SIAM J. COMPUT.
Vol. 28, No. 4, pp. 1433–1459

COMPUTING WITH VERY WEAK RANDOM SOURCES*

ARAVIND SRINIVASAN† AND DAVID ZUCKERMAN‡

Abstract. We give an efficient algorithm to extract randomness from a very weak random source using a small additional number t of truly random bits. Our work extends that of Nisan and Zuckerman [*J. Comput. System Sci.,* 52 (1996), pp. 43–52] in that t remains small even if the entropy rate is well below constant. A key application of this is in running randomized algorithms using such a very weak source of randomness. For any fixed $\gamma > 0$, we show how to simulate RP algorithms in time $n^{O(\log n)}$ using the output of a δ-source with min-entropy R^γ. 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^γ} on each string. If $\gamma > 1/2$, our simulation also works for BPP; for $\gamma > 1 - 1/(k+1)$, 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, a key tool for extracting randomness from weak random sources.

Key words. derandomization, expander graphs, hashing lemmas, hardness of approximation, imperfect sources of randomness, measures of information, pseudorandomness, pseudorandom generators, randomized computation, time-space tradeoffs

AMS subject classifications. 60C05, 68Q15, 94A17

PII. S009753979630091X

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 pseudorandom number generators. Yet even in practice there are reports of algorithms giving quite different results under different pseudorandom generators; see, e.g., [FLW] for such reports on Monte-Carlo simulations, and [Hsu, HRD] 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 unclear whether 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

*Received by the editors March 25, 1996; accepted for publication (in revised form) December 11, 1997; published electronically April 20, 1999. 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, Minneapolis, MN, September 20–24, 1993. A preliminary version of this work appears in *Proc. IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 264–275.

http://www.siam.org/journals/sicomp/28-4/30091.html

†Dept. of Information Systems and Computer Science, National University of Singapore, Singapore 119260, Republic of Singapore (aravind@iscs.nus.edu.sg). The research of this author was done in parts at the Institute for Advanced Study, Princeton, NJ (partially supported by grant 93-6-6 of the Alfred P. Sloan Foundation), at the DIMACS Center, Rutgers University, Piscataway, NJ (partially supported by NSF grant STC91-19999 and by the New Jersey Commission on Science and Technology), and at the National University of Singapore (partially supported by National University of Singapore Academic Research Fund grant RP960620).

‡Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712 (diz@cs.utexas.edu). The research of this author was partially supported by NSF NYI grant CCR-9457799. Part of this work was done while this author was visiting The Hebrew University of Jerusalem and was supported by a Lady Davis Postdoctoral Fellowship.

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 taken in the study of weak random sources. The first is an attempt to describe a weak random source arising in practice. Thus 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. More precisely, we have the following.

DEFINITION 1.1 (see [SV]). *A semirandom source with parameter α outputs bits $X_1X_2,\ldots,X_R$, such that for all $i \le R$, and for all $x_1,\ldots,x_i \in \{0,1\}$,*

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

Simulations must work for all such sources; we do not want to assume any knowledge about the source, except that it is semirandom with the value of the parameter α being known. Santha and Vazirani 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 of simulating randomized algorithms with one semirandom source. Nevertheless, [VV] and [Va2] showed how to efficiently simulate all algorithms in RP and BPP, with one semirandom source, for any constant $\alpha > 0$.

Note that these simulations use $R = poly(n)$ bits from the semirandom source, where n denotes the length of the input to the RP or BPP machine; we shall let n and R denote "length of the input" and "number of bits requested," respectively, when discussing RP or BPP simulations in this paper.

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

DEFINITION 1.2 (see [CG]). *A blockwise δ-source outputs bits as blocks $Y_1,\ldots,Y_s$, where Y_i has length l_i, such that for all $i, y_1,\ldots,y_i$, $Pr[Y_i = y_i | Y_1 = y_1,\ldots,Y_{i-1} = y_{i-1}] \le 2^{-\delta l_i}$.*[1]

Note that δ and the block-lengths l_i are assumed to be known parameters of these sources. Later we will allow the l_i to vary, but for the first four sections we assume that all $l_i = l$. A semirandom 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 blockwise δ-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 does not appear to correspond as 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 (see [CW]).

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

[1] We modify the notation in [CG] to better conform with ours.

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. From an information-theoretic viewpoint, it is necessary for the R bits to have min-entropy only R^γ, for an arbitrary but fixed $\gamma > 0$ [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. Such an abstract model, wherein the "witness set" W can be *any* sufficiently large set, is introduced in Definition 2.3.) Can we achieve this information-theoretic 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^γ 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^γ. For $\gamma > 1/2$, our simulations also work for BPP and approximation algorithms. Moreover, for $\gamma > 1 - 1/(k+1)$, our simulations take time $n^{O(\log^{(k)} n)}$, giving a polynomial-time simulation for $\gamma > 1 - 1/(2\log^* n)$. (Here $\log^{(k)}$ denotes the logarithm 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 the Chor–Goldreich blockwise δ-source, as long as there are at least R^γ blocks and the min-entropy of the R bits is at least R^γ (i.e., $\delta R \geq R^\gamma$), for any fixed $\gamma > 0$. (The second condition may not be implied by the first condition if $\delta l < 1$.) Using $l = 1$, this gives a simulation of BPP and approximation algorithms for the Santha–Vazirani source with min-entropy R^γ. We also generalize Cohen and Wigderson's result on oblivious bit-fixing sources: it is not necessary for n^γ bits to be perfectly independent and uniform, but only "weakly independent" (see section 7). Our BPP simulations also work for approximation algorithms, such as the one for approximating the volume of a convex body [DFK].

Our BPP simulations are corollaries of something even stronger: extractor constructions. An extractor is an algorithm which extracts randomness from a weak source, using a small additional number t of truly random bits. We modify the definition given in [NZ] to account for general families of sources.

DEFINITION 1.5. *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 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. In particular, when $\mathcal{S}$ is the class of δ-sources on $\{0,1\}^n$, E is called an*

TABLE 1

Min-entropy n^γ for	t truly random bits for extractor	(N, M, d, K)-disperser construction
$\gamma > 0$	NA	$(2^n, 2^{\Omega(n^{\gamma/2}/\log n)}, n^{O(\log n)}, 2^{n^\gamma})$
$\gamma > 1/2$	$O(\log^2 n)$	$(2^n, 2^{n^{2\gamma-1}/\log n}, n^{O(\log n)}, 2^{n^\gamma})$
$\gamma > 2/3$	$O((\log n)\log\log n)$	$(2^n, 2^{n^{3\gamma-2}/\log^3 n}, n^{O(\log\log n)}, 2^{n^\gamma})$
$\gamma > 1 - 1/k$	$O((\log n)\log^{(k)} n)$	$(2^n, 2^{n^{k\gamma-k+1}/\log^{2k-1} n}, n^{O(\log^{(k)} n)}, 2^{n^\gamma})$
$\gamma > 1 - 1/(2\log^* n)$	$O(\log n)$	$(2^n, 2^{\sqrt{n}}, n^{O(1)}, 2^{n^\gamma})$

$(n, m, t, \delta, \epsilon)$-*extractor.*[2]

As in [NZ], we observe that an extractor that adds t random bits yields a BPP simulation taking time $2^t poly(n)$.

In the case when $\mathcal{S}$ is the class of δ-sources, it is often convenient to view the extractors graph theoretically, as in [Sip, San, CW]. Namely, construct a bipartite graph on $\{0,1\}^n \times \{0,1\}^m$, where $x \in \{0,1\}^n$ is adjacent to $z \in \{0,1\}^m$ if and only if $z = E(x, y)$ for some y. Then any set in $\{0,1\}^n$ of size at least $2^{\delta n}$ expands almost uniformly into $\{0,1\}^m$. In particular, it yields efficient constructions of graphs, which are called dispersers in [CW].

DEFINITION 1.6. *An (N, M, d, K)-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 K nodes on the left side is connected to at least $M/2$ nodes on the right. By an efficient construction of 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.*

LEMMA 1.7. *If there is an efficient $(n, m, t, \delta, \epsilon)$-extractor for $\epsilon \leq 1/2$, then there is an efficiently constructible $(2^n, 2^m, 2^t, 2^{\delta n})$-disperser.*

Our RP simulation also yields a disperser, although it is not an extractor. In Table 1 we summarize our results for δ-sources. The simulations and running times for the five entries of the table are as follows: The first entry implies an RP simulation, while the other four are for BPP. The respective running times are $n^{O(\log n)}$, $n^{O(\log n)}$, $n^{O(\log\log n)}$, $n^{O(\log^{(k)} n)}$, and $poly(n)$.

