y \geq \epsilon] \leq \epsilon$. In other words, $Pr_Y[S_Y\ is\ \delta' - informative\ to\ within\ \epsilon] \geq 1 - \epsilon$. □

5.2.3 The Block-Wise Converter

We can now define our block-wise converter A. A has parameters n, the size of the input, $(l_1, \ldots, l_s)$, the lengths of the output blocks, and δ, the quality of the source.

1. INPUT: $x \in \{0,1\}^n$; $y_1 \in \{0,1\}^{2\log n}, \ldots, y_s \in \{0,1\}^{2\log n}$.
2. For $i = 1 \ldots s$ do $z_i \leftarrow B(x, y_i)$. (We use B with parameters $n, l_i, \delta/2$.)
3. OUTPUT: $z_1 \circ \cdots \circ z_s$.

Using essentially the same proof as in [NZ], we can show

Lemma 5.12 *Let $l_{min} = \min(l_1, \ldots, l_s)$, and suppose $k \leq (\delta' l_{min})^{1-\beta}$ and $l_1 + \ldots + l_s < \delta n/4$. Then A is an $(n, \delta, sk\log n, (l_1, \ldots, l_s), \delta', \epsilon)$ block-wise converter. Here $\delta' = c\delta/\log\delta^{-1}$ and $\epsilon = 2s(\delta' l_{min})^{-c'\beta k}$, where c' is from Lemma 5.8 and for $c = c'/4$.*

In order to make the error small for the block-wise converter, we set the length of the smallest block $l_s = n^{\Theta(1)}$. This gives:

Corollary 5.13 *Suppose $l_1 + \ldots + l_s < \delta n/4$ and for some constant $\beta > 0$, all $l_i \geq n^\beta(\log\epsilon^{-1})/\delta$. Then we can construct an efficient $(n, \delta, O(s\log\epsilon^{-1}), (l_1, \ldots, l_s), \delta', \epsilon)$ block-wise converter, where $\delta' = c\delta/\log\delta^{-1}$, c from Lemma 5.12.*

Proof. Choose $k = c(\log\epsilon^{-1}/\log n)$ for a large enough constant c. □

5.3 The Basic Extractor

It may help the reader, in the following discussion, to keep in mind the sample parameter values $\delta = n^{-1/4}$ and $\epsilon = 1/n$. We put the corresponding values of the parameters in square brackets. From the discussion about the parameters of Function C, all parameter lengths are determined by the smallest block length l_s. We pick $l_s \geq n^{1/4}$ and large enough so that the error from C is small, but small enough to ensure $s \geq \log_{9/8}(4\delta l_s/r)$. Therefore, by the remark after the description of C, $r_0 = 4\delta' l_0$ and the output m is large enough. These choices are summarized below. E is our main extractor, obtained by combining the converter A with the block-wise extractor C and invoking Lemma 5.3 using the following values for the parameters.

Parameters of E:

1. The parameters n, δ, and ϵ are given. We assume $1/\sqrt{n} \leq \delta \leq 1/2$ and for some constant $\beta > 0$, $2^{-\delta^2 n^{1-\beta}} \leq \epsilon \leq 1/n$. [e.g. $\delta = n^{-1/4}$, $\epsilon = 1/n$, $\beta = 1/4$]

2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.12. [$\delta' = \Theta(n^{-1/4}/\log n)$]

3. r is chosen to be the smallest integer such that $4 \cdot 2^{-r/16} \leq \epsilon/2$. Thus $r = \Theta(\log \epsilon^{-1}) = O(\delta^2 n^{1-\beta})$. [$r = \Theta(\log n)$]

4. $l_s = n^{\beta/2} r/\delta'$. We need $4\delta' l_s \geq r$ for Function C. Also, $l_s = O(\delta n^{1-\beta/2} \log \delta^{-1})$. [$l_s = \Theta(n^{3/8} \log^2 n)$]

