Pseudorandomness for Network Algorithms

Russell Impagliazzo * Noam Nisan † Avi Wigderson ‡

Abstract

We define pseudorandom generators for Yao's two-party communication complexity model and exhibit a simple construction, based on expanders, for it. We then use a recursive composition of such generators to obtain pseudorandom generators that fool distributed network algorithms. While the construction and the proofs are simple, we demonstrate the generality of such generators by giving several applications.

1 Introduction

The theory of pseudorandomness is aimed at understanding the minimum amount of randomness that a probabilistic model of computation actually needs. A typical result shows that n truly random bits used by the model can be replaced by n pseudorandom ones, generated deterministically from $m << n$ random bits, without significant difference in the behavior of the model. The deterministic function stretching the m random bits into n pseudorandom ones is called a pseudorandom generator, which is said to *fool* the given model.

*Dept. of Computer Science, UCSD. Supported by USA-Israel BSF grant 92-00043.

†Institute of Computer Science, Hebrew University of Jerusalem, Israel. This work was supported by USA-Israel BSF grant 92-00043 and by a Wolfeson research award administered by the Israeli Academy of Sciences.

‡Institute of Computer Science, Hebrew University of Jerusalem, Israel. This work was supported by USA-Israel BSF grant 92-00106 and by a Wolfeson research award administered by the Israeli Academy of Sciences.

0

This theory has developed in two (related) branches: conditional and unconditional results. In a conditional result (of the above type), the quality of the generator is based on some complexity theoretic assumption, which is believed but is not known to hold. Such results exist for very strong models, specifically polynomial time computation, under various assumptions [BM82, Yao82, GKL88, ILL89, Has90, NW88, BFNW].

The unconditional results use no unproven assumption, and typically demonstrate that weaker computational models can be fooled by pseudorandom generators. To this class of results belong the pseudorandom generators for various constant-depth circuits [AW85, Nis91, LVW93] and for space-bounded Turing machines [BNS89, Nis92, NZ93]. Our paper adds a significant number of computational models for which such unconditional results can be proved.

We present a new construction of a pseudorandom generator which fools every computational model which can be described as a network of probabilistic processors. The quality of the generator (i.e. the number of truly random bits m it needs to generate its pseudorandom output) essentially depends on the communication bandwidth of the algorithm run by the network. This is determined by two essential factors: the size of separators in the network, and the number of bits communicated by each processor. Note that we care nothing for the computational power of the processors, and care little about their number!

Our generator is constructed recursively, based on a recursive balanced partitioning of the nodes in the given networks by small separators. Thus the key graph-theoretic parameter is the *tree-width* of the network, which determines the largest cut encountered in such a process. The savings in random bits is significant ($m = n^c$ for some $c < 1$) for networks which can be embedded in small dimensional space (such as grids) and low genus topological spaces (such as planar and excluded minor families of graphs). The savings are extremely significant ($m = (\log n)^{O(1)}$)

for networks such as paths, cycles and trees.

As a "base-case" for the recursive construction of the pseudo-random generator, serves a new pseudo-random generator for the two-party communication complexity model of [Y79], which is interesting in its own right. This generator is based on an expanding graph and its security relies on the on following simple observation: The two parties will behave essentially the same, whether they receive two independent random vertices, or the two endpoints of a random edge of the expander. Moreover, if the parties use c communication bits, it suffices that the expander have degree only $exp(O(c))$, regardless of the input length. Moreover, the difference in behavior (traffic on the communication channel) will be bounded by $exp(-c)$.

As mentioned above, a large number of models can be viewed as algorithms performed by communication networks. For these we can apply our general theorem and obtain specific generators. We list these applications below. We mention in passing that it is standard to derive from the generators explicit functions, for which complexity lower bounds (which depend on the quality of the generator) in the given model can be proved.

- **Space-bounded computation** Here the network is a path on which the (random) input resides, and communication is performed by the moving head. We obtain an $O(\log^2 n)$ generator for what we call bounded read-multiplicity Logspace machines. This includes as a special case Nisan's pseudorandom generator for Logspace machines [Nis92]. This also includes as a special case arbitrary two-way finite automata.

