Hardness vs. Randomness

(Extended Abstract)

Noam Nisan [1]

Computer Science division
University of California, Berkeley

Avi Wigderson [2]

Department of Computer Science
Hebrew University of Jerusalem

ABSTRACT

We present a simple new construction of a pseudo-random bit generator. It stretches a short string of truly random bits into a long string that looks random to any algorithm from a complexity class C (eg $P, NC, PSPACE, ...$) using an *arbitrary* function that is hard for C.

This generator reveals an *equivalence* between the problems of proving lower bounds and the problem of generating good pseudo-random sequences. Combining this construction with other arguments we obtain the following consequences (among others).

(1) If EXPTIME cannot be approximated by polynomial size circuits then $BPP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$. (Compare with [Ya])

(2) If PSPACE can not be approximated by NC circuits then $RNC \subset \bigcap_{\varepsilon>0} DSPACE(n^\varepsilon)$. (Compare with [RT])

(3) $RAC^0 \subset DSPACE((\log n)^{O(1)})$. (Compare with [AW])

(4) Almost-NP = AM. (Solves an open problem of Babai and Sipser)

(5) We present a simple new proof that $BPP \subset \Sigma_2 \cap \Pi_2$. (Compare with [Si1])

(6) Either $BPP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$ or $DTIME(T(n)) \subset DSPACE(T^{1-\varepsilon}(n))$ for infinitely many n, some $\varepsilon>0$ and all sufficiently large time bounds $T(n)$. (Compare with [Si2])

1. Introduction

The fundamental idea of trading hardness for randomness is due to Shamir [Sh], who suggested that the RSA function can be used to construct good pseudo-random sequences. The first secure pseudo-random bit-generator was built by Blum and Micali [BlM], who used the intractability of the Discrete Logarithm function. These ideas were generalized by Yao [Ya], who showed that any one-way permutation can be used to construct generators that fool every polynomial time computation. This result gave the first explicit hardness-randomness trade-off: if no poly-size circuit can invert the one- way permutation, then $RP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$.

In all these papers, the generator uses the one-way function f as follows. From a random string X_0 (the seed), it computes a sequence $\{X_i\}$ by $X_{i+1} = f(X_i)$. The output bits b_i depend on this sequence. The heart of the argument is then showing that a small circuit that is not fooled by the bit sequence $\{b_i\}$ can be used to compute f^{-1}, contradicting its assumed hardness. A deterministic simulation of a randomized algorithm then proceeds by trying all possible seeds X_0.

Two striking features of this generator are:

1 Part of this work was done while the first author was visiting the Department of Computer Science at the Hebrew University of Jerusalem.

2 Supported by Israel National Academy of Science grant No. 328071, by the Alon Fellowship, and by NSF grant CCR8612563.

0272-5428/88/0000/0002$01.00 © 1988 IEEE

(1) It is sequential.

(2) It is secure and efficient only if f is easy to compute, i.e. it requires a lower bound for a function in $NP \bigcap co-NP$.

We propose here a new construction of a generator which avoids both problems. It can be based on the hardness of an arbitrary function in $EXPTIME$, and is completely parallel. Simply, each output bit b_i is computed by applying f to a subset S_i of the bits in the seed X_0. These subsets (up to exponentially many) are explicitly described and have the property that every pair of them is nearly disjoint. Here this property is the heart of ensuring the quality of the bits. Clearly, the efficiency of the generator depends (given the complexity of f) on the size of the sets S_i, which is judiciously chosen to equal the time of trying all possible seeds.

This new construction enables us to obtain much sharper hardness-randomness trade-offs, as well as other complexity theoretic results. To state them we use the following notation. For a language $L \subset \{0,1\}^*$, and complexity classes C and C' we say

Definition:

(1) *C cannot approximate L* if for some $k>0$ every algorithm in C that attempts to compute L errs on at least n^{-k} fraction of the inputs of every sufficiently large length n.

