Randomness is Linear in Space*

Noam Nisan † David Zuckerman ‡

Abstract

We show that any randomized algorithm that runs in space S and time T and uses $poly(S)$ random bits can be simulated using only $O(S)$ random bits in space S and time $T + poly(S)$. A deterministic simulation in space S follows.

Of independent interest is our main technical tool: a procedure which extracts randomness from a defective random source using a small additional number of truly random bits.

1 Introduction

The relative power of deterministic and randomized algorithms is a basic question in complexity theory. Despite much effort very little is known. In this paper we consider this question when the complexity measured is *space*. Is randomized-$space(S)$ stronger than deterministic-$space(S)$?

While several nontrivial deterministic simulations of randomized-space are known [2, 15, 16], this question is still completely open. No simulation of randomized-$space(S)$ is known which uses less than $O(S^2)$ deterministic space, a simulation which can be achieved by Savitch's theorem [17].

Indeed, from Savitch's proof it follows that a language accepted by a randomized-$space(S)$ machine using R random bits is also accepted by a deterministic-$space(S \log(R/S))$ machine. There is only one result that improves this bound for some R. Namely, Ajtai, Komlos, and Szemeredi showed that any randomized-$space(S)$ algorithm using only $O(S^2/\log S)$ random bits can be simulated deterministically in $space(S)$ [1]. In this paper we improve upon this result and give a deterministic simulation of algorithms using $poly(S)$ random bits.

What we obtain is a pseudo-random generator. Our generator converts $O(S)$ truly random bits to $poly(S)$ bits that look random to all $space(S)$ machines. The generator can be computed in space S and time polynomial in S. It is thus possible to reduce the number of random bits used by any $space(S)$ algorithm from $poly(S)$ to $O(S)$ without a large penalty in time or space. Our main theorem can be stated as:

Theorem 1 *Any randomized algorithm A that runs in space S and time T and uses $poly(S)$ random bits can be simulated using only $O(S)$ random bits in space S and time $T + poly(S)$. The distribution of the output of the simulation is within statistical distance of $exp(-S^{1-\gamma})$ from the distribution of the output of A. Here $S = S(n) \geq \log n$, $T = T(n) \geq n$, and $\gamma > 0$ is an arbitrary constant.*

---

*A preliminary version of this paper titled "More Deterministic Simulation in Logspace" appeared in the *25th ACM Symposium on Theory of Computing*, 1993, pp. 235-244.

†Institute of Computer Science, Hebrew University, Jerusalem. Supported by USA-Israel BSF grants 89-00126 and 92-00043 and by a Wolfson research award administered by the Israeli Academy of Sciences. Part of this research was done while the author visited IBM Almaden.

‡Dept. of Computer Sciences, The University of Texas at Austin, Austin, TX 78712. Most of this research was done while the author was affiliated with MIT and supported by an NSF Postdoctoral Fellowship, NSF Grant No. 92-12184 CCR, and DARPA Grant No. N00014-92-J-1799. Part of this research was done while the author visited The Hebrew University in Jerusalem, Princeton University through DIMACS, and the International Computer Science Institute in Berkeley.

If one only cares about space then $O(S)$ random bits can clearly be simulated deterministically by running through all possibilities for the random bits.

Corollary 1 *Any language accepted by a randomized Space(S) algorithm that uses only poly(S) random bits can be accepted deterministically in Space(S).*

For polynomial-time algorithms it is probably more natural to state our main result as:

Corollary 2 *Any randomized polynomial time algorithm running in space S can be simulated in polynomial time using only $O(S + n^{\alpha})$ random bits with statistical error $exp(-n^{\alpha'})$, for any constants $\alpha > \alpha' > 0$.*

Several classes of randomized algorithms run naturally in linear space and thus can be simulated using only a linear number of random bits. Examples include walks on "rapidly mixing Markov chains" (as in [12]) and random generation using the "rejection method." A particularly interesting example is uniform generation of prime numbers which, using this corollary, can be approximated (within small statistical distance) using a linear number of random bits (see [15] for more details).

We remark that the results of [15] imply that randomized space S polynomial-time algorithms may be simulated using $O(S \log n)$ random bits, so Corollary 2 is only interesting when $S = n^{\Omega(1)}$.

Our main technical tool is a construction of the following kind of function which we call an *extractor*. To motivate this, suppose we have a set $A \subset \{0,1\}^n$ with $|A| \geq 2^{\delta n}$, and suppose we have a random element from A. Thus, we have δn bits of randomness, but in an unusable form. Our aim is to extract from this distribution a nearly uniform distribution. To do this we will use a small additional number of truly random bits.

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

Definition 2 *Let $E : \{0,1\}^n \times \{0,1\}^t \rightarrow \{0,1\}^m$. E is called a (δ, ϵ)-extractor if for every δ-source D, the distribution of $E(x,y) \circ y$ induced by choosing x from D and y uniformly in $\{0,1\}^t$ is within statistical distance of ϵ from the uniform distribution (on $\{0,1\}^m \times \{0,1\}^t$.)*

The Leftover Hash Lemma of [10] [1] gives an extractor with $t > n$. Our main construction is an extractor with $t << n$.

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

Note that the upper bounds on δ and ϵ are given only to make our expressions simpler. In fact, for smaller δ and ϵ, it is more difficult to construct the extractor, so t is larger and m is smaller. For our application we use δ equal to a constant and $\epsilon = 1/poly(n)$. We therefore advise the reader to ignore the dependence on δ in the first reading.

We also show a lower bound on the quality of any extractor: $t = \Omega(\log \epsilon^{-1} + \log n)$ for constant $\delta < 1$. Thus, ignoring the dependence on δ, the size of t is within an $O(\log^2 n)$ factor of optimal. We can shave off another factor of $\log n$ by using expander graphs. This improvement is not needed for our application so we do not use it here.

In fact, one virtue of our construction is that it is elementary: the only tools we use are the Leftover Hash Lemma and k-wise independence. Our use of these tools is based on the methods of [21]. Indeed, the extractor can be viewed as a simplification and extension of the algorithms in [21], although in one sense the extractor is weaker (see below).

