COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA **12** (4) (1992) 449–461

PSEUDORANDOM GENERATORS FOR SPACE-BOUNDED COMPUTATION

NOAM NISAN*

Received December 3, 1989
Revised June 16, 1992

Pseudorandom generators are constructed which convert $O(S \log R)$ truly random bits to R bits that appear random to any algorithm that runs in $SPACE(S)$. In particular, any randomized polynomial time algorithm that runs in space S can be simulated using only $O(S \log n)$ random bits. An application of these generators is an explicit construction of universal traversal sequences (for arbitrary graphs) of length $n^{O(\log n)}$.

The generators constructed are technically stronger than just appearing random to space-bounded machines, and have several other applications. In particular, applications are given for "deterministic amplification" (i.e. reducing the probability of error of randomized algorithms), as well as generalizations of it.

1. Introduction

Randomness is an important computational resource. There are many problems for which the known randomized algorithms are more efficient than the deterministic ones. The randomized algorithms may use less time, less space, less communication, less of other computational resources, or just be simpler than their deterministic counterparts. Viewing randomness as a resource, it is natural to try to reduce the amount of randomness, the number of random bits, used by randomized algorithms. The most general mechanism for doing such a thing is by using pseudorandom generators ([4,20]).

A pseudorandom generator converts a short truly random seed into a long string which can be used instead of truly random bits in any polynomial time algorithm. It can thus be used to reduce the number of random bits used in any polynomial time algorithm (down to the length of the short random seed). Unfortunately, pseudorandom generators are only known to exist under the unproven assumption that one-way functions exist ([9]). Moreover, the existence of one-way functions, an assumption which is seemingly even stronger than $P \neq NP$, is a necessary requirement for the existence of polynomial time computable pseudorandom generators.

A more specialized approach for saving random bits is to construct pseudorandom generators for some specific *subclass* of algorithms; and to do this *without relying on any unproven assumptions.* The first such result is due to Ajtai and Wigderson ([3]) who construct generators which look random to all polynomial size constant depth circuits. In [15], Nisan and Wigderson give an improved construction, and

AMS subject classification code (1991): 68 Q 15

*This work was done in the Laboratory for Computer Science, MIT, supported by NSF 865727-CCR and ARO DALL03-86-K-017

generalize it to show how "lower bounds" for a complexity class C can be used to construct a generator that looks random to any algorithm from C.

In [5], Babai, Nisan and Szegedy consider classes of space-bounded algorithms and construct generators which look random to all Logspace algorithms (or in general, any "small" space algorithms). In this paper we give an improved construction of a pseudorandom generator for space-bounded computation. Our construction is based upon universal hash functions (Carter, Wegman [7]), and is totally different from the one in [5]. The generator is very efficient as it requires one hashing operation on n-bits strings (typically one modular multiplication) for every n bits generated; it is also in NC.

Theorem 1. *There exists a fixed constant $c > 0$ such that for any R and S there exists an (explicitly given) pseudorandom generator which converts a random seed of length $cS\log R$ to R bits which cannot be distinguished from truly random bits by any algorithm running in space(S).*

In particular $O(\log^2 n)$ random bits are sufficient to produce a polynomial number of bits which look random to any Logspace machine. This is an exponential improvement over the generators constructed in [5], which require $\exp(\sqrt{\log n})$ random bits.

As an immediate corollary we get that any randomized polynomial time algorithm that runs in $Space(S)$ can be simulated in polynomial time using at most $O(S\log n)$ random bits (and $O(S\log n)$ space).

Our generator implies what may be considered a black box version of the randomized analogue of Savitch's theorem: Not only can $RSPACE(S)$ be simulated deterministically in $DSPACE(S^2)$, but the simulation is the same for all randomized *space*(S) algorithms: Simply run the algorithm with every possible output string of the generator.

As shown in [5] pseudorandom generators for space-bounded machines allow explicit constructions of universal traversal sequences (as defined in Aleliunas et al. [1]). The generators obtained in [5] give constructions of universal traversal sequences of length $\exp(\exp(\sqrt{\log n}))$ for arbitrary regular n-vertex graphs. The only other explicit constructions known are for the special cases of degree 2 graphs (a polynomial length construction due to Istrail [10]), and degree $n-1$ graphs (an $n^{O(\log n)}$ construction due to Karloff et al. [13]). We achieve an exponential improvement over known results for general graphs, matching the construction for degree $n-1$ graphs.

Theorem 2. *There exist (explicitly given) universal traversal sequences of length $n^{O(\log n)}$ for regular n-vertex graphs. Moreover, the sequences can be produced by a deterministic Turing machine running in space logarithmic in the length of the sequence.*

The generator we construct actually looks random to a more general class of algorithms. The generator outputs the pseudorandom bits in "blocks" of a certain size. It turns out that the space bound on the algorithm is only necessary *between* the different blocks, while *within* each block no limitation is needed. This fact allows the generator to be used for several other applications.

