Pseudorandomness for Network Algorithms

R. Impagliazzo, N. Nisan, A. Wigderson

Presented by Jean-Philippe Martin

May 9, 2001

1 Introduction

This document records the presentation of this paper, which was given in class. In the interest of completeness, some material has been added. All the statement of theorems are quoted verbatim, except Lemma 1 and 2 which are my own.

1.1 Goal

Impagliazzo, Nisan and Wigderson's paper presents a pseudorandom generator for network algorithms. Pseudorandom generators are a general construction which take as input truly random bits and give as output a larger number of pseudorandom bits which are "close enough to random" for the purpose at hand. We will clarify the notion of "random enough" later in this document.

Pseudorandom generators are useful constructions because random bits are necessary in many applications (probabilistic algorithms and simulations for example), but truly random bits are hard to generate.

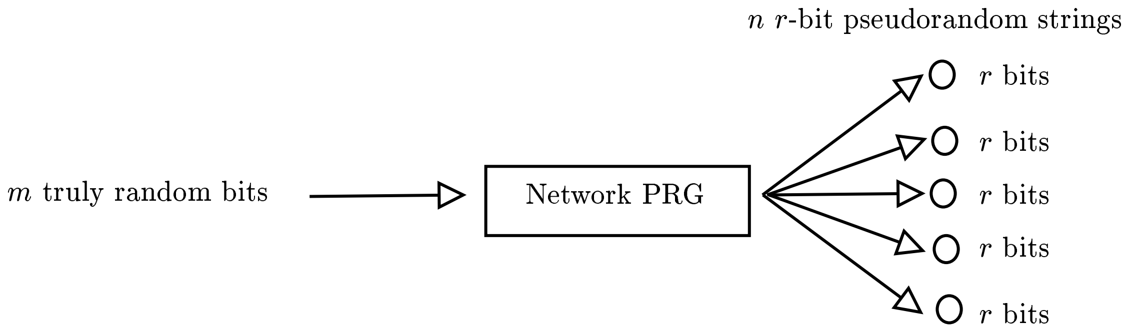
Pseudorandom generators (PRG) are usually built with a given application in mind, so that the pseudorandom bits can have the properties required by the application (for example being pairwise independent). In the case covered by this paper, however, we make no assumption on the algorithm itself, nor on the computational power or the number of the computers. Instead, the authors take advantage of the limited communication between the network nodes. If the nodes communicate only in a limited way, we are be able to "fool" any algorithm using fewer random bits. In the extreme case where the nodes do not communicate at all (so, for example, the final result lies on processor 1) it would not be necessary to provide the other nodes with different pseudorandom bits. Naturally network algorithms will have some communication, and the paper shows how to build the PRG given a bound on communication.

2 Two-Party Generator

2.1 Overview

We first present the two-party generator, which can fool a network algorithm composed of two nodes.

For the remainder of this document, we will call n the number of processors and r the size of the random input that each uses. The pseudorandom generator (PRG) takes as input m truly random bits and outputs n r-bit pseudorandom strings.



2.2 Communication

The PRG is designed for a given amount c of communication, where c is the total number of bits effectively sent and received by all parties. We have to be careful to count all bits of effective communication between parties: for example, if processes do not know the size of the message in advance then that size itself adds to the information transported. To simplify things, we will focus on *c-protocols*: synchronous protocols in which all parties receive messages at each time step and the size of the message is known in advance. Furthermore, the sum of the number of bits received and sent at each time step is at most c.

2.3 Definitions

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

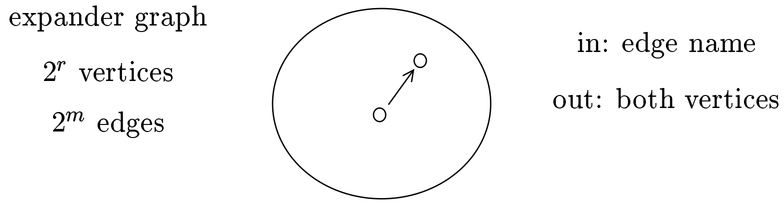
$$|\Pr[P(y_1, y_2)\,accepts] - \Pr[P(g(x))\,accepts]| < \epsilon$$

Where y_1 *and* y_2 *are chosen uniformly at random in* $\{0,1\}^r$, *and* x *is chosen uniformly at random in* $\{0,1\}^m$.