- **Turing machines.** We obtain a $\tilde{O}(\sqrt{T})$ generator for one-tape $\underline{\text{time}}(T)$ Turing machines. We also obtain a $\tilde{O}(\sqrt{ST})$ generator for general time(T)-space(S) bounded Turing machines. These results require a generalization of our basic generator, aimed to handle *average* rather than *worst case* bound on the communication complexity.

- **Boolean circuits.** Here the network is the circuit itself, and we obtain pseudorandom generators that fool planar circuits and read-once formulae. It also results in pseudorandom generators for VLSI model in the AT^2 measure.

- **Parallel and distributed network algorithms.** This obviously suits our model. The most stunning savings we get are for simple architectures like trees and rings for which we can reduce the randomness requirements to polylog. Nontrivial savings in randomness are also achieved for grid architectures.

- **Physical systems and cellular automata** Many models of statistical mechanics assume local interactions (usually on a lattice structure) and random inputs at every site. Monte-Carlo simulations of such systems (e.g. heat propagation, spin-glass models, Maxwell's gas model, ...) are performed in masses. Our results imply that, provably, much less randomness is needed to perform them accurately than it may seem. (Of course, the physicists use even less randomness via the simple generators supplied in every computer, and seem to be satisfied (usually) with the outcome, despite its unproven quality.)

The paper is organized as follows. In section 2 we describe the basic generator, which fools two-party communication complexity. In section 3 we define distributed networks and protocols, and how to construct the generator to fool them. Section 4 describes our applications.

2 Two Parties

Definition 1 *A function $g : \{0,1\}^m \to \{0,1\}^r \times \{0,1\}^r$ is is called a pseudorandom generator for communication complexity c with parameter ϵ, if for every two-party protocol P of at most c bits of communication*

$$|Pr[P(y_1, y_2)\ accepts] - Pr[P(g(x))\ accepts]| \leq \epsilon,$$

where y_1 and y_2 are chosen uniformly at random in $\{0,1\}^r$, and x is chosen uniformly in $\{0,1\}^m$. In short we say that g is a c-generator if it is a pseudorandom generator for communication complexity c with parameter 2^{-c}.

It is clear that for $m = 2r$ it is trivial to get a c-generator for any c (the identity function), and it is also clear that $m \geq r + c$ is a lower bound on m. Our main result in this section is a construction which uses a nearly optimal number of bits $m = r + O(c)$.

The Generator: Fix an easily constructible regular expander graph $H = (V, E)$, with 2^r vertices and degree $D = 2^d$. The input to the generator g is a name of a (directed) edge in E, and the two outputs are the two vertices on the edge. Thus g accepts an $m = r + d$ bit string and produces two r bit strings.

Theorem 1 *g is a c-generator, for $c = (d - \log \lambda)/2$, where λ is the second largest eigenvalue of H. In particular, if H is Ramanujan [LPS86], g is a $(d/4)-$generator.*

Proof: For every graph $H = (V, E)$ of degree D and second largest eigenvalue λ, and for every $S, T \subseteq V$ we have the standard inequality (see e.g. [AC88])

$$|\frac{|E(S,T)|}{|E|} - \frac{|S|}{|V|}\frac{|T|}{|V|}| \leq \frac{\lambda}{D}$$

In our context, the left-hand side is exactly the difference in probability that a pair of vertices (a, b) belongs to the "rectangle" $S \times T$, where in the first expression it is chosen as the endpoints of uniformly random directed edge from E, and in the second each vertex independently from V.

Recall that a c-protocol P partitions the inputs into at most 2^c rectangles, say $S_i \times T_i$, and that the protocol accepts for rectangles $i \in I$. Thus by the definition of our generator,

$$|Pr[P(y_1, y_2) \text{ accepts}] - Pr[P(g(x)) \text{ accepts}]| \leq$$

$$\leq \sum_{i \in I} |\frac{|E(S_i,T_i)|}{|E|} - \frac{|S_i|}{|V|}\frac{|T_i|}{|V|}| \leq \frac{2^c\lambda}{D} \leq 2^{-c}$$

□

Note: It is easy to see that this proof actually implies that this generator also fools what may be called protocols with *effective communication c*, i.e. protocols that have at most 2^c different communication patterns.

