COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA 11 (1) (1991) 63–70

PSEUDORANDOM BITS FOR CONSTANT DEPTH CIRCUITS

NOAM NISAN[1]

Received March 28, 1988

For every integer d we explicitly construct a family of functions (pseudo-random bit generators) that convert a polylogarithmic number of truly random bits to n bits that appear random to any family of circuits of polynomial size and depth d. The functions we construct are computable by a uniform family of circuits of polynomial size and constant depth. This allows us to simulate randomized constant depth polynomial size circuits in $DSPACE(polylog)$ and in $DTIME(2^{polylog})$. As a corollary we show that the complexity class AM is equal to the class of languages recognizable in NP with a random oracle. Our technique may be applied in order to get pseudo random generators for other complexity classes as well; a further paper [16] explores these issues.

1. Introduction

The relation between randomized and deterministic complexity classes is a fundamental question in complexity theory. Most of the known results do not achieve general simulation of randomized algorithms by straight forward deterministic algorithms. Known simulations use either alternation [18], non uniformity [1], [3], unproven assumptions [21], [10], [17], [19] or simulate only specific algorithms. One exception to this is a paper by Ajtai and Wigderson [4].

Ajtai and Wigderson consider deterministic simulation of randomized constant depth circuits (RAC^0). They explicitly construct (in logarithmic space) a series of functions (pseudo-random bit generators) that "stretch" n^ε truly random bits into n bits which appear random to any family of polynomial size constant depth circuits. This allows them to simulate RAC^0 in $DSPACE(n^\varepsilon)$, and thus in $DTIME(2^{n^\varepsilon})$ for any $\varepsilon > 0$.

In this paper we give a different construction of a pseudo random bit generator for RAC^0. Our construction is much simpler than the one in [4], and uses much fewer random bits. We build functions that "stretch" $(\log n)^c$ truly random bits to n bits which appear random to constant depth circuits. This allows us to simulate RAC^0 in $DSPACE(polylog)$ and thus in $DTIME(2^{polylog})$.

Theorem 1. *For any integer d, there exists a family of functions: $\{f_n : \{0,1\}^l \to \{0,1\}^n\}$, where $l = O((\log n)^{2d+6})$ such that:*
(1) $\{f_n\}$ can be computed by a log-space uniform family of circuits of polynomial size and $d+4$ depth.

AMS subject classification (1980): 68 C 25

[1] Part of this work was done while the first author was in U. C. Berkeley, visiting the Hebrew University of Jerusalem.

(2) *For any family* $\{C_n\}$ *of circuits of polynomial size and depth* d, *for any polynomial* $p(n)$, *and for all large enough* n:

$$|Pr\,(C_n(y) = 1) - Pr\,(C_n(f_n(x)) = 1)| \leq \frac{1}{p(n)}$$

where y *is chosen uniformly in* $\{0,1\}^n$, *and* x *is chosen uniformly in* $\{0,1\}^l$.

Denote by $RAC^0(BPAC^0)$ the set of languages that can be recognized by a uniform family of *probabilistic* constant depth, polynomial size circuits, with 1-sided error (2-sided error bounded away from 1/2 by some polynomially small fraction).

Theorem 2.

$$RAC^0 \subset BPAC^0 \subset \bigcup_c DSPACE((\log n)^c) \subset \bigcup_c DTIME(2^{(\log n)^c})$$

As a corollary of the existence of our generator we show that the class AM is equal to the class of languages that can be recognized in NP given a random oracle.

The correctness of our generator is based upon the known lower bounds for constant depth circuits ([2], [11], [22], and we use the results in [14]). The construction is based upon a new idea which allows extraction of many pseudo random bits, from any "hard" function.