Just as extractors and dispersers for constant δ have important applications [NZ, WZ], so, too, do our results for subconstant δ. 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^γ for some $\gamma < 1$, 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

[2] To remember the five parameters, it may be helpful to note that the first three refer to lengths of inputs and outputs in (typically) decreasing order of length. The last two parameters refer to the quality of the sampler.

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 reasons for this are that our extractors require $n^{\Omega(1)}$ min-entropy from an n-bit source and that our extractors do not extract a sufficiently good fraction of the min-entropy.

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 under a slightly stronger assumption than $N\tilde{P} \neq ZP\tilde{P}$. 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 nonexplicit 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 $\tilde{P}$.

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 pseudorandom 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]. By using this with the ideas of [NZ], we get our first main extractor (see Theorem 5.7). Section 5.5 then introduces some new techniques for using such extraction procedures recursively; this helps improve the quality of our extractor when δ is not "too small."

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, and also serves in part as motivation for some of our techniques of sections 5 and 6. Sections 5 and 6 contain some of our main results: simulating BPP and RP algorithms using general weak sources with very low min-entropy. Section 7 uses the result of section 4 to generalize a result of [CW] on oblivious bit-fixing

sources. Applications of our results are presented in section 8. Section 9 concludes with some recent work that our work has led to, in part, and presents open questions. In the Appendix, we show some technical details that are largely borrowed from [NZ].

2. Preliminaries. We use capital letters to denote random variables, sets, distributions, and probability spaces; lowercase letters denote other variables. We often use a correspondence where the lowercase 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 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 chosen 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 chosen 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 $\|D_1 \times E_1 - D_2 \times E_2\| \leq \|D_1 - D_2\| + \|E_1 - E_2\|$. Since this inequality holds for any function, it also holds for random functions f. Next, 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.

We note that $s \geq r - O(\log r)$ random bits are required for this "abstract-RP" problem. For if s random bits are used and the algorithm can output $r^{O(1)}$ r-bit strings, then the number of possible outputs $2^s r^{O(1)}$ must exceed 2^{r-1}. As we can ask for at most $r^{O(1)}$ bits from the source, this is what gives the R^γ 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 blockwise δ-source, where we can request more bits with independence conditions from the first bits, 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 a 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 $1 - r^{-c}$.

Our BPP simulations are actually extractor constructions. 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 2.4. *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 $2^t n^{O(1)}$.*

Remark. In fact, as observed in [Zu2], for δ-sources we need only $\epsilon \leq 1/3$, say, to achieve even an exponentially small error for $\delta' > \delta$.

In order to make our statements cleaner and boost the size of the output from $\Omega(m)$ to m for reasonable m, we use the following lemma, which is a corollary of a lemma in [WZ].

LEMMA 2.5 (see [WZ]). *Suppose $m \leq \delta n/4$ and, for some integer k, $\epsilon \geq 2^{-\delta n/(5k)}$. If there is an efficient $(n, m/k, t, \delta/2, \epsilon/(2k))$-extractor, then there is an efficient $(n, m, kt, \delta, \epsilon)$-extractor.*

Finally, although we focus on extractors that run in polynomial time, all our extractors can actually be made to run in NC; see the remark at the end of section 3.

3. The Leftover Hash Lemma using fewer random bits. 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 $b_1, b_2 \in B$, $Pr[h(a_1) = b_1 \text{ and } h(a_2) = b_2] = 1/|B|^2$, where the probability is over h chosen uniformly at random from H. The Leftover Hash Lemma guarantees that the ordered pair $(h, h(x))$ is almost random.

LEFTOVER HASH LEMMA (see [ILL]). *Let $A \subseteq \{0,1\}^s, |A| \geq 2^t$. 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 way 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 the following.

DEFINITION 3.1 (see [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 for all $I \subseteq \{1, 2, \ldots, n\}, |I| \leq d$, for all $b_1, b_2, \ldots, b_{|I|} \in \{0,1\}$,*

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

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 a 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. A random element from F can be specified using $4(t-k)+O(\log s)$ random bits; given such a specification of any $g \in F$ using $4(t-k) + O(\log s)$ bits and given any $y \in \{0,1\}^s$, $g(y)$ can be computed in time $poly(s, t-k)$.*

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, where $\rho = (\epsilon^2 2^{2k} - 1)2^{-2t+2k}$; ρ is nonnegative since $\epsilon \geq 2^{1-k}$. (That is, F is a family of functions, where each element of F is a function that maps $\{0,1\}^s$ to $\{0,1\}^{t-2k}$.) 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. The lemma's claim about $g(y)$ being efficiently computable follows from the above mentioned fact that individual bits of any string in the support of a small-bias space can be computed efficiently.

We now show that the distribution of $(f, f(x))$ is quasi-random. We follow the proof of the Leftover Hash Lemma due to Rackoff (see [IZ]).

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

For the distribution D of $(f, f(x))$,

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

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,

$$\begin{aligned} cp(D) &= \frac{1}{|F|} Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2)] \\ &\leq \frac{1}{|F|}(Pr_{x_1,x_2 \in A}[x_1 = x_2] + Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2) | x_1 \neq x_2]) \\ &= 2^{-t}/|F| + \frac{1}{|F|} Pr_{x_1,x_2 \in A, f \in F}[f(x_1) = f(x_2) | x_1 \neq x_2] \\ &\leq 2^{-t}/|F| + \frac{1}{|F|} \max_{a_1 \neq a_2} Pr_{f \in F}[f(a_1) = f(a_2)]. \qquad (2) \end{aligned}$$

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

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

Thus, from (2), $cp(D) \leq (1+\epsilon^2)/(|F|2^{t-2k})$. Now, the rest of Rackoff's proof shows that if U is the uniform distribution on $F \times \{0,1\}^{t-2k}$, then $cp(D) \leq (1+\epsilon^2) cp(U) = (1+\epsilon^2)/(|F|2^{t-2k})$ implies the ϵ-quasi-randomness of D. This concludes the proof. □

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 stating 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, page 146 of [LN]). This allows our extractors to run in NC.

4. Simulating BPP using a blockwise δ-source with min-entropy R^γ. We now show how to simulate BPP using a Chor–Goldreich source with min-entropy R^γ for any fixed $\gamma > 0$; in fact, we build an extractor for Chor–Goldreich sources. Note that δ-sources are much weaker than these sources: in these sources, we know that each block has "a lot of randomness," even conditional on the previous blocks' values. In a δ-source, an adversary can, for instance, locate a lot of the randomness in certain positions that are unknown to us. Nevertheless, there are two reasons for the material of this section: to motivate our main results and the reasons for their difficulty and to construct an extractor that works with sources of min-entropy R^γ for *any* fixed $\gamma > 0$. Furthermore, this section will be useful in generalizing a result of [CW] on oblivious bit-fixing sources; see also section 7.

Note that since we are talking about extractors, we use the more usual symbol n (rather than R) in this section to denote the number of random bits requested from the source.

Suppose we are given a blockwise δ-source with $m\ (= n/l) = n^{\Omega(1)}$ blocks and with the min-entropy δn of the n bits being at least $n^{\Omega(1)}$. First note that we may always increase the block length by grouping successive blocks together. Suppose we want the output of E to be quasi-random to within n^{-c}. By choosing the block-length l appropriately, we may assume that the min-entropy of a block, $b = \delta l$, satisfies $b \geq 4(c+2)\log n$. We may also assume that $b = \Theta(\log n)$, since a blockwise δ-source is trivially a blockwise δ-source, if $\delta' < \delta$. We then choose 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 [CG].

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 a blockwise δ-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 that, 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 choosing 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. ∎

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 2.4, we get the following.

THEOREM 4.2. *For any fixed $\gamma > 0$, BPP can be simulated using a blockwise δ-source as long as there are at least n^γ blocks and if the min-entropy of the n bits is at least n^γ (i.e., $\delta n \geq n^\gamma$). 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 $n^{-\Omega(1)}$.*

5. Simulating BPP using a δ-source with min-entropy $R^{1/2+\epsilon}$. We first present some intuition and preliminary ideas behind our extractor construction.

5.1. Intuition and preliminaries. To construct an extractor for a given δ-source D outputting n-bit strings, we follow the same high-level approach as does [NZ]; the crucial differences arise from the fact that the work of [NZ] focused on constant δ, while we need to work with a δ that goes to zero (fairly quickly) with n. We then introduce additional new ideas to bootstrap this construction in subsection 5.5.