Consider the following problem: Given are k randomized algorithms $A_1,\ldots,A_k$, each requiring at most R random bits. Our task is to run all of them such that the

probability that they all succeed is (to within an error of ε) equal to the product of the individual success probabilities. We define a generator to be a *pseudo-independent block generator* if its output can be used for such a task for every choice of $A_1, \ldots A_k$. (In this definition "succeed" can be replaced by "reject", "find a witness", "prints 17", etc; a formal definition appears in section 5.)

A special case of this problem, "deterministic amplification", has been widely studied (see below), but the first nontrivial solution to this general problem is (implicitly) given by Impagliazzo and Zuckerman in [11]. The construction allows running $k = O(\sqrt{R})$ many algorithms with $\varepsilon = \exp(-\sqrt{R})$ while using only $O(R)$ random bits. Our generator can be used to run a much larger number of algorithms, and with a smaller value of ε but incurring a small additional cost in the number of random bits needed.

Theorem 3. *Any k algorithms, each using R random bits, can be run using $O(R \log k)$ random bits, such that the probability that they all succeed is within 2^{-R} of the product of the individual success probabilities.*

The special case where $A_1, \ldots, A_k$ are all repetitions of the same algorithm, and our only aim is to reduce the probability that they all fail (not necessarily close to the optimal value obtained from independent runs) is called the deterministic amplification problem. The possibility of deterministic amplification was pointed out, nonconstructively, by Sipser [18] and Santha [17]. The first constructions were obtained by Karp, Pippenger and Sipser [12], and Chor and Goldreich [6] who reduce the failure probability down to $1/p(n)$ for any polynomial $p(n)$, while still using only $O(R)$ random bits. Reduction to an exponentially small probability of failure ("quasi-perfect pseudorandom generation" in the terminology of Vazirani [19]) was recently obtained by Cohen and Wigderson [8] and Impagliazzo and Zuckerman [11]. The only construction that reduces the error probability as much as we do (to $\exp(-R)$) while using fewer random bits is, as shown in [8,11], due to Ajtai, Komlós and Szemerédi [2] and requires the explicit constructions of constant degree expanders. Our results also apply to randomized algorithms with 2-sided error, and also to algorithms that have only a polynomially small probability of success (the constructions of [2] and of [11] do not yield good bounds in this last case).

Many algorithms can be naturally broken into "stages" such that each stage requires a small number of random bits and the space required *between* the different stages is small. In all these cases the number of random bits required can be reduced by using the output of the generator instead of truly random bits. As an example we show how $O(n \log n)$ random bits suffice for uniformly generating a random n-bit prime number.

The paper is organized as follows. In section 2 we make some necessary definitions including the definition of the class of algorithms that our generator fools. Section 3 contains the construction of the generator. In section 4 we give the applications regarding space-bounded computation. Section 5 discusses pseudo-independent block generators and deterministic amplification. Finally, in section 6, the example of generating random primes is given.

2. Definitions and Notation

2.1. Requirements of the Generator

The generators we construct produce their output broken into blocks, each n bits long. They "fool" every program that accepts its random bits an n-bit block at a time, and that uses at most $Space(w)$ between the different blocks[1]. We say that such a program uses $Space(w)$ with $block-size\ n$. We model such a program by a finite state machine of size 2^w, over the alphabet $\{0,1\}^n$ (with a fixed start state, and an arbitrary number of accepting states). Each state of the FSM corresponds to a possible configuration of the original program between the blocks. Each edge (v,u) of the FSM is labeled by a subset of $\{0,1\}^n$, which is the set of n-bit strings which when accepted as the random block cause the original program to move from configuration v to configuration u; for each vertex v, each n-bit string thus appears in exactly one of the outgoing edges. From this point on we phrase everything in the language of such FSMs.

Definition 1. A generator $G:\{0,1\}^m \to (\{0,1\}^n)^k$ is a *pseudorandom generator for space(w) and block size n with parameter ε* if for every FSM Q of size 2^w over alphabet $\{0,1\}^n$ we have that

$$|Pr_y[Q \text{ accepts } y] - Pr_x[Q \text{ accepts } G(x)]| \leq \varepsilon$$

where y is chosen uniformly at random in $(\{0,1\}^n)^k$ and x in $\{0,1\}^m$.

2.2. Universal Hashing

Our generators are based upon universal hash functions (Carter, Wegman [7]). Formally, let H be a set of functions $h:\{0,1\}^n \to \{0,1\}^m$.