We will use the property of being a c-generator in the following slightly more complex scenario:

Definition 2 *A "two-party protocol with an active channel" is a 3 player protocol: Alice and Bob who hold inputs y_1 and y_2 respectively, and "Channell", who holds no input. All 3 parties have unlimited computational power, may be randomized, and communicate with each other according to some fixed protocol. The communication complexity is defined as the total number of bits communicated.*

A k-bit measurement M on the protocol is a triplet of functions, each which can be computed by one of the parties at the end of the protocol, with the combined range being k bits long. (Each party can compute an arbitrary function of its input, if exists, and of the communication it saw.)

Lemma 1 *Let g be a $(c + k)$-generator. Then for any c-protocol P with an active channel, and any k-measurement M:*

$$\|M(P(y_1, y_2)) - M(P(g(x)))\| \leq 2^{-(c+k)},$$

where y_1 and y_2 are chosen uniformly in $\{0,1\}^r$, x is chosen uniformly in $\{0,1\}^m$, and $\|\|$ denotes statistical difference between distributions.

Proof: (Sketch): When all players are deterministic, it is easy to see that Alice (say) can simulate Channel, and we reduce to the case of Theorem 1. When the players are probabilistic, we reduce to the deterministic case by an averaging argument. □

3 General Distributed Protocols

3.1 The Communication Network

Our generators work against networks which have small separators, and which their subgraphs also do. Formally we need:

Definition 3 *Let $H = (V, E)$ be a graph. A partition tree T for H is a rooted binary tree with a one-to-one onto mapping of V to the leaves of T. A partition tree T is called balanced if the depth of T is $O(\log |V|)$. Every internal node v of T partitions V into 3 sets: A_v, B_v and C_v; these are the vertices of V residing (respectively) in the leaves of the left child of v, right child of v and the remaining leaves of T. The cut associated with v, $cut(v)$, is the subset of E of edges which connect vertices in two different sets. The width of v is the smallest number of vertices which cover all edges in $cut(v)$*[1]*. The width of a tree is the maximum width of an internal node in it. The width of a graph H is the smallest width of a balanced partition tree for H. We also define the total width of a tree to be the maximum over all leaves of the sum of the widths of the nodes on the path to the leaf.*

Examples:

1. Any tree has width $O(1)$.
2. Any graph with "tree-width" k has width $O(k)$ (see e.g. [Re92].)
3. Any planar graph on n vertices has width $O(\sqrt{n})$, and in fact has total width $O(\sqrt{n})$ [LT79]. The same bounds apply also to graphs with bounded genus or with any forbidden minor [AS90].
4. A d-dimensional mesh on N vertices has width (and, in fact, total width) $O(N^{1-1/d})$.

3.2 Protocols

We assume a communication network H with a processor at each node. There are n processors and each processor flips r random bits at most during the protocol. Our aim is to replace these rn random bits by pseudorandom bits.

We will have a bound on the communication of each processor. One must be careful to count all information sent or received by the processor whether explicitly (by sending bits) or implicitly (e.g., by the delay between messages). For concreteness we will

[1] We consider the number of vertices which cover the cut instead of the size of the cut since we will assume a bound on the communication of each processor.

consider a simple class of protocols in which no implicit information is passed [2]. We will assume a synchronous protocol, where at each time unit each processor may send an arbitrary number of bits to any number of its neighbors. We do require though that each processor accepting a message knows *before hand* the length of the message (possibly 0). This way the total number of bits sent/received by a processor captures completely the information that it sent/received. We call such protocols "normal" protocols.

Definition 4 *A c-protocol is a normal protocol in which on every input to the network, and every random choice of processors, every processor sends and receives at most c bits of communication.*

Important examples of normal protocols are "oblivious" protocols, protocols in which the length of every message on every edge in each time step is independent of the inputs to the network and random choices of the processors. A subclass of protocols of this form are protocols which run within time t and each processor is allowed to send one bit to one of its neighbors each time step. In this case the total communication of each processor is bounded by t.