(2) *C cannot approximate C'* if there exists some problem in C' that cannot be approximated by C.

Theorem 2: For every function (size bound) $m \leq s(m) \leq 2^m$ the following are equivalent:

(1) For some $c>0$ EXPTIME cannot be approximated by circuits of size $s(m^c)$.

(2) For some $c>0$ there exists a Pseudo-random generator $\{G_m: \{0,1\}^m \rightarrow \{0,1\}^{s(m^c)}\}$ that runs in time exponential in m and its output looks random to any circuit of size $s(m^c)$.

This allows us to give an efficient simulation of probabilistic algorithms.

Theorem 3: If EXPTIME cannot be approximated by polynomial size circuits then $RP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$.

Comment: All our simulation results are stated for one-sided error classes, but obviously hold for the two-sided analogs as well.

In [Y1], the same consequence is obtained under an assumption which is stronger than $NP \bigcap Co-NP$ cannot be approximated by polynomial size circuits.

Reif and Tygar [RT] considered simulation of probabilistic parallel algorithms under intractability assumptions. They showed how to parallelize the original generator if a particular function (inverse mod p) in P cannot be approximated by NC circuits, and obtain $RNC \subset \bigcap_{\varepsilon>0} DSPACE(n^\varepsilon)$. As our construction is parallel, we can obtain:

Theorem 4: If PSPACE cannot be approximated by NC circuits then $RNC \subset \bigcap_{\varepsilon>0} DSPACE(n^\varepsilon)$

Ajtai and Wigderson [AW] studied the simulation of probabilistic constant- depth circuits, since for them lower bounds existed. They devised a complicated generator, based on the proof methods of the parity lower bound, that gave the first nontrivial simulation result: $RAC^0 \subset \bigcap_{\varepsilon>0} DSPACE(n^\varepsilon)$. Our construction enables us to use directly the best parity lower bound [Ha] and obtain

Theorem 6: $RAC^0 \subset \bigcup_c DSPACE((\log n)^c)$

The power of random oracles is an old subject of interest [BG, BGS]. For a complexity class C, define $almost-C = \{L: L \in C^A$ for almost all oracles $A\}$.

Baker and Gill [BG] proved that $almost-P = BPP$, suggesting that BPP is the right probabilistic analog of P. Babai [Ba] introduced the class AM (Arthur-Merlin games) and proposed it as a probabilistic analog of NP. We justify this intuition, answering an open question of Babai and Sipser (see [BM]) in

Theorem 7: $almost-NP = AM$.

The proof relies on a description of $almost-NP$ as a probabilistic, exponential size, constant-depth circuit, and our generator. A similar consideration, together with Sipser's result that $BPP \subset PH$ [Si1], (which we give a simple new proof for), implies the surprising fact that random oracles do not help the polynomial time hierarchy.

Theorem 8: $almost-PH = PH$.

In [Si2] Sipser made the striking observation that efficient deterministic simulation of probabilistic algorithms is intimately related to efficient simulation of time by space.*

Under an assumption on the possibility to explicitly construct certain strong expanders he proved that either $RP = P$, or else $DTIME(t(n)) \subset DSPACE(t(n)^{1-\varepsilon})$ for some $\varepsilon > 0$. (this space simulation is significantly better than the best unconditional bound of $t(n) / \log t(n)$ of Hopcroft, Paul and Valiant [HPV]).

We use our generator to give a completely different proof of a slightly weaker relation, but using no unproven assumption.

Theorem 10: One of the following holds:
(1) $BPP \subset \bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$
(2) There exist $\varepsilon > 0$ s.t. $DTIME(t(n)) \subset DSPACE(t(n)^{1-\varepsilon})$.

2. The generator

2.1. Hardness

The assumption under which we construct a generator is the existence of a "hard" function. By "hard" we need not only that the function can not be computed by small circuits but also that it can not be *approximated* by small circuits. There are basically two parameters to consider: the size of the circuit and the closeness of approximation.

