Lecture Notes on Probabilistic Algorithms and Pseudorandom Generators [1]

Martin Tompa

Technical Report #91–07–05

July 15, 1991

Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, Washington, U.S.A. 98195

[1]This material is based upon work supported in part by the National Science Foundation under Grant CCR-9002891, and in part by IBM under Research Contract 16980043.

Contents

Preface

These are the lecture notes from CSE 522, a graduate algorithms course I taught at the University of Washington in Winter 1991. The topic of the course was Probabilistic Algorithms and Pseudorandom Generators. These notes are not intended to be a survey of that area, however, as there are numerous important results that I would have liked to cover but did not have time.

I am grateful to Eric Bach, Don Coppersmith, Michael Rabin, Prabhakar Raghavan, Larry Ruzzo, Victor Shoup, Avi Wigderson, and David Zuckerman, all of whom helped me with technical points in the proofs. I am particularly thankful for the many talented students who attended faithfully, served as notetakers, asked embarassing questions, made perceptive comments, and generally make teaching exciting and rewarding.

— Martin Tompa

Lecture 1

Introduction to Probabilistic Algorithms

January 7, 1991
Notes: Ashok Immaneni

1.1. Responsibilities

1. Rotating notetaking: Notes are to be written up in LaTeX, and sent to tompa@june. See the instructions in june:~tompa/522/DIRECTIONS. Notes are due 10 a.m. the day following lecture, except Friday's lecture notes, which are due 9 a.m. the following Monday, so that they may be distributed at the next lecture. The instructor will insert any citations for bibliography entries that don't already appear in the bibliography file.

2. Assignments (open ended)

3. Class participation

1.2. Prerequisites

1. Analysis of Algorithms (CSE 521)

2. Basic probability theory (expectation, conditional probability and expectation, independent and mutually exclusive events)

1.3. Course Outline

1. **Probabilistic algorithms:** Can random numbers possibly be of use in improving the efficiency of deterministic algorithms? One familiar example is Quicksort, but is this an anomoly? In fact, other examples have arisen in areas such as number theory, cryptography, distributed computation, and graph theory. Welsh [41] has written a survey on probabilistic algorithms.

2. **Pseudorandom generators and their interaction with probabilistic algorithms:** In practice, how would you get the random numbers needed by some probabilistic algorithm? You would probably use a pseudorandom number generator, but this has two problems:

   (a) you still need a small amount of true randomness for the initial seed, and

(b) the analysis of the probabilistic algorithm as though it had access to a true source of independent random numbers may no longer be valid.

How can one guarantee that the pseudorandom number generator will not interact badly with the probabilistic algorithm?

1.4. Review of Order Notation

- $f(n) = O(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ |f(n)| \leq cg(n)$
- $f(n) = \Omega(g(n))$ if and only if $(\exists c)(\exists n_0)(\forall n \geq n_0)\ f(n) \geq cg(n)$
- $f(n) = \Theta(g(n))$ if and only if $(\exists c)(\exists c')(\exists n_0)(\forall n \geq n_0)\ cg(n) \leq f(n) \leq c'g(n)$
- $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- $f(n) = \omega(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

1.5. An Example – Matrix Multiplication

Any deterministic algorithm produces the same output for a given input every time it executes. In contrast, a probabilistic algorithm makes use of a random number generator to produce the output, and hence the output is a random function of the input. Surprisingly, such random numbers are sometimes useful for improving the efficiency of deterministic algorithms. A familiar example is Quicksort: If the splitting element is chosen carelessly, its running time is $\Theta(n^2)$. If it is chosen randomly, the expected running time is $O(n \log n)$. Thus, this probabilistic algorithm *usually* executes faster than its deterministic counterpart.

Of course, the deterministic algorithms Mergesort and Heapsort have comparable worst case running times, so Quicksort isn't a very convincing example. This section presents one of the simplest examples of a problem for which we know a more efficient probabilistic solution than any deterministic solution known.

The problem is to determine, given $n \times n$ matrices A, B, and C, whether $AB = C$. The best known deterministic solution is to multiply A and B, and check if the result equals C. This runs in time $O(n^{2.376})$, which is the best known upper bound for matrix multiplication (Coppersmith and Winograd [15]).

In contrast, there is a probabilistic solution due to Freivalds [16] that runs in $O(n^2)$ time:

```
Choose $x \in \{0,1\}^n$ randomly and uniformly;
**if** $ABx \neq Cx$
   **then return** ("$AB \neq C$")
   **else return** ("$AB = C$ probably").
```

Analysis: By associativity, ABx can be computed as $A(Bx)$. Since the product of an $n \times n$ matrix with an n-vector can be computed in $O(n^2)$ time, this is the running time of the entire algorithm.

Correctness:

Case 1: If $AB = C$ then $ABx = Cx$ for all x, so $\Pr(ABx = Cx) = 1$.

Case 2: If $AB \neq C$ then ABx may still be equal to Cx. The following lemma shows that this doesn't happen too often.

Lemma 1.1: Assume $AB \neq C$ and x is chosen randomly and uniformly from $\{0,1\}^n$. Then $\Pr(ABx = Cx) \leq \frac{1}{2}$.

Proof: Since $AB \neq C$, $(AB - C) \neq 0$, so there exist i and j such that $(AB - C)_{ij} \neq 0$. Let $(a_1, a_2, \ldots, a_n)$ be the i^{th} row of $AB - C$, and consider $\sum_{k=1}^{n} a_k x_k$.

$$\begin{aligned}
\Pr((AB - C)x = 0) &\leq \Pr(\sum_{k=1}^{n} a_k x_k = 0) \\
&= \Pr(x_j = \frac{-1}{a_j} \sum_{k \neq j} a_k x_k) \\
&\leq \frac{1}{2},
\end{aligned}$$

since $a_j \neq 0$ and $x_j \in \{0,1\}$ equiprobably. □

This is a rather peculiar notion of "correctness": on some inputs, the algorithm may give the wrong answer half the time. However, this error probability can be decreased to 2^{-k} at the expense of increasing the time to $O(kn^2)$, for any k, by the following simple device. Run the algorithm for k independent trials (i.e., choosing k independent vectors x), and return "$AB \neq C$" if at least one of these trials returns that answer and "$AB = C$ probably" if none of them do.

Open Problem 1: Can this problem be solved deterministically in $O(n^2)$ time? Note that this may be an easier problem than determining if two matrices can be multiplied in $O(n^2)$ time.

1.6. Definitions

There is a fundamental difference between the way Quicksort behaves and the way Freivald's algorithm behaves: Quicksort is always correct, but with small probability runs slowly, whereas Freivald's algorithm always runs fast, but with small probability makes an error in its output. The following definition captures this distinction in types of probabilistic algorithms.

Definition: A probabilistic algorithm M accepts a language L with

- *0-sided error* if and only if $\Pr(M \text{ accepts } x \mid x \in L) = 1$ and $\Pr(M \text{ accepts } x \mid x \notin L) = 0$.
- *1-sided error* if and only if $\Pr(M \text{ accepts } x \mid x \in L) \geq \frac{1}{2}$ and $\Pr(M \text{ accepts } x \mid x \notin L) = 0$.
- *2-sided error* if and only if $\Pr(M \text{ accepts } x \mid x \in L) \geq \frac{2}{3}$ and $\Pr(M \text{ accepts } x \mid x \notin L) \leq \frac{1}{3}$.

Definition: The *error bound* of a 1-sided error algorithm is

$$1 - \mathrm{glb}_x\{\Pr(M \text{ accepts } x \mid x \in L)\},$$

where glb denotes the greatest lower bound.

Definition: The *error bound* of a 2-sided error algorithm is

$$\max\{1 - \mathrm{glb}_x\{\Pr(M \text{ accepts } x \mid x \in L)\},\ \mathrm{lub}_x\{\Pr(M \text{ accepts } x \mid x \notin L)\}\},$$

where glb denotes the greatest lower bound and lub the least upper bound.

Note that the matrix multiplication algorithm of Section 1.5 is a 1-sided error algorithm for the language $\{(A, B, C) \mid AB \neq C\}$ with an error bound of $\frac{1}{2}$. Quicksort cannot technically be classified as a 0-sided error algorithm, since sorting is not a language recognition problem.

Lecture 2

How to Improve 2-Sided Error Algorithms

January 9, 1991
Notes: João Setubal

2.1. Introduction

In the previous lecture we saw how to decrease the error probability of a 1-sided error algorithm by the method of repeated trials. By that strategy, if the algorithm accepts its input in at least one trial, we know for certain that the input is in the language being recognized. On the other hand, if the input is not accepted in any trial, the probability that the algorithm is wrong in all of them is exponentially small.

This method at first does not seem to apply to 2-sided error algorithms, since in this case even if the algorithm accepts its input in a trial, it may be giving the wrong answer. We will see in this lecture that repeated trials can in fact be used to decrease the error probability just as we did for 1-sided error algorithms.

2.2. Decreasing the Error Probability by Repeated Trials

First we state the following theorem, due to Chernoff [13]:

Theorem 2.1: Let $0 < p < 1/2$, $q = 1 - p$, and k be an even integer. Then

$$\sum_{i=k/2}^{k} \binom{k}{i} p^i q^{k-i} \leq (4pq)^{k/2}$$

The interpretation of the theorem is as follows. Suppose $p < 1/2$ is the probability of success of a certain event, and q the probability of failure. If we perform k independent trials, the left-hand side of the inequality above represents the probability that at least half of them are successes, and the theorem asserts that this probability decreases exponentially with the number of trials (since $4pq < 1$).

Given this theorem, we can now show that by running a 2-sided error algorithm multiple times we can in fact decrease its error bound exponentially.