2. The generator

2.1. Main lemma

Our generator is based upon the fact that parity is "hard" for constant depth circuits. We will be using the following theorem which is a corollary of Hastad's [14] results:

Theorem (Hastad). *For any family* $\{C_n\}$ *of circuits of depth* d *and size at most* $2^{\frac{1}{20}n^{\frac{1}{d+1}}}$, *and for all large enough* n:

$$|Pr\,(C_n(x) = parity(x)) - 1/2| \leq 2^{\frac{-1}{20}n^{\frac{1}{d+1}}}$$

When x *is chosen uniformly over all* n*-bit strings.*

Corollary 2.1. *Let* $\{C_n\}$ *be a family of circuits of depth* d, *size* $n^{O(1)}$, *and* $m = (\log n)^{d+2}$ *inputs, then for any polynomial* $p(n)$ *and for all large enough* n:

$$|Pr\,(C_n(x) = parity(x)) - 1/2| \leq \frac{1}{p(n)}$$

when x *is chosen uniformly over all* m*-bit strings.*

This corollary shows that the parity of the input bits "looks random" to any small constant depth circuit. The main idea of the generator is to compute the parity of many *nearly disjoint* subsets of the input bits.

Definition. A collection of sets $\{S_1, \ldots, S_n\}$, where $S_i \subset \{1, \ldots, l\}$ is called a *partial-(k, m)-design* if:
(1) For all i:

$$|S_i| = m$$

(2) For all $i \neq j$:

$$|S_i \cap S_j| \leq k$$

An n by l $0-1$ matrix is called a partial-(k, m)-*design* if its n rows, interpreted as sets over $\{1, \ldots, l\}$, are a partial-(k, m)-design.

Lemma 2.2. *Let $\{C_n\}$ be a family of circuits of depth d and polynomial size, let $m = m(n) = (\log n)^{d+3}$, $l = l(n)$, and let $\{A_n\}$ be a family of n by l matrices which are partial-$(\log n, m)$-designs, then for any polynomial $p(n)$, and for all large enough n:*

$$|Pr\,(C_n(y) = 0) - Pr\,(C_n(A_n x) = 0)| \leq \frac{1}{p(n)} \tag{I}$$

where y is chosen uniformly over all n-bit strings, x is chosen uniformly over all l-bit strings, and $A_n x$ is matrix–vector multiplication done over $GF(2)$.

Proof. We will assume (I) doesn't hold and derive a contradiction to corollary 2.1. We first show, as in [12] and in [21], that if (I) does not hold then one of the bits of $A_n x$ can be predicted from the previous ones.

For any i, $0 \leq i \leq n$, we define a distribution E_i on $\{0,1\}^n$ as follows: the first i bits are chosen to be the first i bits of $A_n x$, where x is chosen uniformly over l bit strings, and the other $n - i$ bits are chosen uniformly. Define

$$p_i = Pr(C_n(z) = 0)$$

where z is chosen according to the distribution E_i. We want to show that $|p_0 - p_n| \leq 1/p(n)$. Assume not, then there exists an i s.t. $|p_{i-1} - p_i| > 1/(np(n))$. Using this fact we can build a circuit that predicts the i'th bit.

Define a circuit D_n, which takes as input the first $i-1$ bits of $A_n x, y_1, \ldots, y_{i-1}$, and predicts y_i. D_n is a probabilistic circuit. It first flips $n - i + 1$ random bits, $r_i, \ldots, r_n$. On input $y =< y_1, \ldots, y_{i-1} >$, it computes $C_n(y_1, \ldots, y_{i-1}, r_i, \ldots, r_n)$. If this evaluates to 1 then D_n will return r_i as the answer, otherwise it will return the complement of r_i. As in [21] it can be shown that

$$\left| Pr\,(D_n(y_1, \ldots, y_{i-1}) = y_i) - \frac{1}{2} \right| > \frac{1}{np(n)}$$

where the probability is taken over all choices of x and of the random bits D_n uses. At this point an averaging argument shows that it is possible to set the private random bits that D_n uses to constants and achieve a deterministic circuit D'_n while preserving the bias.