Definition: Let $f_n:\{0,1\}^n \to \{0,1\}$. Let C be a boolean circuit with n input bits. Define the *bias* C achieves on f to be:

$$bias(C,f) = \left| Pr\left[C(x) = f(x)\right] - Pr\left[C(x) \neq f(x)\right] \right|$$

Where x is chosen uniformly in $\{0,1\}^n$.

Definition: Let $f:\{0,1\}^* \to \{0,1\}$ be a boolean function. Let $\{f_n:\{0,1\}^n \to \{0,1\}\}$ be its restrictions to strings of length n. The *Hardness* of f at n, $H_f(n)$ is defined defined to be the maximum integer S such that for any circuit C of size at most S

$$bias(C,f_n) \leq \frac{1}{S}$$

*) In the sequel we restrict ourselves to a weak notion of efficient space simulation, i.e. one that works for large enough (super exponential) time functions $t(n)$ and infinitely many n.

We have chosen to use only one parameter for the hardness instead of two separate ones, in order to simplify notation and as Yao [Ya] showed robustness under the "closeness of approximation". Yao had shown that by taking multiple copies of a function and XORing the results, the bias achievable on a function can be decreased at will. For example we will note the following corollary of Yao's XOR lemma:

Lemma (Yao): If there is a function f in EXPTIME that cannot be approximated by polynomial (subexponential) size circuits then there exists a function f' in EXPTIME with superpolynomial (exponential (i.e. 2^{n^ε} for some $\varepsilon > 0$)) hardness.

Our generator will be based on a function with a high "hardness".

2.2. The construction

Given a "hard" function, it is easy to generate one pseudo random bit from it since the value of the function must look random to any small circuit. The problem is to generate more than one pseudorandom bit. In order to do that we will compute the function on many different, nearly disjoint subsets of bits.

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

(1) For all i:

$$\left|S_i\right| = m$$

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

$$\left|S_i \bigcap S_j\right| \leq k$$

A $n*l$ 0-1 matrix is called a (k,m)-design if its n rows, interpreted as subsets of $\{1..l\}$ are a (k,m)-design.

Definition: Let A be a l by n 0-1 matrix, let f be a boolean function, and let $x = (x_1 \cdots x_l)$ be a boolean string. Denote by $f_A(x)$ the n bit vector of bits computed by applying the function f to the subsets of the x's denoted by the n different rows of A.

Our generator expands the seed x to the pseudo random string $f_A(x)$. The quality of the bits is assured by the main lemma.

Main Lemma: Let f be a boolean function. Let $\{C_n\}$ be a family of circuits of size n (over n bits). Let $l = l(n)$, $m = m(n)$ where $H_f(m) \geq n^2$, and let $\{A_n\}$ be a family of $n*l$ matrices which

are $(\log n, m)$ designs, then for any constant $\varepsilon > 0$, and for all large enough n:

$$\left| Pr\left[C_n(y)=0\right] - Pr\left[C_n(f_{A_n}(x))=0\right] \right| \leq \varepsilon \quad \text{(I)}$$

Where y is chosen uniformly over all n-bit strings and x is chosen uniformly over all l-bit strings.

Proof: We will assume (I) doesn't hold and derive a contradiction to the hardness assumption. We first show, as in [GM] and in [Ya], that if (I) does not hold then one of the bits of $f_{A_n}(x)$ 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 $f_{A_n}(x)$, where x is chosen uniformly over l bit strings, and the other $n-i$ bits are chosen uniformly. Define

$$p_i = Pr\left[C_n(z)=0\right]$$

where z is chosen according to the distribution E_i. We want to show that $|p_0 - p_n| \leq \varepsilon$. Assume not, then there exists an i s.t. $|p_{i-1} - p_i| > \varepsilon/n$. 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 $f_{A_n}(x)$, $y_1, \cdots, y_{i-1}$, and predicts y_i. D_n is a probabilistic circuit. It first flips $n-i+1$ random bits, $r_i, \cdots, r_n$. On input $y = \langle y_1, \cdots, y_{i-1} \rangle$, it computes $C_n(y_1, \cdots, y_{i-1}, r_i, \cdots, r_n)$. 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 [Ya] it can be shown that

$$\left| Pr\left[D_n(y_1, \cdots, y_{i-1}) = y_i\right] - \frac{1}{2} \right| > \frac{\varepsilon}{n}$$