5. Set $l_i = \max((r/4\delta')(9/8)^{s-i}, l_s)$.

6. s is chosen to be the largest integer such that $\sum_{i=1}^{s} l_i \leq \delta n/4$. Since $s = O(\log n)$, $sl_s = o(\delta n)$; this and $l_0 = \Theta(\delta n)$ imply $(r/4\delta')(9/8)^s = \Theta(\delta n)$. Therefore $s \geq \log_{9/8}(4\delta' l_s/r)$, as required for the function C. [$s = \Theta(\log n)$]

7. k is chosen so that $2s(\delta' l_s)^{-c' k\beta/2} \leq \epsilon/2$, where c' is from Lemma 5.8. Since $\delta' l_s \geq n^{\beta/2}$, $k = \Theta((\log \epsilon^{-1})/\log n)$. Also since $\delta' l_s \geq n^{\beta/2} \log \epsilon^{-1}$, $k \leq (\delta' l)^{1-\beta/2}$. [$k = \Theta(1)$]

8. The length of the second parameter to E is given by $t = s(k \log n) + r$. Thus $t = O((\log n) \log \epsilon^{-1})$. [$t = \Theta(\log^2 n)$]

9. The length of the output of E is given by $m = 4\delta' l_0 - r$. Thus $m = \Omega(\delta^2 n/\log \delta^{-1})$. [$m = \Theta(\sqrt{n}/\log n)$]

Thus, by Lemma 5.3, we deduce:

Theorem 5.14 *For any $\beta > 0$ and any parameters $\delta = \delta(n)$ and $\epsilon = \epsilon(n)$ with $1/\sqrt{n} \leq \delta \leq 1/2$ and $2^{-\delta^2 n^{1-\beta}} \leq \epsilon \leq 1/n$, there is an efficient $(n, \delta, t = O((\log n)\log \epsilon^{-1}), m = \Omega(\delta^2 n/\log \delta^{-1}), \epsilon)$-extractor.*

Then Lemma 2.5 gives

Corollary 5.15 *Any BPP algorithm can be simulated in $n^{O(\log n)}$ time using a δ-source, if the min-entropy of the R output bits is at least $R^{1/2+\epsilon}$ for any fixed $\epsilon > 0$.*

5.4 Bootstrapping to improve the extractor

We now use extractor E above recursively to get extractors which need fewer truly random additional bits, if δ is "much larger" than $n^{-1/2}$, say $\delta = n^{-1/4}$. In particular, we show that BPP can be simulated in polynomial time if $\delta^{\log^* R} R = R^{\Omega(1)}$, where R is the number of random bits requested from the δ-source. Thus, taking $R \geq n^2$, say, as long as $\delta > n^{-1/\log^* n}$, we can simulate BPP in polynomial time: a significant extension of the work of [Zu2]. All of this follows from Lemma 5.16, which shows how to get away with fewer truly random bits than E above needs, by bootstrapping. Basically, we replace one of the hash functions in the function C by the t bits needed for an application of the extractor. Therefore, instead of repeatedly hashing to build up an $n^{\Omega(1)}$-bit string, we need only build up a t-bit string and then apply the extractor.

Lemma 5.16 *Suppose we are given an efficient $(n, \delta, t_1, (n_0, l_1, l_2, \ldots, l_{s-1}), \delta', \epsilon_1)$ block-wise converter A, an efficient $((l_1, \ldots, l_{s-1}), \delta', t_2, m_0, \epsilon_2)$ block-wise extractor C, and an efficient $(n_0, \delta', t_0 = m_0, m, \epsilon_3)$-extractor E. Then we can construct an efficient $(n, \delta, t_1 + t_2, m, \epsilon_1 + \epsilon_2 + \epsilon_3)$-extractor.*

Proof. Use A to add t_1 bits Y_1 and output a block-wise δ'-source with blocks $X_0, X_1, \ldots, X_{s-1}$ with lengths $n_0, l_1, \ldots, l_{s-1}$. Use C to add t_2 bits Y_2 and convert $X_1, \ldots, X_{s-1}$ into a nearly uniform string Y_0 of length $m_0 = t_0$. As in the proof of Lemma 5.4, the distribution of (X_0, Y_0, Y_1, Y_2) is within $\epsilon_1 + \epsilon_2$ of some distribution $D \times U$, where D is a δ'-source and U is the uniform distribution on $t_0 + t_1 + t_2$-bit strings. Therefore, by the extractor property for E, $(E(X_0, Y_0), Y_1, Y_2)$ is quasi-random to within $\epsilon_1 + \epsilon_2 + \epsilon_3$. □