One may think of extractors in various ways and contexts. We briefly sketch some of these below.

Hashing lemmas:

One may view the y's as names of hash functions $h_y : \{0,1\}^n \rightarrow \{0,1\}^m$, by $h_y(x) = E(x,y)$. In this context we obtain very small families of hash functions which still have good properties; specifically, they satisfy a lemma similar to the Leftover Hash Lemma [10].

[1] The term "Leftover Hash Lemma" was coined in [11], which gives a proof due to Rackoff with improved constants.

Expansion:

An extractor E defines in a natural way a bipartite graph on $\{0,1\}^n \times \{0,1\}^m$, where $x \in \{0,1\}^n$ is connected to $z \in \{0,1\}^m$ if there exists $y \in \{0,1\}^t$ such that $E(x,y) = z$. As in the constructions of [20, 21], this graph has good expansion properties, which are *better than what can be obtained using eigenvalue methods*. These ideas are further used in [19].

Weak random sources and deterministic amplification:

Given an extractor and the first parameter x to it, an algorithm may go over all the possible values of y. It is not difficult to see that this can be used to simulate BPP using a δ-source [20, 21], or to do "deterministic amplification" [11, 8]. For the value of t we obtain, however, the running time of this simulation will not be polynomial but only quasi-polynomial. On the other hand, our simulation satisfies a stronger requirement: it truly approximates the acceptance probability of a BPP machine. The result of [3] is similar in this regard, but does not yield an extractor.

2 Definitions and Notation

Throughout this paper, we use the convention that capital letters denote random variables, sets, distributions, and probability spaces; other variables will be in small letters. Exceptions are R and R', denoting numbers of random bits, and S, our space-bound. We often use a correspondence where the small letter denotes an instantiation of the capital letter, e.g. $\vec{x}$ might be a particular input and $\vec{X}$ the random variable being uniformly distributed over all inputs.

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

All logarithms are meant to the base 2.

Distance between Distributions

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

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

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

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

δ-sources

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

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

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

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

where the vector of random variables $X_1 \ldots X_k$ is chosen according to distribution D. A block-wise δ-source is the same as the PRB-source of [6] except that here the block length is allowed to vary.

3 Fooling Randomized Space-Bounded Machines

Our goal in this section is to use an extractor to construct a pseudo-random generator for space bounded computation.

Definition 3 *A generator $G : \{0,1\}^n \rightarrow \{0,1\}^m$ is called a pseudo-random generator for space S with parameter ϵ if, for every randomized space S algorithm A and every input to it,*

$$|Pr[A(y) \text{ accepts}] - Pr[A(G(x)) \text{ accepts}]| \leq \epsilon$$

where x, y are chosen uniformly at random from $\{0,1\}^n, \{0,1\}^m$, respectively.

In this definition it is implied that A accesses y or $G(x)$ as though they were the results of random coin tosses, while also having regular access to its "real" input. We count the space as the total information needed to store the state of the machine, i.e. the space is the logarithm (base 2) of the total number of configurations of the machine. For any space bound $S(n) \geq \log n$, this changes the definition of space by at most a constant factor.

Our generators will run on-line in space(S), in the following sense:

Definition 4 *A generator $G : \{0,1\}^n \rightarrow \{0,1\}^m$ is said to run on-line in space S if its input and output tapes are one-way and it runs in space S.*

In Section 3.1 we will show how a pseudorandom generator that stretches R bits to RS^γ bits for $\gamma < 1$ can be built using an extractor. Then in Section 3.2 we will show how to compose such generators and stretch the number of random bits by a factor of S^c for any constant c.

3.1 Expanding R Bits to RS^γ Bits

Let $0 < \gamma < 1$ be given. We will construct an on-line pseudorandom generator that stretches R bits to $R' = \Omega(RS^\gamma)$ bits (for all $R \geq (c+1)S$ for some constant c described below). Fix the following parameters:

1. $t = S^{1-\gamma}$.
2. n is chosen such that the output of the extractor described in Lemma 1 with input sizes n and t and parameter $\delta = 1/2$ is of length exactly S. Thus $n = cS$ for some constant c. Note that we may also assume $c \geq 4$, since we can always make c larger by ignoring some of the bits output by the extractor.
3. $R' = (R - n)S^\gamma$, and $l = R'/S$. Thus $R' = \Omega(RS^\gamma)$.
4. $\epsilon = l(\epsilon' + 2^{-S})$, where ϵ' is the quality of output of the extractor with input sizes n and t and parameter $\delta = 1/2$. Thus $\epsilon = 2^{-\Omega(S^{1-\gamma}/log^2 S)}$.

Description of G:

1. INPUT: $x \in \{0,1\}^n, y_1, \ldots, y_l \in \{0,1\}^t$.
2. OUTPUT (a string in $\{0,1\}^{R'}$): $E(x, y_1), \ldots, E(x, y_l)$.

Lemma 2 *G is a pseudo-random generator for space S with parameter ϵ running on-line in space $O(S)$.*

Proof: The fact that G runs on-line in space $O(S)$ follows immediately from the fact that E can be computed in space $O(n)$. To prove that G is a pseudorandom generator we will show that it fools any *space*(S) machine M. As in [1], we model M as a layered multi-graph L with a layer for each $0 \leq i \leq l$, where each layer has 2^S vertices. This will represent M reading S random bits at a time: the ith layer of L represents the configuration of M after reading i sets of S bits. More formally, L consists of vertices (i, j), and the edge $< (i,j), (i+1,k) >$ appears with the label r iff the S-bit random string r causes M to go from configuration j to configuration k (so an edge can appear with many labels).

Denote by U_i the distribution on layer i induced by M running on a truly random y. Thus $U_i[j]$ is the probability that M will be in state j after reading iS random bits. Denote by D_i the distribution on layer i induced by M running on the output of the generator. The lemma now follows from the following lemma.

Lemma 3 *For all* $0 \le i \le l$, $||U_i - D_i|| \le i(2^{-S} + \epsilon')$.