By now we have constructed a circuit that predicts y_i from the bits $y_1, \ldots, y_{i-1}$. To achieve a contradiction to corollary 2.1. we will now transform this circuit to a circuit that predicts y_i from the bits $x_1, \ldots, x_l$. W.l.o.g. we can assume that y_i depends on $x_1, \ldots, x_m$, i.e.

$$y_i = \sum_{j=1}^{m} x_j \pmod 2$$

Since y_i does not depend on the other bits of x, it is possible to set the other bits to constants, while leaving the prediction of y_i valid. By an averaging argument there exist constants $c_{m+1}, \ldots, c_l$ such that setting $x_j = c_j$ for all $m < j \leq l$, preserves the prediction probability. At this point, however, each one of the bits $y_1, \ldots, y_{i-1}$ depends only on at most $\log n$ of the bits $x_1, \ldots, x_m$. This is so since the intersection of the sets of $x'_k s$ defininig y_i and y_j is bounded above in size by $\log n$ for each $j \neq i$. Now we can compute each y_j as a CNF (or DNF) formula of a polynomial (in n) size over the bits it uses. This gives us a circuit $D''_n(x_1, \ldots, x_m)$ that predicts y_i which is the parity of $x_1, \ldots, x_m$. It is easy to check that the size of D''_n is still polynomial in n, and that the depth is at most $d+1$. This contradicts corollary 2.1.

2.2. Construction

We will now construct a uniform family of polynomial size constant depth circuits, that uses only $(\log n)^{2d+6}$ truly random bits and produces n bits that appears random to any family of polynomial size depth d circuits. We will first give a construction of a family of n by l matrices $\{A_n\}$ which are partial-$(\log n, m)$-designs, where $l = O((\log n)^{2d+6})$, and $m = O((\log n)^{d+3})$ and then show how to compute $A_n x$.

A partial design

We need to construct n different subsets of $\{1 \cdots l\}$ of size m with small intersections. Let m be a prime power of size approximately $(\log n)^{d+3}$, and let $l = m^2$. Consider the numbers in the range $\{1 \cdots l\}$ as pairs of elements in $GF(m)$, i.e. we construct subsets of $\{< a, b > | a, b \in GF(m)\}$. Given any polynomial f on $GF(m)$, we define a set $S_f = \{< a, f(a) > | a \in GF(m)\}$. The sets we take are all of this form, where f ranges over polynomials of degree at most $\log n$. The following facts can now be easily verified:

(1) The size of each set is exactly m.

(2) Any two sets intersect in at most $\log n$ points.

(3) There are at least n different sets (the number of polynomials over $GF(m)$ of degree at most $\log n$ is $m^{\log n+1} \geq n$).

It should be noted that all that is needed to construct these sets effectively is simple arithmetic in $GF(m)$, and since m has a length of $O(\log \log n)$ bits, everything can be easily computed by a log-space bounded Turing machine.

The circuit

Each bit of the output, $A_n x$, is simply the parity of the corresponding subset of input bits. All the subsets are of size $m = (\log n)^{d+3}$. Computing the parity of this number of variables can be done in depth $d+4$, and polynomial size (in n), by a simple explicit construction.

3. AM vs. almost-NP

The existence of this pseudo random generator has implications concerning the class AM, which was defined by Babai [5].

An AM Turing machine is a machine that may use both randomization and nondeterminism, but in this order only. It first flips as many random bits as necessary and then uses nondeterminism. The machine is said to accept a language L if for every string in L the probability that there exists an accepting computation is at least 2/3, and for every string not in L the probability is at most 1/3 (the probability is over all random coin flips, and the existence is over all nondeterministic choices). The class AM is the set of languages accepted by some AM machine that runs in polynomial time. The randomization stage of the computation is called the "Arthur" stage and the second stage, the nondeterministic one is called the "Merlin" stage. For exact definitions as well as motivation refer to [5], [7], also see [13].

Let C be any complexity class (e.g. P, NP, ...). For an oracle A, we define the class C^A to be the set of languages L that are accepted by some oracle Turing machine M runing with the oracle A in the complexity class C. As in [7] we define the class *almost-C* to be the set of languages L such that:

$$Pr[L \in C^A] = 1$$

where A is an oracle chosen at random.

The following theorem is well known ([15], [6]), and underscores the importance of BPP as the random analogue of P:

Theorem. *BPP=almost-P*

[7] and [13] raised the question of whether AM = almost-NP? This would strengthen the feeling that AM is the probabilistic analogue of NP. Our results imply that this is indeed the case.