The high-level idea is to first convert the output of D into a blockwise δ'-source with suitable block-lengths $l_1, l_2, \ldots, l_s$; this construction is called a *blockwise converter* and is described in subsection 5.3. ($\delta' = \delta^{1-o(1)}$.) This construction is a slight modification of that in [NZ] and hence, many of the details are shown in the Appendix (one important difference is that the l_i values are chosen differently). We output $O(\log n)$ blocks, expending $O(\log n)$ truly random bits for each block; hence, we use $O(\log^2 n)$ random bits here in total. Once this is done, we need to extract quasi-random bits from such a blockwise δ'-source, and such a construction, called a *blockwise extractor*, is presented in subsection 5.2. Here is where we need to replace the application of the Leftover Hash Lemma by our improved version; only $O(\log n)$ truly random bits are needed for this task.

The reason why the above approach works only for min-entropy $n^{1/2+\Omega(1)}$ is as follows. In constructing the blockwise converter, our approach can ensure an output that is (close to) a blockwise δ'-source only if, in particular, $\sum_i l_i = O(\delta n)$. Given $O(\delta n)$ bits from a δ-source, one can intuitively expect these to have at most $O(\delta^2 n)$ bits of randomness; thus, since we wish to extract $n^{\Omega(1)}$ (quasi-)random bits, we need $\delta^2 n = n^{\Omega(1)}$, i.e., $\delta = n^{-1/2+\Omega(1)}$, which is equivalent to min-entropy $n^{1/2+\Omega(1)}$.

The above outline suggests an extractor for min-entropy $n^{1/2+\Omega(1)}$, using $O(\log^2 n)$ bits. How can this be improved (to $O(\log n)$ ideally)? We present a partial solution as a bootstrapping approach in subsection 5.5, which has a lesser randomness requirement for relatively "large" δ, e.g., for $\delta = n^{-1/3+\Omega(1)}$. An interesting open question is whether we can efficiently construct an appropriate blockwise converter that outputs a total of $O(n)$ bits using only $O(\log n)$ truly random bits; this will suffice to give a near-optimal extractor.

We now proceed formally. We 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 blockwise δ-source B. 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 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.

As in [NZ], our extractor first converts a δ-source into a blockwise δ-source with blocks of varying lengths and then extracts good bits from the blockwise δ-source. Unlike [NZ], we define these two intermediate constructions explicitly here.

DEFINITION 5.1. (i) *$E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^{l_1+\cdots+l_s}$ is an $(n, (l_1, \ldots, l_s), t, \delta, \delta', \epsilon)$ blockwise 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 blockwise 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 $((l_1, \ldots, l_s), m, t, \delta, \epsilon)$ blockwise extractor if, for x chosen from a blockwise δ-source with block-lengths $l_1, \ldots, l_s$ 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 5.2, implicit in [NZ], shows how to combine a blockwise converter and a blockwise extractor.

LEMMA 5.2. *Suppose we are given an efficient $(n, (l_1, \ldots, l_s), t_1, \delta, \delta', \epsilon_1)$ blockwise converter and an efficient $((l_1, \ldots, l_s), m, t_2, \delta', \epsilon_2)$ blockwise extractor. Then we can construct an efficient $(n, m, t_1 + t_2, \delta, \epsilon_1 + \epsilon_2)$-extractor.*

Proof. For the 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 5.2, we now focus on constructing an appropriate blockwise converter and a blockwise extractor; these are described in subsections 5.2 and 5.3, respectively.

5.2. A blockwise extractor. Lemma 4.1 gives a blockwise extractor. However, our blockwise converter will be useful only if the number of blocks s in the blockwise 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 blockwise 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 four 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. (To avoid details that may be distracting at this point, we discuss the reasons for the bounds on some of these parameters at the end of this subsection.) C works only 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, as explained in the last paragraph of this subsection,

$$c \log l_s \leq r \leq 3\delta l_s + O(\log l_s) \leq 4\delta l_s \tag{3}$$

for a suitably large constant c.

We define $r_s = r$ and $r_{i-1}/r_i = 9/8$, and then the block lengths $l_i = \max(r_i/(4\delta), l_s)$. Then Lemma 3.2 ensures for each i a fixed family of hash functions $H_i = \{h : \{0,1\}^{l_i} \to \{0,1\}^{\delta l_i/2}\}$ with $|H_i| \leq 2^{r_i}$, 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 $i = s$ down to 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 $s \geq \log_{9/8}(4\delta l_s/r)$. Then, $r_0/(4\delta) = r(9/8)^s/(4\delta) \geq l_s$ and hence, by the definition of l_i, $r_0 = 4\delta l_0$.

LEMMA 5.3. *C is an $((l_1, \ldots, l_s), r_0 - r, r, \delta, 4 \cdot 2^{-r/16})$ blockwise 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 blockwise 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 choosing 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 $2^{1-r_{i+1}/16}$. □

We now explain (3). 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. Next, to understand the equations $l_i = \max(r_i/(4\delta), l_s)$, 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 blockwise converter in Lemma 5.5. Note that Lemma A.1 allows smaller errors for larger values of l.

5.3. A blockwise converter. Our blockwise 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 [BR], we can give a good extractor for a wider range of parameters. The results of this subsection are essentially taken from [NZ], with the only changes being this improved analysis of [BR]. Thus, in order to not obscure the main new ideas, we just present a sketch of the blockwise converter and leave the necessary details to the Appendix.

In order to define our blockwise converter, we first show how to extract one block from a δ-source. The way 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.

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. (In other words, each X_i is uniformly distributed in A_i, and any k of the X_i's are mutually independent.) Our (random) output is $S = \{X_1, \ldots, X_l\}$. Methods to construct such k-wise independent random variables using $k \log n$ random bits are well known; see, e.g., [ABI, Lub].

Extracting one block. The function B. B has two parameters: l, the size of the output, and k, the amount of independence used.

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 using k-wise independence, as described above.
3. OUTPUT (a vector in $\{0,1\}^l$): $x_{i_1}, \ldots, x_{i_l}$ (here x_j is the jth bit of x).

The next key lemma, Lemma 5.4, is proved in the Appendix.

LEMMA 5.4. *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 $\delta' = c\delta/\log \delta^{-1}$, $k \leq (\delta' l)^{1-\beta}$, and $\epsilon = (\delta' l)^{-c\beta k}$ for some sufficiently small positive constant c.*

5.3.1. The blockwise converter. We can now define our blockwise converter A. A has parameters $(l_1, \ldots, l_s)$, the lengths of the output blocks, and k, the amount of independence.

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

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

LEMMA 5.5. *Let $l_{min} = \min(l_1, \ldots, l_s)$, and suppose $k \leq (\delta' l_{min})^{1-\beta}$ and $l_1 + \cdots + l_s < \delta n/4$. Then A is an $(n, (l_1, \ldots, l_s), sk \log n, \delta, \delta', \epsilon)$ blockwise converter. Here $\delta' = c\delta/\log \delta^{-1}$ and $\epsilon = 2s(\delta' l_{min})^{-c'\beta k}$, where c' is from Lemma 5.4 and $c = c'/4$.*

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

COROLLARY 5.6. *Suppose $l_1 + \cdots + l_s < \delta n/4$ and, for some constant $\beta > 0$, all $l_i \geq n^{\beta}(\log \epsilon^{-1})/\delta$. Then we can construct an efficient $(n, (l_1, \ldots, l_s), O(s \log \epsilon^{-1}), \delta, \delta', \epsilon)$ blockwise converter, where $\delta' = c\delta/\log \delta^{-1}$, c from Lemma 5.5.*

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

5.4. Choosing parameters and 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 $\epsilon = 1/n$. The general parameter list for the extractor is given after Corollary 5.8 for reference.

From the discussion about the parameters of function C, all parameter lengths are determined by the smallest block-length l_s. We choose $l_s \geq n^{1/4}$ 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 blockwise extractor C and invoking Lemma 5.2 using the following values for the parameters.

Parameters of E for $\delta = n^{-1/4}$, $\epsilon = 1/n$, and $\beta = 1/4$.

1. The parameter n is given.
2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.5. Thus $\delta' = \Theta(n^{-1/4}/\log n)$.
3. r is chosen to be the smallest integer such that $4 \cdot 2^{-r/16} \leq \epsilon/2$. So, $r = \Theta(\log n)$.
4. $l_s = n^{\beta/2} r/\delta'$; thus $l_s = \Theta(n^{3/8} \log^2 n)$.
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 $\sum_{i=1}^{s} l_i \leq \delta n/4$; thus $s = \Theta(\log n)$.
7. k is chosen so that $2s(\delta' l_s)^{-c'k\beta/2} \leq \epsilon/2$, where c' is from Lemma 5.4. So, $k = \Theta(1)$.
8. The length of the second parameter to E is given by $t = s(k \log n) + r$. Thus $t = \Theta(\log^2 n)$.
9. The length of the output of E is $m = 4\delta' l_0 - r = \Theta(\sqrt{n}/\log n)$.

Thus, by Lemmas 5.2 and 2.5, we deduce the following.