Proof: Let $X, Y_1, \ldots, Y_l$ denote random variables corresponding to the inputs $x, y_1, \ldots, y_l$ being chosen independently and uniformly at random.

We prove this lemma by induction on i. It is true for $i = 0$, since both D_0 and U_0 are simply concentrated at the initial configuration of the machine. Suppose it is true for $i-1$. Define U_i^j and D_i^j to be the distributions on level i conditioned upon the $(i-1)$st vertex being j (where U_i^j is for M running on a truly random input and D_i^j for M running on the output of G). We thus have $U_i = \sum_j U_{i-1}[j]U_i^j$ and $D_i = \sum_j D_{i-1}[j]D_i^j$.

Let B be the set of j for which $D_{i-1}[j] \ge 2^{-2S}$. For a fixed $j \in B$ consider the distribution of X conditioned upon reaching vertex $(i-1, j)$ (and induced by the random choices of X and of $Y_1 ... Y_{i-1}$). Since the conditioning can only increase the probability of each value of X by a factor of at most 2^{2S}, we get that this distribution is a δ-source, for $\delta = (c-2)/c \ge 1/2$. In this case the fact that E is an extractor implies that the distribution of $E(X, Y_i)$ conditioned upon reaching vertex $(i-1, j)$ is quasi-random to within ϵ'. Since the next vertex (on level i) is determined by $E(X, Y_i)$, we get that this vertex (conditioned on visiting $(i-1, j)$) is distributed the same (to within ϵ') in the random and pseudorandom cases. In other words, $||U_i^j - D_i^j|| \le \epsilon'$.

Since there are at most 2^S possible values for j we can bound $\sum_{j \notin B} D_{i-1}[j] \le 2^S 2^{-2S} = 2^{-S}$. We can now bound from above $||U_i - D_i||$. Denote $\sum_k |\alpha_k|$ by $||\alpha||_1$ (thus $||U_i - D_i|| = ||U_i - D_i||_1/2$.) Then,

$$\begin{aligned}
||U_i - D_i||_1 &= ||\sum_j U_{i-1}[j]U_i^j - \sum_j D_{i-1}[j]D_i^j||_1 \\
&\le ||\sum_j U_{i-1}[j]U_i^j - D_{i-1}[j]U_i^j||_1 + ||\sum_j D_{i-1}[j]U_i^j - D_{i-1}[j]D_i^j||_1 \\
&\le (\sum_j |U_{i-1}[j] - D_{i-1}[j]|)||U_i^j||_1 + (\sum_{j \in B} |D_{i-1}[j]|)||U_i^j - D_i^j||_1 + (\sum_{j \notin B} |D_{i-1}[j]|)||U_i^j - D_i^j||_1 \\
&\le ||U_{i-1} - D_{i-1}||_1 \cdot 1 + 1 \cdot 2\epsilon' + 2^{-S} \cdot 2.
\end{aligned}$$

The lemma follows. □

This also concludes the proof of Lemma 2. □

3.2 Composing on-line generators

As is the case for poly-time secure pseudorandom generators, on-line generators can be composed.

Lemma 4 *Let* $G_1 : \{0,1\}^{R_2} \to \{0,1\}^{R_1}$ *be a generator for space* S_1 *with parameter* ϵ_1 *running on-line in space* S_2. *Let* $G_2 : \{0,1\}^{R_3} \to \{0,1\}^{R_2}$ *be a generator for space* $S_1 + S_2$ *with parameter* ϵ_2 *running on-line is space* S_3. *Then,* $G_1 \circ G_2 : \{0,1\}^{R_3} \to \{0,1\}^{R_1}$ *is a pseudorandom generator for space* S_1 *with parameter* $\epsilon_1 + \epsilon_2$ *running on-line in space* $S_2 + S_3$.

Proof: The proof follows from the fact that for any space S_1 algorithm A, $A(G_1(\cdot))$ can be implemented on-line in space $S_1 + S_2$. (Recall that, by definition, A treats $G_1(\cdot)$ as the outcome of random coin flips and thus accesses it on-line.) □

Our main theorem, from which Theorem 1 is immediate, now follows easily:

Theorem 2 *For any constant* $\nu > 0$ *and all polynomials* p*, there is (an explicitly given) pseudo-random generator* $G : \{0,1\}^{O(S)} \to \{0,1\}^{p(S)}$ *for space* S *with parameter* $2^{-S^{1-\nu}}$ *running in time poly*(S) *and space* $O(S)$.

Proof: Let $p(n) = n^c$, and choose some $\gamma < \nu$. We first build a generator G_1 for space $S_1 = S$ that stretches the number of bits by a factor of S^γ and runs on-line in space S_2, as in Section 3.1. We then build a generator G_2 for space $S_1 + S_2$ stretching by a further S^γ factor. This is repeated $(c-1)/\gamma$ times and all of the above generators are composed together. This gives a generator that stretches the random bits by a factor of n^{c-1}. Taking $R = O(S)$ concludes the proof. □

4 A Lower Bound

In this section, we give a lower bound on the quality of any extractor. This means giving a lower bound on t and an upper bound on m.

Theorem 3 *Suppose $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is a (δ,ϵ)-extractor, where $\delta \leq 1 - 1/n$ and $\epsilon < 1/2$. Then $t \geq \max(\log \epsilon^{-1} - 1, \log((1-\delta)n))$ and $m < \delta n + 2\epsilon$.*

Proof: First suppose $\epsilon < 2^{-(t+1)}$. Pick any $S \subseteq \{0,1\}^m$ with $|S| = 2^{m-(t+1)}$. The point is that for each x, the probability that $E(x,y) \in S$ is an integral multiple of 2^{-t}, and hence will differ from $2^{-(t+1)}$ by at least $2^{-(t+1)} > \epsilon$. Thus let A be the set of $x \in \{0,1\}^n$ such that for some y, $E(x,y) \in S$. Then either A or its complement has size at least $2^{\delta n}$, and therefore violates the definition of extractor.