Theorem 3. *AM=almost-NP.*

Proof. Using standard techniques it is easy to see that AM $\subset$ almost-NP since a nondeterministic Turing machine can use the oracle queries to simulate Arthur. We will prove the other direction. We first state the following lemma which is similar to the case of BPP vs. almost-P ([15], BG):

Lemma 3.1. *If $L \in$ almost-NP then exists a specific nondeterministic oracle Turing machine M that runs in polynomial time such that for an oracle A chosen at random:*

$$Pr[M^A \text{accepts} L] \geq 2/3$$

We will simulate this machine by an AM machine. The difficulty in simulating this machine in AM relies in the fact that the machine may access (nondeterministically) an exponential number of locations of the oracle, but AM computations can only supply a polynomial number of random bits. We will use our generator to convert a polynomial number of random bits to an exponential number of bits that "look" random to the machine M.

Let the running time of M be n^k. We can view the computation of M as a large OR of size 2^{n^k} of all the deterministic polynomial time computations occurring for

the different nondeterministic choices. Each of these computations can be converted to a CNF formula of size 2^{n^k} over the oracle entries. All together the computation of M can be written as a depth 2 circuit of size at most 2^{2n^k} over the oracle queries.

Our generator can produce from $O(n^{10k})$ random bits 2^{2n^k} bits that look random to any depth 2 circuit of this size. So the simulation of M on a random oracle proceeds as follows: Arthur will flip $O(n^{10k})$ random bits, and the M will be simulated by Merlin; whenever M makes an oracle query, the answer will be generated from the random bits according to the generator. Note that this is just a parity function of some subset of the bits, which is clearly in P. Since the generator "fools" this circuit, the simulation will accept with approximately the same probability that M accepts on a random oracle.

Exactly the same technique suffices to show that for any computation in PH, the polynomial time hierarchy ([20], [9]), a random oracle can be substituted by an "Arthur" phase. Applying to this a result by Sipser [18] and Gacs showing that $BPP \subset \Sigma_2 \cap \Pi_2$ allows simulation of the "Arthur" phase by one more alternation and thus we get:

Theorem 4. *almost-PH=PH*

4. Further results

How tight are these results?

In order to produce n bits that "look" random to all polynomial size depth d circuits our generator uses $O((\log n)^{c(d)})$ random bits, where $c(d)$ is some function of the depth. It is interesting to ask whether there is a generator that converts $O((\log n)^c)$ random bits for some fixed c to n bits that "fool" *all* constant depth circuits. Such a result is, however, well beyond reach in the current "state of the art". Any proof that a generator that uses only $O((\log n)^{o(d)})$ random bits produces n bits that fool all depth d circuits would also imply a super logarithmic lower bound for circuit depth (for the problem of "inverting" that generator). Thus the number of bits we use is almost optimal in the "current state of the art".

Simulating larger circuits

The parameters in our results can be varied in order to get generators that look random even to larger circuits. For example in order to "fool" all constant depth circuits of size $2^{polylog}$, it also suffices to use a polylog number of random bits. In particular, probabilistic constant depth circuits of size $2^{polylog}$ can be simulated in $DTIME(2^{polylog})$.

Better approximation

The parametes in our results can also be varied in order to get generators which approximate the random distribution more closely. For example in order to achieve a generator that approximates the probability of acceptance of a circuit to within some (additive) exponentially small fraction, $2^{-n^{\delta}}$, it suffices to use n^{ε} random bits for some $\varepsilon = \varepsilon(\delta)$.

Generators for other complexity classes

The technique described here, may be used to produce pseudo-random bit generators for any complexity class, given a function that is "hard" for that class. A further paper [16] explores these possibilities. For example we can show that Yao's result [21] that $BPP \subset DTIME(2^{n^{\varepsilon}})$ for every $\varepsilon > 0$, follows from the much weaker hypothesis than the one required by Yao, namely that there exists a function in EXPTIME such that any polynomial size circuit that attempts to compute it errs on some polynomially small fraction of the inputs.

We also show results to the effect that if such pseudo random generators do not exist then some nontrivial simulation of Turing machine time by space is possible. Specifically we show that if BPP is not contained in $DTIME(2^{n^{\varepsilon}})$ for every $\varepsilon > 0$, then there exists some $\varepsilon > 0$ such that for any time bound $T(n) \geq C^n$, any function in $DTIME(T(n))$ has an algorithm for it that for infinitely many n runs in space $DSPACE(T(n)^{1-\varepsilon})$ on any x of length n. A stronger implication of this form was proven by Sipser [19], but under an unproven assumption about certain kinds of expanders.