Definition 2 *The function* g *is a* c-generator *if it is a pseudorandom generator for communication complexity* c *with parameter* 2^{-c}.

2.4 Construction

The generator g is a regular (directed) expander graph H with 2^r vertices and degree $D = 2^d$. The $m = r + d$ truly random input bits are interpreted as the name of an edge and the output is the name of the vertices at both ends of the edge (r bits each).



Before proving the correctness of this construction, we need to define precisely what it means for a network protocol to accept an input string. This definition was implied from the paper but never explicitly stated; therefore any error here is my own.

We will say that each node can individually accept or reject the input string. Then, as part of the definition of the n-party protocol, a mapping D from $\{0,1\}^n \mapsto \{0,1\}$ is defined which indicates for each set of individual decision what the global state will be (rejection or acceptance).

The proof presented in the paper makes the claim that a c-protocol partitions the input space in 2^c regions, without further explanation. Since this aspect appears to have been a source of confusion during the presentation, we now present this result more formally as a lemma and prove it.

Lemma 1 *A two-party c-protocol can be entirely described by at most 2^c sets S_i and at most 2^c sets T_i such that the protocol accepts if the input of the first party is in S_x and the input of the second party is in T_x for some x ($0 \leq x < 2^c$).*

Proof:

Since we are working with a two-party c-protocol, the total amount of communication is c bits. Because the protocol is deterministic, for each communication pattern i (a c-bit string) we can identify a set of pairs (s_j^i, t_j^i) such that if the first party has input s_j^i and the second has input t_j^i then the communication pattern will be i and the automaton will accept (as specified by the individual decision of the parties and D). We then introduce the following lemma:

Lemma 2 *If the automaton follows communication pattern i and then accepts for initial states (s_1^i, t_1^i) and (s_2^i, t_2^i) then the automaton will follow the same communication pattern and accept for initial states (s_1^i, t_2^i) and (s_2^i, t_1^i).*

Proof: Because the communication between s_1^i and t_1^i and between s_2^i and t_2^i both follow the pattern i in both cases, s_1^i cannot distinguish between t_1^i and t_2^i. Therefore the pairs (s_1^i, t_2^i) and (s_2^i, t_1^i) will accept. □

It follows from Lemma 2 that for a given communication pattern i all the pairs (s_j^i, t_j^i) can be summarized with two sets, S_i and T_i, such that if the first party's input is in S_i and the second party's is in T_i then the protocol will accept.

There is such a pair of sets for each communication pattern which can lead to acceptance. Therefore, the entire protocol can be described with at most 2^c "rectangles" (S_i, T_i) such that the protocol accepts if the input (r_s, r_t) lies within some pair (S_i, T_i). $\square$

Note that all these rectangles are disjoint since the protocol is deterministic.

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, 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

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

For any given S and T, the left side of the difference represents the probability that the generator picks an edge from S to T and the right side is exactly the probability for such an edge to be chosen at random from a complete graph.

This is true for any choice of S and T, so it also applies to the rectangles $S_i \times T_i$ for which the protocol accepts.

$$|\Pr[P(g(x)) \text{ accepts}] - \Pr[P(y_1, y_2)] \text{ accepts}]| = |\sum_{i=0}^{2^c} \frac{|E(S_i, T_i)|}{|E|} - \sum_{i=0}^{2^c} \frac{|S_i|}{|V|}\frac{|T_i|}{|V|}|$$

$$\leq \sum_{i=0}^{2^c} |\frac{|E(S_i, T_i)|}{|E|} - \frac{|S_i|}{|V|}\frac{|T_i|}{|V|}| \leq 2^c \frac{\lambda}{D}$$

$\square$

This proof is restricted to c-protocols, but it is easy to see that it also applies to any protocol with c bits of effective communication, because such a protocol can be modeled by a c-protocol.

The paper then builds a slightly more powerful definition and lemma, on which the generalized generator is based. We reproduce them here.