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

$$y_i = f(x_1 \cdots x_m)$$

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 $c_{m+1}, \cdots, c_l$ 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 $y_1, \cdots, y_{i-1}$ depends only on at most $\log n$ of the bits $x_1, \cdots, x_m$. This is so since the intersection of the set of x_k's defined by y_i and by y_j is bounded from above by $\log n$ for each $i \neq j$. Now we can compute each y_i as a CNF (or DNF) formula of a linear (in n) size over the bits it uses. This gives us a circuit $D''_n(x_1, \cdots x_m)$ that predicts y_i which is $f(x_1, \cdots, x_m)$. It is easy to check that the size of D''_n is at most $O(n^2)$, and the bias achieved is more than n^{-2}, which contradicts the assumption that $H_f(m) > n^2$. ■

2.3. Construction of nearly disjoint sets

Given a hard function f, with hardness $H_f(n)$, we need to build designs for it for every n, where the requirement that was given by the previous lemma is that $H_f(m) \geq n^2$. Our aim will be to minimize l, the number of random bits used.

For every n and every order of m in the range $n \geq m \geq \log n$, we construct a a family of n by l matrices $\{A_n\}$ which are $(\log n, m)$-designs, where $l = m^2$.

A design

We need to construct n different subsets of $\{1 \cdots l\}$ of size m with small intersections. Assume m is a prime power, 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 $\{\langle a,b \rangle \mid a,b \in GF(m)\}$. Given any polynomial q on GF(m), we define a set $S_q = \{\langle a, q(a) \rangle \mid a \in GF(m)\}$. The sets we take are all of this form, where q 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 $O(\log n)$ bits, everything can be easily computed by a log-space bounded Turing machine.

Other designs

The previous design is optimal up to a factor of $\log n$, i.e for given m and n, l is within a log factor of the design with the smallest value of l. For most values of m this small added factor is not so important, however when $\log n \le m < \log^2 n$ we may wish to do better. In particular when $m = O(\log n)$ we want to achieve $l = O(\log n)$.

One way to achieve a better design for small values of m is to consider multivariate polynomials over finite fields. These multinomials may define sets in a similar manner as in the previous design, and for small values of m, l can be reduced up to about $m \log m$. We leave the details to the interested reader.

For the case $m = C \log n$ for some $C > 1$, we do not have an explicit construction of a design where $l = O(\log n)$, but we note that such a design can be computed in polynomial time! A simple counting argument shows that for $l = 2C^2 \log n$, a design can be obtained by *greedily* choosing subsets of cardinality m, which intersect each of the previously chosen sets at less than $\log n$ points.

2.4. Main Theorem

The previous sections have shown how to generate pseudo random bits, given a "hard" function. The generation is not always in polynomial time, but if the original hard function had, say, exponential complexity then the generator works in time exponential in m and not in n! Thus we first get the following theorem:

Theorem 1: If there exists a function f computable in $DTIME(T(n))$ with hardness $H_f(n)$ then there exists a family of functions (pseudo random bit generators) $\{G_n : \{0,1\}^l \to \{0,1\}^n\}$ with the following properties:

(1) $l = (H_f^{-1}(n^2))^2$ where H_f^{-1} is the inverse function of H_f.

(2) G_n can be computed in $DTIME(nT(l))$.