THEOREM 5.7. *For any $\beta > 0$ and any parameters $\delta = \delta(n)$ and $\epsilon = \epsilon(n)$ with $1/\sqrt{n} \le \delta \le 1/2$ and $2^{-\delta^2 n^{1-\beta}} \le \epsilon \le 1/n$, there is an efficient $(n, m = \delta^2 n/\log \delta^{-1}, t = O((\log n)\log \epsilon^{-1}), \delta, \epsilon)$-extractor.*

Proof. Using Lemma 5.2 gives output $m = \Omega(\delta^2 n/\log \delta^{-1})$; by applying Lemma 2.5 we can improve this to $m = \delta^2 n/\log \delta^{-1}$ while increasing t by a constant factor. □

Then Lemma 2.4 gives Corollary 5.8.

COROLLARY 5.8. *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^γ for any fixed $\gamma > 1/2$.*

We now present the parameter list of E in full generality.

Parameters of *E*. General case.

1. The parameters n, δ, and ϵ are given. We assume $1/\sqrt{n} \le \delta \le 1/2$ and for some constant $\beta > 0$, $2^{-\delta^2 n^{1-\beta}} \le \epsilon \le 1/n$.
2. $\delta' = c\delta/\log \delta^{-1}$, where c is from Lemma 5.5.
3. r is chosen to be the smallest integer such that $4 \cdot 2^{-r/16} \le \epsilon/2$. Thus $r = \Theta(\log \epsilon^{-1}) = O(\delta^2 n^{1-\beta})$.
4. $l_s = n^{\beta/2} r/\delta'$. We need $4\delta' l_s \ge r$ for function C. Also, $l_s = O(\delta n^{1-\beta/2} \log \delta^{-1})$.
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 $\sum_{i=1}^{s} l_i \le \delta n/4$. 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 $s \ge \log_{9/8}(4\delta' l_s/r)$, as required for the function C.
7. k is chosen so that $2s(\delta' l_s)^{-c'k\beta/2} \le \epsilon/2$, where c' is from Lemma 5.4. Since $\delta' l_s \ge n^{\beta/2}$, $k = \Theta((\log \epsilon^{-1})/\log n)$. Also, since $\delta' l_s \ge n^{\beta/2} \log \epsilon^{-1}$, $k \le (\delta' l)^{1-\beta/2}$.
8. The length of the second parameter to E is given by $t = s(k \log n) + r$. Thus $t = O((\log n)\log \epsilon^{-1})$.
9. The length of the output of E is given by $m = 4\delta' l_0 - r$. Thus $m = \Omega(\delta^2 n/\log \delta^{-1})$.

5.5. 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 $R \ge n^2$, say, as long as $\delta > n^{-1/\log^* n}$, we can simulate BPP in polynomial time; this is a significant extension of the work of [Zu2]. All of this follows from Lemma 5.9, which shows how, by bootstrapping, to get away with fewer truly random bits than E above needs. Basically, we replace one of the hash functions in the function C by the t bits output by an extractor. This way, we replace truly random bits by quasi-random bits that we extract from the source itself (i.e., we use the source's own randomness to further extract more bits). 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 5.9. *Suppose we are given an efficient $(n, (n_0, l_1, l_2, \ldots, l_{s-1}), t_1, \delta, \delta', \epsilon_1)$ blockwise converter A, an efficient $((l_1, \ldots, l_{s-1}), m_0, t_2, \delta', \epsilon_2)$ blockwise extractor C, and an efficient $(n_0, m, t_0 = m_0, \delta', \epsilon_3)$-extractor E. Then we can construct an efficient $(n, m, t_1 + t_2, \delta, \epsilon_1 + \epsilon_2 + \epsilon_3)$-extractor.*

Proof. Use A to add t_1 bits Y_1 and output a blockwise 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 5.3, 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 $\epsilon_1 + \epsilon_2 + \epsilon_3$. □

Ideally, the extractor would add $O(\log \epsilon^{-1})$ truly random bits (for $\epsilon \leq 1/n$). The following corollary shows how an extractor using $u \log \epsilon^{-1}$ truly random bits can be improved to one using $O((\log u)(\log \epsilon^{-1}))$ additional bits.

COROLLARY 5.10. *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 5.5. Then, given an efficient $(n_0, m, t = u \log \epsilon^{-1}, \delta', \epsilon')$-extractor for $t \leq c'\delta n/\log n$ for a sufficiently small constant c', we can construct an efficient $(n, m, O((\log u)(\log \epsilon^{-1})), \delta, \epsilon + \epsilon')$-extractor.*

Proof. We first modify the blockwise extractor defined in subsection 5.2 so that its output is of length $t = u \log \epsilon^{-1}$. This requires only $s = O(\log u)$ blocks. More precisely, we define l_s and l_i as in subsection 5.4, but we choose s to ensure that the output length $r_0 - r = t$. Since $r = \Theta(\log \epsilon^{-1})$ and $r_{i-1}/r_i = 9/8$, this gives $s = O(\log u)$, as claimed. We therefore have an $((l_1, \ldots, l_s), t, O(\log \epsilon^{-1}), \delta', \epsilon/2)$ blockwise extractor.

We then use an $(n, (n_0, l_1, l_2, \ldots, l_{s-1}), O((\log u) \log \epsilon^{-1}), \delta, \delta', \epsilon/2)$ blockwise converter defined in subsection 5.3. Note that we have $n_0 + l_1 + l_2 + \cdots + l_s < \delta n/4$ as needed for Lemma 5.5. This inequality follows from $l_1 + l_2 + \cdots + l_s \leq s \cdot t \leq c' \cdot O(\delta n)$, and we can choose c' small enough. Now apply Lemma 5.9. □

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

THEOREM 5.11. *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 $(n, m = \delta^k n/(\log \delta^{-1})^{2k-3}, t = O((\log^{(k-1)} n) \log \epsilon^{-1}), \delta, \epsilon)$-extractor. For the value $k = \log^* n - 1$, this gives an efficient $(n, m = (\delta/\log^2 \delta^{-1})^{\log^* n} n, t = O(\log \epsilon^{-1}), \delta, \epsilon)$-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 5.10 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 5.10 causes the output length to decrease by a factor of $m(n_0, \delta')/m(n, \delta)$, which in our case is $\Theta(\delta/\log^2 \delta^{-1})$. Finally, use Lemma 2.5 to eliminate the Ω in front of the output m. □

COROLLARY 5.12. *For any constant $\gamma > 1 - 1/(k+1)$, any BPP algorithm can be simulated using a δ-source with min-entropy R^γ in time $n^{O(\log^{(k)} n)}$. For $\gamma > 1 - 1/(2 \log^* n)$, any BPP algorithm can be simulated using a δ-source in polynomial time.*

Remark. By applying random walks on expanders instead of k-wise independence, as in Lemma A.3, we can construct extractors for slightly smaller values of ϵ than given in Theorem 5.11: $\epsilon \geq 2^{-\delta^{2 \log^* n} n}$. However, this is not usually in the range of interest.

6. Simulating RP using a δ-source with min-entropy R^γ. Recall the RP simulation problem. We are given some fixed $\gamma > 0$, an error parameter $\kappa \in [0, 1)$, any r, and some hidden $W \subseteq \{0,1\}^r$ such that $|W| \geq 2^{r-1}$; we want to use a distribution on $\{0,1\}^R$ with min-entropy R^γ to produce a set of strings which intersects W with probability at least $1 - \kappa$. (Corollary 5.8 solves this for $\gamma > 1/2$; we now focus on an arbitrary fixed $\gamma > 0$.) Call this problem RPSIM(R, γ, κ, r). Since r will not change throughout our discussion (but R will), we let $T(R, \gamma, \kappa)$ denote its time complexity.

We shall focus on this problem for $R = R_0 \doteq r^{c(\gamma)}$, where the constant $c(\gamma)$ will be spelled out later; henceforth, R will denote an arbitrary integer in $[r^\gamma, R_0]$.

The difficulty in achieving any simulation with min-entropy less than $\sqrt{R}$ is that the basic extractor outputs a string of length less than $\delta^2 R$. One factor of δ is lost by the blockwise extractor, Lemma 5.3, and the other by the blockwise converter, Lemma 5.5. Our approach is to have the converter output a larger string with the hope of getting a larger blockwise source. If this works, we are done; if not, we show that the converter's output is a higher quality source than the original source. We can then proceed recursively.