3.3 Pseudorandomness

We are assuming a c-protocol running on a communication network $H = (V, E)$ of width w on n processors. We assume that every processor uses at most r random bits. We would like to reduces the number of total random bits to less than nr. It is not clear what exactly should the requirement from a pseudorandom generator be. Clearly every single processors should not be able to distinguish between a random and a pseudorandom run of the protocol. However, since we assume no limit on the computational power of a single processor, a large number of processors combining their knowledge can. It will be convenient to introduce another parameter denoting how many bits collected together from different processors need to be fooled.

Definition 5 *A k-measurement M on a protocol is a k-bit function where each bit of the function can be computed at the end of the protocol by a single processor (i.e. it depends on the input to that processor and on the communication it received.)*

[2] Our results hold also for arbitrary protocols as long as we consider the "effective communication" of each processor: i.e. the logarithm of the number of different possible communication patterns it may see. Here "different" is in the eye of the receiving processor. In general for almost any model one may think of, the "effective communication" is at most a logarithmic (in the time and number of processors) factor larger than the actual number of bits sent.

Let $H = (V, E)$ be a communication graph. $G : \{0,1\}^m \to (\{0,1\}^r)^V$ is called (ϵ, k)-pseudorandom for c-protocols on H, if for any c-protocol P and any k-measurement M,

$$\|M(P(y)) - M(P(G(x)))\| \leq \epsilon$$

where y is a uniformly chosen vector of n r-bit strings, x is chosen uniformly in $\{0,1\}^m$, and $\|\|$ denotes statistical difference.

It is possible to use the definition of (ϵ, k)-pseudorandomness directly or indirectly. An example of direct use is that the communication traffic seen by any single processors is the same (to within ϵ) on a random run or on a $(\epsilon, 1)$-pseudorandom run. The simplest application is to computations whose outcome resides at a single processor. Another application of this might be if the random bits are used (in some arbitrary way) for randomized routing. Then, we can argue that in a pseudorandom run every packet would reach its destination on time with approximately the same probability that it reaches it in the random case.

Sometimes one wishes to argue that certain properties of the run which depend on the whole network remain the same under the psuedorandom generator. The property may be the number of processors that reach a certain state, e.g those who received their packets on time in the above example. This can be argued by imagining a second phase of the protocol which tries to compute this property (in a way which now becomes a k-measurement.) If such a phase can be implemented with low communication then we we are guaranteed that this property behaves the same under the pseudorandom generator. For example, counting the number of processors that reach a certain state can be done with $O(\log n)$ bits of communication per processor.

3.4 The Generator

Let $H = (V, E)$ be the communication network with $|V| = n$, and let T be a balanced partition tree for H of width w. Let k and ϵ be given parameters, let c be the bound on the communication, and define $p = (wc + \log(n/\epsilon) + k)$. We will define our pseudorandom generator recursively, where for a subtree S of T, G^S will have a range of $(\{0,1\}^r)^U$, where U is the subset of the vertices of G that are mapped to leaves in S

1. For a leaf f of the tree, $G^f(x)$ is defined to be the first r bits of x.

2. For a subtree S of T with left and right subtrees L and R respectively,

$$G^S(x) = G^L(g^{left}(x)) \circ G^R(g^{right}(x)),$$

where g^{left} and g^{right} are respectively the two halves of the output of a p-generator g from section 2, and $\circ$ denotes concatenation.

Our main theorem is:

Theorem 2 *G^T is (ϵ, k)-pseudorandom for c-protocols on H. G^T converts $m = r + O(p \log n) = O(r + (wc + \log(n/\epsilon) + k) \log n)$ random bits to n r-bit pseudorandom strings.*

Note: One may slightly improve on this number of random bits used as follows. In the recursive construction we use a different p at each internal node, one defined by taking w to be the width of the root of the subtree instead of the global bound of the width. In this case the total number of bits used is reduced to $O(r + w_{total}c + (\log(n/\epsilon) + k) \log n)$, where w_{total} is the total width of the tree. This may save a up to a factor of $\log n$ for certain networks.