(3) For any family of circuits $\{C_n\}$ of size n, for any $\varepsilon > 0$ and for all large enough n:

$$Pr\left[\left[C(y)=1\right] - Pr\left[C(G(x))=1\right]\right] < \varepsilon$$

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

Proof: The generator will be $G_n(x) = f_{A_n}(x)$, where A_n is the design described in section 2.3 for $m = H^{-1}{}_f(n^2)$. The parameters of the design ensure part (1); part (2) is a consequence of the fact that the function f is evaluated n times, each time on at most l bits, and of the fact that the design can be constructed easily; and part (3) is the statement of the main lemma. ■

Our construction has many consequences. Exact statements of the corollaries as well as sketches of their proofs are given in the next section.

3. Main corollaries

3.1. Sequential computation

Our construction is nearly optimal. Theorem 2 gives the Hardness-Randomness tradeoff. It also shows that the problem of getting a single extra pseudo random bit (designing an *extender*) is equivalent to getting as many of them as needed (this remains true even in parallel!).

Theorem 2: For every function (size bound) $m \le s(m) \le 2^m$ the following are equivalent:

(1) For some $c > 0$ EXPTIME cannot be approximated by circuits of size $s(m^c)$.

(2) For some $c > 0$ There exists a function in EXPTIME with hardness $s(m^c)$.

(3) For some $c > 0$ there exists an *extender* $\{G_m : \{0,1\}^m \to \{0,1\}^{m+1}\}$ that runs in time exponential in m and its output looks random to any circuit of size $s(m^c)$.

(4) For some $c > 0$ There exists a Pseudo-random generator $\{G_m : \{0,1\}^m \to \{0,1\}^{s(m^c)}\}$ that runs in time exponential in m and its output looks random to any circuit of size $s(m^c)$.

Proof: (1) -> (2) is exactly Yao's Xor lemma. (2) -> (4) is the statement of theorem 1. (4) -> (3) is trivial. We will show (3) -> (1).

Let $G = \{G_m\}$ be an extender as in (3). Consider the problem of "Is y in the range of G?". It can be easily seen that this can be computed in exponential time; however, no circuit of size $s(m^c)$ can approximate it since that circuit would distinguish between the output of G

and between truelly random strings. ■

We can now give our results regarding the simulation of BPP.

Theorem 3: If there exists a function computable in $DTIME(2^n)$,

(1) That cannot be approximated by polynomial size circuits. Or

(2) That cannot be approximated by circuits of size 2^{n^ε} for some $\varepsilon>0$. Or

(3) with hardness $2^{\varepsilon n}$ for some $\varepsilon>0$.

Then

(1) $BPP\subset\bigcap_{\varepsilon>0} DTIME(2^{n^\varepsilon})$

(2) $BPP\subset DTIME(2^{(\log n)^c})$ for some constant c.

(3) BPP = P

respectively.

3.2. Parallel computation

The construction of the generator was very general, it only depended on the the existence of a function that was hard for the class the generator is intended for. In particular the depth of the circuit D_n in the proof of the main lemma only increased slightly. Thus we can get similar simulation results for other complexity classes under the analogues assumptions. For example:

Theorem 4: If there exists a function in PSPACE that

(1) cannot be approximated by NC circuits. Or

(2) cannot be approximated by circuits of *depth* n^ε (for some constant $\varepsilon>0$).

Then

(1) $RNC\subset\bigcap_{\varepsilon>0} DSPACE(n^\varepsilon)$.

(2) $RNC\subset DSPACE(polylog)$.

Respectively.

A special case of interset is the class of constant depth circuits. For this class of circuits lower bounds are known!! The following section will give results regarding simulation of randomized constant depth circuits which are proven without any unproven assumption!

3.3. Constant depth circuits

The results in this subsection and in the following one appear in a more complete form in a previous paper of ours [NW].

Our generator is based on the known lower bounds for constant depth circuits computing the parity function. We will use directly the strongest bounds known due to Hastad [Ha].