Definition 2. (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

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

It is only important for this paper that it is possible to efficiently give small universal families of hash functions, i.e. such that each $h \in H$ can be represented by at most $O(n+m)$ bits, and such that computing $h(x)$ given the representations of h and x is efficient. An example of such a family is convolution: Let x be an n-bit string, a an $m+n-1$ bit string, and b an m-bit string. Denote by $a*x$ the m-bit convolution of a and x (i.e. the j'th bit of $a*x$ is $\sum_{i=1}^{n} a_{i+j-1}x_i \pmod 2$), and by $c+b$ the bit-wise exclusive-or of the vectors b and c. The family $H=\{(a*x)+b | a,b\}$ is a universal family of hash functions (see e.g. [14]).

---

[1] Our measurement of space includes all information regarding the configuration of the machine. For Turing machines this includes the state of the finite control, the location of the heads, and the contents of the work tapes, all measured in bits. This inflates by at most a constant factor the space requirement of any machine that uses at least space $S(n)=\Omega(\log n)$.

3. The Generator

Before describing the generator we need to prove a certain useful property of universal families of hash functions. This property is of independent interest, and indeed a variant of it has already been used for proving time-space tradeoffs by Mansour et al. in [14].

3.1. A Property of Universal Hashing

The main trick used in the pseudorandom generator is to replace the usage of two random strings x and y with one random string x, and use $h(x)$ for y. For this to work we require some kind of "independence" between the values of x and $h(x)$. This independence is not information theoretic, but rather only relative to a specific application. We now define exactly the kind of independence we require:

Definition 3. Let $A \subset \{0,1\}^n$, $B \subset \{0,1\}^m$, $h: \{0,1\}^n \to \{0,1\}^m$, and $\varepsilon > 0$. We say that h is (ε, A, B)-*independent* if

$$|Pr_{x \in \{0,1\}^n}[x \in A \text{ and } h(x) \in B] - \varrho(A)\varrho(B)| \leq \varepsilon$$

where $\varrho(A) = |A|/2^n$ and $\varrho(B) = |B|/2^m$.

Lemma 1. *Let $A \subset \{0,1\}^n$, $B \subset \{0,1\}^m$, H be a universal family of hash functions $h: \{0,1\}^n \to \{0,1\}^m$, and $\varepsilon > 0$, then*

$$Pr_{h \in H}[h \text{ isn't } (\varepsilon, A, B)\text{-independent}] \leq \frac{\varrho(A)\varrho(B)(1 - \varrho(B))}{2^n \varepsilon^2}$$

where h is chosen uniformly at random in H.

Proof. Consider the matrix M, having a row for each $x \in \{0,1\}^n$ and a column for each $h \in H$, given by $M(x,h) = 1$ if $h(x) \in B$, and $M(x,h) = 0$ otherwise. Define now $f(h) = E_{x \in A} M(x,h) = Pr_{x \in A}[h(x) \in B]$ and denote $p = \varrho(B)$. First note that the expected value of f (over all $h \in H$) is p – this is simply because for every fixed x, $h(x)$ is uniformly distributed when h is chosen at random. Next, observe that by the definition of (ε, A, B)-independence, h is (ε, A, B)-independent iff

$$|p - f(h)| \leq \varepsilon / \varrho(A)$$

We will bound the variance of f from above, and will then be able to conclude that for "most" h, $f(h)$ is indeed very close to p.

$$Var(f) = E_{h \in H}(p - E_{x \in A} M(x,h))^2 =$$
$$E_{x_1 \in A, x_2 \in A} E_{h \in H}(p - M(x_1,h))(p - M(x_2,h))$$

Rearranging, and recalling that for every x, $E_{h \in H} M(x,h) = p$, we get that

$$Var(f) = E_{x_1 \in A, x_2 \in A} E_{h \in H} M(x_1,h) M(x_2,h) - p^2$$

This quantity is evaluated by looking at the two cases: Case 1 (happens with probability $1 - 1/|A|$): $x_1 \neq x_2$. In this case we use the definition of universal hash functions, and since $h(x_1)$ and $h(x_2)$ are distributed independently when h is chosen at random we have that $E_{h \in H} M(x_1,h) M(x_2,h) = p^2$. Case (2) (happens

with probability $1/|A|$): $x_1 = x_2$. In this case $M(x_1,h)M(x_2,h) = M(x_1,h)$ and thus $E_{h\in H}M(x_1,h)M(x_2,h) = p$. We thus get

$$Var(f) = (1 - 1/|A|)p^2 + (1/|A|)p - p^2 = \frac{\varrho(B)(1-\varrho(B))}{|A|}$$

By Chebychev's theorem applied to the random variable $f(h)$, for any $\delta > 0$ we have that:

$$Pr_{h\in H}[|p - f(h)| \geq \delta] \leq \frac{\varrho(B)(1-\varrho(B))}{|A|\delta^2}$$

The lemma is implied by letting $\delta = \varepsilon/\varrho(A)$. ∎

Consider the special case where $m = 1$, $B = \{1\}$, and $|H| = 2^n$. In this case our matrix M is essentially a Hadamard matrix (with entries 0,1 instead of 1,-1), and the statement of the lemma is a well known property of Hadamard matrices. Our lemma can be thought of as a natural generalization of this fact.