Definition 3 *A* "two-party protocol with an active channel" *is a 3 players protocol: Alice and Bob who hold inputs y_1 and y_2 respectively, and "Channel", 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-bits 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 it exists, and of the communication it saw).*

Lemma 3 *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): Since Channel holds no input, it is easy to see that a player (say, Alice) can simulate it and reduce to the case of Theorem 1. Each of the k bits of the measurement can be modeled by a different protocol P_α, which accepts if the bit is "1" and rejects otherwise. Since each of these automaton is within $2^{-(c+2k)}$ of the random distribution (by Theorem 1), their combination cannot be more than $2^{-(c+k)}$ away from the random distribution. The following should make the author's intentions more clear.

$$||M(P(y_1,y_2)) - M(P(g(x)))||$$

$$= \sum_{\alpha=0}^{\alpha=2^k} |\Pr[M(P(y_1,y_2)) = \alpha] - \Pr[M(P(g(x))) = \alpha]|$$

$$= \sum_{\alpha=0}^{\alpha=2^k} |\Pr[M(P_\alpha(y_1,y_2)) \text{ accepts}] - \Pr[M(P_\alpha(g(x))) \text{ accepts}]|$$

$$\leq \sum_{\alpha=0}^{\alpha=2^k} \frac{2^c\lambda}{D} = \sum_{\alpha=0}^{\alpha=2^k} 2^{(-c-2k)} = 2^{-(c+k)}$$

The last line follows from Theorem 1, after computing D for the (c+k)-generator (in this case, $D = 2^{2(c+k)}\lambda$).

In the case when the players are probabilistic, we reduce to the deterministic case by an averaging argument.

□

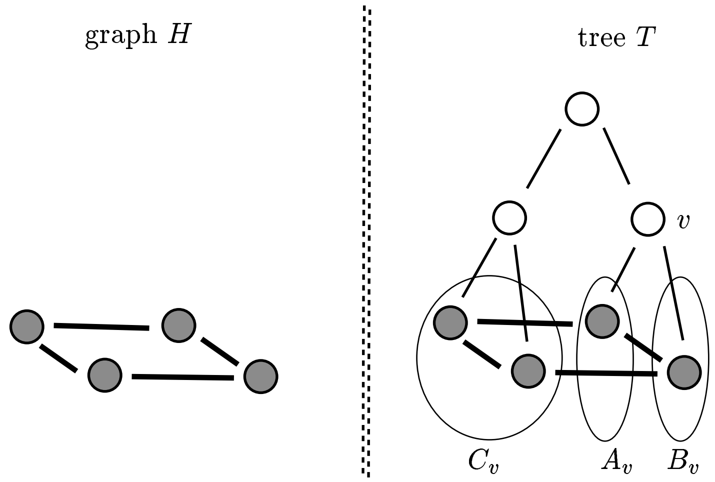
3 Generalized PRG

The generalized construction builds on the previous section's generator to fool network protocols with an arbitrary number of parties. The intuition behind this result is that the network is partitioned using a tree on which lemma 3 can be applied at each level. The generator will be build recursively, following the tree structure. As the authors points out, this generator works best against networks which have small separators and whose subgraphs also do. As is the case with lemma 3, we will use k-measurement to quantify the security of this generator.

We will assume that the communication graph $H = (V, E)$ contains n nodes, each of which uses at most r random bits. Our goal is to generate these bits from fewer than nr truly random bits.

3.1 Definitions

For this construction, we build a balanced binary rooted tree T on top of the network graph $H = (V, E)$ such that the vertexes of H are now the leaves of T (Remember that a tree is called balanced if its depth is $O(\log |V|)$). The resulting tree is shown below. Every internal node v of T partitions V into three sets: the nodes on the left of v (A_v), the nodes on the right (B_v) and the other nodes (C_v).



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 minimal number of vertices of V sufficient to cover all edges in $cut(v)$. The *width of a tree* is the maximal width of any internal node in it. The *width of a graph* H is the smallest width of a balanced partition tree for H. The *total width* of a tree is the maximum over all leaves of the sum of the widths of the nodes on the path to the leaf.

We can then formalize the notion that we will use to indicate that the pseudorandom generator can "fool" any protocol.

Definition 4 *Let $H = (V, E)$ be a communication graph. $G : \{0,1\}^m \mapsto (\{0,1\}^r)^{|V|}$ 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.