Corollary 2.2: If a probabilistic algorithm M accepts language L using 2-sided error, time T, and error bound $\epsilon \leq 1/3$, then for all k there is a probabilistic algorithm M' that accepts L in time $O(kT)$ and with error bound $2^{-\Omega(k)}$.

Proof: Since the bounds are asymptotic, we can assume without loss of generality that k is even. Given an input x, M' runs k independent trials of M on x and accepts x if and only if the majority of these trials accept x. This gives the desired time bound. We prove the error bound as follows.

Let $\epsilon(x) = \Pr(\text{M is wrong on input } x)$. Since $\epsilon(x) \leq \epsilon \leq 1/3$ (by the definition of error bound), $\epsilon(x)(1-\epsilon(x)) \leq (1/3)\cdot(2/3)$. Note that if M' gives the wrong answer on input x, then in at least half of the trials M' got wrong answers from M. That is,

$$\begin{aligned}
\Pr(M' \text{ is wrong on input } x) &\leq \sum_{i=k/2}^{k} \binom{k}{i} (\epsilon(x))^i (1-\epsilon(x))^{k-i} \\
&\leq (4\epsilon(x)(1-\epsilon(x)))^{k/2} && \text{(Theorem 2.1)} \\
&\leq \left(4\cdot\frac{1}{3}\cdot\frac{2}{3}\right)^{k/2} \\
&= \left(\frac{8}{9}\right)^{k/2} = 2^{-\Omega(k)}
\end{aligned}$$

□

The proof of Corollary 2.2 shows that there is nothing magical about using 1/3 in the definition of 2-sided error: any constant less than 1/2 would do. We note that if ϵ were not bounded away from 1/2 by a constant, a constant number of trials would not be enough to decrease the error bound by as much as we wish. For example, if $\epsilon = 1/2 - 1/n^2$, we would have to run M a polynomial number of times simply to get the error probability bounded away from 1/2 by a constant. If $\epsilon = 1/2 - 1/2^n$ we would have to run M an exponential number of times.

Lecture 3

From Expected Time to Worst Case Time

January 11, 1991
Notes: Donald Chinn

3.1. Error Bounds Not Bounded Away From 1/2

Last lecture we showed a way to reduce the error in a 2-sided error algorithm by repeatedly running the algorithm. But if the error is not bounded away from 1/2 by a constant, then the number of times we would have to repeat the algorithm to bound the error away from 1/2 depends on how close ϵ is to 1/2. For example, if $\epsilon = \frac{1}{2} - \frac{1}{n^2}$, then it takes n^4 trials to bound the error away from 1/2.

To see this, note that the Chernoff bound (Theorem 2.1) says that the probability that the wrong answer is given in the majority of k trials is less than $(4\epsilon(1-\epsilon))^{k/2}$. But

$$\begin{aligned} & (4\epsilon(1-\epsilon))^{k/2} \\ = \; & \left(4\left(\frac{1}{2}-\frac{1}{n^2}\right)\left(\frac{1}{2}+\frac{1}{n^2}\right)\right)^{k/2} \\ = \; & \left(1-\frac{4}{n^4}\right)^{k/2} \approx \frac{1}{e} \qquad \text{when } k \approx n^4/2 \end{aligned}$$

since $\lim_{x\to\infty}(1-1/x)^x = 1/e$.

Another question that arose concerning error bounds not bounded away from 1/2 by a constant is where these bounds arise in practice. A possible example is if the algorithm needs an n-bit prime number. The most straightforward way to do this is to choose a random n-bit integer x, because the primes are reasonably densely distributed: by the prime number theorem, $\Pr(x \text{ is prime }) \approx 1/n$. This could conceivably lead to an algorithm whose error is bounded away from 1/2 by only $1/n$.

3.2. Turning Expected Time into Worst Case Time

We now turn to the following question: is there a way, given a probabilistic algorithm with 0-sided error and expected time $T(n)$, to construct an algorithm that runs in worst case running time not much greater than $T(n)$? First, we need to define what expected and worst case times are.

Definition: A probabilistic algorithm M runs in *expected time* $T(n)$ if and only if, for all x, the expected running time of M on input x is $T(|x|)$.

Notice that the expectation in this definition is over the outputs of the random number generator, not over the inputs to M.

Definition: A probabilistic algorithm M runs in *worst case time* $T(n)$ if and only if, for all x, the worst case running time of M on input x is $T(|x|)$.

As we shall see, the answer to the question above is 'yes', but we introduce a small error into the output (that is, we produce an algorithm with 1-sided error). Before we get to the proof, however, we need the following lemma, which bounds the probability that a random variable greatly exceeds its expected value.

Lemma 3.1 (Markov's inequality): Let X be a discrete nonnegative random variable with expectation (that is, there is an expected value $E(X)$ of X). Then, for all positive k,

$$\Pr(X \geq kE(X)) \leq \frac{1}{k}.$$

Proof: Suppose $\Pr(X \geq kE(X)) > \frac{1}{k}$. Then

$$\begin{aligned}
E(X) &= \sum_i i \cdot \Pr(X = i) \\
&= \sum_{i<kE(X)} i \cdot \Pr(X = i) + \sum_{i \geq kE(X)} i \cdot \Pr(X = i) \\
&\geq 0 + \sum_{i \geq kE(X)} kE(X) \Pr(X = i) \\
&= kE(X) \sum_{i \geq kE(X)} \Pr(X = i) \\
&= kE(X) \Pr(X \geq kE(X)) \\
&> E(X). && \text{(by assumption } \Pr(X \geq kE(X)) > \tfrac{1}{k}\text{)}
\end{aligned}$$

This contradiction establishes the lemma. □

An immediate consequence of Markov's inequality is the answer to our original question.

Corollary 3.2: If L is accepted by a probabilistic algorithm M with 0-sided error and expected time $T(n)$, then L is accepted by a probabilistic algorithm M' with 1-sided error and worst case time $O(T(n))$, assuming $T(n)$ is time-constructible.

(A function $T(n)$ is *time-constructible* if and only if there is an algorithm that, given an input of length n, outputs the value $T(n)$ within $O(T(n))$ steps. In practice, all time bounds that one encounters are time-constructible. For example, n^3 and $\log n$ are time-constructible.)

Proof: Construct M' as follows. M' runs M for $2T(n)$ steps (which M' can compute since $T(n)$ is time-constructible) and accepts if and only if M accepts within that time.

Correctness:

Case 1:

$$\begin{aligned}\Pr(M' \text{ accepts } x \mid x \notin L) &= \Pr(M \text{ accepts } x \text{ within } 2T(n) \text{ steps } \mid x \notin L) \\ &= 0.\end{aligned}$$

Case 2:

$$\begin{aligned}\Pr(M' \text{ rejects } x \mid x \in L) &= \Pr(M \text{ rejects } x \text{ within } 2T(n) \text{ steps } \mid x \in L) \\ &\quad + \Pr(M \text{ runs for } > 2T(n) \text{ steps on } x \mid x \in L) \\ &\leq 0 + \frac{1}{2},\end{aligned}$$

by Lemma 3.1. □

If one wanted to reduce the error bound in Corollary 3.2, M' could of course be run multiple times. Another way to reduce the error would be to run M directly for a longer period of time than just $2T(n)$ (say, $100T(n)$). The natural question to ask, then, is which is better: repeating the $2T(n)$ version of M $\frac{k}{2}$ times or running M for $kT(n)$ steps? The answer is that the error will decay exponentially with k for the first case, whereas Markov's inequality can only guarantee a decay of $\frac{1}{k}$ in the second case (as in case 2 of Corollary 3.2).

3.3. Polynomial Time Complexity Classes

In this section we introduce the probabilistic analogues of the complexity class P. It is convenient to classify the various kinds of probabilistic algorithms (depending on the level of error and whether we are looking at worst case or expected time). There are six classes when we vary these two orthogonal concepts.

Definition:

- $ZPP = \{L \mid L$ is accepted by some probabilistic algorithm with 0-sided error and polynomial expected time$\}$. (ZPP stands for "Zero-sided Probabilistic Polynomial time".)
- $RP = \{L \mid L$ is accepted by some probabilistic algorithm with 1-sided error and polynomial worst case time$\}$. (RP stands for "Randomized Polynomial time".)
- $BPP = \{L \mid L$ is accepted by some probabilistic algorithm with 2-sided error and polynomial worst case time$\}$. (BPP stands for "Bounded error Probabilistic Polynomial time".)

The other three classes already have names.

1. $\{L \mid L$ is accepted by some probabilistic algorithm with 0-sided error and polynomial worst case time$\}$ is just P since, for any machine M that accepts any language in the former set, there is a deterministic machine M' that accepts the same language by simulating M as though all the random bits were 1. Containment in the other direction is obvious.

2. $\{L \mid L$ is accepted by some probabilistic algorithm with 1-sided error and polynomial expected time$\}$ is just RP, since we can just cut off the computation and reject after $3T(n)$ steps and use repeated trials to correct the error bound, invoking Lemma 3.1.

3. Similar comments hold for $\{L \mid L$ is accepted by some probabilistic algorithm with 2-sided error and polynomial expected time$\}$, which is just BPP.

The next natural question to ask is what the relationships are among these complexity classes.

Theorem 3.3: $P \subseteq ZPP \subseteq RP \subseteq NP$, and $RP \subseteq BPP$.

Proof: $P \subseteq ZPP$: The machines that accept languages in P are restrictions of machines that accept languages in ZPP (both in the random number generator and the running time).

$ZPP \subseteq RP$: Corollary 3.2.

$RP \subseteq NP$: Given a probabilistic algorithm whose language is in RP, we can convert it into a nondeterministic algorithm by guessing a sequence of random bits that lead to acceptance. Imagine a *computation tree* (left undefined formally here) on an input x, where at each step of the computation, we flip a coin and have two branches in the tree corresponding to heads and tails. If $x \in L$, then at least half the leaves accept, and thus the NP algorithm will accept (it only needs one accepting leaf to accept). If $x \notin L$, then the RP algorithm will not accept, meaning all the leaves of the computation tree are rejecting branches, and thus our NP algorithm will reject.

