1 Introduction

1.1 Randomized Complexity

In the last decade randomization became an important tool in the design of algorithms. There are many problems for which efficient randomized algorithms have been given even though no efficient deterministic algorithm is known. This apparently enhanced power which randomization provides has become a major research topic in complexity theory.

For almost any natural complexity class, a corresponding randomized class may be defined (in fact several randomized variants can be defined). In most cases no nontrivial relationship is known between the corresponding randomized and deterministic classes, and in many cases there are some interesting problems known to lie in the randomized class but not known to lie in the corresponding deterministic one. Primality testing is known to lie in randomized polynomial time [SS77, Rab80, AH87], but not known to lie in deterministic polynomial time. Construction of a perfect matching in a graph is known to lie in random-NC [KUW86, MVV87], but not known to lie in NC. Undirected connectivity is known to lie in random-Logspace [AKL+79], but not known to lie in Logspace. Graph non-isomorphism is known to lie in AM, the randomized analogue of NP [GMW86], but not known to lie in NP.

This thesis continues the investigation into the power of randomization, and the relationships between randomized and deterministic complexity classes. The line of attack we pursue is the idea of *emulating* randomness, known as *pseudorandom generation*.

1.2 Pseudorandom Generators

The major conceptual idea behind pseudorandom generation is that sequences of events may *look* random even though they are not truly random in the information theoretic sense. Sequences may look random to any observer that does not have enough *computational* power to "understand" them.

This revolutionary idea was introduced and formalized in the early '80s by Blum and Micali [BM84], who were influenced by Shamir [Sha81], and by Yao [Yao82]. Blum and Micali and Yao proposed the idea of *pseudorandom generators*, functions which stretch a short string of truly random bits into a long string of bits which looks random to observers having limited computational power.

There are several motivations for research into pseudorandom generators. Perhaps the most broadly stated one is merely getting better insight into the nature of randomness from a *behavioral* point of view. More specifically, pseudorandom generation is perhaps

the most general and natural method to reduce or eliminate the randomness required by algorithms. Pseudorandom sequences may replace the random sequences required by algorithms without changing the results in any way. This reduces the number of random bits required for solving the problem. Moreover, the trivial exponential deterministic simulation of randomized algorithms can now be used to obtain rather fast deterministic simulations.

Another motivation for research into pseudorandom generation is cryptography. A standard element implicit in many cryptographic protocols is to generate messages that do not give any information to your opponent, in other words, messages that appear random to him. Pseudorandom generators are a general framework to study these issues. Indeed many cryptographic protocols may be based on pseudorandom generators (see, e.g., [LR86, ILL89]).

Finally, perhaps the practical motivation should be mentioned: in reality many computer programs use random numbers. Hardly ever are truly random bits supplied by the computer (or do they even claim to be supplied by some physical means such as Zener diodes). Usually some pseudorandom generator supplies numbers which are hoped to be as good as truly random ones. It is important to know how much and in what cases this can be really justified.

1.3 Basic Definitions

Blum and Micali [BM84] and Yao [Yao82] were the first to define pseudorandom generators. They were concerned with pseudorandom generators that look random to polynomial time Turing machines, or to polynomial size circuits. We will consider the natural extensions and give general definitions of pseudorandom generators that look random to an arbitrary class of machines.

Another way in which we modify the definitions given by Blum-Micali and by Yao is the requirement regarding the running time of the pseudorandom generator. Blum-Micali and Yao defined pseudorandom generators to be computed in polynomial time. We will remove this requirement from the definition. Of course, for our pseudorandom generators to be interesting, we will need to show that they can indeed be computed somewhat efficiently.

The definitions that appear here are generic, and more precise specific definitions for particular complexity classes will appear where we use them.

Blum-Micali and Yao gave competing definitions of what a pseudorandom generator should be. These two definitions turned out to be equivalent. Yao's definition is perhaps the strongest one imaginable: that the pseudorandom bits will behave just like truly

random ones in any way measurable by machines in the class.

Definition 1 A function $G = \{G_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n\}$ is called a *pseudorandom* generator for class C if for every algorithm A in C, every polynomial $p(n)$, and for all sufficiently large n:

$$|Pr[A(y) = 1] - Pr[A(G_n(x)) = 1]| \leq 1/p(n) \tag{1.3.1}$$

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

The definition given by Blum-Micali seems to be a rather minimalist one: that given any prefix of the pseudorandom sequence, the next bit would look random.