Proof: Let the network H and its balanced partition tree T be given. To prove the theorem, we shall use the standard hybrid argument, which looks at intermediate distributions "between" the all–random string y and all–pseudorandom string $G^T(x)$. The twist is that we have a tree structure, and thus hybrids will be associated with the internal vertices of the tree T. The best way to describe them is via a post-order traversal of the internal notes of T.

```
postorder(v) =
If v is not a leaf Then
    postorder left(v)
    postorder right(v)
    Apply g to v
```

We now explain the meaning of applying g to an an internal node v of T. Let $T(v)$ be the subtree rooted at v, and a, b its left and right children, resp. Assume that inductively we assigned the leaves of $T(a)$ (resp. $T(b)$) pseudorandom inputs according to $G^{T(a)}(\alpha)$ (resp. $G^{T(b)}(\beta)$), with independent random strings α, β. Then applying g to v replaces α and β by the output of $g(\gamma)$ for a random γ, i.e. $\alpha = g^{left}(\gamma)$ and $\beta = g^{right}(\gamma)$. By definition, this assigns the pseudorandom bits $G^{T(v)}(\gamma)$ to the leaves of $T(v)$.

Let us now define a sequence of distributions $D_1 ... D_n$ (on n-tuples of r-bit strings) as follows: D_1 is the distribution where each processor gets r independent random bits. D_i is the distribution obtained after the $(i-1)$'st application in the postorder of g, with the interpretation above. Thus D_n is the pseudorandom distribution obtained as the output of the generator G^T. We now fix any c-protocol P on H, and any k-measurement M. We see that simple induction on i will conclude the theorem from the following inductive step:

Claim: For every i, $||M(P(D_{i+1})) - M(P(D_i))|| \leq \epsilon/n$, where $||||$ denotes statistical distance between distributions.

proof of Claim: Let v be the i'th node on which g was applied. Let a, b be respectively his left and right children, A, B respectively the leaves of $T(a), T(b)$, and C the remaining leaves of T.

But this leads us naturally into the situation of Lemma 1, a two-party protocol with an active channel. Let Alice, Bob and Channel simulate the protocol P by simulating the sets of processors A, B, C respectively. Note that in D_i all three get independent random inputs, whereas in D_{i+1} Alice and Bob share the output of g, just like in Lemma 1. A k measurement M translates naturally to a k-measurement in this scenario.

Now is the time to recall that $p = wc + k + \log(\epsilon/n)$, the fact that $cut(v) \leq w$, that P was a c-protocol (so altogether wc bits are communicated in the simulated protocol), and that M is a k-measurement to see that lemma 1 implies the claim. $\square$

4 Applications

In all our applications the generators given are explicit and easy to compute as they are just special cases (or variants) of the generator described above. The exact complexity of computing the generators depends on the construction of the expanders used, but, using the known constructions, they can certainly be computed in polynomial time and polylog space in the size of the output of the generator.

One should also note that the security of the pseudorandom generator implies a lower bound in the model which this generator fools. The lower bound is for the problem of deciding whether a string is in the range of the generator, and is very strong since it also shows that no significant advantage can even be obtained on this problem.

4.1 Space Bounded Computation

Definition 6 *A Space(S) machine with bounded read-multiplicity is a Turing machine which runs in space S, with an extra read-only input tape (containing the random or pseudorandom bits). The head on the read-only input tape may visit any cell on the input tape at most a constant number of times. We also allow the machine to be non-uniform, formally by giving it access to an arbitrary oracle.*

We have two examples in mind: the first is an arbitrary space(S) machine which treats its input tape as though it was a source of random bits, i.e. in one-way fashion. This is the model considered in

[BNS89, Nis92]. The second example is an arbitrary two-way finite automata.

Theorem 3 *There exists a generator $G : \{0,1\}^m \to \{0,1\}^n$ which is pseudorandom, to within ϵ, for space(S) bounded read-multiplicity machines, where $m = O((S+\log(n/\epsilon))\log n)$.*

Proof: A bounded read-multiplicity space(S) machine can be simulated by a network of processors on a line. Each cell of the read-only input tape will be a processor, and whenever the head on the read-only input tape moves from one cell to its neighbor the first cell will send the total state of the machine (including the contents of all work tapes) to the neighbor. This requires $O(S)$ bits of communication, and may happen at most a constant number of times for each cell. Thus we have an $O(S)$-protocol on the line, which has width $O(1)$. Finally, acceptance is determined in one cell (the one in which the head resides at the end of the computation), so it is a 1-measurement. □

4.2 Turing Machines

In this section we consider generators which aim to fool general Turing machines (no bound on read-multiplicity). Clearly we can not hope to do better than the current lower bound techniques yield. Thus we will obtain generators for the two cases for which lower bounds for Turing machines are known: (1) For single tape Turing machines: a quadratic lower bound for time is known; (2) For arbitrary Turing machines: a quadratic lower bound on the time-space product is known. The lower bounds implied by our generators almost match this, yet our bounds are in some sense stronger as they hold even for getting any significant advantage.