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

$$\left|Pr\left[C_n(x)=parity(x)\right]-1/2\right| \le 2^{n^{\frac{1}{d+1}}}$$

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

Applying to this our construction we get:

Theorem 5: For any integer d, there exists a family of functions: $\{G_n:\{0,1\}^l\to\{0,1\}^n\}$, where $l=O((\log n)^{2d+6})$ such that:

(1) $\{G_n\}$ can be computed by a log-space uniform family of circuits of polynomial size and $d+4$ depth.

(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:

$$\left|Pr\left[C_n(y)=1\right] - Pr\left[C_n(G_n(x))=1\right]\right| \le \frac{1}{p(n)}$$

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

Proof: Again $G_n=f_{A_n}$ where f is the parity function, and A_n is the design described in section 2.3 for $m=(\log n)^{d+3}$. Notice that: (1) the generator can be computed by polynomial size circuits of depth $d+4$ since it is just the parity of sets of bits of cardinality $(\log n)^{d+3}$. (2) All the considerations in the proof of correctness of the generator apply also to constant depth circuits. In particular the depth of the circuit D_n in the proof increases only by one. ■

We can now state the simulation results we get for randomized constant depth circuits. 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 6:

$$BPAC^0, RAC^0 \subset \bigcup_c DSPACE((\log n)^c)$$

and

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

■

Denote by #DNF the problem of counting the number of satisfying assignments to a DNF formula, and by Approx-#DNF the problem of computing a number which is within a factor of 2 (or even $1+n^{-k}$) from the correct value. Clearly #DNF is #P complete. However, our results imply that:

Corollary 3.1: Approx-#DNF $\in DTIME(2^{(\log n)^{14}})$

Proof: Karp and Luby [KLu] give a probabilistic algorithm for Approx-#DNF. It is not difficult to see that their algorithm can be implemented by RAC^0 Circuits of depth 4.

3.4. Random oracles

The existence of a pseudo random generator for constant depth circuits has implications concerning the power of random oracles for classes in the polynomial time heirarchy.

Let C be any complexity class (e.g. P, NP, ...). As in [BM] we define the class $almost-C$ to be the set of languages L such that:

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

where A is an oracle chosen at random. The class $almost-C$ can be thought of as a natural probabilistic analogue of the class C.

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

Theorem: BPP = almost-P ■

Babai [Ba] introduced the class AM. An AM Turing machine is a machine that may use both randomization and nondeterminisim, but in this order only, first flip as many random bits as necessary and then use nondeterminisim. 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 [Ba], [BM], also see [GS].

[BM] and [GS] 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 7: AM = almost-NP.

Proof: the fact that AM $\subset$ almost-NP is trivial 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 ([Ku], [BG]):

Lemma 3.2: If L $\in$ almost-NP then there exists a specific nondeterministic oracle Turing machine M that runs in polynomial time such that for an oracle A chosen at random:

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

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 (non deterministically) 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 $2n^{10k}$ 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 $2n^{10k}$ random bits, and then 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 ([St], [CKS]), a random oracle can be substituted by an "Arthur" phase. Applying to this Sipser's [Si1] result that $BPP \subset \Sigma_2 \bigcap \Pi_2$ allows simulation of the "Arthur" phase by one more alternation and thus we get:

Theorem 8: almost-PH = PH ■

3.5. BPP and the polynomial time hierarchy

In [Si1] Sipser showed that BPP could be simulated in the polynomial time hierarchy. Gacs improved this result and showed simulation is possible in $\Sigma_2 \bigcap \Pi_2$. In this section we give a new simple proof of this fact.

Theorem 9 (Sipser, Gacs): $BPP \subset \Sigma_2 \bigcap \Pi_2$.

Proof: It suffices to show that $BPP \subset \Sigma_2$. The main idea is that a pseudorandom generator that stretches $O(\log n)$ random bits to n pseudorandom bits can be constructed in Σ_2. To simulate BPP then, a Σ_2 machine will then run over all of the polynomially many possabilities of the random seed.