To see that $t > \log((1-\delta)n)$, denote by $V(x)$ the 2^t-bit long vector obtained by concatenating the first bit of $E(x,y)$ for all values of $y \in \{0,1\}^t$; also for $v \in \{0,1\}^{2^t}$ denote $A_v = \{x | V(x) = v\}$. It is clear that for any fixed v, if x is uniformly chosen from A_v then the first bit of $E(x,y)$ is completely determined by y, and thus $E(x,y) \circ y$ is not quasi-random. This implies that $|A_v| < 2^{\delta n}$. As the A_v's are a partition of $\{0,1\}^n$ we have $2^{2^t} 2^{\delta n} \geq 2^n$ so $t > \log((1-\delta)n)$.

To see the upper bound on m, we note that if D is quasi-random to within ϵ on $\{0,1\}^r$, then for some z, $D(z) < 2^{-s}$, where $s = r - 2\epsilon$. Otherwise D would place positive probability on at most 2^s strings, so the variation distance from D to uniform would be $1 - 2^{s-r} = 1 - 2^{-2\epsilon} > \epsilon$.

Applying this to the $(m+t)$-bit output $E(x,y) \circ y$, we see that some string must be output with probability at most $2^{2\epsilon - m - t}$. On the other hand, any such string must also have probability at least $2^{-\delta n - t}$. Thus $2^{2\epsilon - m - t} \geq 2^{-\delta n - t}$.

□

5 Extracting Randomness

In this section we describe the extractor. There are two main parts: "converting" a δ-source into a distribution close to a block-wise δ-source, and using hashing techniques to extract bits from a block-wise δ-source. Because this second part is easier, we present it first in Subsection 5.2, just after presenting our tools in Subsection 5.1. The first part is described in Subsections 5.3 and 5.4, where everything is put together.

5.1 Tools

5.1.1 k-wise independent distributions

We will need to choose, sufficiently randomly but using few random bits, l elements out of n given elements. The property we wish to have from the random choice is that, with high probability, it intersects every given subset of size δn in at least $\delta l/2$ places. The simplest way to do this is using k-wise independent distributions (see e.g. [7, 13, 4, 14]). In order to ensure that no duplicate elements are chosen, we do the following.

Choosing l out of n elements:
We divide the n elements into l disjoint sets $A_1, \ldots, A_l$ of size $m = n/l$, i.e.

$$A_i = \{(i-1)m + 1, (i-1)m + 2, \ldots, im\}.$$

We then use $k \log n$ random bits to choose $X_1, \ldots, X_l$ k-wise independently, where the range of X_i is A_i, and set $S = \{X_1, \ldots, X_l\}$.

The property we will require is:

Lemma 5 *Let $T \subseteq \{1, 2, \ldots, n\}$, $|T|/n \geq \delta$. Suppose $k \leq \delta l/6$. If S is chosen at random as described above, then*

$$Pr[|S \cap T| \geq \delta l/2] \geq 1 - e^{-\lfloor k/2 \rfloor}.$$

We use the following lemma, which is a special case of Theorem 2.5 from [18]:

Lemma 6 *Let $Y_1, \ldots, Y_l$ be k-wise independent 0-1 random variables, $Y = \sum_{i=1}^{l} Y_i$, and $\mu = EY$. Let $\alpha = \sqrt{ke^{1/3}/\mu}$, and suppose $\alpha \leq 1$. Then*

$$Pr[|Y - \mu| > \alpha\mu] \leq e^{-\lfloor k/2 \rfloor}.$$

Proof of Lemma 5: Define the random variables Y_i to be 1 iff $X_i \in T$, and 0 otherwise. Let $\delta_i = EY_i = |T \cap A_i|/m$. Then for $Y = \sum_{i=1}^{l} Y_i$, $EY = \sum_{i=1}^{l} \delta_i \geq \delta l$. Setting $\alpha = 1/2$ (so $\alpha^2 e^{-1/3} \geq 1/6$) in Lemma 6 concludes the proof. □

In the above lemma we used $t = O(k \log n)$ random bits to generate the k-wise independent random variables $Y_1, Y_2, \ldots, Y_n$. By using more sophisticated techniques based on random walks on constant degree expanders, we can reduce the number of random bits to $O(k + \log n)$ for constant δ. ("Almost k-wise independent" sample spaces do not appear to give this.) This is done below, but we do not use it further in this paper.

Lemma 7 *Suppose $ck \leq \delta^2 l$. Then we can use $O(k/\delta + \log n)$ random bits to pick l random variables $X_1, \ldots, X_l$ in $\{1, 2, \ldots, n\}$ such that*

$$Pr[\geq \delta^2 l/16 \text{ of the } X_i\text{'s lie in } T] \geq 1 - 2^{-k}.$$

Proof: We combine 10-wise independence and random walks on expanders in a manner similar to [3]. We divide $\{1, 2, \ldots, n\}$ into m disjoint sets $A_1, \ldots, A_m$ of size $p = n/m$, where $m = 14k/\delta$. Within each set A_i, we use $10 \log p$ bits to pick a set S_i of size $l' = l/m$ using 10-wise independence, as in Lemma 5 (in fact, pairwise independence would suffice, but we wish to quote Lemma 5).

Let $T_i = T \cap A_i$, and $\delta_i = |T_i|/p$. We say that dimension i is *important* if $\delta_i \geq \delta/2$. There must be at least $\delta m/2 \geq 7k$ important dimensions. Now set the constant c in the statement of the lemma large enough so that $\delta l' \geq 120$. By Lemma 5, if i is important then $Pr[|S_i \cap T_i| \geq \delta l'/4] \geq .99$. Call such a set S_i *good.*

We now use an explicitly constructible constant-degree expander graph G on p^{10} nodes, with second largest eigenvalue in absolute value at most $1/10$. For example, we can use a power of the one in [9] with sufficiently many self loops to eliminate the negative eigenvalues. We then take a random walk for m steps from a uniformly random start vertex. The vertex visited at the ith step defines a set $S_i \subseteq A_i$ as above. We set $S = \cup_{i=1}^{m} S_i$.