Theorem 4 *There exists a generator $G : \{0,1\}^m \to \{0,1\}^n$ for Turing machines running in simultaneous time T and space S, which is pseudorandom, to within $2^{-\sqrt{TS}}$, where $m = O((\sqrt{TS}\log T)$.*

Theorem 5 *There exists a generator $G : \{0,1\}^m \to \{0,1\}^n$ for one–tape Turing machines running in time T, which is pseudorandom, to within ϵ, where $m = O(\sqrt{T\log(T/\epsilon)}\log T)$.*

The generators we have designed so far will not do the trick since they require some bound on the information transfer between any two processors in the network. If we try to simulate a general (or a 1-tape) Turing machine by a network of processors we will not be able guarantee that: the problem is that the head of the Turing machine may move many times between two neighboring cells, thus in the simulation much communication will be needed between these two cells. Instead what we may obtain for the simulation of Turing machines is an upper bound on the *average* communication. To take advantage of this we will modify our generator as follows.

The Improved Generator

It will be built recursively as the previous generator for the special case where the network is a line. The base case is trivial $G^1(x) = x$ for r-bit strings x. In the recursion step we will use a variant:

$$G^n(x, z) = G^{(n-c)/2}(g^{left}(x)) \circ z \circ G^{(n-c)/2}(g^{right}(x))$$

Here, g is a c-generator for communication complexity which takes an $m + O(c)$ bit string and produces two m-bit strings; x is an $m + O(c)$-bit string; z is a c-bit string; $G^{(n-c)/2}$ takes an m-bit input and produces $(n - c)/2$ r-bit outputs; and thus G^n takes an $m + O(c)$ bit input and produces n r-bit outputs.

Recursively, the number of bits of input needed to produce n r-bit strings is still $r + O(c\log n)$.

Proof: (of theorem 4 – sketch) We use the improved generator with $c = O(\sqrt{TS})$, and $n = T$. The proof proceeds exactly like the proof of theorem 2 using the hybrid method. The only difference is the induction step in which we make use of the extra c random bits (z) injected between the left and right outputs of the generator. In the simulation Alice and Bob will simulate respectively the left and right parts of the output of g. Movement of the head on the read-only input tapes from Alice's part to Bob's part requires $c = \sqrt{TS}$ time steps, and only such movement transfers information between Alice and Bob. Each such movement transfers $O(S)$ bits of information (i.e. the contents of all work tapes.) The total communication is thus $S \cdot T/\sqrt{TS} \le c$ and the induction step follows from the security of g. □

Proof: (of theorem 5 – sketch) Again we use the improved generator, this time with $c = O(\sqrt{T\log(T/\epsilon)})$, and $n = T$. The only difference from the proof of the last theorem is that the Turing machine is allowed to write on the section of the tape between Alice's part and Bob's part. We thus find it difficult to control the information passed between Alice and Bob (i.e. simulate the TM directly by Alice and Bob with low communication). Our solution to this is the following randomized protocol for Alice and Bob: they will choose a random point in the middle section and each will simulate the TM on their side of that point. Since there are c cells in the middle section, and the total running time is bounded by T, the expected number of times the head crosses this point is $O(T/c)$. Thus the expected communication needed for the simulation is $O(T/c) = O(\sqrt{T/\log(T/\epsilon)})$. To finish the proof we need to show that g can fool two-party protocols with *expected* communication $O(\sqrt{T/\log(T/\epsilon)}) = O(c/\log(T/\epsilon))$. This is true since any such protocol can be turned into one that runs in worst case $O(c)$ communication by trying $O(\log(T/\epsilon))$ independent runs, each abandoned after it exceeds the expected running time by a factor of 2. This introduces a probability ϵ/T of failure. $\square$