This approach will lead to some problems with the error κ becoming too large. We handle this by using a remark in section 2.3 of [Zu2]. That remark implies the following useful inequality, which holds for any R, $\gamma' < \gamma$, and $\kappa < 1$:

$$T(R, \gamma, \kappa 2^{R^{\gamma'} - R^\gamma}) \leq T(R, \gamma', \kappa). \tag{4}$$

Here, it will be useful to think of κ as "large," i.e., close to 1. The bound (4) therefore says that we can get a high-quality solution for γ (i.e., the "error" $\kappa 2^{R^{\gamma'} - R^\gamma}$ is very low) as long as we can get even a rather low-quality solution (i.e., the error κ is "large") for an appropriate $\gamma' < \gamma$. Concretely, define $T_1(R, \gamma) \doteq T(R, \gamma, 1 - 1/\log^2 R)$ and $T_0(R, \gamma) \doteq T(R, \gamma, 1/\log^2 R_0)$ (the subscripts 0 and 1 refer to κ close to 0 and 1); recall that R denotes an arbitrary integer in $[r^\gamma, R_0]$. Then (4) shows, for instance, that

$$T_0(R, \gamma) \leq T_1(R, \gamma' = \gamma - 1/\log R_0). \tag{5}$$

Our approach will yield an algorithm upper bounding $T_1(R, \gamma')$ in terms of T_0, thus leading to a recurrence for T_0 via (5). Before that, we present some useful preliminaries.

6.1. Some useful results. Given random variables X and Y and elements x and y in the respective supports of X and Y, let $\mathcal{P}_{X,Y}(y|x)$ denote $Pr[(Y = y)|(X = x)]$. Given this, we can define $\mathcal{P}(Y|X)$ to be the random variable which, for all x and y in the respective supports of X and Y, takes on the deterministic value of $\mathcal{P}_{X,Y}(y|x)$ if $X = x$ and $Y = y$. Thus, for instance,

$$Pr_{X,Y}[\mathcal{P}(Y|X) > b] = \sum_{x,y:\ \mathcal{P}_{X,Y}(y|x) > b} Pr[(X = x) \wedge (Y = y)].$$

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 and S_2, respectively, and let D_1 be the distribution induced on S_1. If $Pr_X[D(X) \leq 2^{-\ell}] \geq p$, then for any $p' \in [0, p]$, either $Pr_{X_1}[D_1(X_1) \leq 2^{-\ell/2}] \geq p'$ or $Pr_{X_1,X_2}[\mathcal{P}(X_2|X_1) \leq 2^{-\ell/2}] \geq p - p'$.*

Proof. Given any $x \in \{0,1\}^R$, let x_{S_1} and x_{S_2} denote its restrictions to S_1 and S_2, respectively. Now,

$$Pr_X[D(X) \leq 2^{-\ell}] = \sum_{x \in \{0,1\}^R:\ D(x) \leq 2^{-\ell}} D(x). \tag{6}$$

For any $x \in \{0,1\}^R$, $D(x) = D_1(x_{S_1}) \cdot \mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1})$; thus, if $D(x) \leq 2^{-\ell}$, then $D_1(x_{S_1}) \leq 2^{-\ell/2}$ or $\mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1}) \leq 2^{-\ell/2}$. Thus,

$$\sum_{x\in\{0,1\}^R:\ D(x)\le 2^{-\ell}} D(x) \le \sum_{x\in\{0,1\}^R:\ D_1(x_{S_1})\le 2^{-\ell/2}} D(x) + \sum_{x\in\{0,1\}^R:\ \mathcal{P}_{X_1,X_2}(x_{S_2}|x_{S_1})\le 2^{-\ell/2}} D(x). \tag{7}$$

However, by definition, the first and second terms in the right-hand side are, respectively, $Pr_{X_1}[D_1(X_1) \le 2^{-\ell/2}]$ and $Pr_{X_1,X_2}[\mathcal{P}(X_2|X_1) \le 2^{-\ell/2}]$. The lemma now follows from (6) and (7), using the given assumption that $Pr_X[D(X) \le 2^{-\ell}] \ge p$. □

LEMMA 6.2. *Suppose random variables $U \in \{0,1\}^{r_1}$ and $V \in \{0,1\}^{r_2}$ are such that $Pr_{U,V}[\mathcal{P}(V|U) > 2^{-\ell}] \le p$. Then, if $r_2 \ge \ell$, there is a random variable $W \in \{0,1\}^{r_2}$ such that* (i) *for all u and w in the respective supports of U and W, $Pr_{U,W}[(W = w)|(U = u)] \le 2^{-\ell}$; and* (ii) *the distribution of $U \circ V$ is within p of the distribution of $U \circ W$.*

Proof. Fix any u in the support of U, and let D_u denote the distribution of V conditional on $U = u$. Let $V_u = \{v : \mathcal{P}_{U,V}(v|u) > 2^{-\ell}\}$. Consider a distribution D'_u obtained by altering D_u such that for all $v \in V_u$, $D'_u(v) = 2^{-\ell}$; this is done by increasing the probabilities $D_u(v')$ for $v' \in \{0,1\}^{r_2} - V_u$ in some way. Now the condition $r_2 \ge \ell$ guarantees a way of doing this such that for all $v' \in \{0,1\}^{r_2} - V_u$, $D'_u(v') \le 2^{-\ell}$; it is now immediate that we have satisfied requirement (i) of the lemma.

In the above process, the only strings $u \circ v$ whose probabilities were decreased were those such that $\mathcal{P}_{U,V}(v|u) > 2^{-\ell}$. Now, it is easily seen that for any two distributions D_1 and D_2 on the same set S,

$$\|D_1 - D_2\| = \sum_{a\in S: D_1(a)>D_2(a)} (D_1(a) - D_2(a)) \le \sum_{a\in S: D_1(a)>D_2(a)} D_1(a).$$

Thus, since $Pr_{U,V}[\mathcal{P}(V|U) > 2^{-\ell}] \le p$ by assumption, requirement (ii) of the lemma is proved. □

6.2. The algorithm. We now present an algorithm for RPSIM$(R, \gamma' = \gamma - 1/\log R_0, 1 - 1/\log^2 R, r)$; we will bound its running time $T_1(R, \gamma')$ in terms of T_0. Fix a δ-source outputting R-bit strings with min-entropy $R^{\gamma'}$; thus, $\delta = \delta(R, \gamma')$ is given by $\delta R = R^{\gamma'}$ (so $\delta = R^{\gamma'-1}$). We may assume that the number of bits used to describe a hash function in the function C is $r = 4\delta l_s$, because a larger l_s was necessary only to reduce the error ϵ of the extractor. Since $r = \Theta(\log R)$, we have $l_s = \Theta(R^{1-\gamma'}/\log R)$. We also set $k = 2$ in the function B, i.e., use pairwise independence; thus, the error parameter ϵ in the statement of Lemma 5.4 is at most $R^{-\alpha}$, where α is a positive constant that depends only on γ. We also let δ' denote $c(\delta/2)/\log(2/\delta) = \Theta(\delta/\log R)$, where c is from the statement of Lemma 5.4. The block-lengths l_i are given by $l_i = l_s(9/8)^{s-i}$; we defer the presentation of s for now, but just note here that s will be $\Theta(\log R)$.

Since $k = 2$, the function B from subsection 5.3 uses strings of length $2\log R$ to index l-element subsets of $\{1, 2, \ldots, R\}$. For $y \in \{0,1\}^{2\log R}$, denote this subset by $S(l, y)$. Given $x \in \{0,1\}^R$ and any $S \subseteq \{1, 2, \ldots, R\}$, let x_S denote the sequence of bits of x indexed by S. Thus $B_l(x, y) = x_{S(y,l)}$ (note that we are subscripting the function B by the output length l). Recall that the output of the blockwise converter A, with parameters $(l_1, \ldots, l_s)$ and $k = 2$, is $A(x, (y_1, \ldots, y_s)) = B_{l_1}(x, y_1) \circ B_{l_2}(x, y_2) \circ \cdots \circ B_{l_s}(x, y_s)$.

The following lemma will be crucial.

LEMMA 6.3. *Let $\vec{X}$ denote a random string drawn from the given δ-source. For each i, $1 \le i \le s$, at least one of the following holds:*

(P1) *There exist $y_1, y_2, \ldots, y_i \in \{0,1\}^{2\log R}$ such that the distribution of $B_{l_1}(\vec{X}, y_1) \circ \cdots \circ B_{l_i}(\vec{X}, y_i)$ is within $i(\epsilon + 1/(3s))$ of the distribution of a blockwise δ'-source with block-lengths $l_1, l_2, \ldots, l_i$; or*

(P2) *there exist $y_1, y_2, \ldots, y_{i-1} \in \{0,1\}^{2\log R}$ such that the distribution of $B_{l_1}(\vec{X}, y_1) \circ \cdots \circ B_{l_{i-1}}(\vec{X}, y_{i-1})$ is within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+l_2+\cdots+l_{i-1}}$ with min-entropy $R^{\gamma'}/2$.*

Proof. The proof is by induction on i. (P1) is true for the base case $i = 1$, by Lemma 5.4. We assume the lemma for $i = j \ge 1$ and prove for $i = j + 1$. If (P2) is true for $i = j$, so it is for $i = j + 1$; so we assume that (P1) is true for $i = j$ and prove the lemma for $i = j + 1$.

