

## Lecture 1: Introduction and Overview

September 7, 2004

Scribe: Valentine Kabanets

## 1 Some Applications of Randomness

Randomness plays central part in Algorithm Design, Cryptography, Computational Complexity, and Combinatorics.

- **Algorithms** Randomized algorithms are usually very fast and simple. For some computational problems, randomized algorithms are the only known efficient algorithms; for others, randomized algorithms are simply faster than any known deterministic algorithms.
- **Cryptography** Cannot avoid the use of randomness if we want to hide some information. Intuitively, we want to encode secret information in an unpredictable way. For achieving unpredictability, randomness is essential.
- **Complexity** Randomness is an important computational resource (in addition to the “classical” resources such as time and space). Some of the most exciting results of modern Complexity Theory involve randomness (e.g., results on Interactive Proofs, and Probabilistically Checkable Proofs).
- **Combinatorial Structures** The “probabilistic method” can be used to argue the existence of a variety of combinatorial objects. The idea is to define a probability distribution on combinatorial objects of interest, and argue that picking a random object from that distribution results in a “good” object with *non-zero* probability. Hence, a “good” object must exist. (We will be interested in *explicit* constructions of such combinatorial objects. That is, it is not enough for us to know that a “good” object exists. We actually would like to have an efficient deterministic algorithm for producing a “good” object.)

The main question of the course is: What is the power of randomness in computation? Can we reduce (or eliminate) randomness without losing efficiency of our algorithms?

## 2 Motivation

Basically, we want to be frugal with our computational resources, including randomness. That is, we want to use as few random bits as possible in our computation. The limit of this would be complete derandomization — using zero random bits.

Another reason is that we need explicit constructions of interesting combinatorial objects (such as expander graphs, or error-correcting codes). While the probabilistic method guarantees the existence of such objects, we need to be able to construct them explicitly (i.e., efficiently deterministically) for our applications.

Finally, there may not be any physical source of true randomness. That is, we may have to deal with sources of randomness that produce correlated and biased bits. As long as such sources contain some kind of randomness in them, we would like to be able to “squeeze out” that randomness and use it.

### 3 Pseudorandom Objects

The word *pseudo* means *false* in Greek. When we say that something is pseudorandom, we mean that it “looks” random, without actually being truly random. There will be different interpretations of what it means to “look random”, depending on the type of objects.

In this course, we’ll be looking at the following pseudorandom objects.

- **Pseudorandom Generators (PRGs):** These are efficient deterministic algorithms that stretch short random strings into much longer “pseudorandom” strings, so that no efficient algorithm can tell the difference between an output of the PRG and a truly random string. The main application of PRGs are in Cryptography and Derandomization. (There is no known provably good PRG. The reason is that proving that something is a PRG involves proving that a certain task, namely distinguishing outputs of the PRG from random strings, is hard to do. But computer scientists don’t know yet how to prove such limitation results. We’ll have to settle for conditional constructions of PRGs.)
- **Expanders:** These are sparse, yet highly connected graphs. In particular, we want graphs on  $n$  vertices that have constant degree (and hence  $O(n)$  edges), but, at the same time, any bisection of the graph will have to cut a constant fraction of edges of the graph. These two properties seem contradictory, and it’s quite amazing that graphs satisfying these two properties actually exist.  
Expanders are used in many applications: fault-tolerant networks, error-correcting codes, parallel sorting algorithms, etc.
- **Randomness Extractors:** These are efficient algorithms that convert a distribution with high enough “entropy” into a distribution that is close to uniform; in general, such conversion requires the use of a short truly random string as a “catalyst”. The main application of extractors is to take a “weak” source of randomness and turn it into a source of uncorrelated and unbiased bits, which can later be used in randomized algorithms.
- **Error-Correcting Codes (ECCs):** These are algorithm for encoding messages so that, even if some bits of the encoded message get corrupted due to a noisy transmission channel, we can still correctly recover the original message. The obvious application of ECCs is in data storage and transmission.
- **Hard Boolean functions:** These are Boolean functions that cannot be computed by Boolean circuits (involving the operations AND, OR, and NOT) using fewer than an exponential number of the operations. In other words, if you think of a Boolean function on  $n$  variables as a truth table of length  $2^n$  (the bit in the  $i$ th position is the value of the function at input  $i$ ), then a hard function is the one whose truth table cannot be significantly compressed. That is, to describe such a function, essentially the best you can do is to write down its exponentially long truth table.

All of the objects listed above are pseudorandom in various senses that will be made precise later in the course. The surprising fact is that all of them turn out to be very closely related to each other, even though they come from different areas of computer science. Exploring these interconnections will be the main focus of our course.