Corollary 5.17 *Suppose n, δ, and ϵ are such that $1/\sqrt{n} \leq \delta \leq 1/2$ and for some constant $\beta > 0$, $2^{-\delta^2 n^{1-\beta}} \leq \epsilon \leq 1/n$. Set $n_0 = \delta n/8$ and $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.12. Then given an efficient $(n_0, \delta', t = u \log \epsilon^{-1}, m, \epsilon')$-extractor for $t \leq c'\delta n/\log n$ for a sufficiently small constant c', we can construct an efficient $(n, \delta, O((\log u)(\log \epsilon^{-1})), m, \epsilon + \epsilon')$-extractor.*

Proof. For $s = O(\log u)$, we use an $(n, \delta, O((\log u) \log \epsilon^{-1}), (n_0, l_1, l_2, \ldots, l_{s-1}), \delta', \epsilon/2)$ block-wise converter defined in Section 5.2, and an $((l_1, \ldots, l_{s-1}), \delta', O(\log \epsilon^{-1}), t, \epsilon/2)$ block-wise extractor defined in Section 5.1. Note that we have $n_0 + l_1 + l_2 + \ldots + l_{s-1} < \delta n/4$ as needed for Lemma 5.12. This inequality follows from $l_1 + l_2 + \ldots + l_{s-1} < s \cdot t \leq c' \cdot O(\delta n)$. Now apply Lemma 5.16. □

Let $\log^{(k)}$ denote the logarithm iterated k times. We can now show:

Theorem 5.18 *For any $\beta > 0$ and any parameters $\delta = \delta(n)$, $\epsilon = \epsilon(n)$, and $k = k(n)$ with $n^{-1/k} \leq \delta \leq 1/2$ and $2^{-\delta^k n^{1-\beta}} \leq \epsilon \leq 1/n$, there is an efficient $(n, \delta, t = O((\log^{(k-1)} n) \log \epsilon^{-1}), m = \Omega(\delta^k n/(\log \delta^{-1})^{2k-2}), \epsilon)$-extractor. For the value $k = \log^* n - 1$, this gives an efficient $(n, \delta, t = O(\log \epsilon^{-1}), m = \delta^{\log^* n} n, \epsilon)$-extractor for $\delta \geq n^{-1/2\log^* n}$ and $2^{-\delta^{\log^* n} n^{1-\beta}} \leq \epsilon \leq 1/n$.*

Proof. Apply Corollary 5.17 repeatedly k times. Letting $m(n, \delta)$ denote the output length of the current extractor as a function of the input n and the quality δ, we see that each application of Corollary 5.17 causes the output length to decrease by a factor of $m(n_0, \delta')/m(n, \delta)$, which in our case is $\Theta(\delta/\log^2 \delta^{-1})$. $\square$

Corollary 5.19 *For any constant $\epsilon > 1 - 1/(k+1)$, any BPP algorithm can be simulated using a δ-source with min-entropy R^ϵ in time $n^{O(\log^{(k)} n)}$. For $\epsilon > 1 - 1/(2\log^* n)$, any BPP algorithm can be simulated in polynomial time.*

Remark: By applying random walks on expanders instead of k-wise independence, as in Lemma 5.7, for constant δ we can achieve an $(n, \delta, t = O(\log \epsilon^{-1}), m = \delta^{\log^* n} n, \epsilon)$-extractor for $2^{-\delta^{2\log^* n} n} \le \epsilon \le 1/n$ and n large enough. For smaller values of ϵ we may also use Lemma 3.2.