Definition 2 A function $G = \{G_n : \{0,1\}^{m(n)} \rightarrow \{0,1\}^n\}$ passes all class C *prediction tests* if for every algorithm A in C, every $1 \leq i \leq n$, every polynomial $p(n)$, and all sufficiently large n:

$$|Pr[A(y_1, ...y_{i-1}) = y_i] - 1/2| \leq 1/p(n) \tag{1.3.2}$$

Where y_j is the j'th bit output by G_n, and the probability is taken over a random input to G_n.

It was proven by Yao [Yao82] that these two definitions are equivalent.

THEOREM 1 (Yao) G is a pseudorandom generator for class C iff it passes all class C prediction tests.

This fact is extremely helpful, since the usual method of proving that a generator is pseudorandom is to show that it satisfies the weaker definition, and then to conclude that it has all the nice properties of the stronger one.

1.4 Previous Work

Most work regarding pseudorandom generators has been directed towards pseudorandom generators for P, polynomial time Turing machines. The first pseudorandom number generator was designed by Blum and Micali [BM84]. It is based on the unproven assumption that computation of the "discrete log" function cannot be done in polynomial time. They first showed that, under this assumption, the most significant bit of the discrete log cannot be approximated by any polynomial time computation, and they proceeded to give a general scheme to produce many pseudorandom bits using this fact.

Yao [Yao82] generalized this construction. He showed how any *one-way permutation* can be used in place of the discrete log function.

Definition 3 A function $f = \{f_n : \{0,1\}^n \rightarrow \{0,1\}^n\}$ is called a one-way permutation if (1) For all n, f_n is one-one and onto (2) f can be computed in polynomial time (3) Any polynomial size circuit that attempts to invert f errs on at least a polynomially large fraction of the inputs.

Yao first showed how to "amplify" the "unpredictability" of condition 3 and obtain a "hard bit", a bit that cannot be predicted at all. This amplification is achieved by taking multiple copies of the hard function on disjoint sets of input bits and xor-ing them; Yao shows that this operation indeed "amplifies" the unpredictability of the bits. Once a "hard bit" is obtained, a pseudorandom generator can be designed using the Blum-Micali scheme:

A seed x_0 of size n^ϵ bits is given as the random input. The one-way permutation is applied to the seed repeatedly, generating a sequence $x_0, f(x_0), f(f(x_0)), \ldots, f^{(n)}(x_0)$. The pseudorandom output of the generator is obtained by extracting the "hard bit" from each string in this sequence. The proof that this is indeed a pseudorandom generator proceeds by showing how a test that this sequence fails can be used to invert the one-way permutation f.

THEOREM 2 (Yao) If a one-way permutation exists then for every $\epsilon > 0$ there exists a polynomial time computable pseudorandom generator $G : \{0,1\}^{n^\epsilon} \rightarrow \{0,1\}^n$.

Recently, Impagliazzo, Levin and Luby [ILL89] and Hastad [Has90] proved that the existence of any one-way function (not necessarily a permutation) suffices for the construction of pseudorandom generators. This condition is also necessary for the existence of pseudorandom generators that can be computed in polynomial time. The pseudorandom generators described so far can indeed be computed in polynomial time.

All the work described so far concerns generators that pass all *polynomial time* tests; pseudorandom generators for P. Some work has also been done regarding the construction of pseudorandom generators for other complexity classes.

Reif and Tygar [RT84] describe a generator that passes all NC tests and, moreover, can itself be computed in NC. This generator is based on the assumption that "inverse mod p" cannot be computed in NC. The main innovation here is showing that for this particular function, the original Blum-Micali-Yao generator can be parallelized.

Ajtai and Wigderson [AW85] considered pseudorandom generators for AC^0. They use an ad-hoc construction based on the lower bound methods for constant depth circuits to construct a pseudorandom generator that passes all AC^0 tests. The significance of this result is that it does not require any unproven assumptions. Instead, it builds upon the known, proven lower bounds for constant depth circuits.