$RP \subseteq BPP$: Run the RP algorithm twice in order to increase the probability of accepting $x \in L$ from $\frac{1}{2}$ to $\frac{3}{4}$ (which is is greater than $\frac{2}{3}$). □

3.4. Some History

Now that we know some of the relationships among these classes, it would be nice to know if any of these classes were actually different from each other. For example, it would be nice to know if there are languages in ZPP that are not in P.

Solovay and Strassen [38] proved that the set of composite integers is in RP. It was this startling result that drew many algorithms researchers into studying probabilistic algorithms. On hearing of Solovay and Strassen's result, Rabin [32] provided an entirely different proof. A decade later, Adleman and Huang [2] proved that the composites are in ZPP.

Lecture 4

Verifying Identities

January 14, 1991
Notes: George Forman

The examples given so far of probabilistic algorithms are not very dramatic: in both quicksort and matrix multiplication, the advantage gained by using randomization is only a small polynomial speedup. In this lecture we will see an example of a problem for which randomization seems to make the difference between polynomial and superpolynomial time.

Schwartz [36] discovered a general method for using randomization to verify large identities, similar to Freivalds' probabilistic algorithm for verifying the matrix identity $AB = C$ from Lecture 1. In this lecture we will give one concrete example of Schwartz's method, namely verifying purported counterexamples to Fermat's Last Theorem. The analysis in this lecture follows that of Karp and Rabin [24]. The current version is a simplification and strengthening of the original lecture, due to Chinn, Thrash, and Walkup [personal communication].

Suppose you were given a counterexample to Fermat's Last "Theorem", which maintains that there are no positive integers a, b, c, k such that $a^k + b^k = c^k$ for $k > 2$. How could you verify this counterexample if each number had 500 digits? More generally, given n-bit numbers a, b, c and k, check whether $a^k + b^k = c^k$. It is not at all clear how to do this deterministically in time polynomial in n, because for example c^k has exponentially many bits. But we can solve this probabilistically in polynomial time:

Theorem 4.1: $\{(a, b, c, k) \mid a^k + b^k \neq c^k\} \in RP$.

Proof:

CONSTRUCTION: Let $L = \max(\lceil k \log_2(a + b + c)\rceil, 89)$. Choose a random positive integer $m \leq L^2$ and accept if and only if $a^k + b^k \not\equiv c^k \pmod{m}$.

ANALYSIS: This algorithm runs in polynomial time by Lemma 4.2 below, since m has $O(n)$ bits.

CORRECTNESS:

1. It is easy to see that $\Pr(a^k + b^k \not\equiv c^k \pmod{m} \mid a^k + b^k = c^k) = 0$.
2. The remaining part, that $\Pr(a^k + b^k \equiv c^k \pmod{m} \mid a^k + b^k \neq c^k) < \frac{1}{2}$, will be proved in Lemma 5.1 of next lecture, with the help of the lemmas proved in the remainder of this lecture.

$\square$

Lemma 4.2: If a, k, and m are n-bit integers, then $a^k \bmod m$ can be computed deterministically in polynomial time (in fact, in polynomially many operations on individual bits).

Proof: Left as an exercise. Hint: beginning with a, alternately square and reduce modulo m. □

For the proof of correctness of Theorem 4.1, we will need the following three facts about prime numbers, all of which can be found in Rosser and Schoenfeld [34].

Theorem 4.3 (Prime Number Theorem): Let $\pi(x)$ denote the number of primes not exceeding x. Then for all $x \geq 17$,

$$\frac{x}{\ln x} \leq \pi(x) \leq 1.25506\frac{x}{\ln x}.$$

Theorem 4.4 (Mertens): There exists a constant B such that, for all positive integers x,

$$\ln\ln x + B - \frac{1}{2\ln^2 x} < \sum_{\substack{p \leq x \\ p \text{ prime}}} \frac{1}{p} < \ln\ln x + B + \frac{1}{\ln^2 x}.$$

Theorem 4.5: For all $x \geq 41$,

$$\sum_{\substack{p \leq x \\ p \text{ prime}}} \ln p > x(1 - \frac{1}{\ln x}).$$

Corollary 4.6: For all $x \geq 41$,

$$\prod_{\substack{p \leq x \\ p \text{ prime}}} p > 2^x.$$

Proof: $x(1 - 1/\ln x) > x \ln 2$, provided $x \geq 27$. The corollary then follows from Theorem 4.5 by exponentiating both sides of the inequality. □

Definition: An integer x is called *M-fat* if and only if $1 \leq x \leq M$ and x has a prime factor exceeding $\sqrt{M}$. Let $F(M)$ be the number of M-fat integers.

Lemma 4.7: For all $L \geq 89$, $F(L^2) > \frac{1}{2}L^2$.

Proof: If x is L^2-fat, then exactly one prime $p > L$ divides x. Hence,

$$\begin{aligned}
F(L^2) &= \sum_{\substack{L<p\leq L^2 \\ p \text{ prime}}} \left\lfloor \frac{L^2}{p} \right\rfloor \\
&\geq \sum_{\substack{L<p\leq L^2 \\ p \text{ prime}}} \left(\frac{L^2}{p} - 1 \right) \\
&= \left(L^2 \sum_{\substack{L<p\leq L^2 \\ p \text{ prime}}} \frac{1}{p} \right) - \left(\sum_{\substack{L<p\leq L^2 \\ p \text{ prime}}} 1 \right).
\end{aligned}$$

Applying Theorems 4.3 and 4.4,

$$\begin{aligned}
F(L^2) &> L^2\left(\left(\ln\ln L^2 + B - \frac{1}{2\ln^2 L^2}\right) - \left(\ln\ln L + B + \frac{1}{\ln^2 L}\right)\right) - \left(1.25506\frac{L^2}{\ln L^2} - \frac{L}{\ln L}\right) \\
&= L^2\left(\ln(2\ln L) - \ln\ln L - \frac{1.25506}{2\ln L} - \frac{1}{8\ln^2 L} - \frac{1}{\ln^2 L}\right) + \frac{L}{\ln L} \\
&= L^2\left(\ln 2 - \frac{0.62753}{\ln L} - \frac{9}{8\ln^2 L}\right) + \frac{L}{\ln L} \\
&> \frac{1}{2}L^2.
\end{aligned}$$

The last inequality holds for all $L \geq 89$ since it holds for $L = 89$, and the difference between the last two lines is an increasing function of L for $L \geq 89$. $\square$

Lemma 4.8: For all $L \geq 89$, the following holds: if you choose any set T of at least $\frac{1}{2}L^2$ integers from $\{1, 2, 3, ..., L^2\}$, their least common multiple l exceeds 2^L.

Proof: Without loss of generality, assume that T minimizes l among all sets of at least $\frac{1}{2}L^2$ integers from $\{1, 2, 3, \ldots, L^2\}$. By Lemma 4.7, T contains some L^2-fat integer. That is, some prime $r > L$ divides l. But then every prime $q \leq r$ must also divide l: if not, then each factor of r in T could be decreased to q, contradicting the minimality of l. The result then follows from Corollary 4.6, since

$$l \geq \prod_{\substack{p \leq r \\ p \text{ prime}}} p > 2^r > 2^L.$$

$\square$

The constant 89 in Lemma 4.8 is overly conservative. By using a combination of other clever techniques and some computer assistance, Thrash [personal communication] proved that Lemma 4.8 holds for all $L \geq 3$, which is best possible.

Lecture 5

Traversal of Undirected Graphs

January 16, 1991
Notes: Tracy Kimbrel

5.1. Verifying Identities (conclusion)

All that remains to prove from the previous lecture is that the algorithm accepts any input in $\{(a,b,c,k) \mid a^k + b^k \neq c^k\}$ with probability greater than 1/2.

Lemma 5.1: Assume that $a^k + b^k \neq c^k$, and let $L = \max(\lceil k \log_2(a+b+c) \rceil, 89)$. Suppose m is chosen randomly and uniformly from $\{1, 2, \ldots, L^2\}$. Then $\Pr(a^k + b^k \equiv c^k \pmod{m}) < \frac{1}{2}$.

Proof: Suppose that at least $\frac{1}{2}L^2$ values of m satisfy $a^k + b^k \equiv c^k \pmod{m}$, and let l be their least common multiple. Since each m divides $a^k + b^k - c^k$, so does their least common multiple; that is, $a^k + b^k \equiv c^k \pmod{l}$. By Lemma 4.8, $l > 2^L \geq 2^{\lceil k \log_2(a+b+c) \rceil} \geq (a+b+c)^k$. Since l is greater than either side of the equivalence $a^k + b^k \equiv c^k \pmod{l}$, $a^k + b^k = c^k$, a contradiction. □

Open Problem 2: Is $\{(a,b,c,k) \mid a^k + b^k \neq c^k\}$ in P? Perhaps there is a clever way of deterministically choosing polynomially many moduli m so that it suffices to check the equivalence modulo each of them. Failing this, is this problem in *ZPP*?

5.2. Introduction to Traversal of Undirected Graphs

The next topic is another probabilistic algorithm, one that solves a problem in graph theory, namely, the traversal of undirected graphs. The problem is to determine, given an undirected graph $G = (V, E)$ and a pair of distinguished vertices $s, t \in V$, whether there exists a path from s to t in G. (This problem is known as USTCON, for Undirected $s-t$-Connectivity.)