6 Simulating RP using a δ-source with min-entropy R^ϵ

We now show that RP can be simulated in $n^{O(\log n)}$ time with a δ-source on R bits whose min-entropy is R^ϵ, for *any* fixed $\epsilon > 0$ (compare with Corollary 5.15); here, as usual, n is the length of the input to the RP language. To simulate RP, it suffices to build an extractor which, on input t truly random bits and n bits from a source with min-entropy n^ϵ, outputs an n^α-bit string that is quasi-random to within β, for any fixed $\alpha > 0$, $\beta < 1$ (since we can initially make the number of accepting strings to be some constant strictly greater than an $1 - \beta$ fraction of the total sample space, by running the RP algorithm on independent random strings). We produce a constant number of strings, the distribution of at least one of which is quasi-random to within β; this suffices since we may then try the RP algorithm on all of these strings. We need two simple lemmas first.

Lemma 6.1 *Given a source outputting an R-bit string X with associated distribution D, partition $\{1, 2, \ldots, R\}$ into any two sets S_1 and S_2. Let X_1 and X_2 be the restrictions of X to S_1 to S_2, respectively, D_1 the distribution induced on S_1, $D_2(\cdot|y)$ the distribution of X_2 conditional on $X_1 = y$. If $Pr[D(X) \le 2^{-\ell}] \ge p$, then either $Pr[D_1(X_1) \le 2^{-\ell/2}] \ge p/2$ or $Pr[D_2(X_2|X_1) \le 2^{-\ell/2}] \ge p/2$.*

Proof. $D(X) = D_1(X_1) \cdot D_2(X_2|X_1)$ and hence, $Pr[D(X) \le 2^{-\ell}] \le Pr[D_1(X_1) \le 2^{-\ell/2}] + Pr[D_2(X_2|X_1) \le 2^{-\ell/2}]$. Thus, at least one of the two probabilities on the right-hand-side must be at least $p/2$. $\square$

Lemma 6.2 *Suppose a distribution D on a finite set S is such that $Pr[D(X) \le 2^{-\ell}] \ge p$ for a random $X \in S$ drawn according to D. Then assuming that $|S|2^{-\ell} \ge 1$ (which is necessary and sufficient for the existence of a distribution with min-entropy ℓ on S), D is within $1 - p$ of a source with min-entropy ℓ.*

Proof. Let $Y = \{y \in S : D(y) > 2^{-\ell}\}$. Clearly, $D(Y) = Pr[D(X) > 2^{-\ell}] \le 1 - p$, by assumption. Consider a distribution D' obtained by altering D such that $\forall y \in Y$, $D'(y) = 2^{-\ell}$; this is done by increasing the probabilities $D(z)$ for $z \in S - Y$ in some way.

Now the condition $|S|2^{-\ell} \geq 1$ guarantees a way of doing this in such a way that $\forall z \in S - Y$, $D'(z) \leq 2^{-\ell}$; thus D' has min-entropy ℓ. Clearly, $\|D - D'\| = D(Y) - |Y|2^{-\ell} \leq D(Y) \leq 1 - p$. $\square$

We reduce the case of building an extractor for a δ-source on R-bit strings with min-entropy R^ϵ for an arbitrary (but fixed) $\epsilon > 0$ to the case for some fixed $\epsilon' > 1/2$ (say $\epsilon' = 2/3$); we may then invoke Corollary 5.15 to handle this case.

We now show how to extract a constant number of strings, one of which is quasi-random to within $1 - \Omega(1)$. Fix a δ-source S outputting R-bit strings, with min-entropy R^ϵ for some fixed $\epsilon > 0$; as usual let R be a sufficiently large polynomial in n. We may assume $r = 4\delta l_s$ in the function C, because a larger l_s was necessary only to reduce the error ϵ of the extractor.