To analyze this, we need the following modification of a lemma from [11] (see also [8]):

Lemma 8 *Suppose that for $1 \leq i \leq 7k$, $W_i \subseteq \{1, 2, \ldots, N\}$, $|W_i| \geq .99N$, and G_i is a regular expander multigraph on N nodes with corresponding transition matrix having second largest eigenvalue in absolute value at most $1/10$. Perform a random walk from a random initial start vertex, and then use graphs $G_1, \ldots, G_{7k}$ to take the next $7k$ steps and visit vertices $v_1, \ldots v_{7k}$. Then*

$$Pr[> 7k/2 \text{ of } v_i \in W_i] \geq 1 - 2^{-k}.$$

Now consider only the first $7k$ important dimensions. Let G_i denote G^{r_i} (G to the power r_i, corresponding to a walk on G for r_i steps), where r_i is the number of dimensions between the $(i-1)$st important dimension and the ith. Then Lemma 8 applies, and

$$Pr[> \delta m/4 \text{ of } S_i \text{ are good}] \geq 1 - 2^{-k}.$$

Note that if there are $\delta m/4$ good S_i, then $|S \cap T| \geq \delta^2 l/16$, and the proof is complete.

We remark that we can improve the dependence on δ in two ways: first, by using the generator of [15] for the bits of the random walk; and second, by redefining i as important if $\delta_i \in I_j = (2^{-(1+j)}, 2^{-j}]$, where j is chosen so that the sum of the δ_i in this interval is maximum.

□

5.1.2 Universal Hashing

We will use universal hash functions [5]. Formally, let H be a set of functions $h : \{0,1\}^n \to \{0,1\}^m$.

Definition 5 *(Carter-Wegman) H is called a universal family of hash functions if for any $x_1 \neq x_2 \in \{0,1\}^n$ and $y_1, y_2 \in \{0,1\}^m$ we have that*

$$Pr_{h \in H}[h(x_1) = y_1 \text{ and } h(x_2) = y_2] \;=\; 2^{-2m}.$$

We will require the Leftover Hash Lemma of [10].

Lemma 9 *(Leftover Hash Lemma [10]) Let $X \subset \{0,1\}^n, |X| \geq 2^r$. Let $k > 0$, and let H be a universal family of hash functions mapping n bits to $r - 2k$ bits. Then the distribution $(h, h(x))$ is quasi-random within $1/2^k$ (on the set $H \times \{0,1\}^{r-2k}$), where h is chosen uniformly at random from H, and x uniformly from X.*

The following is a corollary of the proof of the Leftover Hash Lemma.

Corollary 3 *Let D be a distribution on $\{0,1\}^n$ such that for all $x \in \{0,1\}^n$, $D(x) \leq 2^{-r}$. Let $k > 0$, and let H be a universal family of hash functions mapping n bits to $r - 2k$ bits. Then the distribution $(h, h(x))$ is quasi-random within $1/2^k$ (on the set $H \times \{0,1\}^{r-2k}$), where h is chosen uniformly at random from H, and x according to D.*

5.2 Hashing to get quasi-randomness

In this subsection we present a function which extracts a quasi-random string from a block-wise δ-source.
Function C:
The function has 3 parameters: δ, the quality of the source; l_1, l_s, the largest and smallest block sizes.

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

Lemma 10 *Let D be a block-wise δ-source on $\{0,1\}^{l_1+\cdots+l_s}$. If $\vec{X} = X_1 \ldots X_{l_s}$ is chosen according to D and Y is chosen uniformly at random in $\{0,1\}^{2l_s}$, then the distribution of $C(X,Y) \circ Y$ is quasi-random to within $2 \cdot 2^{-\delta l_s/4}$.*

Proof: We will prove by induction from $i = s$ down to $i = 0$ the following claim, which clearly implies the lemma.
Claim: For any sequence of values $x_1 \ldots x_i$, the distribution of h_i conditioned on $X_1 = x_1, \ldots, X_i = x_i$, is quasi-random to within ϵ_i, where $\epsilon_i = \sum_{j=i+1}^{s} 2^{-\delta l_j/4}$.

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

Thus, the distribution of h_i is within ϵ_{i+1} of the distribution obtained by picking x_{i+1} according to D_{i+1}, and h_{i+1} independently and uniformly at random in H_{i+1}. Using Corollary 3 this second distribution is quasi-random to within $2^{-\delta l_{i+1}/4}$. □

The algorithm is more subtle than at first appears. In particular, it is important that the above algorithm proceeds "backwards," i.e. that the block-wise δ-source outputs the biggest blocks first, but we start hashing with the smallest blocks first. Otherwise, say the distribution of X_{s-1} could be an arbitrary δ-source depending on x_s. Then since h_{s-1} also depends on x_{s-1}, h_{s-1} and x_{s-1} would not be close to independent, and we could not apply Corollary 3.

5.3 Extracting a block

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

The idea to do this is as follows. Intuitively, a δ-source has many bits which are somewhat random. We wish to obtain l 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.

The function B:

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

1. INPUT: $x \in \{0,1\}^n$; $y \in \{0,1\}^t$ (where $t = O(k \log n)$).
2. Use y to choose a set $\{i_1 \ldots i_l\} \subset \{1 \ldots n\}$ of size l as described in Section 5.1.1.
3. OUTPUT (a vector in $\{0,1\}^l$): $x_{i_1} \ldots x_{i_l}$ (here x_j is the jth bit of x).

Lemma 11 *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, where $\delta' = c\delta / \log \delta^{-1}$ and $\epsilon = \max(2^{-ck}, 2^{-c\delta' l})$ for some sufficiently small positive constant c.*

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

The rest of this section is devoted to proving the slightly stronger conclusion, that the output is near a δ'-source. Our proof tries to retain the structure of the above proof but, since we do not have the powerful tools of information theory at our disposal, the proof is not very simple. The difficulty is perhaps best appreciated by observing that it is possible that for all $\vec{y}$, $B(\vec{X}, \vec{y})$ is not a δ'-source (for any δ'), but only statistically close to a δ'-source.