There are well known deterministic algorithms that solve this problem in time $O(n+e)$, where $n = |V|$ and $e = |E|$, namely breadth-first search and depth-first search. Since this time is optimal, why do we need a probabilistic algorithm? Breadth-first search and depth-first search each require $\Omega(n)$ space (at least one bit for each vertex, to record whether or not the vertex has been visited); perhaps a probabilistic algorithm can use less space.

How can an algorithm run in sublinear space, when representing the input itself requires $\Omega(n)$ bits? Our model of computation assumes that the input is read-only, and that the space it occupies is not counted in the space bound. Some justifications for this model include the following:

- The input may be stored elsewhere on a real machine, e.g. in someone else's file directory or on some other file system.
- Subroutine parameters that are not copied by the subroutine are counted only once if they are not charged to the subroutine.
- Computations using external storage devices, such as database search problems, fit this model.
- Traversal of implicitly defined graphs, such as AI search-space problems, fit this model.
- From a theoretical point of view, sublinear space bounds have very interesting properties, which it would be impossible to study otherwise. For instance, there is a close relationship between space bounds and parallel time bounds. (See, for instance, Chandra and Stockmeyer [12].) By studying sublinear space bounds, we learn about the corresponding sublinear parallel time bounds.

The workspace bounds will be measured in bits (rather than, say, registers each capable of containing an integer). Returning to the question of whether USTCON can be solved in sublinear space, in a result that was very surprising 20 years ago, Savitch [35] showed that it can be solved in $O(\log^2 n)$ space; in fact, the directed version can be solved in this space bound. There is a drawback to this algorithm, however: it uses superpolynomial time.

Open Problem 3: Can USTCON be solved in $O(\log n)$ space? Can it be solved in $O(\log^{1+\epsilon} n)$ space for $\epsilon < 1$? Can it be solved in $O(\log^2 n)$ space and polynomial time simultaneously?

5.3. Log Space Complexity Classes

Definition:

- *ZPLP* = $\{L \mid L$ is accepted by some probabilistic algorithm with 0-sided error, polynomial expected time, and $O(\log n)$ space$\}$.
  (ZPLP stands for "Zero-sided Probabilistic Log space and Polynomial time".)
- *RLP* = $\{L \mid L$ is accepted by some probabilistic algorithm with 1-sided error, polynomial worst case time, and $O(\log n)$ space$\}$.
  (RLP stands for "Randomized Log space and Polynomial time".)
- *BPLP* = $\{L \mid L$ is accepted by some probabilistic algorithm with 2-sided error, polynomial worst case time, and $O(\log n)$ space$\}$.
  (BPLP stands for "Bounded error Probabilistic Log space and Polynomial time".)

The goal for the next few lectures will be to show a 1-sided error, $O(\log n)$ space, $O(ne)$ time probabilistic algorithm for USTCON:

Theorem 5.2 (Aleliunas, Karp, Lipton, Lovász, Rackoff [4]): USTCON $\in$ *RLP*.

Unlike earlier algorithms for graph traversal, this result seems to differentiate between the complexities of the undirected and directed cases, since we don't know how to prove the analogous result for directed $s-t$-connectivity, nor do many researchers believe it possible.

Open Problem 4: Is directed $s-t$-connectivity in BPLP?

5.4. Random Walks on Graphs

Aleliunas *et al.*'s probabilistic algorithm for USTCON takes a "random walk" on G, starting at s and looking for t. Informally, a random walk is simply a random path in which the next vertex is chosen equiprobably among all neighbors of the current vertex.

Definition: A *random walk* on a graph $G = (V, E)$ starting at s is a random sequence $v_0, v_1, v_2, \ldots$ of vertices such that $v_0 = s$ and, for all $i \geq 0$,

$$\Pr(v_{i+1} = w \mid v_i = u) = \begin{cases} \frac{1}{deg(u)} & \text{if } \{u, w\} \in E \\ 0 & \text{otherwise} \end{cases} .$$

The algorithm for USTCON is as follows: Take a random walk on G starting at s for time $O(ne)$. If vertex t is reached, accept; if not, reject (possibly incorrectly).

As an example, consider a graph consisting of an $\frac{n}{2}$-clique (i.e., $\frac{n}{2}$ vertices with each possible edge present) connected at one of its vertices v to one end of an $\frac{n}{2}$-chain, with s in the clique, and t at the opposite end of the chain from v. Note that, even when the walk reaches v, the probability that the walk escapes the clique and enters the chain is only $2/n$. Even if the walk does escape the clique and move a few vertices toward t, it is much more likely to fall back into the clique than it is to reach t. Incredibly, even in this pathological case, the algorithm works.

Exercise: What is the expected time for a random walk to reach t from s if the graph is an n-clique? An n-chain with s and t at opposite ends?

Lecture 6

Random Commutes on Graphs

January 18, 1991
Notes: Joan Lawry

6.1. Commutes

Recall that our goal from the previous lecture is to show a 1-sided error, $O(\log n)$ space, $O(ne)$ time probabilistic algorithm for USTCON. We first need some preliminary results on random walks.

Definition: A *commute* from i to j is a path that starts at i and ends the first time it returns to i after having visited j. A *random commute* is a random walk that forms a commute.

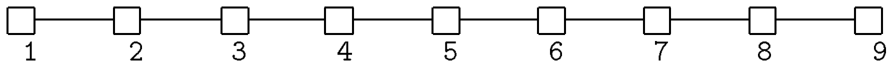

Figure 6.1: Chain of 9 Vertices

Example 6.1: A commute from 2 to 3 on the chain in Figure 6.1 might look like 2, 1, 2, 1, 2, 3, 4, 5, 4, 5, 4, 3, 2. Note that it may visit i multiple times before encountering j, and then j multiple times before its return to i.

6.2. Relevant Measures in Random Commutes

Definition: For all i, j, u, v, with $\{u, v\} \in E$, let c_{ijuv} be a random variable that is the number of times a random commute from i to j crosses the edge $\{u, v\}$ in the direction from u to v. Let $\theta_{ijuv} = \mathrm{E}(c_{ijuv})$.

Example 6.2: For the graph in Figure 6.1, $\theta_{2323} = 1$, and $\theta_{2343} = 1$. The former is straightforward from the definition, and the latter can be seen as follows:

$$\Pr(\{3,4\} \text{ is crossed}) = 1/2$$

$$\mathrm{E}(c_{2343} \mid \{3,4\} \text{ is crossed}) = 2$$

$$\theta_{2343} = \mathrm{E}(c_{2343}) = \mathrm{E}(c_{2343} \mid \{3,4\} \text{ is crossed}) \Pr(\{3,4\} \text{ is crossed}) = 1.$$

Although one might expect θ_{2389} to be much less, it turns out that it is also 1. Intuitively, though it is unlikely that the random commute would reach $\{8, 9\}$, once there the commute would probably traverse $\{8, 9\}$ several times before returning to 2.

6.3. Independence of θ_{ijuv} from u and v

Example 6.2 suggests that, in a random commute between neighboring chain vertices, the expected number of times any edge is crossed in either direction is 1. Not only is this true, but in fact a surprising generalization holds for all graphs: For any connected, undirected graph $G = (V, E)$, and for any two vertices $i, j \in V, \theta_{ijuv}$ is independent of u and v (i.e., it is the same for all edges $\{u, v\}$). This result was originally proved by Göbel and Jagers [18], using fundamental theorems from the theory of Markov chains. The proof we will present is due to Zuckerman, and is elementary and self-contained.

First, we need some preliminary lemmas.

Lemma 6.3: For any vertices i, j, and u, θ_{ijuv} is independent of v (i.e., it is the same for all neighbors v of u).

Lecture 7

Random Commutes on Graphs, continued

January 23, 1991
Notes: Elizabeth Walkup

Proof of Lemma 6.3: We begin by extending a random commute into an infinite process. Let $y_0, y_1, y_2, \ldots$ be an infinite sequence of random vertices, whose prefix $y_0, y_1, \ldots, y_m$ is a random commute from i to j and for which $y_k = i$ for all $k > m$.

For all $k \in \mathcal{N}$ (the nonnegative integers) and for all $\{u, v\} \in E$, define the random variable X_{kuv} as

$$X_{kuv} = \begin{cases} 1 & \text{if } y_k = u \ \& \ y_{k+1} = v \\ 0 & \text{otherwise} \end{cases} .$$

Then

$$\begin{aligned}
\theta_{ijuv} &= \mathrm{E}(c_{ijuv}) \\
&= \mathrm{E}\left(\sum_{k=0}^{\infty} X_{kuv}\right) \\
&= \sum_{k=0}^{\infty} \mathrm{E}\left(X_{kuv}\right) && \text{(Chow and Teicher [14, page 89, Corollary 2])} \\
&= \sum_{k=0}^{\infty} \left(1 \cdot \Pr\left(X_{kuv} = 1\right) + 0 \cdot \Pr\left(X_{kuv} = 0\right)\right) \\
&= \sum_{k=0}^{\infty} \Pr\left(y_k = u \ \& \ y_{k+1} = v\right) \\
&= \sum_{k=0}^{\infty} \Pr\left(y_k = u \ \& \ y_{k+1} \neq u \ \& \ y_{k+1} = v\right) \\
&= \sum_{k=0}^{\infty} \Pr\left(y_{k+1} = v \mid y_k = u \ \& \ y_{k+1} \neq u\right) \cdot \Pr\left(y_k = u \ \& \ y_{k+1} \neq u\right) \\
&= \sum_{k=0}^{\infty} \frac{1}{\deg(u)} \cdot \Pr\left(y_k = u \ \& \ y_{k+1} \neq u\right)
\end{aligned}$$

which leaves us with an expression independent of v. □

As a general rule, if you are trying to get a bound on the expected number of times some event happens (in this case it was the number of times $\{u, v\}$ is crossed), an approach that is often successful is to introduce a binary valued random variable for each time step, replace the expected sum of these variables by the sum of their expectations, and then simplify. The reason this approach is successful is that it allows you to treat the expectations of the binary valued random variables individually, even though they may not be independent (as was certainly the case in this proof).