3.2 Generator

Let $H = (V, E)$ be the given communication network, with $n = |V|$ and T is the minimal width tree associated with H. k and ϵ are given parameters (the measurement size and tolerated error, respectively) and communication is bound by c, i.e. for each processor the sum of the number of bits sent and received is at most c. The pseudorandom generator is defined recursively, such that at each node v of T we define a PRG which can fool the vertices of H which are under v. The upper-level PRG uses the two lower-level ones and requires a few more bits: this amount turns out to be $p = (wc + \log(n/\epsilon) + k)$. This recursive construction leads us to define a function G^S for each subtree S, such that G^S has range $(\{0,1\}^r)^{|U|}$, where U is the subset of the vertices of G that are mapped to leaves in S (Note that the construction of this generator is very similar to Nisan's construction using hash functions which we discussed in lecture 17). Here is the generator itself:

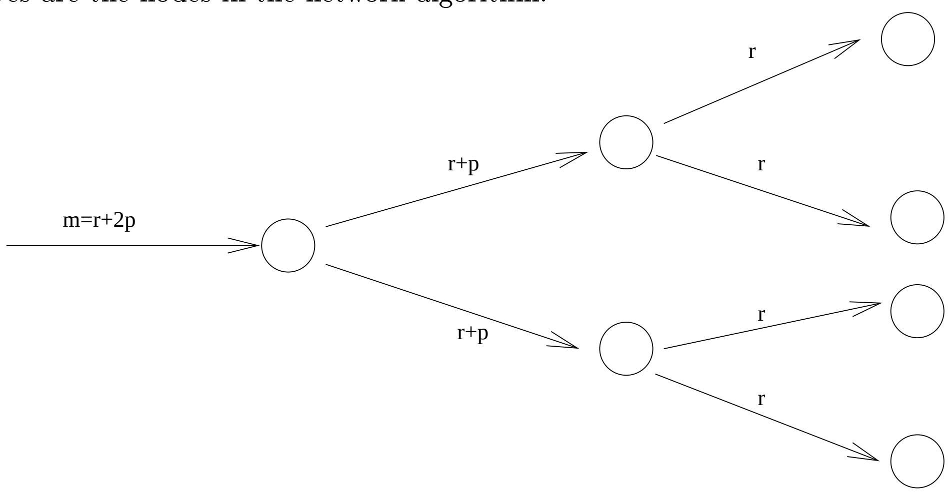
"

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."

Intuitively, at each stage the generator consumes $r + p$ truly random bits and outputs $2r$ pseudorandom bits, which are sent to the next stage. The untouched truly random bits are duplicated and sent along. This is shown in the figure below: the tree represents the generalized generator and the leaves are the nodes in the network algorithm.



Theorem 2 *G^T is $(2\epsilon, k)$-pseudorandom[1] 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.*

Proof: For a node v of T, let T_v be the subtree rooted at v and L_v and R_v its left and right subtree, respectively. We will write $|T_v|$ to mean the number of vertices in the subtree T_v, not only the leaves. We prove this theorem indirectly, by means of the following lemma:

Lemma 4 *Let y_u, $u \in C_v$ be arbitrary r bit strings. Let D_{pseudo} denote the distribution of $M(P)$ when we use the y_u's as inputs to the processors in C_v, and use G^{T_v} with a random input to produce inputs for the processors in $A_v \cup B_v$. Let D_{random} denote the distribution of $M(P)$ when we use truly random inputs for the processors in $A_v \cup B_v$ and we use the y_u's as inputs to the processors in C_v. Then $||D_{pseudo} - D_{random}|| \leq |T_v|\epsilon/n$, where $||$ denotes statistical difference.*

Proof: The base case (leaf PRG) holds trivially, since the generator returns the r truly random bits unmodified and has the same distribution as the truly random choice. We will now prove that the lemma holds for the parent node's generator G^P, knowing that its two children's PRG, G^L and G^R satisfy the lemma.

[1]The paper claims that G^T is (ϵ, k)-pseudorandom. The presented proof, however, did not support this.

To compare D_{pseudo} and D_{random}, we go through two intermediate distributions: D_A and D_{AB}. We quote their definition directly from the paper:

"Let D_A be the distribution of $M(P)$ where we set the values for processors in A_v by $G^{L_v}(x_0)$ for a random x_0 and those in B_v at random (those in C_v still get the y_u's). Let D_{AB} be the distribution where we set the values for processors in A_v by $G^{L_v}(x_0)$ and those in B_v by $G^{R_v}(x_1)$ for random x_0, x_1."