As in Theorem 5.14, given a string x from S, we use pairwise independence (and $O(\log^2 R) = O(\log^2 n)$ truly random bits as before) to pick blocks $B_1, B_2, \ldots, B_k$ of length $l_s, (9/8)l_s, (9/8)^2 l_s, \ldots$ etc. but instead of stopping at $l_0 = \Omega(\delta R) = \Omega(R^\epsilon)$, we continue in this fashion until we arrive at a block B_k of length at least $R^{1-\epsilon/2}/2$. Thus, we get $O(\log n)$ blocks, and $\Theta(R^{1-\epsilon/2})$ bits altogether. We then run function C on $B_k \circ B_{k-1} \circ \cdots \circ B_1$ by using $O(\log n)$ purely random bits as before, to produce a final string s of length $\Omega(\delta' |B_k|) = \Omega(R^{\epsilon/2}/\log n)$. All this is as before, with the difference that this time we have picked many more blocks. By Lemma 6.1, there are two cases:

(a) Before picking each block, the remaining bits we had to choose from (from x) were within $1/2$ of a source with min-entropy $R^\epsilon/2$. If so, the string s output by our extractor is quasi-random to within $1/2 + o(1)$, by Lemma 6.2. The errors do not add, since the bits remaining at each step form a subset of the bits remaining at the previous step. Thus if D is a distribution where, before picking the last block, the remaining bits form a source with min-entropy at least ℓ, then before picking *each* block the remaining bits have min-entropy at least ℓ.

(b) If Case (a) does not hold, then if D_1 denotes the distribution on the $R^{1-\epsilon/2}$ bits X_1 that we picked, $Pr[D_1(X_1) \leq 2^{-R^\epsilon/2}] \geq 1/2$, by Lemma 6.1. In this case, we recursively run our procedure on these bits.

We do not know which case holds, so we must try both cases. If we repeat our procedure c times, we end up with $c + 1$ strings, at least one of which is quasi-random to within $1 - 2^{-c} + o(1)$. If $s(R, \epsilon)$ is the number of bits used by this extractor, then

$$s(R, \epsilon) = O(\log^2 R) + s(R^{1-\epsilon/2}, \frac{\epsilon(1 - o(1))}{1 - \epsilon/2}).$$

The initial condition, given by our extractor E, is, say, $s(R, 2/3) = O(\log^2 R)$. Thus, $s(R, \epsilon) = O(\log^2 R)$. Hence we get

Theorem 6.3 *Any RP algorithm can be simulated in $n^{O(\log n)}$ time using a δ-source, if the min-entropy of the R output bits is at least R^ϵ for any fixed $\epsilon > 0$.*

7 Applications and Open Questions

Our applications heavily rely on previous work involving these applications. It is sometimes helpful to view our results graph-theoretically, as in [Sip, San, CW]. We modify the definitions in [CW] to suit our needs.

Definition 7.1 *An (N, M, d, α, β)-disperser is a bipartite graph with N nodes on the left side, each with degree at most d, and M nodes on the right side, such that every subset of N^α nodes on the left side is connected to at least βM nodes on the right. By an efficient construction of either a disperser, we mean that given a node on the left side, its neighbor set can be found in poly$(\log N + \log M + d)$ time deterministically.*

As was implicit in [CW], disperser constructions are equivalent to RP and BPP simulations using a δ-source. We list the relevant implications for us below.

Lemma 7.2 *A simulator for RP using R bits from a δ-source and outputting d r-bit queries yields an efficient construction of a $(2^R, 2^r, dk, \delta, 1 - 2^{-k})$-disperser, for any positive integer k (representing the number of repetitions of the simulator).*

7.1 Time-Space Tradeoffs

Our first application is to time-space tradeoffs. Sipser defined the class Strong-RP [Sip]:

Definition 7.3 *$A \in$ Strong-RP if there is an RP machine accepting A using $q(n)$ random bits and achieving an error probability of at most $2^{-(q(n)-q(n)^\alpha)}$ for some fixed $\alpha < 1$.*

He then showed

Theorem 7.4 ([Sip]) *P equals Strong-RP or for some $\epsilon > 0$ and for any time bound $t(n) \geq n$, all unary languages in DTIME$(t(n))$ are accepted infinitely often in SPACE$(t(n)^{1-\epsilon})$.*

We would like to replace Strong-RP in the above theorem by RP. Note the relevance of δ-sources to Strong-RP:

Lemma 7.5 *Strong-RP equals RP if and only if RP can be simulated using a δ-source with min-entropy R^α for some $\alpha < 1$. (For the equivalence, we assume non-oblivious simulations, i.e., the simulation could be different for different languages.)*