Exercise: Prove that $E(X+Y) = E(X)+E(Y)$, where X and Y are discrete random variables that are not necessarily independent.

From this, it follows by induction that the expectation of a finite sum equals the sum of the expectations. The proof needed for an infinite sum, as in the proof of Lemma 6.3, is subtler.

Lemma 7.1 (Zuckerman): For any vertices i, j, u, and v, $\theta_{ijuv} = \theta_{ijvu}$.

Proof: For any commute from i to j

$$C = \left(\underline{i, w_1, \ldots, w_a,} i, x_1, \ldots, x_b, \underline{j, y_1, \ldots, y_c,} j, z_1, \ldots, z_d, i\right)$$

where, for all k, $x_k \neq i$, $y_k \neq i$, and $z_k \neq i$, and $w_k \neq j$, $x_k \neq j$, and $z_k \neq j$, define the *reversal* C^R of C as

$$C^R = \left(\underline{i, w_a, \ldots, w_1,} i, z_d, \ldots, z_1, \underline{j, y_c, \ldots, y_1,} j, x_b, \ldots, x_1, i\right).$$

(It is possible that either or both of the underlined sequences are missing from C, in which case the corresponding underlined sequence(s) are deleted from C^R.) Then C^R has the following properties:

1. C^R is a commute from i to j.

2. $(C^R)^R = C$, so that there is a one-to-one correspondence between commutes and their reversals.

3. Let Γ_{ij} be a random commute from i to j. Then $\Pr(\Gamma_{ij} = C) = \Pr(\Gamma_{ij} = C^R)$. This is because the probability of a given commute C occurring is the product of the reciprocals of the degrees of all the vertices in the commute, excluding the last vertex. C and C^R both end at i and both contain the same vertices (with the same multiplicity of each), so the products are identical.

4. Let $N(C, u, v)$ be the number of times C crosses the edge $\{u, v\}$ in the direction from u to v. Then $N(C, u, v) = N(C^R, v, u)$.

From these four facts, if Γ_{ij} is a random commute from i to j,

$$\begin{aligned}
\theta_{ijuv} &= \sum_{\text{commutes } C} \Pr(\Gamma_{ij} = C)\, N(C, u, v) \\
&= \sum_{\text{commutes } C} \Pr(\Gamma_{ij} = C^R) N(C^R, v, u) \\
&= \sum_{\text{commutes } C^R} \Pr(\Gamma_{ij} = C^R) N(C^R, v, u) \\
&= \theta_{ijvu}.
\end{aligned}$$

□

Theorem 7.2 (Göbel and Jagers [18]): For any connected, undirected graph $G = (V, E)$, and for any two vertices $i, j \in V$, θ_{ijuv} is independent of u and v (i.e., it is the same for all edges $\{u, v\}$ and each of the two directions $\{u, v\}$ could be crossed).

Proof: Let $\{u, v\}$ and $\{u', v'\}$ be two edges, and let $P = (u, v, w, \ldots, u', v')$ be a path containing both. (Such a path exists because G is connected.) Applying Lemmas 6.3 and 7.1 to P alternately, we have the following:

$$\theta_{ijuv} = \theta_{ijvu} = \theta_{ijvw} = \theta_{ijwv} = \cdots = \theta_{iju'v'}.$$

□

Lecture 8

General Random Walks on Graphs

January 25, 1991
Notes: Wendy Thrash

8.1. From Commutes to General Random Walks

Following a two-lecture tour of random commutes, we complete our own commute by returning to the topic of general random walks on graphs, introduced three lectures ago. The following result of Aleliunas *et al.* is a rather surprising application of random commutes.

Theorem 8.1 (Aleliunas, Karp, Lipton, Lovász, Rackoff [4]): Let $G = (V, E)$ be a connected, undirected graph with $n = |V|$ and $e = |E|$. For any starting vertex, the expected number of steps that a random walk needs in order to visit every vertex in V is at most $4ne$.

Proof: Consider an infix traversal of any spanning tree of G. (Such a spanning tree exists because G is connected.) We shall show that the expected time to visit the vertices of G *in the order given by this traversal* is no more than $4ne$.

Let T_{ij} be the expected number of steps for a random walk to get from vertex i to vertex j, where i and j are adjacent in the spanning tree. Certainly T_{ij} is no greater than the expected number of steps in a random commute from i to j, which is equal to the expected total number of crossings (in both directions) of all edges in a random commute from i to j. That is,

$$\begin{aligned}
T_{ij} &\leq \sum_{\substack{(u,v)\in V\times V \\ \{u,v\}\in E}} \theta_{ijuv} \\
&= \sum_{\substack{(u,v)\in V\times V \\ \{u,v\}\in E}} \theta_{ijij} && \text{(Theorem 7.2)} \\
&\leq \sum_{\substack{(u,v)\in V\times V \\ \{u,v\}\in E}} 1 \\
&= 2e. && \text{(each edge is counted twice)}
\end{aligned}$$

There are $n-1$ edges in a spanning tree, and each is traversed twice in an infix walk, so the expected number of steps in the entire walk is at most $2(n-1)\cdot 2e \leq 4ne$. $\square$

Exercise: The analogous result for directed graphs does not hold. Show this by finding a strongly connected directed graph and two vertices s and t such that a random walk from s requires exponential expected time to reach t.

The long-awaited proof of Theorem 5.2 (USTCON $\in$ *RLP*) follows easily from Theorem 8.1:

Proof of Theorem 5.2 [4]: Start at vertex s and take a random walk of $8ne$ steps, accepting if and only if t has been reached in that time. Clearly $\Pr((G, s, t)$ is accepted $\mid G$ has no $s - t$ path$)$ $= 0$, so it remains to show that $\Pr((G, s, t)$ is rejected $\mid G$ has an $s - t$ path$) \leq 1/2$.

Suppose that the graph has an $s - t$ path, and let X be a random variable representing the number of steps a random walk requires to reach t from s. Theorem 8.1 applied to the connected component containing s and t shows that $\mathrm{E}(X) \leq 4ne$, so

$$\begin{aligned} \Pr((G,s,t) \text{ is rejected}) &= \Pr(X > 8ne) \\ &\leq \Pr(X > 2\mathrm{E}(X)) \\ &\leq 1/2 \qquad \text{(Lemma 3.1)} \end{aligned}$$

Since the algorithm need only record a constant number of vertex names and a step counter that does not exceed $8ne$, $O(\log n)$ space suffices, and the proof is complete. □

Having finally established that USTCON $\in$ *RLP*, we close this section with a subsuming result that demonstrates 0-sided rather than 1-sided error:

Theorem 8.2 (Borodin, Cook, Dymond, Ruzzo, Tompa [8]): USTCON $\in$ *ZPLP*.

One interesting point about the proof of this theorem is that it combines Warshall's algorithm [40], the random walk of Aleliunas *et al.* (Theorem 8.1), and the Immerman [20]–Szelepcsényi [39] method of inductive counting, thus bringing to bear material from both algorithms and complexity.

8.2. Using a Random Number Generator

We have assumed so far that we can get uniformly distributed random integers in any desired range, for example, that we would be able to choose with equal probability the neighbors of a vertex of degree d. Begging the question of whether we can generate random numbers at all, how might we use a random number generator to make this equiprobable selection?

1. **Get someone else to solve the problem:** Perhaps the random number generator takes an integer d, and returns a uniformly chosen integer between 0 and $d - 1$.
2. **Generate random floating-point numbers:** Divide the interval [0,1] into d equal subintervals, generate a random number between 0 and 1, and choose a vertex accordingly. Because of finite precision, it is not possible to do this exactly, so the resulting assignments are not quite equiprobable. It might be possible to bound the deviations from equiprobable assignment and rework the probability statements in our theorems accordingly, but there is an easier solution.

3. **Throw away some random numbers:** Suppose we want random integers between 0 and $d-1$, but our random number generator gives us only a random and uniform bit. Let D be the next power of 2 that is at least d, and by $\log_2 D$ calls on the random bit generator construct a random and uniform integer x between 0 and $D-1$. If $0 \le x \le d-1$, use x; if not, discard x and repeat until x is between 0 and $d-1$. Since $D < 2d$, the expected number of repetitions is less than 2. If, for example, we were counting calls to the random number generator as the measure of time in Theorem 8.1, this process would raise the expected time bound from $4ne$ to $8ne \log_2 n$.

Lecture 9

Introduction to Boolean Circuits

January 30, 1991
Notes: Tor Jeremiassen

9.1. Random Walks (conclusion)

The question arose as to whether the bound of Theorem 8.1 is tight, in light of the fact that the analysis seems to be rather sloppy. It is not hard, however, to construct graphs for which the bound is asymptotically tight. As an example, consider any graph H with $n/2$ vertices and $\Theta(e)$ edges, connected at one of its vertices v to one end of an $\frac{n}{2}$-chain, with s in H, and t at the opposite end of the chain from v.

On the other hand, there are graphs for which this bound is not tight. A tighter upper bound in many cases is given by the following fascinating theorem:

Theorem 9.1 (Chandra, Raghavan, Ruzzo, Smolensky, Tiwari [11]): Let the *electrical resistance* R of a connected, undirected graph be the maximum effective resistance between any pair of vertices when a one ohm resistor is substituted for each edge. Then for any starting vertex, the expected number of steps that a random walk needs in order to visit every vertex is $O(eR\log n)$.

This bound is quite tight for *all* graphs, since for some starting vertex the expected number of steps must be at least eR [11].

9.2. Boolean Circuits

The last topic needed in order to understand the literature on pseudorandom generators is Boolean circuit complexity.