The strings in D_A are composed of the lower level's generator's bits concatenated with truly random bits. The latter bits have a uniform distribution, identical to that of D_{random}. The former, by the induction hypothesis, have a distribution that lies within $|L_v|\epsilon/n$ of uniform. Therefore $||D_{random} - D_A|| \leq |L_v|\epsilon/n$. Consider now D_{AB}. Since x_0 and x_1 are chosen independently, for any given value for processors in A_v the generated bits lie within $|R_v|\epsilon/n$ of uniform in D_{AB}. In D_A, these bits were uniformly distributed. Therefore $||D_A - D_{AB}|| \leq |R_v|\epsilon/n$. We now use Lemma 3 to show that $||D_{AB} - D_{pseudo}|| \leq \epsilon/n$.

"Consider the following protocol with an active channel: Alice is given x_0, which she uses as an input to G^{L_v} to generate inputs for all nodes in A_v. Bob is given x_1, and uses $G^{R_v}(x_1)$ as inputs for nodes in B_v. Channel simulates the remaining nodes in C_v using the specified inputs y_u. Alice, Bob and Channel then simulate the distributed protocol, sending messages only when a node in one group send a message to a node in another group. Note that, by the definition of the width of a partition tree, there are at most w vertices covering the edges connecting different groups[2], and each vertex can communicate at most c bits. Thus, there are at most cw bits communicated in total during this protocol. The k-measurement M on the network protocol translates naturally to a k-measurement on this 3-party protocol. Therefore, using Lemma 3 and since $p \geq wc + k$, the statistical distance between the distribution on the k-measurement when x_0, x_1 are independent and when $< x_0, x_1 > = g(x)$ for a random x is at most $2^{-p} \leq \epsilon/n$. This is exactly the same as the statistical difference $||D_{AB} - D_{pseudo}||$."

By the triangle inequality between distributions, we can deduce from the above that $||D_{random} - D_{pseudo}|| \leq (|L_v| + |R_v| + 1)\epsilon/n = |T_v|\epsilon/n$. □

Because $|T_v| \leq 2n$, the theorem is the lemma when v is the root of the tree. □

The authors also note that this result can be improved by using a different p at each level of the tree, thus adding fewer bits if that level's root has a low width. This technique can reduce the size of m up to a factor of $\log n$.

4 Applications

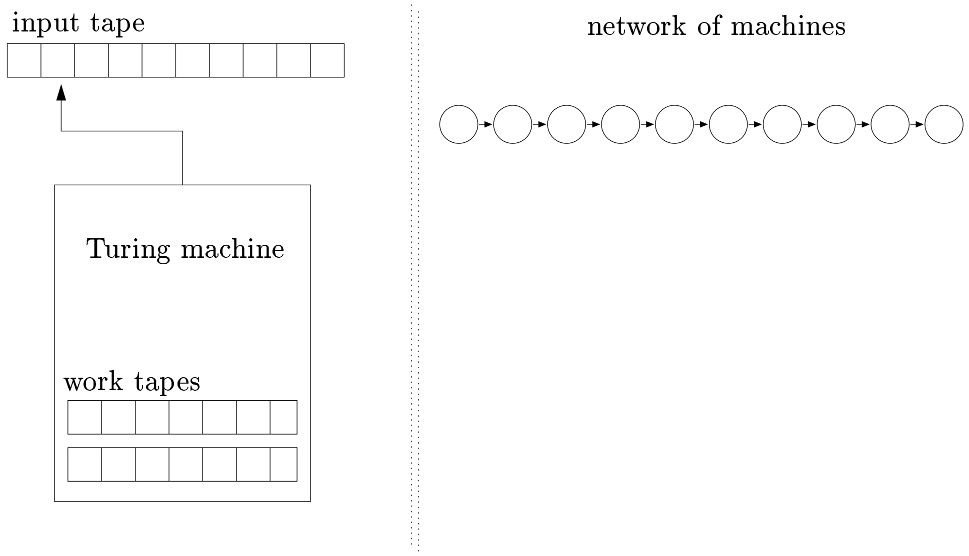
The paper presents a number of different applications of the generator. These applications are especially interesting because some of them introduce variants to be to the generator or use it for other purposes than just using fewer truly random bits.