3.2. More Notation

Let Q be a FSM with 2^w states over alphabet $\{0,1\}^n$ and let D be any distribution on $(\{0,1\}^n)^k$ (sequences of k n-bit strings). We denote by $Q(D)$ the matrix whose (i,j)'th entry is the probability of getting from node i in Q to node j via a random $y \in (\{0,1\}^n)^k$ drawn according to distribution D. We denote by U_n the uniform distribution on n-bit strings, and by $(U_n)^k$ the uniform distribution on sequences of k n-bit strings.

It will be convenient to measure the distance between the effects of two distributions D_1, D_2 on a FSM Q by the $1-norm$ of the difference matrix $Q(D_1) - Q(D_2)$. For a vector $x \in \mathcal{R}^s$ we define $\|x\| = \sum |x_i|$, and for a $s \times s$ real matrix M we define

$$\|M\| = \sup_{0 \neq x \in \mathcal{R}^s} \frac{\|xM\|}{\|x\|}$$

All norms appearing in this paper are these 1-norms. The following facts are standard:

1. $\|M+N\| \leq \|M\| + \|N\|$
2. $\|MN\| \leq \|M\|\|N\|$
3. $\|M\| = \max_i \sum_j |M_{ij}|$
4. If each entry of an $s \times s$ matrix M is bounded in absolute value by ε then $\|M\| \leq s\varepsilon$.
5. If M is a transition probability matrix, i.e. all entries are non negative, and the sum of entries in each row is 1, then $\|M\| = 1$.

3.3. The Generator

Fix H, a universal family of hash functions $h: \{0,1\}^n \to \{0,1\}^n$. For every integer $k \geq 0$ we define a generator

$$G_k : \{0,1\}^n \times H^k \to (\{0,1\}^n)^{2^k}$$

G_k is defined recursively by

$$G_0(x) = x$$

and

$$G_k(x, h_1, \ldots, h_k) = G_{k-1}(x, h_1, \ldots, h_{k-1}) \circ G_{k-1}(h_k(x), h_1, \ldots, h_{k-1})$$

Here $\circ$ means concatenation of the two sequences of strings.

For any fixed choice of $h_1, \ldots, h_k$, denote by $G_k(*, h_1, \ldots, h_k)$ the distribution of $G_k(x, h_1, \ldots, h_k)$ induced by a random choice of x.

The generator has the property that for almost all choices of $h_1, \ldots, h_k$, the distribution $G_k(*, h_1, \ldots, h_k)$ is "close" to the uniform distribution. Here the closeness property is relative to a fixed FSM Q. We now define this exactly.

Definition 4. Let Q be a FSM over alphabet $\{0,1\}^n$, $\varepsilon > 0$, and $h_1, \ldots, h_k : \{0,1\}^n \to \{0,1\}^n$. Then the sequence $(h_1, \ldots, h_k)$ is called (ε, Q)-*good* if

$$\|Q(G_k(*, h_1, \ldots, h_k)) - Q((U_n)^{2^k})\| \leq \varepsilon$$

Lemma 2. *Let H be a universal family of hash functions $h : \{0,1\}^n \to \{0,1\}^n$. Let Q be a FSM of size 2^w over alphabet $\{0,1\}^n$, let $\varepsilon > 0$, and let k be any integer then*

$$Pr[(h_1, \ldots, h_k) \text{ is not } ((2^k - 1)\varepsilon, Q)\text{-good}] \leq \frac{2^{6w} k}{\varepsilon^2 2^n}$$

where $h_1, \ldots, h_k$ are chosen uniformly at random in H.

Proof. The proof is by induction on k. For $k = 0$ the statement is trivial. Assume it is true for $k-1$ and prove for k.