Definition: A *(Boolean* or *combinational) circuit* is an acyclic, oriented, directed graph with two distinguished subsets of vertices, called *inputs* and *outputs*. ("Oriented" means that the edges directed into any particular vertex are ordered.) The vertices have indegrees and labels as follows:

- *Inputs* have indegree 0 and are labeled consecutively from $\{x_1, x_2, \ldots, x_n\}$.
- All noninput vertices are called *gates*. A gate with indegree d is labeled by any function $f : \{0,1\}^d \to \{0,1\}$.

In what follows, unless specified otherwise it is assumed that the maximum indegree d is a constant with respect to the number n of input labels. Such circuits are said to have *bounded fanin*.

Definition: The *value* $\mathrm{val}(u) \in \{0,1\}$ of vertex u in a circuit on input $(b_1, b_2, \ldots, b_n) \in \{0,1\}^n$ is defined as follows:

- If u is an input with label x_i then $\mathrm{val}(u) = b_i$.
- If u is a gate with label $f : \{0,1\}^d \rightarrow \{0,1\}$, and the d vertices adjacent to u are $u_1, u_2, \ldots, u_d$, then $\mathrm{val}(u) = f(\mathrm{val}(u_1), \mathrm{val}(u_2), \ldots, \mathrm{val}(u_d))$.

A circuit with 1 output vertex u is said to *output* b on input $(b_1, b_2, \ldots, b_n)$ if the value of u on $(b_1, b_2, \ldots, b_n)$ is equal to b.

9.3. Families of Boolean Circuits

We would like to think about Boolean circuits as computing devices, but individual circuits can only take fixed size inputs. Thus we need to introduce the notion of a family of Boolean circuits.

Definition: An infinite vector $C = (c_0, c_1, \ldots, c_n, \ldots)$ is called a *family* of circuits if and only if, for all $n \in \mathcal{N}$, c_n is a circuit with n uniquely labeled inputs.

Definition: A family $C = (c_0, c_1, \ldots, c_n, \ldots)$ of one-output circuits *recognizes* the language $L \subseteq \{0,1\}^*$ if, and only if, for each n, on input $x \in \{0,1\}^n$, c_n outputs $\begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$.

9.4. Complexity of Circuits

Definition: The *size* of a circuit is the number of gates in the circuit. The *depth* of a circuit is the length of a longest path from any input to any output.

The notion of size can be thought of as a measure of sequential time, provided the circuit is looked upon as a sequential computing device. Depth can be considered a measure of parallel time, provided the circuit is looked upon as a parallel computing device, in which case size becomes a measure of the hardware requirement.

Definition: $\mathrm{SIZE}(S(n))$ is the set of all languages L recognized by some family $C = (c_0, c_1, \ldots, c_n, \ldots)$ of circuits such that for all $n \in \mathcal{N}$, the size of c_n is $O(S(n))$. $\mathit{PSIZE} = \cup_{k>0}\mathrm{SIZE}(n^k)$

Thinking of circuits as sequential computing devices, *PSIZE* is the circuit analog of *P*. However, families of circuits have an unrealistic power compared to algorithms: the structure of c_{n+1} need bear no resemblance to that of c_n. This property is called "nonuniformity", and leads to anomolies such as the following simple fact:

Proposition 9.2: There is an uncomputable language in SIZE(1).

Proof: Let $L \subseteq \{0,1\}^*$ be any uncomputable language. Consider $L' = \{x \mid \text{bin}(|x|) \in L\}$, where $\text{bin}(n)$ is the binary encoding of n. L' cannot be computable, since if it were then, given y, one could test if $y \in L$ by asking if $0^y \in L'$. Note that $L' \cap \{0,1\}^n$ can be computed by the single constant gate $\begin{cases} 1 & \text{if bin}(n) \in L \\ 0 & \text{otherwise} \end{cases}$. □

Lecture 10

Circuits and Uniformity

February 4, 1991
Notes: Juan Alemany

10.1. Intuition Behind the Power of Nonuniform Circuits

In Proposition 9.2 we saw an example demonstrating the unrealistic capabilities of nonuniform circuits. In particular we saw how nonuniform circuits of constant size can decide uncomputable languages. What follows is some intuition as to why these circuits are so powerful.

Recall that the power of nonuniformity is that circuits for a given input size need not resemble circuits of *any* other size. When we think of a family of circuits that computes a simple function (say, the sum of its inputs) we think of circuits with similar structure. For example, circuits for this sum might look like balanced trees that differ only in their size. Nonuniform families of circuits, on the other hand, may have different built–in ("hardwired") information in each circuit.

In Proposition 9.2, nonuniformity allowed us to hide the complexity of deciding whether a string is in the language in the construction of the nth circuit, rather than in its size or depth. In particular, we could hardwire all the membership information for strings of a given length n (in this case, a single bit) into the nth circuit.

10.2. Algorithms + Circuits = Uniform Circuits

In some sense using nonuniform circuits amounts to cheating. In real life we must describe a single procedure that will work for every input length. For example, a graph search algorithm does the "same thing" on every graph regardless of its size. We do not have a different algorithm for graphs of size 10, 17, and so on. An intuitive definition of a "uniform" family of circuits is one in which any member can be generated by a single algorithm running in a reasonable amount of time that depends on the size of the circuit that is generated. We now formalize this intuition.

Definition: A family $C = (c_0, c_1, \ldots, c_n, \ldots)$ of circuits is *(P–) uniform* if and only if there is an algorithm that outputs c_n in time polynomial in $n + |c_n|$, where $|c_n|$ denotes the size of c_n.

Definition: *(P–) uniform PSIZE* is the subset of *PSIZE* restricted to uniform families of circuits. In other words it is the set of languages accepted by some P–uniform family of circuits $C = (c_0, c_1, \ldots, c_n, \ldots)$, where $|c_n| = n^{O(1)}$.

Beware not to confuse the two polynomials in this definition. One polynomial corresponds to the size of the nth circuit. The other polynomial is the time to compute the nth circuit.

10.3. Relating Circuit Size to Algorithm Time

Now that we have defined uniform circuits several questions come to mind. How do these uniform circuits relate to problems known to be in P? Can we simulate algorithms by circuits and circuits by algorithms? We know that nonuniform circuits are unreasonably powerful, so we cannot simulate them with (uniform) algorithms. It seems it should be possible to simulate algorithms by circuits. In the next theorem we show that not only is it possible, but the circuit need not be very large relative to the running time of the algorithm.

Theorem 10.1: $P \subseteq PSIZE$.

We will omit the proof of the theorem. A careful proof requires defining what a polynomial time algorithm is, which we have studiously avoided. A clean way to do the proof carefully would be by simulating deterministic Turing machines by circuits. The best such simulation is given by Pippenger and Fischer [31]. In its place, we will give a sketch of the proof, highlighting the relevant issues.

10.3.1. Circuits, Algorithms, and Obliviousness

In one sense circuits are weaker than algorithms: algorithms can adapt their behavior according to the particular values of their inputs. For instance, an algorithm can do something like "if the first input is 3 then jump to label l", whereas circuits have a prescribed computation once the input size is fixed. We refer to this characteristic as the *obliviousness* of a circuit, the idea being that circuits carry on a fixed computation, oblivious to their input values. (Sometimes oblivious programs are referred to as "nonadaptive" or "straight-line".)

It is possible, however, to simulate any algorithm by an algorithm that is "oblivious", in the sense that the operation the algorithm performs at step t, and the memory locations holding its arguments, are independent of the particular value of the input. Furthermore, this oblivious algorithm will have a computation time polynomial in the computation time of the original algorithm.

Once we have an oblivious algorithm we need only show how to simulate it by a circuit of size approximately equal to its running time. The idea is to compute the result of the instruction at step t by a small subcircuit C_t. The inputs to C_t are themselves the results of the last instructions that modified the arguments of the current instruction. For example, if the step t operation is "add the contents of register 5 to register 7", we will construct a subcircuit C_t that performs an addition of two inputs. Now, what should the inputs to this subcircuit be? By obliviousness, we know the last time t_1 that register 5 was modified, and the last time t_2 that register 7 was modified. Then the two inputs to C_t will be the outputs of subcircuits C_{t_1} and C_{t_2}.

10.3.2. What Constitutes an Algorithm?

Having digressed to the level of registers and instructions, it is reasonable to pause to ask what a reasonable instruction set for polynomial time algorithms might be. If you had to define such a set, you might well include the instructions load, store, add, subtract, multiply, indirect addressing, and conditional branching. You might also charge one time unit per instruction executed.

However, if we charge one time unit for each multiplication regardless of the length of the operands, we quickly run into trouble. Each multiplication of binary numbers can double the length of the operands. Thus in polynomial time we can have numbers that are exponentially long, which cannot be handled by polynomial size circuits. Although this only suggests that the simulation by circuits might need to be improved, in fact there are more inherent problems associated with charging unit time to the multiplication of large numbers [19].

One possible solution is to charge time $\log n$ for each operation, where n is the length of the greatest operand for that operation. A second solution is simply to restrict each register to hold only polynomially many bits.

10.3.3. A Partial Converse

Although Proposition 9.2 shows that one cannot hope for a converse of Theorem 10.1, the proof of Theorem 10.1 does in fact support the converse if the circuits are forced to be uniform:

Theorem 10.2: $P =$ uniform *PSIZE*.

Lecture 11

BPP $\subseteq$ PSIZE; Pseudorandom Generators

February 6, 1991
Notes: Rakesh Kumar Sinha

11.1. Relating Circuits and Probabilistic Complexity Classes

Theorem 10.1 showed that the class P is contained in *PSIZE* — the class of languages accepted by polynomial size circuits. Adleman [1] extended this result by showing that the class *RP*, which is a superset of P, is also contained in *PSIZE*. Since we have natural examples of problems in *RP* not known to be in P (for example, the set of composite integers [38]), Adleman's result was very surprising: it says such problems can be solved deterministically (though nonuniformly). Bennett and Gill [6] further strengthened this result by showing that the class *BPP* is also contained in *PSIZE*.