Proof. Let $L \in RP$, and suppose M recognizes L using a δ-source with min-entropy R^α for some $\alpha < 1$. Then M errs on fewer than 2^{R^α} R-bit strings. Setting $q(n) = R$ shows that M is a Strong-RP machine recognizing L. Conversely, suppose Strong-RP equals RP, and again let $L \in RP$. Say M accepts L with error probability at most $2^{-(q(n)-q(n)^\alpha)}$ for some $\alpha < 1$. Then M errs on at most $2^{q(n)^\alpha}$ strings. Thus for a fixed β, $\alpha < \beta < 1$, M accepts L with error probability at most $2^{q(n)^\alpha - q(n)^\beta}$ if the random bits come from a δ-source with min-entropy R^β. $\square$

As we did not quite show that RP equals Strong-RP, substituting our result into his proof gives

Theorem 7.6 *$RP \subseteq \bigcap_k DTIME(n^{\log^{(k)} n})$ or, for any time bound $t(n) \geq n$, all unary languages in DTIME$(t(n))$ are accepted infinitely often in $\bigcap_k SPACE(t(n)^{1-1/\log^{(k)} n})$.*

7.2 Explicit Expanders and Related Problems

Our second application is to improve the expanders constructed in [WZ], and hence all of the applications given there. Call an N-vertex undirected graph N^δ-expanding if there is an edge connecting every pair of disjoint subsets of the vertices, of size N^δ each. In [WZ], such graphs with essentially optimal maximum degree $N^{1-\delta+o(1)}$ were constructed in polynomial time. They were used to explicitly construct some useful combinatorial structures, as mentioned in Section 1. All of these results are optimal to within factors of $N^{o(1)}$. In [WZ], these $N^{o(1)}$ factors were $2^{(\log N)^{2/3+o(1)}}$. Our results improve these $N^{o(1)}$ factors to $2^{(\log N)^{1/2+o(1)}}$.

We first borrow the following lemma from the final version of [WZ].

Lemma 7.7 ([WZ]) *(i) If there is an $(n, \delta, t, m, 1/4)$-extractor computable in linear space, then there is an N^δ-expanding graph on $N = 2^n$ nodes with maximum degree $N2^{1+2t-m}$ constructible in Logspace.*
(ii) Fix positive integers n and k. Suppose that for each $\delta \in [\eta, 1]$ we are given an efficient $(n, \delta, t(\delta), m(\delta), \epsilon(\delta))$-extractor, where t and ϵ are non-increasing functions of δ. Let $f(\delta) = m(\delta)/(\delta n)$. Let $r = \ln(\delta/\eta)/f(\eta)$ or, if f grows at least linearly (i.e., $f(c\delta) \geq cf(\delta)$), let $r = 2/f(\eta)$. Then we can construct an efficient $(n, \delta, r \cdot t(\eta), (\delta - \eta)n - k, r(\epsilon(\eta) + 2^{-k}))$-extractor.

We can now prove our improved construction:

Theorem 7.8 *There is a polynomial-time algorithm that, on input N (in unary) and δ, where $0 < \delta = \delta(N) < 1$, constructs N^δ-expanding graphs on N nodes with maximum degree $N^{1-\delta}2^{(\log N)^{1/2+o(1)}}$.*

Proof. Assume without loss of generality that N is a power of 2, with $N = 2^n$. Set $\eta = (\log^3 n/n)^{1/2}$, $\epsilon = 1/n$, and $k = \log n$. If $\delta < 2\eta$, then the complete graph satisfies the theorem. Otherwise, apply Lemma 7.7(ii) to the extractor given by Theorem 5.14 to build an

$$(n, \delta, t = O(\log^2 n \log \eta^{-1}/\eta), m = (\delta - \eta)n - \log n, \epsilon = O(1/\eta n)) - \text{extractor}.$$

Then Lemma 7.7(i) gives an N^δ-expanding graph with maximum degree $N^{1-\delta}2^{O(n\eta)}$, i.e., $N^{1-\delta}2^{(\log N)^{1/2+o(1)}}$. □