Let $y_1^*, y_2^*, \ldots, y_j^*$ be the values of $y_1, y_2, \ldots, y_j$ that make (P1) true for $i = j$. Let $S = S(y_1^*, l_1) \cup S(y_2^*, l_2) \cup \cdots \cup S(y_j^*, l_j)$, and let D_1 denote the distribution placed by the given δ-source on $\vec{X}_S$. Substituting $p = 1$, $p' = 1/(3s)$, $\ell = R^{\gamma'}$, $S_1 = S$, and $S_2 = \{1, 2, \ldots, n\} - S_1$ in Lemma 6.1, we see that one of two cases holds: (a) $Pr_{\vec{X}}[D_1(\vec{X}_S) \le 2^{-R^{\gamma'}/2}] \ge 1/(3s)$ or (b) $Pr_{\vec{X}}[\mathcal{P}(\vec{X}_{S_2} | \vec{X}_S) > 2^{-R^{\gamma'}/2}] \le 1/(3s)$.

If case (a) holds, we see by substituting $r_1 = 0$ in Lemma 6.2 that (P2) holds for $i = j + 1$, with $y_1^*, y_2^*, \ldots, y_j^*$ being the corresponding values of $y_1, y_2, \ldots, y_j$.

So, let us suppose case (b) holds. Then, it is easy to see that $Pr_{\vec{X}}[\mathcal{P}(\vec{X} | \vec{X}_S) > 2^{-R^{\gamma'}/2}] \le 1/(3s)$. So Lemma 6.2 shows that the distribution of $\vec{X}_S \circ \vec{X}$ is within $1/(3s)$ of the distribution of $\vec{X}_S \circ V$, where (i) $V \in \{0,1\}^R$; and (ii) for all x in the support of $\vec{X}_S$ and for all $v \in \{0,1\}^R$, $Pr[(V = v) | (\vec{X}_S = x)] \le 2^{-(\delta/2)R}$. Thus, by Lemma 5.4, there is at least one $y_{j+1}^* \in \{0,1\}^{2\log R}$ such that the distribution of $\vec{X}_S \circ \vec{X}_{S(y_{j+1}^*, l_{j+1})}$ is within $\epsilon + 1/(3s)$ of the distribution of $\vec{X}_S \circ V'$, where (i′) $V' \in \{0,1\}^{l_{j+1}}$; and (ii′) for all x in the support of $\vec{X}_S$ and for all $v' \in \{0,1\}^{l_{j+1}}$, $Pr[(V' = v') | (\vec{X}_S = x)] \le 2^{-\delta' l_{j+1}}$. This, combined with the inductive assumption that $y_1^*, \ldots, y_j^*$ are values of $y_1, \ldots, y_j$ that make (P1) true for $i = j$, shows that (P1) is also true for $i = j + 1$, with $y_1 = y_1^*, y_2 = y_2^*, \ldots, y_{j+1} = y_{j+1}^*$. □

Recall that if (P2) is true for i, it is also true for $i + 1$. Thus, substituting $i = s$ in Lemma 6.3 and noting that the min-entropy does not decrease if we add more bits, we deduce the following.

COROLLARY 6.4. *There exist $y_1, y_2, \ldots, y_s \in \{0,1\}^{2\log R}$ such that the distribution of $A(\vec{X}, (y_1, \ldots, y_s)) = B_{l_1}(\vec{X}, y_1) \circ B_{l_2}(\vec{X}, y_2) \circ \cdots \circ B_{l_s}(\vec{X}, y_s)$ is either* (a) *within $s\epsilon + 1/3 = 1/3 + o(1)$ of the distribution of a blockwise δ'-source with block-lengths $l_1, l_2, \ldots, l_s$; or* (b) *within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+l_2+\cdots+l_s}$ with min-entropy $R^{\gamma'}/2$.*

Recall that we want an algorithm for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$. Also recall that $l_i = l_s(9/8)^{s-i}$; we now choose $s = \Theta(\log R)$ as the largest integer such that $\sum_{i=1}^{s} l_i \le R^{1-\gamma'/2}$. We first apply our function C (we deterministically cycle through all the $R^{O(1)}$ possible choices for the random input seed for C) one by one on $\vec{X}_{S(y_1,l_1)} \circ \vec{X}_{S(y_2,l_2)} \circ \cdots \circ \vec{X}_{S(y_s,l_s)}$, for all the $R^{O(\log R)}$ possible choices for $(y_1, y_2, \ldots, y_s)$. Thus, if case (a) of Corollary 6.4 were true, at least one of the strings output would be quasi-random to within $1/3 + o(1)$. Note that all the output strings will have length $\Omega(\delta' l_1) = \Omega(R^{\gamma/2}/\log R)$. Thus, as long as this is at least r, the probability of at least one of the output strings hitting W is at least $1/2 - 1/3 - o(1) = 1/6 - o(1)$, which is much greater than the required $1 - (1 - 1/\log^2 R) = 1/\log^2 R$ for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$.

However, since we do not know if case (a) of Corollary 6.4 holds, we also have to consider the remaining possibility (case (b) of Corollary 6.4) that there exist $y_1, y_2, \ldots, y_s \in \{0,1\}^{2\log R}$ such that the distribution of $A(\vec{X}, (y_1, \ldots, y_s)) = B_{l_1}(\vec{X}, y_1) \circ B_{l_2}(\vec{X}, y_2) \circ \cdots \circ B_{l_s}(\vec{X}, y_s)$ is within $1 - 1/(3s)$ of a distribution on $\{0,1\}^{l_1+\cdots+l_s}$ with min-entropy $R^{\gamma'}/2$. To handle this possibility, we once again exhaustively consider all the $R^{O(\log R)}$ possible choices for $(y_1, y_2, \ldots, y_s)$; for each such choice we run RPSIM$(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}, 1/\log^2 R_0, r)$ on $A(\vec{X}, (y_1, \ldots, y_s))$. Thus, if case (b) of Corollary 6.4 holds, then by the definition of RPSIM, we will hit W with probability at least $1 - (1 - 1/(3s)) - 1/\log^2 R_0 = \Theta(1/\log R)$, which is again greater than the required $1/\log^2 R$ for RPSIM$(R, \gamma', 1 - 1/\log^2 R, r)$.

The total time taken is

$$R^{O(\log R)} + R^{O(\log R)} T\left(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}, 1/\log^2 R_0\right),$$
$$= R^{O(\log R)} T_0\left(R^{1-\gamma'/2}, \frac{\gamma' - 1/\log R}{1-\gamma'/2}\right).$$

Thus, noting that $\frac{\gamma' - 1/\log R}{1-\gamma'/2} = (\gamma - \Theta(1/\log r))/(1 - \gamma/2)$, the time complexity of this algorithm can be summarized as

$$T_1(R, \gamma') \leq R^{O(\log R)} T_0(R^{1-\gamma'/2}, (\gamma - \Theta(1/\log r))/(1 - \gamma/2)).$$

Combining with (5), we see that

$$T_0(R, \gamma) \leq R^{O(\log R)} T_0(R^{1-\gamma'/2}, (\gamma - \Theta(1/\log r))/(1 - \gamma/2)). \tag{8}$$

The termination condition for this recurrence given by our extractor E is, say, $T_0(R, 2/3) = r^{O(\log r)}$. Letting $\gamma_0 = \gamma > 0$ be the initial value of γ, the sequence of values taken by the second argument in (8) is given by $\gamma_{i+1} \geq (\gamma_i - \Theta(1/\log r))/(1 - \gamma_i/2)$. Since the $\Theta(1/\log r)$ term can be made sufficiently small relative to γ_0 by taking r large enough, it is not hard to prove by induction on i that $\gamma_i \geq \gamma_0/(1-\gamma_0/4)^i$. Hence, the termination condition $\gamma_i \geq 2/3$ is achieved after a constant number of iterations, for any given constant $\gamma_0 > 0$; thus, $T_0(R, \gamma) = r^{O(\log r)}$. It is also not hard to check that all the output strings have length $\Omega(R^{\gamma/2}/\log R)$. Thus, there are constants $c_1(\gamma)$ and $c_2(\gamma)$ such that as long as we choose R_0 such that $R_0^{\gamma/2}/\log R_0 \geq c_1(\gamma) r$, i.e., as long as $R_0 \geq c_2(\gamma)(r \log r)^{2/\gamma}$, then all the output strings will have length at least r, which suffices for the above process to work.