Choose $h_1, \ldots, h_k$ at random from H. For every fixed choice of $h_1, \ldots, h_{k-1}$, and for every two nodes i, j of Q define the sets:

$$B_{i,j}^{h_1, \ldots, h_{k-1}} = \{x |\ G_{k-1}(x, h_1, \ldots, h_{k-1}) \text{ takes } i \text{ to } j\}$$

Consider the following two events:

1. $(h_1, \ldots, h_{k-1})$ is $((2^{k-1} - 1)\varepsilon, Q)$-good. (Informally, $h_1, \ldots, h_{k-1}$ were chosen "well".)
2. For every triplet of nodes i, l, j: h_k is $(2^{-2w}\varepsilon, B_{i,l}^{h_1, \ldots, h_{k-1}}, B_{l,j}^{h_1, \ldots, h_{k-1}})$-independent. (Informally, h_k is chosen "well".)

We claim that (1) The probability that both events happen simultaneously is at least $1 - \frac{2^{6w} k}{\varepsilon^2 2^n}$, and (2) When both events happen, $(h_1, \ldots, h_k)$ is $((2^k - 1)\varepsilon, Q)$-good. These two claims imply the lemma.

Proof of Claim 1. The probability of event 1 not happening is bounded by the induction hypothesis to be at most $\frac{2^{6w}(k-1)}{\varepsilon^2 2^n}$. We now bound the probability of event 2 not happening. Consider a fixed choice of $h_1, \ldots, h_{k-1}$. By Lemma 1 we get that for any fixed triplet of nodes i, l, j the probability that h_k is not $(2^{-2w}\varepsilon, B_{i,l}^{h_1, \ldots, h_{k-1}}, B_{l,j}^{h_1, \ldots, h_{k-1}})$-independent is bounded by

$2^{4w}\varrho(B_{i,l}^{h_1,\ldots,h_{k-1}})\varepsilon^{-2}2^{-n}$. Summing up over all triplets i,l,j, and recalling that for every fixed i $\sum_l \varrho(B_{i,l}^{h_1,\ldots,h_{k-1}}) = 1$, we obtain that the probability of event 2 not happening is bounded by $\dfrac{2^{6w}}{\varepsilon^2 2^n}$. Adding the probabilities of events 1 and 2 not happening, we conclude the proof of claim 1.

Proof of Claim 2. Assume that events 1 and 2 hold. We can estimate $\|Q(G_k(*,h_1,\ldots,h_k)) - Q((U_n)^{2^k})\|$ from above by:

$$\|Q(G_k(*,h_1,\ldots,h_k)) - Q((U_n)^{2^k})\| \leq$$
$$\|Q(G_k(*,h_1,\ldots,h_k)) - Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))^2\| +$$
$$\|Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))^2 - Q((U_n)^{2^k})\|.$$

We will bound the first summand by ε, and the second by $(2^k-2)\varepsilon$. This implies claim 2, and will conclude the proof of the lemma.

Consider the matrix $Q(G_k(*,h_1,\ldots,h_k))$. By the definition of G_k, its i,j entry is given by

$$\sum_l Pr_x[x \in B_{i,l}^{h_1,\ldots,h_{k-1}} \text{ and } h_k(x) \in B_{l,j}^{h_1,\ldots,h_{k-1}}].$$

On the other hand, consider the matrix $Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))^2$. Its i,j entry is given by

$$\sum_l \varrho(B_{i,l}^{h_1,\ldots,h_{k-1}})\varrho(B_{l,j}^{h_1,\ldots,h_{k-1}}).$$

Since event 2 holds we get that each entry i,j of the matrix $Q(G_k(*,h_1,\ldots,h_k)) - Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))^2$ is bounded in absolute value by $2^{-w}\varepsilon$. It follows that the norm of this matrix is at most ε.

Now consider the second summand. Note that $Q((U_n)^{2^k}) = Q((U_n)^{2^{k-1}})^2$, thus the second summand can be rewritten as

$$\|Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))^2 - Q((U_n)^{2^{k-1}})^2\|.$$

Denote $Q(G_{k-1}(*,h_1,\ldots,h_{k-1}))$ by M, and $Q((U_n)^{2^{k-1}})$ by N, then we can bound

$$\|M^2 - N^2\| \leq \|M\|\|M-N\| + \|M-N\|\|N\|.$$

Event 1 means that $\|M-N\|$ is bounded by $(2^{k-1}-1)\varepsilon$. The norms of M and N are 1 since they are transition probability matrices. It follows that the second summand is bounded by $(2^k-2)\varepsilon$. ∎

In conclusion we have:

Lemma 3. *There exists a constant $c>0$ such that for all integers n and $k \leq cn$ we have that $G_k : \{0,1\}^n \times H^k \to (\{0,1\}^n)^{2^k}$ is a pseudorandom generator for space(cn) and block-size n with parameter 2^{-cn}.*

Proof. Let Q be a FSM, our aim is to bound the difference between $Pr[Q \text{ accepts } y]$ where y is chosen uniformly in $(\{0,1\}^n)^{2^k}$, and $Pr[Q \text{ accepts } G_k(x,h_1,\ldots,h_k)]$ where

x is chosen uniformly in $\{0,1\}^n$ and $h_1,\ldots,h_k$ in H. This difference we bound from above by

$$Pr[(h_1,\ldots,h_k) \text{ is not } (\varepsilon,Q)\text{-good}]+$$
$$\mid Pr[Q \text{ accepts } y] - Pr[Q \text{ accepts } G_k(x,h_1,\ldots,h_k) \mid h_1,\ldots,h_k \text{ is } (\varepsilon,Q)\text{-good}] \mid$$

(where ε is chosen below.)