7.3 The hardness of approximating NP-hard problems

Our third application is to the hardness of approximating $\log\log\omega(G)$, where $\omega(G)$ is the clique number of G. In [Zu2], it was shown that if $N\tilde{P} \neq \tilde{P}$, then approximating $\log\omega(G)$ to within any constant factor is not in $\tilde{P}$ (recall that $\tilde{P}$ denotes quasi-polynomial time). In [Zu3], a randomized reduction was given showing that any iterated log is hard to approximate; in particular if $N\tilde{P} \neq ZP\tilde{P}$, then approximating $\log\log\omega(G)$ to within a constant factor is not in $co-R\tilde{P}$. This used the fact that with high probability, certain graphs are dispersers. The disperser implied by our RP construction is almost as good. This makes the last reduction above deterministic, with a slight loss of efficiency: if $N\tilde{P} \not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. We now present a lemma, which is implicit in the results of [Zu2, Zu3] on the hardness of approximation:

Lemma 7.9 ([Zu2, Zu3]) *Suppose there is an explicit construction of an $(N, n^{O(1)}, d, \delta, 1/2)$-disperser, for any integer n. For some pair of functions $g_1, g_2 : \Re^+ \to \Re^+$ such that $g_1(y) \leq g_2(y)$ for all $y \in \Re^+$ and for any input graph G, suppose a number $h(G) \in [g_1(\omega(G)), g_2(\omega(G))]$ can be computed in $\tilde{P}$. Then if $g_1(N) > g_2(N^\delta)$, we have $NP \subseteq poly(N, 2^d)$.*

Theorem 7.10 *If $N\tilde{P} \not\subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$, then approximating $\log\log\omega(G)$ to within any constant factor is not in $\tilde{P}$. In other words, if we can compute, for some fixed $t > 1$, a number in the range*

$$[2^{(\log\omega(G))^{1/t}}, 2^{(\log\omega(G))^t}],$$

in $\tilde{P}$, then $N\tilde{P} \subseteq DTIME(2^{(\log n)^{O(\log\log n)}})$.

Proof. For any fixed $\epsilon \in (0, 1]$, Theorem 6.3 in conjunction with Lemma 7.2, implies that an $(N, n^{O(1)}, d, \delta, 1/2)$-disperser is efficiently constructible in the notation of Lemma 7.9, where $N = 2^{(\log n)^{O(\log\log n)}}$, $d = (\log n)^{O(\log\log n)}$, and $\delta = (\log N)^{\epsilon-1}$. Thus, by taking $\epsilon < 1/t^2$, $g_1(y) = 2^{(\log y)^{1/t}}$ and $g_2(y) = 2^{(\log y)^t}$, we invoke Lemma 7.9 to conclude that NP and hence $N\tilde{P}$, by a simple padding argument, is contained in $DTIME(2^{(\log n)^{O(\log\log n)}})$. □

The obvious open questions left open by this work are to simulate RP and BPP using δ-sources with min-entropy R^ϵ, for any fixed $\epsilon > 0$. As mentioned earlier, the former problem has been solved recently [SSZ]. By building on our work, substantial progress towards the second question has been made in [Ta-S], where an $R^{O(\log^{(k)} R)}$ algorithm is given for every fixed positive integer k. A still more interesting problem is to efficiently construct, for the class of δ-sources with min-entropy R^ϵ for any fixed $\epsilon > 0$, efficient extractors which use $O(\log R)$ purely random bits to extract as many as $(1 - o(1))R^\epsilon$ bits, which are quasi-random to within $R^{-\Theta(1)}$. It is easy to show that such extractors exist, non-constructively. In [Ta-S] it is shown that polylog(R) bits suffice to do this. Finally, we mention that ideas from this paper have been extended to give optimal extractors for constant-rate sources, as well as randomness-optimal samplers [Zu4].

Acknowledgement. We thank Avi Wigderson for helpful discussions.