THEOREM 6.5. *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^γ for any fixed $\gamma > 0$. Correspondingly, there is an efficient construction of a $(2^n, 2^{\Omega(n^{\gamma/2}/\log n)}, n^{O(\log n)}, 2^{n^\gamma})$-disperser.*

7. 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 the following definition.

DEFINITION 7.1. *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,*

$$\alpha \leq Pr[X_i = 0 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}, X_{i+1} = x_{i+1}, \ldots, X_n = x_n] \leq 1 - \alpha.$$

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 semirandom source of [SV]: we look at a bit conditioned on all the other bits, not just on the previous bits. Indeed, it is not necessarily true that every bit of a semirandom source with parameter α has weak independence close to α.

THEOREM 7.2. *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 $\gamma > 0$, 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 $\sum_{i=1}^n \alpha_i \geq n^\gamma$, where $t = O(\log n)$ and $m = n^{\Omega(1)}$; the output of the extractor is $n^{-\Theta(1)}$-quasi-random. The extractor need only know the value γ and not the quantities α_i.*

Note that this is best possible: if the X_i's are independent with $Pr(X_i = 0) = \alpha_i$, then the entropy of $(X_1, X_2, \ldots, X_n)$ is $k \doteq \sum_{i=1}^n H(\alpha_i)$, where $H(x) = -x \log_2 x - (1-x)\log_2(1-x)$ is the usual binary entropy function, with $H(0) = H(1) \doteq 0$. If $\sum_{i=1}^n \alpha_i = \beta$, then k is maximized when each α_i equals β/n, by the concavity of H; so $k = O(\beta \log(n/\beta))$. Thus if $\beta = n^{o(1)}$, then the entropy of $(X_1, X_2, \ldots, X_n)$ is also $n^{o(1)}$; hence such an extractor construction would not be possible. Thus we indeed need $\sum_{i=1}^n \alpha_i \geq n^\gamma$, for some fixed $\gamma > 0$.

Proof. Although the min-entropy is at least n^γ, we do not know as much about the "location" of the "good bits" as we do for Chor–Goldreich sources. 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 $\sum_{i=1}^n \alpha_i = n^\gamma$, 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 $m = n^{\gamma/3}$ blocks $B_1, \ldots, B_m$ of length $l = n^{1-\gamma/3}$, 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 $E[Y_j] = \alpha_j l/n = \alpha_j/n^{\gamma/3}$ and hence, $E[W_i] = \sum_{j=1}^n E[Y_j] = n^{2\gamma/3}$. 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, $j \neq k$,

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

Thus, the variance $Var[W_i]$ of W_i is

$$\begin{aligned} Var[W_i] &= \sum_j (E[Y_j^2] - (E[Y_j])^2) + 2\sum_{j<k}(E[Y_j Y_k] - E[Y_j]E[Y_k]) \\ &\leq \sum_j (E[Y_j^2] - (E[Y_j])^2) \leq \sum_j E[Y_j^2] = \sum_j \frac{\alpha_j^2}{n^{\gamma/3}} \leq n^{2\gamma/3}/2. \end{aligned}$$

So, using Chebyshev's inequality, $Pr[W_i < n^{2\gamma/3}/2] \leq 2n^{-2\gamma/3}$ and hence, $Pr[(\exists i)W_i < n^{2\gamma/3}/2] \leq 2n^{-\gamma/3}$. Hence, with probability at least $1 - 2n^{-\gamma/3}$, 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 $\epsilon' + 2n^{-\gamma/3} = n^{-\Theta(1)}$, where $\epsilon' = n^{-\Theta(1)}$ is the amount of quasi-randomness introduced by the extractor of Lemma 4.1. □

8. Applications. Our applications rely heavily on previous work involving these applications. It is often helpful to use the graph-theoretic, or disperser, view of our results.

8.1. Time-space tradeoffs. Our first application is to time-space tradeoffs. Sipser defined the class strong-RP [Sip] as follows.

DEFINITION 8.1. *$A \in$ 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 the following.

THEOREM 8.2 (see [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 as follows.

LEMMA 8.3. *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 nonoblivious simulations; i.e., the simulation could be different for different languages.)*

Proof of Lemma 8.3. 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. □

As we did not quite show that RP equals strong-RP, substituting our result into Sipser's proof gives the following.

THEOREM 8.4. *$RP \subseteq \bigcap_k DTIME(n^{\log^{(k)} n})$ or, for any time bound $t(n) \geq n$, all unary languages in $DTIME(t(n))$ are accepted infinitely often in $\bigcap_k SPACE(t(n)^{1-1/\log^{(k)} n})$.*

8.2. Explicit expanders and related problems. Our second application is to improving the expanders constructed in [WZ], and hence all 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 lemmas from the final version of [WZ].

LEMMA 8.5 (see [WZ]). *If there is an $(n, m, t, \delta, 1/4)$-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.*

The next lemma shows how to modify an extractor so it extracts almost all the randomness of a δ-source. The intuition is that if x is output from a δ-source and the output of the extractor $E(x,y)$ has length $m=\beta n$, then the string $E(x,y)\circ x$ is close in distribution to a uniform m-bit string concatenated with a $(\delta-\beta)$-source. Thus, we can apply an extractor E' for a $(\delta-\beta)$-source with an independent t-bit string y', and output $E(x,y)\circ E'(x,y')$. The following is based on recursing on this idea.

LEMMA 8.6 (see [WZ]). *Fix positive integers n and k. Suppose that for each $\delta\in[\eta,1]$ we are given an efficient $(n,m(\delta),t(\delta),\delta,\epsilon(\delta))$-extractor, where t and ϵ are nonincreasing 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 $(n,(\delta-\eta)n-k,r\cdot t(\eta),\delta,r(\epsilon(\eta)+2^{-k}))$-extractor.*

We can now prove our improved construction.

THEOREM 8.7. *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 8.6 to the extractor given by Theorem 5.7 to build an

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

Then Lemma 8.5 gives an N^{δ}-expanding graph with maximum degree $N^{1-\delta}2^{O(n\eta)}$, i.e., $N^{1-\delta}2^{(\log N)^{1/2+o(1)}}$. □

8.3. 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 $\tilde{P}$.

Let quasi-poly(x) be shorthand for $2^{(\log x)^{O(1)}}$. We now present a lemma, which is implicit in the results of [Zu2, Zu3] on the hardness of approximation.

LEMMA 8.8 (see [Zu2, Zu3]). *Suppose there is an explicit construction of an $(N=N(n),n^{\Theta(1)},d=d(n),K=K(n))$-disperser, for all integers n. Let $g_1,g_2\colon[1,\infty)\to\Re^+$ be functions satisfying $g_1(y)\leq g_2(y)$ for all $y\in[1,\infty)$. Suppose that for any input graph G, a number $h(G)\in[g_1(\omega(G)),g_2(\omega(G))]$ can be computed in $\tilde{P}$. Then if $g_1(N)>g_2(K)$, we have $NP\subseteq DTIME($ quasi-poly $(N+2^d))$.*

THEOREM 8.9. *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 $\gamma \in (0,1]$, Theorem 6.5 implies that an $(N, n^{\Theta(1)}, d, K)$-disperser is efficiently constructible in the notation of Lemma 8.8, where $N = 2^{(\log n)^{O(1/\gamma)}}$, $d = (\log n)^{O(\log\log n)}$, and $K = 2^{(\log N)^{\gamma}}$. Thus, by taking $\gamma < 1/t^2, g_1(y) = 2^{(\log y)^{1/t}}$, and $g_2(y) = 2^{(\log y)^t}$, we invoke Lemma 8.8 to conclude that NP and hence $N\tilde{P}$, by a simple padding argument, is contained in $DTIME(2^{(\log n)^{O(\log\log n)}})$. □

9. Later work and open problems. Some of our main contributions—the improved Leftover Hash Lemma and its use in extractors—have served as building blocks for several recent results. First, Saks, Srinivasan, and Zhou [SSZ] have improved our RP simulation to $poly(n)$ time. Second, substantial progress on the BPP simulation question has been made by Ta-Shma [Ta-S], where an $R^{O(\log^{(k)} R)}$ algorithm is given for every fixed positive integer k. Third, ideas from this paper have been extended by Zuckerman to give optimal extractors for constant-rate sources, as well as randomness-optimal samplers [Zu4]. In an exciting new result, Andreev et al. have shown how to simulate BPP using δ-sources with min-entropy R^{γ} for any fixed $\gamma > 0$, in polynomial time [AC+].