Let us first evaluate the second term. Fix $\varepsilon > 0$ and any choice of $(h_1,\ldots,h_k)$ which is $(\varepsilon,Q)-good$. Let 1_{start} be the probability distribution concentrated on the starting state of Q, and let 1_{accept} be the vector having 1's on all the accepting states of Q (and 0's elsewhere). The probability that Q accepts y is given by $1_{start}\cdot Q((U_n)^{2^k})\cdot 1_{accept}$ and the probability that Q accepts $G(x,h_1,\ldots,h_k)$ is given by $1_{start}\cdot Q(G(*,h_1,\ldots,h_k))\cdot 1_{accept}$. The difference is thus given by

$$1_{start}\cdot(Q(G(*,h_1,\ldots,h_k)) - Q((U_n)^{2^k}))\cdot 1_{accept}$$

Since 1_{start} has a 1-norm of 1, and 1_{accept} has all of its entries bounded in absolute value by 1, the above expression is bounded in absolute value by

$$\|Q(G(*,h_1,\ldots,h_k)) - Q(U_n{}^{2^k})\| \le \varepsilon$$

Using Lemma 2, the probability that $(h_1,\ldots,h_k)$ is not (ε,Q)-good is bounded from above by $\dfrac{2^{2k}2^{6cn}k}{\varepsilon^2 2^n}$. We require the total difference in probability of acceptance to be at most 2^{-cn}, for all $k \le cn$. This is obtained when $\varepsilon + \dfrac{2^{2cn}2^{6cn}cn}{\varepsilon^2 2^n} \le 2^{-cn}$, an inequality which can be satisfied for e.g. $c=0.05$ and $\varepsilon = 2^{-cn-1}$. ∎

4. Generators for Space-Bounded Computation

Definition 5. A generator $G:\{0,1\}^m \to \{0,1\}^n$ is called a *pseudorandom generator for space(S) with parameter ε* if for every randomized *space(S)* algorithm A and every input to it we have that

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

where y is chosen uniformly at random in $\{0,1\}^n$, and x uniformly in $\{0,1\}^m$.

In the above definition, it is implied that the algorithm A is being run on its input while accessing the bits of y or of $G(x)$ as the random coin tosses. For a more detailed definition and discussion of pseudorandom generators for space bounded computation refer to [5].

Proposition 1. *Let $G:\{0,1\}^m \to (\{0,1\}^n)^k$ be a pseudorandom generator for space(S) and block size n with parameter ε then G is a pseudorandom generator for space(S) with parameter ε (where we concatenate the strings output by G to obtain a kn-bit long string).*

Proof. Given a *space(S)* algorithm A and an input to it, we build a FSM Q of size 2^S consisting of all of A's configurations. We label each edge (i,j) with the set of n-bit strings that cause A to move from configuration i to configuration j after being accepted as the next n coin tosses. The proposition now follows from definitions. ∎

By using the generators G_k obtained in section 3, using a family of universal hash functions with linear size descriptions we get:

Theorem 1. *For any $R=R(n)$ and $S=S(n)$ there exists an (explicitly given) pseudorandom generator $G:\{0,1\}^{O(S\log(R/S))}\to\{0,1\}^R$ for space(S) with parameter 2^{-S}. Moreover, G can be computed in polynomial time (in R and S) and $O(S\log R)$ space.*

This implies that randomized polynomial time algorithms that run in space(S) can be simulated using $O(S\log n)$ random bits.

Corollary 1. *Any randomized algorithm running in space(S) and using R random bits may be converted to one that uses only $O(S\log R)$ random bits (and runs in space$(O(S\log R))$).*

In fact, the extra $\log R$ factor in the space is only due to the necessity of storing the random bits. If the random bits are given as input (say, on a special tape of the Turing machine with 2-way access to it allowed) then the generator can be computed in space(S), and so can the simulation.