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

Definition 6 *For a string $\vec{x} \in \{0,1\}^n$ and an index $1 \leq i \leq n$, let*

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

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

The part of the definition with $p_i(\vec{x}) = 1/2$ is to ensure that exactly one of $x_i = 0$ and $x_i = 1$ is good, for a given prefix. This is used in Lemma 14.

Definition 7 *$\vec{x}$ is α-good if there are at least αn indices which are good in x.*

Definition 8 *For $S \subseteq \{1, 2, \ldots, n\}$, $\vec{x}$ is α-good in S if there are at least $\alpha|S|$ indices in S which are good in $\vec{x}$.*

Definition 9 *S is α-informative to within β if*

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

Denote by S_y the set of l indices chosen using the random bits $\vec{y}$ in the manner described in Section 5.1.1. We will prove two lemmas which together clearly imply Lemma 11.

Lemma 12

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

Lemma 13 *Fix a set of indices $S = \{i_1 \ldots i_l\}$ that is δ'-informative to within ϵ. Then, the distribution of $X_{i_1} \ldots X_{i_l}$ induced by choosing $\vec{X}$ according to D is ϵ-near a δ'-source.*

5.3.1 Proof of Lemma 12

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

Lemma 14

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

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

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

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

for c_1 small enough. □

Proof of Lemma 12: Denote $k' = \min(k, \delta l/6)$. For any fixed α-good string $\vec{x}$, we can apply Lemma 5 to the set of good indices and obtain

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

Using Lemma 14 it follows that

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

Set $\delta' = \alpha/2$ and $\epsilon = \sqrt{4e^{-k'/2} + 2^{-c_1\delta n}}$. We will now use Markov's inequality in the following way. Let $A_y = Pr_{\vec{X} \in D}[\vec{X}\ is\ not\ \delta' - good\ in\ S_y]$. Thus A_Y is a random variable determined by Y. From the above analysis, $E_Y[A_Y] \leq \epsilon^2$. Therefore, by Markov, $Pr_Y[A_Y \geq \epsilon] \leq \epsilon$. In other words,

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

□

5.3.2 Proof of Lemma 13

Proof: We will divide the probability space P corresponding to D into many subspaces P_r, and prove that in each P_r the corresponding distribution D_r is near a δ'-source. To do this, first observe that we can alter P a negligible amount by assuming that all conditional probabilities $Pr[X_i = 0 | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}]$ are rational. Then we can view P as generated by the following process: first, n numbers $r_1, \ldots, r_n$, are chosen independently and uniformly in the range $\{1 \ldots R\}$ for some R. Then the string $\vec{x}$ is generated deterministically bit by bit according to some deterministic functions: $x_i = f_i(x_1, \ldots, x_{i-1}, r_i)$. (The function f_i simply fixes the conditional probability of x_i given $x_1 \ldots x_{i-1}$.)

Our subspaces will be obtained by fixing the values of r_i for all indices i *not in* S. For any $(n-l)$-tuple of numbers $\vec{r}$, we consider the space P_r to be obtained by fixing r_i to be the value specified by $\vec{r}$ for all $i \notin S$; choosing r_i at random for all $i \in S$; and then generating $\vec{x}$ using the functions f_i. Let E_r be the distribution of

$X_{i_1} \ldots X_{i_l}$ induced when X is chosen according to D_r, and let E be the distribution of $X_{i_1} \ldots X_{i_l}$ induced when X is chosen according to D.

We say that $\vec{x}$ is consistent with $\vec{r}$ if $Pr_{\vec{X} \in D_r}[\vec{X} = \vec{x}] > 0$. This implies in particular that for all $i \notin S$, $x_i = f(x_1, \ldots, x_{i-1}, r_i)$.

For each $\vec{r}$ define

$$\epsilon_r = Pr_{\vec{X} \in D_r}[\vec{X} \text{ is not } \delta' - \text{good in } S].$$

We will show

Lemma 15 *For any fixed r, E_r is ϵ_r-near a δ'-source.*

Before we prove this lemma, let us see how this implies Lemma 13. For each $\vec{r}$ let $\tilde{E}_r$ be a δ'-source that is ϵ_r close to E_r. It is easy to see that the distribution $\tilde{E}$ which is defined to be the average of the $\tilde{E}_r$'s is a δ'-source and its distance from E is the average of the ϵ_r's, which is exactly $Pr_{\vec{X} \in D}[\vec{X} \text{ is not } \delta' - \text{good in } S] \leq \epsilon$. □

Before we prove Lemma 15, we require the following lemma.

Lemma 16 *If $\vec{x}$ is δ'-good in S and consistent with $\vec{r}$, then $E_r(x_{i_1} \ldots x_{i_l}) \leq 2^{-\delta' l}$.*

Proof: By definition,

$$E_r(x_{i_1} \ldots x_{i_l}) = Pr_{\vec{X} \in D_r}[X_{i_1} = x_{i_1}, \ldots, X_{i_l} = x_{i_l}].$$

We compute the above quantity as

$$\prod_{j=1}^{l} Pr[X_{i_j} = x_{i_j} | X_{i_1} = x_{i_1}, \ldots, X_{i_{j-1}} = x_{i_{j-1}}].$$

Note that for each $i \notin S$ we have that

$$Pr_{\vec{X} \in D_r}[X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}] = 1,$$

since $X_i = f(X_1 \circ \cdots \circ X_{i-1}, r_i) = x_i$. It follows that for each j,

$$Pr_{\vec{X} \in D_r}[X_{i_j} = x_{i_j} | X_{i_1} = x_{i_1}, \ldots, X_{i_{j-1}} = x_{i_{j-1}}]$$

is equal to

$$Pr_{\vec{X} \in D_r}[Pr[X_{i_j} = x_{i_j} | X_1 = x_1, \ldots, X_{i_j - 1} = x_{i_j - 1}].$$

For each $i \in S$, the distribution of X_i conditioned on $X_1 = x_1 \ldots X_{i-1} = x_{i-1}$ is exactly the same in D_r and in D as in both spaces it is given by $f(x_1, \ldots, x_{i-1}, r_i)$, where r_i is chosen at random. It follows that the above expression is equal to

$$Pr_{\vec{X} \in D}[Pr[X_{i_j} = x_{i_j} | X_1 = x_1, \ldots, X_{i_j - 1} = x_{i_j - 1}].$$

The assumption that $\vec{x}$ is δ'- good in S means that for at least $\delta' l$ of the possible j's the corresponding factor is less or equal to $1/2$. This completes the proof of the claim. □

We are now ready to prove Lemma 15.