4.1 Space Bounded Computation

We consider space-bound Turing machines with a read-only input tape, each cell of which can be read only a certain number of times. Note that this is a generalization of the traditional space-bound

[2]The paper says w edges, but this is clearly a typo since the definition of width considers vertices only.

Turing model in which the read-only (random) input tape can be read only once.



As you can see in the figure above, we model the Turing machine (TM) with a network protocol arranged in a line. Each processor of the network represents a single cell of the input tape (the example shows ten). Together they simulate the Turing machine in the following way: initially, only the leftmost machine performs computation, by simulating the Turing machine. If at some point the TM moves the read head to the next cell then the processor will sent its entire state (of size S) to the neighbor machine, which will continue the simulation.

Since each cell can be visited at most a certain number of times i, the amount of communication on the network between two processors is at most iS. We can therefore use the generalized construction. The acceptance of the machine is signified by a single cell, so we need to perform a 1-measurement. Substituting these values in Theorem 2 yields (n is the size of the input tape):

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

Note that using known constructions for expanders, the generator G can be computed in polynomial time and polylog space.

4.2 Turing Machines

What if we did not specify any bound on the read multiplicity? Certainly, our previous argument would not hold anymore since the communication would not be bound. However, the paper shows that we can still obtain some results by using a variant of the generator, which leverages the fact that the *average* communication between any two neighbor nodes is known. c is the average communication between two nodes.

4.2.1 Goal

While the results presented in this section show how to fool Turing machines, they are more interesting when viewed from the point of view of a different problem: finding functions which are "hard" to compute, that is functions for which a lower bound on computation time exists.

In this case, the hard function is going to be: "given an input, is this input in the domain of the generator"? We will explore this question for two different types of Turing machines, either time and space bound or one-tape Turing machine. In both cases we show that the minimum time (or time-space) to answer the question is about m^2. As indicated by the authors, lower bounds for TM are already known in these two cases and they are slightly better than these. However, what makes this result interesting is the fact that we can show that if the computation time is shorter than the bound then the attacker cannot gain any significant advantage.

This result is achieved by building a generator which can fool any program running on the aforementioned Turing machines, in particular the program which tries to determine whether the input is pseudorandom is truly random. As Theorem 3 (below) will describe in more detail, the improved generator presented in the next section can fool the program to within $2^{-\sqrt{TS}}$.

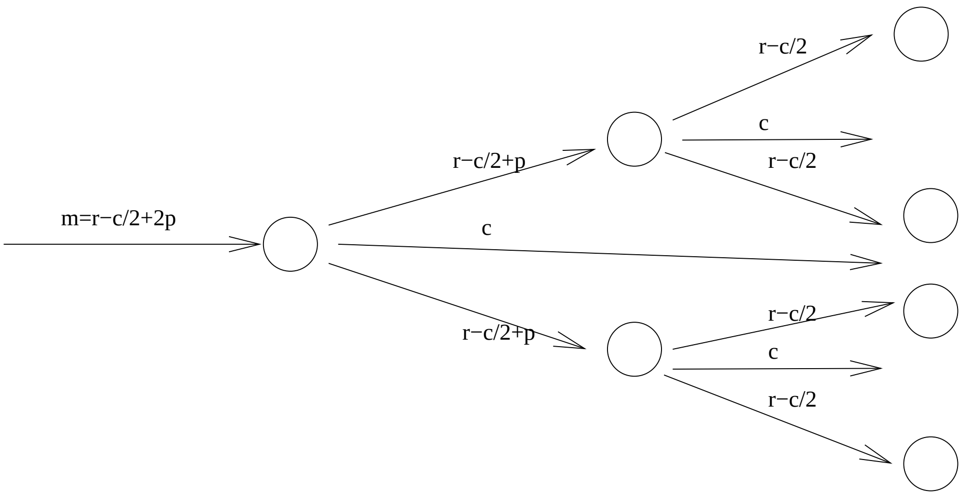
4.2.2 Improved Generator

The improved generator is still built recursively, but it is optimized for the case where the network is a line. The base case is $G^1(x) = x$ for r-bit strings x. In the recursion step:

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

"Here, g is a c-generator which takes 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. 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 $r + O(c \log n)$."