Both these results exploit the nonuniformity of the circuit model in a strong way. The general idea is to start with a probabilistic algorithm and reduce its error exponentially, after which a simple counting argument guarantees the existence of a nonuniform algorithm that never makes a mistake.

Theorem 11.1 (Bennett and Gill [6]): *BPP* $\subseteq$ *PSIZE*.

Proof: Let $L \in BPP$ be recognized by a probabilistic algorithm in time $T(n) = n^{O(1)}$. By Corollary 2.2, there is a probabilistic algorithm A that recognizes L in time $O(nT(n))$ and with error probability *strictly less* than 2^{-n}. (The significance of 2^{-n} will be clear shortly.)

For any n, we want to compute the expected number of inputs of length n on which A makes a mistake. For each input x, we define a random variable

$$C_x = \begin{cases} 1 & \text{if } A \text{ errs on input } x \\ 0 & \text{otherwise} \end{cases} .$$

Then

$$\begin{aligned} \text{E(number of inputs of length } n \text{ on which } A \text{ errs)} &= \text{E}\left(\sum_{x\in\{0,1\}^n} C_x\right) \\ &= \sum_{x\in\{0,1\}^n} \text{E}(C_x) \end{aligned}$$

$$\begin{aligned}
&= \sum_{x\in\{0,1\}^n} \Pr(C_x = 1) \\
&= \sum_{x\in\{0,1\}^n} \Pr(A \text{ errs on input } x) \\
&< \sum_{x\in\{0,1\}^n} 2^{-n} \\
&= 1.
\end{aligned}$$

This says that the expected number of inputs of length n on which A makes a mistake is *strictly less* than 1. So, there must be at least one output $g \in \{0,1\}^{O(nT(n))}$ of the random number generator such that, if A uses g in place of random bits, then A makes a mistake on strictly less than 1 input of length n, since otherwise the average over all such g couldn't possibly be strictly less than 1. In other words, if A uses g in place of random bits then A is correct on all inputs of length n. To construct the nth circuit C_n that simulates A on inputs of length n, build g into C_n, and then simulate A as in Theorem 10.1, since A behaves deterministically once g is fixed. $\square$

Notice that this family of circuits is nonuniform, as we do not know of any way to compute g from n in polynomial time. It is possible to compute g in exponential time, though, so the circuit family is not totally nonuniform.

This brings to an end the first half of the course. So far, we have always assumed that we have an unlimited source of perfect random numbers at our disposal. This assumption may be unreasonable in practice. In the remainder of this course, we will see how the performance of our algorithms is affected if we have less than perfect random numbers.

11.2. Pseudorandom Generators

Anyone who considers arithmetical methods for producing random digits is, of course, in a state of sin.

– *von Neumann*

A *pseudorandom generator* is a deterministic algorithm that take as input a "small" number k of truly random bits (called the *seed*) and outputs a "large" number m of bits (called *pseudorandom bits*). The string of m pseudorandom bits is, of course, a random variable (in the sense that it takes on various values with certain probabilities), but it is by no means uniformly distributed over $\{0,1\}^m$, since it assumes at most 2^k distinct values with nonzero probability.

The hope is to generate bits efficiently which may be used in place of random numbers without significantly degrading the performance of the algorithm. For example, g in our last proof can be thought of as the output of a degenerate pseudorandom generator that takes no seed at all and does not affect the performance of the algorithm. The catch, of course, is that we do not know of any efficient way to generate g.

Bach [5] was the first person to consider the effect of having pseudorandom generators on the performance of probabilistic algorithms. He confined himself to algorithms for number-theoretic problems such as compositeness testing. We will discuss subsequent results of Karloff and Raghavan [23], who studied the effect of pseudorandom generators on more familiar algorithms such as Quicksort. We need the following definition before we can describe their first result.

Definition: A *linear congruential generator* has three parameters m, a, and c. It takes a random seed $X_0 \in \mathcal{Z}_m$ and outputs the pseudorandom sequence $X_0, X_1, \ldots, X_i, \ldots$, where $X_i = (aX_{i-1} + c) \bmod m$ for $i \geq 1$.

Linear congruential generators have been quite effective in practice, and are advocated by Knuth [27]. If one were implementing Quicksort, it would be most natural to use a linear congruential generator as the source of randomness, with $m = \Theta(n)$ and a and c chosen so as to make the period of the linear congruential generator as close to m as possible. Therefore, it came as a surprise when Karloff and Raghavan proved that a natural way of using such a linear congruential generator to implement Quicksort results in quadratic expected running time (where the expectation, as usual, is taken over the random seed, not over the input to be sorted).

For concreteness and ease of analysis, they make the following assumptions about the implementation of Quicksort:

(Q1) Whenever we have two recursive subproblems, we solve the smaller one first.

(Q2) The partition subprocedure is stable (that is, the order of inputs within a subproblem resulting from a partition is the same as the order within the original problem).

(Q3) Every subproblem of size at least one calls the random partition subprocedure. That is, a subproblem of size L consumes exactly L pseudorandom numbers.

Q1 is a popular heuristic to minimize the height of the stack. Q2 and Q3 simplify the analysis. Although the usual in-place partitioning procedure is not stable (Q2), stability in sorting is a desirable feature that other sorting algorithms strive to achieve [26]. Although it is not particularly realistic to use a pseudorandom number on subproblems of size 1 (Q3), their result even with this assumption must make one suspicious of the combination of Quicksort and linear congruential generators. More generally, we will see that their result does not rule out an $O(n \log n)$ expected time implementation of Quicksort using linear congruential generators; it says rather that one must be very careful in order to achieve such an implementation.

Lecture 12

Quicksort with Linear Congruential Generators

February 8, 1991
Notes: George Forman

Last lecture we listed the assumptions about the implementation of Quicksort addressed in the upcoming theorem of Karloff and Raghavan. Next we list the assumptions about the linear congruential generator used. (Recall that a, c, and m are the parameters of the generator, and n is the number of items to sort.)

(L1) $m > n$.

(L2) $\gcd(a, m) = 1$.

(L3) $c = 0$.

(L4) $\min\{t > 0 \mid a^t \equiv 1 \pmod{m}\} > n/4$.

L1 – L4 are all intended to ensure that the "period" of the generator (that is, the number of pseudorandom numbers generated before the sequence begins to repeat itself) is $\Omega(n)$. Since Quicksort consumes n pseudorandom numbers (Q3), this seems desirable. Certainly the period cannot exceed m, so that L1 is necessary. A period of m is achieved only if $\gcd(a, m) = \gcd(c, m) = 1$ [27, Theorem A, page 16]; if instead $c = 0$ is chosen, $\gcd(a, m) = 1$ is still necessary to maximize the period [27, Theorem B, page 19]. Although assumption L3 is chosen to make the analysis simpler, Karloff and Raghavan claim that the analysis when $\gcd(c, m) = 1$ is similar.

Assumption L4 says that the period of the generator is reasonably large. In L4, the minimum t can be interpreted as the period when the initial seed X_0 is 1; the period is the same for any other value of X_0 such that $\gcd(X_0, m) = 1$, since the sequence of pseudorandom numbers is then

$$X_0, aX_0 \bmod m, a^2X_0 \bmod m, a^3X_0 \bmod m, \ldots, a^{t-1}X_0 \bmod m, a^tX_0 \bmod m = X_0,$$

and $a^iX_0 \equiv a^jX_0 \pmod{m}$ for $0 \le i < j < t$ would imply that $a^i(a^{j-i} - 1)X_0 \equiv 0 \pmod{m}$, which in turn implies that $a^{j-i} \equiv 1 \pmod{m}$ (since $\gcd(X_0, m) = \gcd(a, m) = 1$), contradicting the minimality of t.

A question was raised about whether assumptions L2 – L4 can ever be satisfied simultaneously. One way to show that they can is to select m to be prime, in which case $\phi(m-1)$ values of a achieve period $m-1$ for all $m-1$ seeds $X_0 \neq 0$, where ϕ is Euler's totient function [29, page 305]. This is a large enough number of values of a that choosing a at random has a good chance of working.

Next we need to consider how the pseudorandom numbers from $\mathcal{Z}_m$ are to be used by Quicksort in selecting an arbitrary pivot for a subproblem that has only L values.

(H1) Let $\text{hash}(y, L) = \lfloor L \cdot \frac{y}{m} \rfloor$, where $y \in \mathcal{Z}_m$. Partition the L elements by the pivot element in cell $\text{hash}(y, L)$, where y is the next pseudorandom number.

This is the method advocated by Knuth [27] for converting a pseudorandom number in $\mathcal{Z}_m$ to one in $\mathcal{Z}_L$.

Theorem 12.1 (Karloff and Raghavan [23]): For implementations of Quicksort using a linear congruential generator satisfying Q1 – Q3, L1 – L4, and H1, there is an input permutation requiring $\Omega\left(\frac{n^4}{m^2} + n \log n\right)$ expected time (averaged over all seeds X_0).

Consequence: If $m = O(n)$, the expected time is $\Omega(n^2)$, which is asymptotically as slow as Quicksort can run.

Open Problem 5: Prove that, for some sufficiently large m that is still polynomial in n (say $m = n^2$), Quicksort with a linear congruential generator runs in $O(n \log n)$ expected time.

Open Problem 6: Prove that Quicksort with the usual unstable partition procedure (rather than Q2) still satisfies Theorem 12.1.