Proof: Let $\vec{z} \in \{0,1\}^l$. We say that $\vec{z}$ is *big* if $E_r(\vec{z}) > 2^{-\delta' l}$. We first claim that:

$$Pr_{\vec{Z} \in E_r}[\vec{Z} \text{ is big}] \leq \epsilon_r.$$

To see this first observe that by definition

$$Pr_{\vec{Z} \in E_r}[\vec{Z} \text{ is big}] = Pr_{\vec{X} \in D_r}[X_{i_1} \ldots X_{i_l} \text{ is big}].$$

By Lemma 16, and the fact that $Pr_{\vec{X} \in D_r}[\vec{X} \text{ is consistent with } \vec{r}] = 1$, we have that the above quantity is bounded from above by

$$Pr_{\vec{X} \in D_r}[\vec{X} \text{ is not } \delta' - \text{good in } S] = \epsilon_r.$$

We can now obtain a δ'-source $\tilde{E}_r$ which is ϵ_r-close to E_r, by simply taking each big $\vec{z}$ and dividing it's probability amongst the $\vec{z}$'s with lowest probability. The distance from E_r is at most the probability of these big $\vec{z}$'s, which is at most ϵ_r. □

5.4 Description of the extractor

The only thing left to explain is how extracting small slightly random blocks can be used to obtain a distribution close to a block-wise δ-source. We do this after describing the extractor more precisely.

Unfortunately, the extractor requires a large number of parameters. How they are chosen is summarized below. The reader is advised to skip to the extractor description, and only use this parameter list as a reference.

Parameters:

1. The parameters n, ϵ and δ are given. We assume $1/n \le \delta \le 1/2$ and $2^{-\delta n} \le \epsilon \le 1/n$.
2. $\delta' = c(\delta/2)/\log(2/\delta)$, where c is from Lemma 11. Thus $\delta' = \Theta(\delta/\log\delta^{-1})$.
3. l_0 is the largest integer such that $\sum_{i=1}^{\infty} \frac{l_0}{(1+\delta'/4)^i} \le \delta n/4$. This is used in Lemma 17. Thus $l_0 = \Omega(\delta^2 n/\log\delta^{-1})$.
4. For each i, set $l_i = l_{i-1}/(1+\delta'/4)$. This is needed to define the function C.
5. k is chosen so that $2^{-ck} = (\epsilon/8n)^2$, where c is from Lemma 11. Thus $k = O(\log\epsilon^{-1})$.
6. s is chosen to be the largest integer such that $l_s \ge k/\delta'$. This is needed to apply Lemma 11. (This also implies $2^{-\delta' l_s} \le \epsilon/4$, as needed to apply Lemma 10 in the proof of Lemma 1.) Thus $l_s = O(\log\epsilon^{-1}\log\delta^{-1}/\delta)$, and $s = O(\log n \log\delta^{-1}/\delta)$.
7. $t_1 = k\log n$. Thus $t_1 = O(\log\epsilon^{-1}\log n)$.
8. $t_2 = 2l_s$. Thus $t_2 = O(\log\epsilon^{-1}\log\delta^{-1}/\delta)$.
9. The length of the second parameter to E is given by $t = st_1 + t_2$. Thus $t = O(\log\epsilon^{-1}\log^2 n\log\delta^{-1}/\delta)$.
10. The length of the output of E is given by $m = 2l_0 - 2l_s$. Thus $m = \Omega(\delta^2 n/\log\delta^{-1})$.

Description of E:

1. INPUT: $x \in \{0,1\}^n$; $y_1 \in \{0,1\}^{t_1}, \ldots, y_s \in \{0,1\}^{t_1}$; $y_0 \in \{0,1\}^{t_2}$.
2. For $i = 1 \ldots s$ do $z_i \leftarrow B(x, y_i)$. (We use B with parameters $n, l_i, k, \delta/2$.)
3. OUTPUT (a vector in $\{0,1\}^m$): $C(z_1 \ldots z_s, y_0)$. (We use C with the parameters δ', l_1, l_s.)

The following lemma tells us that the distribution of the z_i's is close to a block-wise δ-source.

Lemma 17 *For all but $\epsilon/4$ fraction of possible values of $y_1 \ldots y_s$, the distribution of $Z_1 \circ \cdots \circ Z_s$ induced by choosing X according to distribution D is within $\epsilon/4$ of a block-wise δ'-source.*

Before we prove this lemma let us see how it implies the main lemma.

Main lemma (Lemma 1): For any δ-source D the distribution of $E(X,Y)\circ Y$ induced by choosing X according to D and Y uniformly in $\{0,1\}^t$ is ϵ-quasi-random on $\{0,1\}^m \times \{0,1\}^t$.

Proof: By Lemma 17 for all but $\epsilon/4$ fraction of values of $y_1 \ldots y_s$ the distribution on the z's is within $\epsilon/4$ of a block-wise δ'-source. For each such value of the y's, by Lemma 10, the output concatenated with y_0 is quasi-random within $\epsilon/2 + \epsilon/4$. Add the $\epsilon/4$ "bad" y's and the lemma follows. $\square$

We now return to the proof of Lemma 17.

Proof of Lemma 17: Let us first give the intuition. Lemma 11 tells us how to extract one slightly-random block from a δ-source. When we extract the second (or sth) block, however, we must ensure that it is still slightly random conditional on the first block. The reason the last blocks are slightly random is that we are conditioning on at most $l_1 + l_2 + \ldots + l_{s-1} < \delta n/4$ bits of information, so we are still left with a $3\delta/4$-source, and with high probability a $\delta/2$-source. We now formalize this.