As shown in the image below, c random bits are inserted between each pair of nodes at each level. The total number of additional bits is therefore $c \log n$.



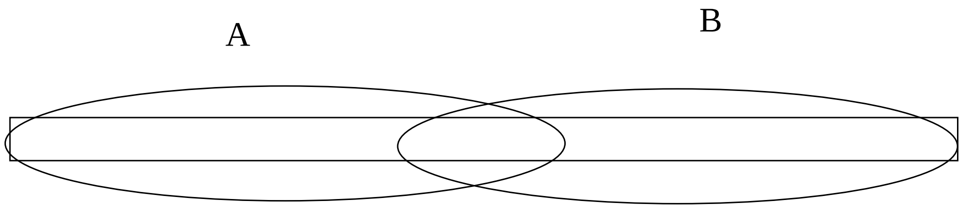
The generator G, in the case of a line, would use c random bits for each level. In the improved generator, we insert c additional random bits. The usefulness of these bits will become obvious with the proof below. Intuitively, they are used to provide a "buffer" zone which both nodes can simulate, therefore reducing the need for communication.

Theorem 3 *There exists an efficient generator $G : \{0,1\}^m \mapsto \{0,1\}^n$ for Turing machines running in simultaneous time T and space S, which is pseudorandom to within $2^{-\sqrt{TS}}$, where $m = O(\sqrt{TS}\log T)$.*

Proof: (sketch): Again, we model the Turing machine as a network arranged in a line. Because the TM runs in time T, it cannot possibly use more than T random bits: therefore we set $n = T$ and build a network with n processors. We choose $c = \sqrt{TS}$ and show later why this is a bound on communication.

The proof proceeds recursively, as before. The base case is trivially true, and we build the distributions D_A, D_{AB} and D_{pseudo} as before. Using the same argument as the previous proof we can show that $||D_{random} - D_A|| \leq |L_v|\epsilon/n$ and $||D_A - D_{AB}|| \leq |R_v|\epsilon/n$.

To compare D_{AB} and D_{pseudo}, consider the following protocol: Again, we simulate the Turing machine using the two party protocol with an active channel. This time, however, the nodes simulated by Alice and Bob overlap in the c-bit central zone. Alice only sends a message to Bob when the TM's read head leaves Alice's area, and Bob only sends a message to Alice once the read head has moved back sufficiently to leave Bob's area.



For each message sent, the head must move by at least c steps. Each time this happens, the entire TM state (including the contents of all work tapes) must be transmitted, for a total of S bits. The total communication is at most $S \cdot T/c = S \cdot T/\sqrt{TS} = \sqrt{TS}$, so in the network the total communication is bounded by c. We can therefore apply Lemma 3 as previously and complete the proof. $\square$

Theorem 4 *There exists an efficient generator $G : \{0,1\}^m \mapsto \{0,1\}^n$ for one-tape Turing machines running in time T which is pseudorandom, to within ϵ, where $m = O(\sqrt{T \log(T/\epsilon)} \log T)$.*

Proof: (sketch): We use the improved generator again, with $n = T$ and this time $c = O(\sqrt{T \log(T/\epsilon)})$. In the previous case, we could assume that the Turing machine did not write on the input tape (it does not need to, having work tapes available to it). The single-tape TM, however, must be allowed to write on the tape. We cannot therefore use the same technique as previously to limit the communication between Alice and Bob. The authors propose the following randomized algorithm:

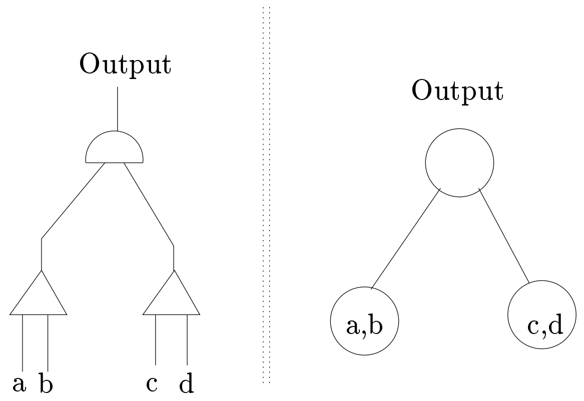
"Let Alice and Bob choose a random point in the middle section and each simulate the TM on their side of that point. Since there are c cells in the middle section and the total running time is bounded by T, the expected number of times the head crosses the point is $O(T/c)$. Thus the expected communication 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." $\square$