An important open question is to efficiently construct, for the class of δ-sources with min-entropy R^{γ} for any fixed $\gamma > 0$, efficient extractors which use $O(\log R)$ purely random bits to extract as many as $(1 - o(1))R^{\gamma}$ bits, which are quasi-random to within $R^{-\Theta(1)}$. It is easy to show that such extractors exist nonconstructively. In [Ta-S] it is shown that polylog(R) bits suffice to do this. Constructing a near-optimal family of dispersers may be an interesting step in this direction.

See [Nis] for a survey of some of the recent results in this area.

Appendix. Details of the blockwise converter. A property of such k-wise independent random variables that we will require follows.

LEMMA A.1. *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 [BR].

LEMMA A.2. *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 $\alpha > 0$, $Pr[|Y - \mu| \geq \alpha] \leq 8((k\mu + k^2)/\alpha^2)^{k/2}$.*

Proof of Lemma A.1. Define the random variables Y_i to be 1 if and only if $X_i \in T$, and 0 otherwise. Then, $E[Y_i] = |T \cap A_i|/m$. Thus for $Y = \sum_{i=1}^{l} Y_i$, $E[Y] = \sum_{i=1}^{l} |T \cap A_i|/m = |T|/m \geq \delta l$. Setting $\alpha = \delta l/2$ in Lemma A.2 concludes the proof. □

We remark that for certain parameters of the extractor (e.g., if $\delta = \Omega(1)$), it may be better to use the following lemma from [NZ]. The parameters where this is useful are mostly uninteresting: $\epsilon = 2^{-\delta^2 n^{1-o(1)}}$.

LEMMA A.3 (see [NZ]). *There is an absolute constant $c > 0$ such that the following holds: 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 $T] \geq 1 - 2^{-k}$.*

Proof of Lemma 5.4**.** To prove Lemma 5.4, we proceed as in [NZ]. Fix a δ-source D. We need the following definitions that are relative to D.

DEFINITION A.4. *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 $p_i(\vec{x}) = 1/2$ ensures that exactly one of $x_i = 0$ and $x_i = 1$ is good, for a given prefix.

DEFINITION A.5. *$\vec{x}$ 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 α-good in $S] \geq 1 - \beta$.*

Denote by S_y the set of l indices chosen using the (k-wise independent) random bits $\vec{y}$, as described in section 5.3. 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 Lemma A.6, will clearly prove Lemma 5.4.

LEMMA A.6. *$Pr_{\vec{Y}}[S_Y$ is δ'-informative to within $\epsilon] \geq 1 - \epsilon$.*

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

$$Pr_{\vec{X} \in D}[\vec{X} \text{ is not } \alpha\text{-good}] \leq 2^{-c_1 \delta n}, \tag{9}$$

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 A.1 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 (9), it follows that

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

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 δ'-informative to within $\epsilon] \geq 1 - \epsilon$. □

Note that in the sources considered in section 4, we know that each block has "many" good bits; thus, since we know the block boundaries, working with the good bits was much easier there. Since we have no idea of the location of good bits in general δ-sources, we have to work much harder here.

Acknowledgments. We thank Avi Wigderson for helpful discussions, and the two referees for their detailed and helpful suggestions.

REFERENCES

[ABI] N. ALON, L. BABAI, AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, J. Algorithms, 7 (1986), pp. 567–583.

[AC+] A. E. ANDREEV, A. E. F. CLEMENTI, J. P. D. ROLIM, AND L. TREVISAN, *Weak random sources, hitting sets, and BPP simulations*, in Proc. 38th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 264–272.

[AG+] N. ALON, O. GOLDREICH, J. HÅSTAD, AND R. PERALTA, *Simple constructions of almost k–wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–303.

[BR] M. BELLARE AND J. ROMPEL, *Randomness-efficient oblivious sampling*, in Proc. 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 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, Greenwich, CT, 1989, pp. 91–115.

[Blu] M. BLUM, *Independent unbiased coin flips from a correlated biased source: A finite Markov chain*, Combinatorica, 6 (1986), pp. 97–108.

[CG] B. Chor and O. Goldreich, *Unbiased bits from sources of weak randomness and probabilistic communication complexity*, SIAM J. Comput., 17 (1988), pp. 230–261.

[CG+] B. Chor, O. Goldreich, J. Håstad, J. Friedman, S. Rudich, and R. Smolensky, *The bit extraction problem or t-resilient functions*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 396–407.

[CW] A. Cohen and A. Wigderson, *Dispersers, deterministic amplification, and weak random sources*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 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 (1991), pp. 1–17.

[FLW] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong, *Monte Carlo simulations: Hidden errors from "good" random number generators*, Phys. Rev. Lett., 69 (1992), pp. 3382–3384.

[GW] O. Goldreich and A. Wigderson, *Tiny families of functions with random properties: A quality-size trade-off for hashing*, Random Structures Algorithms, 11 (1997), pp. 315–343.

[HRD] T.-S. Hsu, V. Ramachandran, and N. Dean, *Parallel implementation of algorithms for finding connected components in graphs,* in Parallel Algorithms, 3rd DIMACS Implementation Challenge, October 17–19, 1994, DIMACS, Ser. Discrete Math. Theor. Comput. Sci. 30, Sandeep N. Bhatt, ed., AMS, Providence, RI, 1997, pp. 23–41.

[Hsu] T.-S. Hsu, *Graph Augmentation and Related Problems: Theory and Practice*, Ph.D. thesis, Department of Computer Sciences, University of Texas at Austin, Austin, TX, October 1993.

[ILL] R. Impagliazzo, L. Levin, and M. Luby, *Pseudo-random generation from one-way functions*, in Proc. 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 12–24.

[IZ] R. Impagliazzo and D. Zuckerman, *How to recycle random bits*, in Proc. 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 248–253.

[KKL] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proc. 29th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 68–80.

[LLS] D. Lichtenstein, N. Linial, and M. Saks, *Some extremal problems arising from discrete control processes*, Combinatorica, 9 (1989), pp. 269–287.

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

[Lub] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[NN] J. Naor and M. Naor, *Small–bias probability spaces: Efficient constructions and applications*, SIAM J. Comput., 22 (1993), pp. 838–856.

[Nis] N. Nisan, *Extracting randomness: How and why*, in Proc. IEEE Conference on Computational Complexity (formerly Structure in Complexity Theory), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 44–58.

[NZ] N. Nisan and D. Zuckerman, *Randomness is linear in space*, J. Comput. System Sci., 52 (1996), pp. 43–52.

[San] M. Santha, *On using deterministic functions in probabilistic algorithms*, Inform. Comput., 74 (1987), pp. 241–249.

[SV] M. Santha and U. Vazirani, *Generating quasi-random sequences from slightly random sources*, J. Comput. System Sci., 33 (1986), pp. 75–87.

[Sip] M. Sipser, *Expanders, randomness, or time versus space*, J. Comput. System Sci., 36 (1988), pp. 379–383.

[SSZ] M. Saks, A. Srinivasan, and S. Zhou, *Explicit OR-dispersers with polylogarithmic degree*, J. ACM, 45 (1998), pp. 123–154.

[Ta-S] A. Ta-Shma, *On extracting randomness from weak random sources*, in Proc. 28th Annual ACM Symposium on Theory of Computing, ACM, New York, 1996, pp. 276–285.

[Va1] U. Vazirani, *Efficiency considerations in using semi-random sources*, in Proc. 19th Annual ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 160–168.

[Va2] U. Vazirani, *Randomness, Adversaries and Computation*, Ph.D. thesis, University of California, Berkeley, CA, 1986.

[Va3] U. Vazirani, *Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources*, Combinatorica, 7 (1987), pp. 375–392.

[VV] U. Vazirani and V. Vazirani, *Random polynomial time is equal to slightly-random polynomial time*, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1985, pp. 417–428. See also U. Vazirani and V. Vazirani, *Random Polynomial Time is Equal to Semi-random Polynomial Time,* Tech. Report 88-959, Department of Computer Science, Cornell University, Ithaca, NY, 1988.

[WZ] A. Wigderson and D. Zuckerman, *Expanders that beat the eigenvalue bound: Explicit construction and applications*, Combinatorica, to appear. Also see Technical Report CS-TR-95-21, Computer Science Dept., The University of Texas at Austin, Austin, TX. Preliminary version appears in Proc. 25th Annual ACM Symposium on Theory of Computing, ACM, New York, 1993, pp. 245–251.

[Zu1] D. Zuckerman, *General weak random sources*, in Proc. 31st Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 534–543.

[Zu2] D. Zuckerman, *Simulating BPP using a general weak random source*, Algorithmica, 16 (1996), pp. 367–391.

[Zu3] D. Zuckerman, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.

[Zu4] D. Zuckerman, *Randomness-optimal oblivious sampling*, Random Structures Algorithms, 11 (1997), pp. 345–367.