4.3 Circuits and VLSI

The models we discuss here are nonuniform, as oppose to the Turing machines above, so we should clarify the interest in pseudorandom generators. One answer is that they would work equally well for uniform families of circuits. Another answer, which is better, is that an efficient generator provides an efficient algorithm for the following problem: Given a circuit C and a real number $\epsilon > 0$, compute the fraction of inputs accepted by C, up to an additive error ϵ. The idea is that if C takes n input bits, and $G : \{0,1\}^m \to \{0,1\}^n$ fools C within ϵ, then computing the fraction of inputs from the image of the generator accepted by C will do. This will take deterministic time $2^m Time(G)$, where $Time(G)$ is the complexity of evaluating G.

This approach was used to approximately count the number of accepted inputs to constant depth circuits [Nis91] and to $GF(2)$ polynomials in [LVW93]. Since our generators are efficient (computable in polynomial time), the two theorems below add nontrivial algorithms for this approximate counting problem for two other circuit classes: read-once formula and planar circuits. Moreover, both results hold for any choice of finite basis for the gates of the circuit.

Definition 7 *A read-once formula is a formula in which every input variables occurs exactly once.*

Note: Observe that while this model is obviously weak, it can compute functions that cannot be computed, say, in AC^0.

Theorem 6 *There exist generators $G : \{0,1\}^m \to \{0,1\}^n$ which are pseudorandom to within ϵ for read-once formula of size n, where $m = O((\log n)(\log(n/\epsilon)))$.*

Note: The exact mapping of the output of the generator to the leaves of the formula depends on the structure of the formula (i.e. on the underlying tree) but not on the specific gates used. These may be arbitrary (finite) gates.

Proof: The processors of our protocol will reside at the gates and simulate the computation of the circuit in the obvious way. The result follows since it is an $O(1)$-protocol on a tree (which has width $O(1)$), and the output is a 1-measurement. $\square$

Definition 8 *A circuit C is planar if the graph describing it is planar.*

Note: We do not care here about the distinction usually made between having the inputs presented at the boundary of the planar embedding or not [We87]. Also, as in the space results, we cannot hope to beat the best known lower bounds for planar circuits, which are quadratic. Again, our generator essentially meets this bound.

Theorem 7 *There exist generators $G : \{0,1\}^m \to \{0,1\}^n$ which are pseudorandom to within ϵ for every planar circuit of size n, where $m = O((\sqrt{n} + \log(1/\epsilon))\log n))$.*

Proof: The only difference from the previous proof is that in a planar circuit we use the bound on the total width, which is $O(\sqrt{n})$ ([LT79]). $\square$

Essentially the same theorem can be proved for the VLSI model, where instead of the size of the circuit (n) the relavent parameter is AT^2. We omit the details.

4.4 Randomized Parallel Network Algorithms

This is the most natural family of applications. Consider a synchronous network of processors running on a network H of n processors. Each processor receives an input, and in each time step may flip α random bits, and communicate α bits to a neighbor. The processors run for T steps and then the first processor decides if to accept or reject the input. A generator which provides pseudorandom bits to the processors *fools* the algorithm to within ϵ, if the acceptance probability does not change by more then ϵ for any input.

Theorem 8 *There exist generators which fool any algorithm of the type above to within ϵ. The generator for network H uses only $O(T\alpha w \log(n/\epsilon))$ random bits, where w is the width of H.*

The most intereting implication is for fast algorithms, with $T, \alpha = (\log n)^{O(1)}$, running on simple architectures like trees and rings (more generally with width $w = (\log n)^{O(1)}$). These can be fooled to within $n^{-O(1)}$ with only polylog(n) random bits. As our generator works in polylog space, these randomized algorithms can be simulated in DSPACE$((\log n)^{O(1)})$. Note that while deterministic algorithms of this type can be easily simulated in this space bound, our result does not follow at all from randomized analogs of Savitch's theorem.

4.5 Physical Systems and Cellular Automata

Our generators can be used in various simulations of physical systems. Instead of formally defining classes of such simulations, we prefer to describe them informally. The basic idea is that many physical systems are simulated by breaking "space" into many cells which interact with each other like probabilistic cellular automata. Cellular automata are natural examples of distributed systems, and thus our generator can fool them. Let us elaborate a little about what we have in mind.

