

# CMSC 858T: Randomized Algorithms

Spring 2003

## Handout 1: Main issues that we aim to learn

In this handout, we list the main issues that we aim to learn in this course. The actual list of key issues that we cover in the course may be slightly different from the list here. The main purpose of listing these is that (like in many other fields of science and engineering) it is useful for our topic of study to have a *systematic* collection of techniques with which to attack problems. This list is by no means exhaustive, but does cover many important paradigms and approaches. Please try to bring this handout to every class.

We first detail some main *paradigms* that make randomized algorithms useful. The examples listed here are mostly toy examples, but they give a glimpse of why making random choices can help algorithms.

- (P1) *Abundance of witnesses.* Suppose you have an unsorted array of  $n$  elements, each element of which is 0 or 1. You are told that at least 90% of the elements are 1, and asked to output the index of at least one “1” in the array. Since a large proportion of the array contains what we want, a natural approach is to look at a *randomly chosen* entry of the array: the probability that it is not a 1 is at most  $1/10$ . Suppose we repeat this six times independently; the probability of being unsuccessful all six times is  $10^{-6}$ , i.e., one-in-a-million. In contrast, it is easy to show that any *deterministic* algorithm needs to examine at least  $n/10$  locations of the array in the worst case. Thus, in many cases, we can get substantial gains in speed (or in other computational resources) by allowing for a small probability of error.
- (P2) *Symmetry-breaking.* This is especially useful in settings where there is limited communication among computing agents, such as in distributed and parallel computing. Consider two identically-programmed wireless devices, say, that wish to use a shared communication channel, such as air. The channel can only take a message from one of the devices at a time; if both try sending, their messages “collide” (get garbled), and have to be re-sent. Here is a simple randomized solution to the problem that requires no communication between the devices: each device flips a coin locally, and tries to send its message iff it got a Heads (and repeats this protocol every step until it successfully sends its message). Convince yourself why both devices would have successfully transmitted their messages within a small number of steps, with high probability. We will see generalizations to settings where a large number of agents try to achieve their own goals without much co-ordination.
- (P3) *Laws of large numbers.* Informally, such laws state that when a “large enough” number of “almost-independent” random variables are aggregated suitably (e.g., summed), then the aggregate random variable is very close to its expected value, with high probability. This phenomenon leads to several applications via random sampling. A well-known example from the real world: to find out how many people support a social issue, say, we query a small random subset of the people and estimate the actual answer by the sampled one. We will study variations on this theme, especially in the context of approximation algorithms.
- (P4) *Randomly ordering the data.* Many types of algorithms have to process a sequence of items in order; if the algorithm is unlucky, the order it chooses may not be a good one. However, if we can show that “most” orderings of the data are good, the algorithm can

first pick a random ordering, and then proceed. We will see applications of this natural idea to some graph-theoretic problems.

**(P5)** *Construction of combinatorial structures using randomization.* Probabilistic approaches help in the construction of many types of discrete structures: e.g., error-correcting codes, networks with good fault-tolerance properties, counterexamples to some graph-theoretic conjectures, etc. We will study a few important examples.

We may look upon the above as a quick answer to the question “why does randomization add any power at all to algorithms?”

We next see some useful principles for the design of (de)randomized algorithms.

**(M1)** The first moment method / linearity of expectation / union bound.

**(M2)** The second moment method.

**(M3)** Alteration.

**(M4)** Random sampling.

**(M5)** Randomly ordering the data.

**(M6)** Fingerprinting.

**(M7)** Derandomization: the method of conditional probabilities,  $k$ -wise independence.

Finally, the following are some of the main tools we will use in analyzing randomized algorithms. Not only are these important for analysis: often, a (rough) calculation using such tools of what one might expect from a certain type of randomized algorithm, drives the detailed design of the algorithm.

**(T1)** Linearity of expectation, union bound, truncated inclusion-exclusion.

**(T2)** Large-deviation bounds:

- the bounds of Markov, Chebyshev, Chebyshev-Cantelli, Chernoff, and Hoeffding;
- exploiting negative correlation and “near-independence”;
- Martingale tail bounds.

**(T3)** The FKG and Janson inequalities, and the Poisson paradigm;

**(T4)** The Lovász Local Lemma.

Using the above, we will study applications to randomized approximation algorithms, distributed algorithms, network design, random graphs, etc.