5 Circuits and VLSI

5.1 Overview

For this application, we consider non-uniform circuits. In particular, the results apply to both read-once formulae and planar circuits (defined below). The goal this time is to estimate the fraction of the inputs which are accepted by the circuit, within an additive error ϵ.

The principle is that to estimate this fraction, instead of sampling the total input space it is sufficient to sample the PRG's input space, which is smaller. The transformation from a circuit to a network algorithm is done by replacing each gate with a node, as shown below (a, b, c, d are the inputs).



These results can be applied to the VLSI model as well, the only difference being that the size of the circuit is then measured as AT^2 instead of n.

5.2 Definitions

Definition 5 *A* read-once formula *is a formula in which every input variable occurs exactly once.*

Definition 6 *A circuit C is* planar *if the graph describing it is planar.*

Even though for planar graphs it is often also required that the inputs be presented on the edges of the graph, this restriction does not apply here.

5.3 Results

Theorem 5 *There exist efficient generators $G : \{0,1\}^m \mapsto \{0,1\}^n$ which are pseudorandom to within ϵ for read-once formula of size n, where $m = O((\log n)(\log(n/\epsilon)))$.*

Proof: Since the formula is read-once, the circuit can be converted into a network protocol on a tree with n leaves, where n is the number of inputs. The result of the computation can be gathered by a 1-measurement and the width of any tree is $O(1)$: substituting these numbers into the generator (Theorem 2) completes the proof. □

Theorem 6 *There exist efficient generators $G : \{0,1\}^m \mapsto \{0,1\}^n$ 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 proof is identical to Theorem 5's proof, except for the fact that the width for planar graphs is $O(\sqrt{n})$. $\square$

6 Randomized Parallel Network

6.1 Overview

This application is very close to the assumptions of the network protocol generator. The parallel network allows the n processors to use α random bits at each step and then communicate α bits with their neighbors. The total execution time is T and the randomized parallel network then either accepts or rejects the input.

Our goal is to use the PRG to provide these $nT\alpha$ bits from a smaller pool of truly random bits, with the constraint that the probability of accepting must remain within ϵ of what it would be using truly random bits.

6.2 Results

Theorem 7 *There exist efficient generators which fool any randomized parallel network algorithm 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 authors points out that this theorem also shows that fast algorithms running on simple graphs ($T, \alpha, w = (\log n)^{O(1)}$) can be fooled to within $n^{-O(1)}$ using polylog(n) random bits. Since the generator runs in polynomial space, randomized parallel networks can be simulated in DSPACE($(\log n)^{O(1)}$).

7 Physical Systems and Cellular Automata

7.1 Overview

Network computation can be used to simulate physical systems. In these simulations, the environment is partitioned into a number of *cells* and each such cell is simulated by a computer which communicates with the neighboring cells to simulate the natural interactions. This layout is also known as *cellular automaton.*

7.2 Results

Assuming that each time step uses $O(1)$ random bits and communicates $O(1)$ bits, then if the simulation is run for t time steps on n cells, a naive result would use $O(tn)$ bits. The paper shows that generators exist for $m = O(t\sqrt{n} + \log \epsilon^{-1})$ in the two dimensional case and $m = O(tn^{2/3} + \log \epsilon^{-1})$ for the three dimensional.

8 Conclusion

We have seen how to build a pseudorandom generator which can fool network algorithms, regardless of the power of the individual nodes. This generator is built from a two-party generator using a recursive construction similar to the one used for Nisan's generator using hash functions. The new construction shows that expanders and pseudorandom generators are linked.

Intuitively, the PRG works because of the bound on communication between the nodes. However, the various applications show that it is also possible to leverage bounded computation time or average communication in a similar way. They have also shown that the generator can be used to obtain functions with a quadratic minimal computation time as well as techniques to estimate the fractions of inputs that a given circuit will accept.