Pseudorandom generators for Logspace can be used for explicit constructions of universal traversal sequences. For definitions of universal traversal sequences see [1]. In [5] it is shown that the concatenation of all possible output strings of a pseudorandom generator for Logspace is a universal traversal sequence. Using our generator we obtain:

Theorem 2. *For all n and $2\le d\le n-1$, there exist (explicitly given) universal traversal sequences of length $n^{O(\log n)}$ for d-regular n-vertex graphs. Moreover, the sequences can be produced by a Turing machine running in space logarithmic in the length of the sequence.*

5. Pseudo-independent Block Generators

We are interested in generators whose output may be used to run several randomized algorithms, with the property that it "looks" as though the different algorithms got independent random strings. Formally:

Definition 6. Let $G:\{0,1\}^m\to(\{0,1\}^n)^k$, and $\varepsilon>0$, G is called a *pseudo-independent block generator with parameter* ε if for any sequence of sets $A_1,\ldots,A_k\subset\{0,1\}^n$ we have that

$$|Pr[y_1\in A_1 \text{ and } \ldots \text{ and } y_k\in A_k]-p_1\ldots p_k|\le\varepsilon$$

where y_i denotes the i'th n-bit string produced by G, $p_i=\frac{|A_i|}{2^n}$, and the probability is taken over a random input to G.

By going over the proofs of [11], it is not difficult to verify that the generator proposed there is in fact a pseudo-independent block generator. The generator uses only linearly (in R) many random bits and produces $k=O(\sqrt{R})$ strings with parameter $\varepsilon=\exp(-\sqrt{R})$. Our generator is also a pseudo-independent block generator, and can be used for larger values of R and smaller values of ε. It requires, however, slightly more random bits.

Proposition 2. *Let* $G : \{0,1\}^m \to (\{0,1\}^n)^k$ *be a pseudorandom generator for space*$(\log(k+2))$ *and block size* n *with parameter* ε*, then* G *is a pseudo-independent block generator with parameter* ε.

Proof. We build a FSM Q with states $0 \ldots k$ and an extra *fail* state. The start state is state 0, and the only accepting state is state k. Each edge $(i-1, i)$ is labeled with the set of n-bit strings in A_i, and each edge $(i-1, fail)$ is labeled with the strings not in A_i. The proposition now follows from definitions. ∎

Using our generator we obtain:

Theorem 3. *There exists a constant* $c > 0$ *such that for any integers* R *and* $k \leq 2^R$ *there exists an (explicitly given) pseudo-independent block generator with parameter* 2^{-R} *that converts* $cR\log k$ *random bits into* k *strings of length* R.

Proof. Use the generator assured by Lemma 3, for space R, block size R, parameter 2^{-R}, that produces k R-bit strings. (In fact, Lemma 3 gives larger block size, but excess bits can be thrown away). Then apply Proposition 2. ∎

As a special case we obtain deterministic amplification: Given any randomized algorithm that uses R random bits and has success probability $1/2$ and given $k \leq R$, we can run the algorithm k times using the output of our generator. This requires only $O(R\log k)$ random bits and will reduce the probability of failure to $(1+o(1))2^{-k}$. Notice also that even if the original algorithm had a success probability of only $1/poly(k)$ we could still run the algorithm $poly(k)$ times, reducing the failure probability to 2^{-k} and still using only $O(R\log k)$ random bits.

These results are all stated for algorithms with one-sided error[2], but they easily extend to algorithms which have 2-sided error (BPP-type algorithms)[3].

6. Other Applications

Many algorithms can be naturally broken into stages with the property that only small space is required *between* the different stages and each stage uses only a small number of random bits. In all these algorithms it is possible to reduce the number of random bits used by using the output of our generator instead of truly random bits (each stage of the algorithm gets a block that is output by the generator). We now give an example how a random n-bit prime can be chosen using only $O(n\log n)$ random bits. We believe that this is in itself interesting, but the main point we wish to make is that the techniques used are very general, and can be used for a variety of problems.

The following algorithm is the standard one used for uniformly generating prime numbers in the range $1 \ldots N$:

1. Repeat until success

[2] I.e. the algorithm is always correct on inputs not in the language and is correct with probability at least $1/2$ on inputs in the language. Thus if any run of it says "yes" we can immediately conclude that the input is in the language

[3] I.e. algorithms which are always correct with probability of at least $2/3$. In this case amplification is achieved by taking a majority vote of all runs of the algorithm.

1.1. Choose a random integer x in the range $1 \ldots N$.
1.2. Test: is x prime? If "yes" then success:=true.
2. Output x.

The expected number of times that the loop is performed until a prime number is found is approximately $\ln N = \Theta(n)$ (since the density of prime numbers in $1 \ldots N$ is approximately $1/\ln N$). Even assuming, for now, that we have a deterministic primality test, the algorithm requires an expected $\Theta(n^2)$ random bits. Using our generator we can reduce the number of random bits used to $O(n \log n)$ without assuming a deterministic primality test, but using, e.g., the Rabin-Miller randomized primality test [16].