1.5 Hardness vs. Randomness

The second chapter of this thesis is devoted to a new general construction of pseudorandom generators for arbitrary complexity classes. This construction overcomes two basic limitations of the known pseudorandom generators:

- They require a strong unproven assumption. (The existence of a one-way function, an assumption which is even stronger than $P \neq NP$.)
- They are sequential, and cannot be applied to an arbitrary complexity class. E.g. there is no known construction of pseudorandom generators for NC that is based upon a general complexity assumption about NC.

The construction which we propose avoids both problems: It can be applied to any complexity class C, it is based on an arbitrary function which is hard for C, and gives a pseudorandom generator that looks random to the class C. Although our generator cannot necessarily be computed in the class C, we will show that it can be computed efficiently enough for our *simulation* purposes.

Perhaps the most important conceptual implication of this construction is that it proves the *equivalence* between the problem of proving lower bounds for the size of circuits approximating functions in EXPTIME and the problem of constructing pseudorandom generators which run "sufficiently fast". This should be contrasted with the results of Impagliazzo, Levin, Luby and Hastad [ILL89, Has90] showing the equivalence of proving the existence of *one-way* functions and constructing pseudorandom generators that run in polynomial time. Our construction requires much weaker assumptions but yields less efficient pseudorandom generators. This loss does not have any effect when using pseudorandom generators for the deterministic simulation of randomized algorithms.

This construction has many implications, and a large part of the second chapter describes them. We first show that efficient deterministic simulation of randomized algorithms is possible under much weaker assumptions than previously known. The efficiency of the simulation depends on the strength of the assumption and can be good enough to show $P = BPP$. Since the assumptions required for our generator are so weak and natural, we believe that this work provides overwhelming evidence that the gap between deterministic and randomized complexity is not large.

We then turn to pseudorandom generators for constant depth circuits. Since lower bounds for constant depth circuits are known (e.g., [Has86]), our construction yields an unconditionally proven pseudorandom generator for constant depth circuits. This generator improves upon the known generator, due to Ajtai and Wigderson [AW85], and implies much better deterministic simulation of randomized constant depth circuits.

Our generator for constant depth circuits turns out to have some interesting conse-

quences regarding the power of random oracles for complexity classes in the polynomial time hierarchy. We show that NP with a random oracle is exactly the class AM, solving an open problem of Babai [BM88]. We also show that a random oracle does not add power to the polynomial time hierarchy.

A new proof is given of the fact that BPP is in the polynomial time hierarchy. Our final application is a surprising connection between simulation of "time by space" and simulation of "randomness by determinism". We show that one of these simulations can be substantially improved over known simulations.

The results in this chapter have appeared in [Nis91a, NW88] and are joint work with Avi Wigderson.

1.6 Pseudorandom Generators for Logspace

The third chapter in this thesis describes the construction of a pseudorandom generator for Logspace. This result is unique in that it is not based on any unproven assumptions. The only other class for which pseudorandom generators were unconditionally proven to exist is AC^0.

In order to prove the correctness of our pseudorandom generator we use the following multiparty communication game, first introduced by Chandra, Furst and Lipton [CFL83]: Let $f(x_1 \ldots x_k)$ be a Boolean function that accepts k arguments each n bits long. There are k parties who wish to collaboratively evaluate f; the i'th party knows each input argument except x_i; and each party has unlimited computational power. They share a blackboard, viewed by all parties, where they can exchange messages. The objective is to minimize the number of bits written on the board.

We first prove lower bounds for the number of bits that must be written on the board in order to get even a small advantage on computing certain functions. We then show how to use these lower bounds in order to construct a pseudorandom generator for Logspace.

We conclude by giving some applications of our pseudorandom generator. We describe a construction of universal sequences for arbitrary regular graphs; no nontrivial such construction was previously known. We also show that random Logspace machines with two-way access to the random bits are better, in some specific sense, than random Logspace machines with the usual one-way access to the random bits.

The results in this chapter have appeared in [BNS89] and are joint work with Laszlo Babai and Mario Szegedy.

1.7 Recent Results

Since this thesis was originally written (in December 1989), several new results related to problems considered here were obtained. The most relevant ones are a different, improved construction for pseudorandom generators for Logspace [Nis90] and a further sharpening of the conditions under which the pseudorandom generators presented in chapter two can be constructed [BFNW91].

The fourth chapter briefly describes these as well as other recent results related to the subject of this thesis.