To get such a psedorandom generator, using our construction, we only need a function with exponential hardness (specifically we want a function on $O(\log n)$ bits with hardness which is $\Omega(n^2)$). Such a function can be found in Σ_2: A simple counting argument shows that such a function exists (although non uniformly), and verifying that a function on $O(\log n)$ bits has indeed a high hardness can easily be seen to be in Co-NP. (The function can be described by a polynomial size table, and the verification can be done by nondeterministically trying all circuits of size n^2).

Thus the simulation of BPP will proceed as follows: (1) Nondeterministically guess a function on $O(\log n)$ bits with high hardness (first alternation). (2) Verify it is indeed hard (Second alternation). (3) Use it as a basis for the pseudorandom generator, using our construction. (4) Try all possable seeds. ■

Actually, this proves a slightly stronger statement, namely that $BPP \subset ZPP^{NP}$. (ZPP^{NP} is the class of languages that have polynomial time, randomized, *zero error* algorithms, using an NP-complete oracle).

3.6. Randomness and Time vs. Space

Our generator is based on assumption that there exists a function in, say, $DTIME(2^n)$, that can not be approximated by small circuits. In this section we show that if this assumption does not hold then some nontrivial simulation of space by time is possible.

Consider the following function: On input $<M,x,t>$ the output is a representation of what Turing Machine M does on input x at time t. Where the representation includes the state the machine is in and the location of the heads. Moreover, consider a language L which encodes this function.

Hypothesis H1: L has circuits for it of size $2^{(1-\varepsilon)n}$ for some $\varepsilon > 0$.

We will show that if hypothesis H1 is true then some non trivial simulation of time by space is possible, and if it is false then we can use our construction to get a pseudo random bit generator.

Lemma 3.3: If Hypothesis H1 is true for all sufficiently large n then for some constants $C > 1$ and $\varepsilon > 0$, and for every function $T(n) = \Omega(C^n)$,
$DTIME(T(n)) \subset DSPACE(T^{1-\varepsilon}(n))$.

(Compare with [KLi])

Lemma 3.4: If Hypothesis H1 is false for all sufficiently large n, then for every $\varepsilon > 0$ and every $c > 0$, there exists a polynomial time generator that converts n^ε truelly random bits to n bits that look random to any circuit of size n^c.

Proof (of lemma 3.3): We will show that (1) if Hypothesis H1 is true for all n then for some $\varepsilon > 0$, $L \in DSPACE(2^{(1-\varepsilon)n})$ and (2) this implies the lemma.

(1) A space-efficient algorithm for L is as follows: The machine tries all circuits of size $2^{(1-\varepsilon)n}$; for each one it checks whether this is indeed the circuit for L. Once it finds the correct circuit, it uses it to look up the answer. Note that checking

whether the circuit is the correct one is easy, since it only needs to be consistent between consecutive accesses to the same cell.

(2) Consider any Turing machine M running in $DTIME(T(n))$ where $T(n)=2^{t(n)}$. The result of the Turing machine can be derived by looking whether $<M,x,T(n)>\in L$. This can be done is $DSPACE(2^{(1-\varepsilon)m})$ where m is the size of the input which in this case is $n+t(n)+K$, where K is the length of the description of M. It can be easily checked that the statement of the lemma follows. ■

Note: Actually a stronger statement can be made, as under the assumption H1 the simulation mentioned can even be performed in $\Sigma_2-TIME(T^{(1-\varepsilon)}(n))$.

Proof (of lemma 3.4): First note that if H1 is false then every circuit of size $2^{n/2}$ errs on at least $2^{-\varepsilon n}$ fraction of the inputs, since otherwise there would be at most $2^{(1-\varepsilon)n}$ errors which could be corrected by a table. Next, Yao's Xor lemma allows amplification of the unpredictability by Xoring disjoint copies of L. By taking $2^{k\varepsilon n}$ disjoint copies (for some small constant k) a function with arbitrary large polynomial hardness can be reached, which can be used as the basis for our generator. The details can be verified reviewing the exact statement of Yao's Xor lemma. ■

The exact statement of the theorem we obtain is thus:

Theorem 10: One of the 2 following possibilities holds:

(1) $BPP\subset\bigcap_{\varepsilon>0} DTIME(2^{n^{\varepsilon}})$.