These kinds of results can also be obtained *in a unified fashion* for all other natural complexity classes. For example if there exists some function in PSPACE that requires circuit *depth* of n^{ε} in order to compute it (with some polynomially small fraction of error) then randomized circuits of poly-logarithmic depth can be simulated (uniformly) by deterministic circuits of poly-logarithmic depth. in particular $RNC \subset DSPACE(polylog)$.

5. Acknowledgements

I would like to thank László Babai for suggesting AM=almost-NP? as an application of our result. I would like to thank Avi Wigderson for much help and collaboration on this work.

References

[1] L. ADLEMAN: Two theorems on random polynomial time, *19th FOCS* pp. 75–83, 1978.

[2] M. AJTAI: Σ_1^1 formulas on finite structures, *Annals of Pure and Applied Logic* **24**, pp. 1–48. 1983.

[3] M. AJTAI, and M. BEN-OR: A theorem on probabilistic constant depth computations, *16th STOC*, pp. 471–474, 1984.

[4] M. AJTAI, and A. WIGDERSON: Deterministic simulation of probabilistic constant depth circuits *26th FOCS*, pp. 11–19, 1985.

[5] L. BABAI: Trading group theory for randomness *17th STOC*, pp. 421–429, 1975.

[6] C. H. BENETT, and J. GILL: Relative to a random oracle A, $P^A \neq NP^A \neq Co-NP^A$ with probability 1. *SIAM J. Comp.* **10**, 1981.

[7] L. BABAI, and S. MORAN: Arthur Merlin games: a randomized proof system, and a hierarchy of complexity classes, *J. Computer Sys. Sci.* **36**, pp. 254–276, 1988.

[8] M. BLUM, and S. MICALI: How to generate cryptographically strong sequences of pseudo random bits. *23rd FOCS*, pp. 112–117, 1982.

[9] A. CHANDRA, D. KOZEN, AND L. STOCKMEYER: Alternation, *J. ACM*, **28**, 1981.

[10] M. FURST, R. J. LIPTON, and L. STOCLMEYER: Pseudo random number generation and space complexity, *Information and Control*, **64**, 1985.

[11] M. FURST, J. SAXE, and M. SIPSER: Parity, Circuits, and the polynomial time hierarchy, *22nd FOCS*, pp. 260–270, 1981.

[12] S. GOLDWASSER, and S MICALI: Probabilistic Encryption, *JCSS*, **28**, No. 2, 1984.

[13] S. GOLDWASSER, and M. SIPSER: Private coins vs. Public voins in interactive proof systems, *18th STOC*, pp. 59–68, 1986.

[14] J. HASTAD: *Lower Bounds for the Size of Parity Circuits*, Ph.D. Thesis, M.I.T., 1987.

[15] S. A. KURTS: A note on randomized polynomial time, *SIAM J. Comp.*, **16**, No. 5, 1987.

[16] N. NISAN, and A. WIGDERSON: Hardness vs. Randomness, *29th FOCS*, 1988.

[17] J. H. REIF, and J. D. TYGAR: Towards a theory of parallel randomized computation, *TR-07-84, Aiken Computation Lab.*, Harvard University, 1984.

[18] M. SIPSER: A complexity theoretic approach to randomness, *15th STOC*, 330–335, 1983.

[19] M. SIPSER: Expanders, Randomness, or Time vs. Space, Structure in Complexity Theory, Lecture notes in Computer Science, No. 223, Ed. G. Goos, J. Hartmanis, pp. 325–329.

[20] L. STOCKMEYER: The polynomial time hierarchy, *Theor. Comp. Sci.* **3**, No. 1, 1976.

[21] A. C. YAO: Theory and applications of trapdoor functions, *23rd FOCS*, pp. 80–91, 1982.

[22] A. C. YAO: Separating the polynomial time hierarchy by oracles, *26th FOCS*, pp. 1–10, 1985.

Noam Nisan

Department of Computer Science
Hebrew University of Jerusalem
Israel