References

[AG+] N. Alon, O. Goldreich, J. Håstad, and R. Peralta, "Simple Constructions of Almost k–wise Independent Random Variables," *Random Structures and Algorithms*, 3(3):289–303, 1992.

[BR94] M. Bellare and J. Rompel, "Randomness-Efficient Oblivious Sampling," *Proc. 35th IEEE Symposium on Foundations of Computer Science*, 1994, pp. 276-287.

[BL] M. Ben-Or and N. Linial, "Collective Coin Flipping," in *Advances in Computing Research 5: Randomness and Computation*, S. Micali, ed., JAI Press, 1989.

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

[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.

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

[CW] A. Cohen and A. Wigderson, "Dispersers, Deterministic Amplification, and Weak Random Sources," *Proc. 30th IEEE 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.

[FLW] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, Monte Carlo Simulations: Hidden Errors from "Good" Random Number Generators, *Physical Review Letters*, 69(23):3382–3384, 1992.

[GW] O. Goldreich and A. Wigderson, "Tiny Families of Functions with Random Properties: A Quality-Size Trade-off for Hashing," *Proc. 26th ACM Symposium on Theory of Computing*, 1994, pp. 574-583.

[HRD94] T.-s. Hsu, V. Ramachandran, and N. Dean, Parallel Implementation of Algorithms for Finding Connected Components, *Proc. DIMACS International Algorithm Implementation Challenge*, 1994, pp. 1-14.

[Hsu93] T.-s. Hsu, *Graph Augmentation and Related Problems: Theory and Practice*, PhD thesis, Department of Computer Sciences, University of Texas at Austin, October 1993.

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

[IZ] R. Impagliazzo and D. Zuckerman, "How to Recycle Random Bits," *Proc. 30th IEEE Symposium on Foundations of Computer Science*, 1989, pp. 248-253.

[KKL] J. Kahn, G. Kalai, and N. Linial, "The Influence of Variables on Boolean Functions," *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 68-80.

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

[LN] R. Lidl and H. Niederreiter, *Finite Fields*, Addison-Wesley, 1983.

[NN] J. Naor and M. Naor, "Small–Bias Probability Spaces: Efficient Constructions and Applications," *SIAM J. Comput.*, 22(4): 838-856, 1993.

[NZ] N. Nisan and D. Zuckerman, "More Deterministic Simulation in Logspace," *Proc. 25th ACM Symposium on Theory of Computing*, 1993, pp. 235-244. Final version to appear in *Journal of Computer and System Sciences.*

[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.

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

[SSZ] M. Saks, A. Srinivasan, and S. Zhou, "Explicit Dispersers with Polylogarithmic Degree," *Proc. 27th ACM Symposium on Theory of Computing*, 1995, pp. 479-488.

[Ta-S] A. Ta-Shma, "On extracting randomness from weak random sources," *Proc. 28th Annual ACM Symposium on Theory of Computing*, 1996.

[Va1] U. Vazirani, "Efficiency Considerations in Using Semi-Random Sources," *Proc. 19th ACM Symposium on Theory of Computing*, 1987, pp. 160-168.

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

[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 Slightly-Random Polynomial Time," *proc. 26th IEEE Symposium on Foundations of Computer Science*, 1985, pp. 417-428. See also U. Vazirani and V. Vazirani, "Random polynomial time is equal to semi-random polynomial time", Technical Report 88-959, Department of Computer Science, Cornell University, 1988.

[WZ] A. Wigderson and D. Zuckerman, "Expanders that Beat the Eigenvalue Bound: Explicit Construction and Applications," Technical Report CS-TR-95-21, Computer Science Dept., The University of Texas at Austin. Preliminary version appeared in *Proc. 25th ACM Symposium on Theory of Computing*, 1993, pp. 245-251.

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

[Zu2] D. Zuckerman, "Simulating BPP Using a General Weak Random Source," *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, 1991, pp. 79-89. Final version to appear in *Algorithmica* (copies available on request from the author).

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

[Zu4] D. Zuckerman, "Randomness-Optimal Sampling, Extractors, and Constructive Leader Election," *Proc. 28th Annual ACM Symposium on Theory of Computing*, 1996.