(2) There exist $\varepsilon>0$ and $C>1$ such that for any function $T(n)=\Omega(C^n)$, every language in $DTIME(T(n))$ has an algorithm for it that for infinitely many n, runs in SPACE (actually even Σ_2-TIME) $T^{(1-\varepsilon)}(n)$ on all inputs of length n.

Proof: If Hypothesis H1 holds for only finitely many n then lemma 3.4 holds. Otherwise the algorithm in lemma 3.3 will work for infinitely many n. ■

4. Acknowledgements

We would like to thank Laszlo Babai for suggesting AM=almost-NP? as an application of our result.

5. References

[Ad] L. Adleman, "Two theorems on random polynomial time", 19th FOCS, pp. 75-83, 1978.

[AB] M. Ajtai and M. Ben-Or, "A theorem on probabilistic constant depth computations", 16th STOC, pp. 471-474, 1984.

[AW] M. Ajtai and A. Wigderson, "Deterministic simulation of Probabilistic constant depth circuits", 26th FOCS, pp. 11-19, 1985.

[Ba] L. Babai, "Trading group theory for randomness", 17th STOC, pp. 421-429, 1975.

[BG] C.H. Bennett 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.

[BM] L. Babai and S. Moran, "Arthur Merlin games: a randomized proof system, and a hierarchy of complexity classes", Manuscript, 1988.

[BlM] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo random bits", 23rd FOCS, pp. 112-117, 1982.

[CKS] A. Chandra, D. Kozen and L. Stockmeyer, "Alternation", J. ACM 28, 1981.

[FLS] M. Furst, R.J. Lipton and L. Stockmeyer, "Pseudo random number generation and space complexity", Information and Control, Vol. 64, 1985.

[GM] S. Goldwasser and S. Micali, "Probabilistic Encryption", JCSS Vol. 28, No. 2, 1984.

[GS] S. Goldwasser and M. Sipser, "Private coins vs. public coins in interactive proof systems", 18th STOC, pp. 59-68, 1986.

[Ha] J. Hastad, "Computational limitations for small depth circuits", Ph.D. thesis, M.I.T. press, 1986.

[HPV] J. Hopcroft, W. Paul and L. Valiant, "On time versus space and related prob-

lems", 16th FOCS, 1975.

[KLi] R. M. Karp and R. Lipton, "Turing machines that take advice", Enseign. Math. 28, pp. 191-209, 1982.

[KLu] R.M. Karp and M. Luby, "Monte-Carlo algorithms for enumeration and reliability problems", 24th FOCS, pp. 56-64, 1983.

[Ku] S. A. Kurtz, "A note on randomized polynomial time, SIAM J. Comp., Vol. 16, No. 5, 1987.

[NW] N. Nisan and A. Wigderson. "Pseudo random bits for constant depth circuits", manuscript, 1988.

[RT] J.H. Reif and J.D. Tygar, "Towards a theory of parallel randomized computation", TR-07-84, Aiken computation lab., Harvard university, 1984.

[Si1] M. Sipser, "A complexity theoretic approach to randomness", 15th STOC, 330-335, 1983.

[Si2] 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.

[Sh] A. Shamir, "On the generation of cryptographically strong pseudo-random sequences", 8th ICALP, Lecture notes in Comp. Sci. 62, Springer-Verleg, pp. 544-550, 1981.

[St] L. Stockmeyer, "The polynomial time hierarchy", Theor. Comp. Sci. 3, No. 1, 1976.

[Ya] A.C. Yao, "Theory and applications of trapdoor functions", 23rd FOCS, pp. 80-91, 1982.