The basic step in this primality test uses $O(n)$ random bits and detects non-primes with probability of at least $1/2$. To get a test that fails with exponentially small probability, the test is repeated $O(n)$ times. We can now break the algorithm into stages: Choosing a random x is a stage, and a basic primality test is a stage. Note that now each stage requires $O(n)$ random bits, and that the space required between any two stages is also $O(n)$ (for storing x). We can thus use the output of our generator for space $O(n)$ and block size $O(n)$, with parameter $2^{-\Omega(n)}$, instead of truly random bits. This requires only $O(n \log n)$ random bits.

Acknowledgements. I'd like to thank Russell Impagliazzo, Yishay Mansour, Avi Wigderson, Oded Goldreich, Shafi Goldwasser, Muli Safra, Mike Sipser, and Martin Tompa for helpful suggestions and discussions.

References

[1] R. ALELIUNAS, R. M. KARP, R. J. LIPTON, L. LOVÁSZ, and C. RACKOFF: Random walks, universal sequences and the complexity of maze problems, in: 20^{th} *Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico*, 1979, 218–223.

[2] M. AJTAI, J. KOMLÓS, and E. SZEMERÉDI: Deterministic simulation in logspace, In: *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing, New York City*, 1987, 132–141.

[3] M. AJTAI and A. WIGDERSON: Deterministic simulation of probabilistic constant depth circuits, In: 26^{th} *Annual Symposium on Foundations of Computer Science, Portland, Oregon*, 1985, 11–19.

[4] M. BLUM and S. MICALI: How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comp.*, **13**, (1984) 850–864.

[5] L. BABAI, N. NISAN, and M. SZEGEDY: Multiparty protocols and logspace-hard pseudorandom sequences, In: *Proceedings of the* 21^{st} *Annual ACM Symposium on Theory of Computing, Seattle, Washington*, 1989, 1–11.

[6] B. CHOR and O. GOLDREICH: On the power of two points biased sampling, Manuscript, 1986.

[7] L. CARTER and M. WEGMAN: Universal hash functions, *J. Comp. and Syst. Sci.* **18**, (1979) 143–154.

[8] A. COHEN and A. WIGDERSON: Dispersers, deterministic amplification, and weak random sources, In: 30^{th} *Annual Symposium on Foundations of Computer Science, Reseach Triangle Park, NC*, 1989, 14–19.

[9] R. IMPAGLIAZZO, L. LEVIN, and M. LUBY: Pseudorandom generation from one-way functions, In: *Proceedings of the* 21^{st} *Annual ACM Symposium on Theory of Computing, Seattle, Washington*, 1989, 12–24.

[10] S. ISTRAIL: Polynomial traversing sequences for cycles are constructable, In: *Proceedings of the* 20^{th} *Annual ACM Symposium on Theory of Computing*, 1988, 491–503.

[11] R. IMPAGLIAZZO and D. ZUCKERMAN: How to recycle random bits, In: 30^{th} *Annual Symposium on Foundations of Computer Science, Reseach Triangle Park, NC*, 1989, 248–253.

[12] R. KARP, N. PIPPENGER, and M. SIPSER: A time-randomness tradeoff, In: *AMS Conference on Probabilistic Computational Complexity*, 1985.

[13] H. KARLOFF, R. PATURI, and J. SIMON: Universal sequences of length $n^{O(\log n)}$ for cliques, *Inf. Proc. Let.* **28**, (1988) 241–243.

[14] Y. MANSOUR, N. NISAN, and P. TIWARI: The computational complexity of universal hashing, In: *Proceedings of the* 22^{nd} *Annual ACM Symposium on Theory of Computing*, 1990, 235–243.

[15] N. NISAN and A. WIGDERSON: Hardness vs. randomness, In: 29^{th} *Annual Symposium on Foundations of Computer Science, White Plains, New York*, 1988, 2–12.

[16] M. O. RABIN: Probabilistic algorithm for testing primality, *J. of Number Theory* **12**, (1980) 128–138.

[17] M. SANTHA: On using deterministic functions to reduce randomness in probabilistic algorithms, Manuscript, 1986.

[18] M. SIPSER: Expanders, randomness or time vs. space, *JCSS* **36** (1988), 379–383.

[19] U. VAZIRANI: Efficiency considerations in using semi-random sources, In: *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing, New York City*, 1987, 160–168.

[20] A. C. YAO: Theory and applications of trapdoor functions, In: 23^{rd} *Annual Symposium on Foundations of Computer Science*, 1982, 80–91.

Noam Nisan

Department of Computer Science,
Hebrew University of Jerusalem.
91904 Jerusalem, Israel.
noam@cs,huji.ac.il