We consider some physical-like "space" e.g. the real plane, 3-dimensional real space, the surface of some simple body (like a submarine or a spaceship), or even a 1-dimesional line. Many simulations start by breaking this space into n different "cells", each with its own "state". Examples we have in mind are spin in an Ising model, molecule speed (and type, number) in thermodynamic computations, or stress in a finite-element analysis. The simulation proceeds one "time-step" after another for t steps. In each time step each cell changes state, in some probabilistic manner, depending on the state of its neighbor cells. E.g. the spin may flip with probability proportional to the number of neighbor cells with different spin. Let us assume for concreteness that the information passed to each cell each time unit is $O(1)$ bits and that the amount of randomness needed each time unit for each cell is $O(1)$ bits. (Perhaps $O(\log \delta^{-1})$, where δ is some precision parameter of the simulation, is more realistic here, but this will change our results only by a similar factor.) At the end of the simulation we are usually interested in some "simple" property of the system, like the number of cells in some given state (e.g. the total magnetic moment), or the state of some given cell.

A direct simulation of such a system uses $O(tn)$ random bits. Since we can view this system as a network of processors, we can directly use our generator to reduce the randomness requirements of the simulation. Specifically, to get a simulation which is, provably, within ϵ from the truly random simulation, it suffices to use $O(t\sqrt{n} + \log \epsilon^{-1})$ random bits for 2-dimensional spaces (the real plane or the surface of a simple body) and $O(tn^{2/3} + \log \epsilon^{-1})$ for 3-dimensional bodies.

Acknowledgments

We thank Oded Goldreich for discussions on the tree-generator.

References

[AC88] N. Alon and F. R. K. Chung. Explicit Constructions of Linear Sized Tolerant Networks. In *Discrete Mathematics* 72, pp. 15-19, 1988.

[AS90] N. Alon, P. Seymour. A Separator Theorem for Graphs with an Excluded Minor and its Applications. In *Proc. of the 22nd STOC*, pp. 293–299, 1990.

[AW85] M. Ajtai and A. Wigderson. Deterministic simulation of probabilistic constant depth circuits. In *26th FOCS*, pages 11–19, 1985.

[BFNW] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. Bpp has subexponential time simulations unless exptime has publishable proofs. *To appear in Journal of Complexity Theory.*

[BM82] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. In *23rd FOCS*, pages 112–117, 1982.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and Logspace-hard pseudorandom sequences. In *21st STOC*, pages 1–11, 1989.

[GKL88] O. Goldreich, H. Krawcyzk, M. Luby. On the Existence of Pseudorandom Generators. In *29th FOCS*, pp. 12–24, 1988.

[Has90] J. Hastad. Pseudorandom generators under uniform assumptions, In *22nd STOC*, pp 395-404, 1990.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random number generation from one-way functions. In *21st STOC*, pages 12–24, 1989.

[LPS86] A. Lubotzky, R. Phillips, and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *18th Annual ACM Symposium on Theory of Computing*, pages 240–246, 1986.

[LT79] R. J. Lipton, R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.* 36, pp. 177–189, 1979.

[LVW93] M. Luby, B. Velickovic, A. Wigderson. Deterministic Approximate Counting of Depth–2 Circuits. In *Proc. of the 2nd ISTCS (Israeli Symposium on Theoretical Computer Science)*, pp. 18–24, 1993.

[Nis91] N. Nisan. Pseudorandom bits for constant depth circuits. In *Combinatorica* 11 (1), pp. 63–70, 1991.

[Nis92] N. Nisan. Pseudo-random sequences for space bounded computation. In *Combinatorica* 12 (4), pp 449–461, 1992.

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. In *FOCS*, 1988.

[NZ93] N. Nisan and D. Zuckerman. More deterministic simulation in Logspace. In *STOC*, 1993.

[Re92] B. Read. Finding Approximate Separators and Computing Tree-Width Quickly. In *Proc. of the 24th STOC*, pp. 221–228, 1992.

[We87] I. Wegener. The complexity of Boolean functions, Wiley-Teubner, 1987.

[Y79] A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of* 11^{th} *STOC*, pp. 209-213 1979.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *FOCS*, pages 80–91, 1982.