Call a vector $y_1 \ldots y_i$ *good* if the distribution of $Z_1 \ldots Z_i$ is within $i\epsilon/(4n)$ from a block-wise δ'-source. We now prove by induction on i that all but a $i\epsilon/(4n)$ fraction of $y_1 \ldots y_i$ are good. As $s \leq n$, this suffices to prove the lemma.

Fix a vector $y_1 \ldots y_{i-1}$ that is good. We will show that for all but $\epsilon/(4n)$ fraction of y_i's, the vector $y_1 \ldots y_i$ is also good. We call the vector of values $z_1, \ldots, z_{i-1}$ *tiny* if

$$Pr[Z_1 = z_1 \textit{ and } \cdots \textit{ and } Z_{i-1} = z_{i-1}] \leq 2^{-\delta n/2}.$$

Since there are at most $2^{l_1+\ldots+l_{i-1}} \leq 2^{\delta n/4}$ possible values for $z_1 \ldots z_{i-1}$, $Pr[Z_1 \ldots Z_{i-1} \textit{ is tiny}] \leq 2^{-\delta n/4} \leq \epsilon/(16n)$.

For any $z_1 \ldots z_{i-1}$ consider the distribution $D_{z_1 \ldots z_{i-1}}$ defined to be the distribution on X conditioned on $Z_1 = z_1 \ldots Z_{i-1} = z_{i-1}$. It is clear that if $z_1 \ldots z_{i-1}$ is not tiny then for all $x \in \{0,1\}^n$: $D_{z_1 \ldots z_{i-1}}(x) \leq 2^{-\delta n/2} D(x)$, and thus $D_{z_1 \ldots z_{i-1}}$ is a $\delta/2$-source. Let $E_{y_i, z_1 \ldots z_{i-1}}$ denote the distribution of Z_i induced by choosing X according to $D_{z_1 \ldots z_{i-1}}$. Applying Lemma 11 to $D_{z_1 \ldots z_{i-1}}$, we conclude that for every non-tiny choice of $z_1 \ldots z_{i-1}$ for all but an $(\epsilon/8n)^2$ fraction of y_i's, $E_{y_i, z_1 \ldots z_{i-1}}$ is within $(\epsilon/8n)^2$ of a δ'-source.

Applying Markov's inequality in a way analogous to that in Lemma 12, we conclude that for all but at most an $\epsilon/(4n)$ fraction of y_i's,

$$\begin{aligned} &Pr_{z_1 \ldots z_{i-1}}[E_{y_i, z_1 \ldots z_{i-1}} \text{ is not within } (\tfrac{\epsilon}{8n})^2 \text{ of a } \delta'\text{-source}] \\ &\leq \epsilon/(16n) + Pr[z_1 \ldots z_{i-1} \text{ is tiny}] \\ &\leq \epsilon/(8n). \end{aligned}$$

We conclude the induction step by observing that the above inequality implies that $y_1 \ldots y_i$ is good. To get a block-wise δ'-source that is close to the distribution of $Z_1 \ldots Z_i$, we start with the block-wise source that is close to the distribution of $Z_1 \ldots Z_{i-1}$ (given by the induction hypothesis), and "extend it" by choosing a δ'-source on Z_i for each value of $z_1 \ldots z_{i-1}$. If $E_{y_i, z_1 \ldots z_{i-1}}$ is within $(\epsilon/8n)^2$ of a δ'-source, we choose the close δ'-source. All the other possible values of $z_1 \ldots z_{i-1}$ occur with low probability, so we can use e.g. the uniform distribution. The total error is: $(i-1)\epsilon/(4n)$ for the original block-wise source, $(\epsilon/8n)^2$ on the "good" z's and $\epsilon/(8n)$ for the "bad" z's, all together less than $i\epsilon/(4n)$. □

6 Acknowledgements

We thank Mauricio Karchmer, Nati Linial, Mike Luby, Muli Safra, and Avi Wigderson for helpful discussions. We also thank the anonymous referees for a careful reading and helpful comments.

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi, Deterministic Simulation of Logspace, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987, pp. 132-140.

[2] L. Babai, N. Nisan, and M. Szegedy, Multiparty Protocols, Pseudorandom Generators for Logspace, and Time-Space Tradeoffs, *J. Comp. Syst. and Sci.* **45(2)** (1992), pp. 204-232.

[3] M. Bellare, O. Goldreich, and S. Goldwasser, Randomness in Interactive Proofs, *Computational Complexity* **5** (1993): 319-354.

[4] B. Berger and J. Rompel, Simulating $(\log^c n)$-Wise Independence in NC, *JACM* **38(4)**:1026-1046, 1991.

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

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

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

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

[9] O. Gabber and Z. Galil, Explicit Construction of Linear-Sized Superconcentrators, *J. Comp. and Sys. Sci* 22 (1981), pp. 407-420.

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

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

[12] M. Jerrum and A. Sinclair, Approximating the Permanent, *SIAM J. Comput.* **18** (1989):1149-1178.

[13] M. Luby, Removing Randomness in Parallel Computation Without a Processor Penalty, *Proceedings of the 29th Symposium on Foundations of Computer Science*, 1988, pp. 162-173.

[14] R. Motwani, J. Naor, and M. Naor, The Probabilistic Method Yields Deterministic Parallel Algorithms, *Proceedings of the 30th Symposium on Foundations of Computer Science*, 1989, pp. 8-13.

[15] N. Nisan, Pseudorandom generators for space-bounded computation, *Combinatorica* **12(4)** (1992), pp. 449-461.

[16] N. Nisan, $RL \subseteq SC$, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992, pp. 619-623.

[17] W.J. Savitch, Relationships between nondeterministic and deterministic space complexities, *J. Comp. and Syst. Sci.* **4(2)** (1970):177-192.

[18] J.P. Schmidt, A. Siegel, A. Srinivasan, Chernoff-Hoeffding Bounds for Applications with Limited Independence, *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993, pp. 331-340.

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

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

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