Note that if we were to assume a random distribution on the ordering of the input values, we would not even need a random number generator to achieve $O(n \log n)$ expected running time: simply selecting the first element as the pivot value would be a sufficiently random pivot. This explains why Theorem 12.1 addresses the behavior of Quicksort when given the worst case permutation. In fact, though, the proof will demonstrate that there are many input permutations that lead to the same bad expected running time.

The proof Theorem 12.1 will be presented next lecture. We conclude the remainder of this lecture with an overview of the proof.

It is easy to find a permutation such that there is *one* seed X_0 (say $X_0 = 1$) that forces Quicksort to use $\Omega(n^2)$ time: simply order the inputs so that the predetermined sequence of pivots encountered by the generator with this seed are in sorted order. However, this would only yield expected time $\Omega\left(\frac{n^2}{m} + n \log n\right)$, since we have to average over all m seeds. We will show instead that there is a permutation for which $\frac{n^2}{16m}$ seeds each reach the same subproblem of size $\frac{n}{4}$, each doing so exactly at the point when the linear congruential generator is about to output the value 1. Furthermore, this subproblem will then take time $\Omega(n^2)$, by the construction outlined at the beginning of this paragraph. This yields the following lower bound on expected time:

$$\frac{\frac{n^2}{16m} \cdot \Omega(n^2) + \left(m - \frac{n^2}{16m}\right) \cdot \Omega(n \log n)}{m} = \Omega\left(\frac{n^4}{m^2} + n \log n\right),$$

using L1 and the fact that $\Omega(n \log n)$ is the best running time that Quicksort can achieve for any sequence of pivots.

Lecture 13

Quicksort with Linear Congruential Generators, cont.

February 11, 1991
Notes: Denise Draper

Proof of Theorem 12.1:

Case 1: $m > n^2/32$. Then $n^4/m^2 = O(1)$, and the best time that Quicksort can achieve for any set of pivots is $\Omega(n \log n) = \Omega\left(n^4/m^2 + n \log n\right)$.

Case 2: $m \leq n^2/32$. Let $A[0 \mathinner{..} n-1]$ be the input vector to Quicksort. We will construct A to be a permutation of the integers $\{0, 1, \ldots, n-1\}$ such that its expected time is $\Omega(n^2)$ for $n^2/(16m)$ distinct seeds.

Let x satisfy $a^{\lfloor n/4 \rfloor} x \equiv 1 \pmod{m}$. Such an x exists, because L2 specifies that $\gcd(a, m) = 1$, which guarantees that a (and hence any power of a) is invertible modulo m. Furthermore, note that x is also invertible modulo m (its inverse being $a^{\lfloor n/4 \rfloor}$), so that $\gcd(x, m) = 1$.

Let $y_i = a^i x \bmod m$ for all $0 \leq i \leq \lfloor n/4 \rfloor$. Note that the y_i's are distinct: since $\gcd(x, m) = 1$, the analysis on page 34 (which depends on L4) holds.

Let $k = \lfloor n^2/(8m) \rfloor - 1$. Let $K = \{\text{hash}(y_0, n), \text{hash}(y_1, n), \ldots, \text{hash}(y_{\lfloor n/4 \rfloor}, n)\} - \{0\}$. $|K|$ may be less than $\lfloor n/4 \rfloor$, since distinct values of y_i may hash to the same value. We will show, however, that $|K| \geq k$. By the definition of the hash function (H1), at most $\lceil m/n \rceil$ distinct values in $\mathcal{Z}_m$ hash into a single value in $\mathcal{Z}_n$. Therefore,

$$\begin{aligned}
|K| &\geq \frac{\lfloor n/4 \rfloor + 1}{\lceil m/n \rceil} - 1 \\
&> \frac{n/4}{(m+n)/n} - 1 \\
&= \frac{n^2}{4(m+n)} - 1 \\
&\geq \frac{n^2}{8m} - 1 \qquad \text{(L1)} \\
&\geq k.
\end{aligned}$$

Thus, there are k values $y_{i_1}, y_{i_2}, \ldots, y_{i_k} \in \{y_0, y_1, \ldots, y_{\lfloor n/4 \rfloor - 1}\}$ such that the k hashed values $\text{hash}(y_{i_j}, n)$ are distinct and nonzero.

Now construct the input permutation A as follows:

1. Let $A[\text{hash}(y_{i_j}, n)] = n - (\lfloor n/4 \rfloor - i_j)$, for all $1 \le j \le k$.
2. Let σ be a permutation of $\{0, 1, \ldots, \lfloor n/4 \rfloor - 1\}$ requiring $\Omega(n^2)$ time when the pseudorandom sequence is $a, a^2 \bmod m, a^3 \bmod m, \ldots, a^{\lfloor n/4 \rfloor} \bmod m$. (The existence of such a permutation was shown last lecture.) Let $A[0] = \lfloor n/4 \rfloor$. Assign $\sigma(0), \sigma(1), \ldots, \sigma(\lfloor n/4 \rfloor - 1)$ in this order to the $\lfloor n/4 \rfloor$ leftmost available cells of A.

Note that the values placed are distinct, since those in item (1) are at least $3n/4$ and those in item (2) are at most $n/4$. Note also that the locations in A in which they are placed are distinct. At this point the number of locations filled is at most $k + \lfloor n/4 \rfloor + 1 \le n/8 + n/4$ (L1); complete A to a permutation of $\{0, 1, \ldots, n-1\}$ arbitrarily. (Note that the indeterminacy here implies that Quicksort will run slowly not just on one input permutation, but in fact on at least $\lfloor 5n/8 \rfloor!$ permutations.)

Now consider what happens when the seed X_0 is y_{i_j}, for any $1 \le j \le k$. Quicksort first partitions A on $A[\text{hash}(y_{i_j}, n)] = n - (\lfloor n/4 \rfloor - i_j)$, consuming one pseudorandom number. Note again that this value is at least $3n/4$, which means that the smaller subproblem will be to sort those numbers greater than $n - (\lfloor n/4 \rfloor - i_j)$. Q1 requires that we sort this smaller subproblem first, and Q3 states that doing so will consume exactly as many pseudorandom numbers as it has elements, namely $(n-1) - (n - (\lfloor n/4 \rfloor - i_j)) = -1 + \lfloor n/4 \rfloor - i_j$. Thus, after completing this subproblem the pseudorandom generator is in the state in which its next output will be

$$\begin{aligned} X_{\lfloor n/4 \rfloor - i_j} &= y_{i_j} a^{\lfloor n/4 \rfloor - i_j} \bmod m \\ &= a^{i_j} x a^{\lfloor n/4 \rfloor - i_j} \bmod m \\ &= a^{\lfloor n/4 \rfloor} x \bmod m \\ &= 1. \end{aligned}$$

Let $L = n - (\lfloor n/4 \rfloor - i_j)$ be the size of the remaining subproblem. Quicksort partitions next on

$$\begin{aligned} A[\text{hash}(X_{\lfloor n/4 \rfloor - i_j}, L)] &= A[\text{hash}(1, L)] \\ &= A[\lfloor L/m \rfloor] && \text{(H1)} \\ &= A[0]. && \text{(L1)} \end{aligned}$$

By Q2, the contents of $A[0]$ will not have changed since Quicksort started, namely $A[0] = \lfloor n/4 \rfloor$. After partitioning on $\lfloor n/4 \rfloor$ we will have $A[\lfloor n/4 \rfloor] = \lfloor n/4 \rfloor$, and $A[i] = \sigma(i)$, for all $0 \le i < \lfloor n/4 \rfloor$ (again by Q2). This smaller subproblem is sorted next (Q1), and requires $\Omega(n^2)$ time, by the construction of σ, since the next pseudorandom numbers generated will be $a, a^2 \bmod m, a^3 \bmod m, \ldots, a^{\lfloor n/4 \rfloor} \bmod m$.

In summary, for each of the $k = \lfloor n^2/(8m) \rfloor - 1$ seeds y_{i_j}, Quicksort requires time $\Omega(n^2)$. Since, by the assumption of case 2, $m \le n^2/32$, it follows that $k \ge n^2/(16m)$. Averaging over all m seeds yields a lower bound on expected time of $\Omega(n^4/m^2 + n \log n)$, as shown at the end of the previous lecture. $\square$

Open Problem 7: Improve the lower bound on $|K|$ from $\Omega(n^2/m)$ to $\Omega(n)$, using the idea that $y_0, y_1, \ldots, y_{\lfloor n/4 \rfloor}$ will hash quite uniformly into the n cells, since they form a geometric series modulo m. This would improve Theorem 12.1 to $\Omega(n^3/m + n \log n)$.

Lecture 14

Quicksort with the 5-Way Generator

February 13, 1991
Notes: Ashok Immaneni

The standard probabilistic Quicksort algorithm uses $O(n \log n)$ bits of randomness, and runs in $O(n \log n)$ expected time. Theorem 12.1 shows that Quicksort with common linear congruential generators (that use only $O(\log n)$ bits of randomness) requires $\Omega(n^4/m^2)$ expected time. In contrast, Karloff and Raghavan [23] also showed that Quicksort can be implemented to run in $O(n \log n)$ expected time, using only $O(\log n)$ bits of randomness, by introducing a novel pseudorandom generator called the "5-way generator". In this lecture we will see what a 5-way generator is, and the assumptions Karloff and Raghavan make about the implementation of Quicksort.

Definition: A *5-way generator* has one parameter p, a prime. It takes a random seed $(a, b, c, d, e) \in \mathcal{Z}_p^5$ and outputs the pseudorandom sequence $X_0, X_1, \ldots, X_i, \ldots$, where $X_i = (a + bi + ci^2 + di^3 + ei^4) \bmod p$, for $i \geq 0$.

Definition: A set $\{x_1, x_2, \ldots, x_n\}$ of random variables is *mutually independent* if and only if

$$\Pr\left(\bigwedge_{i=1}^{n} x_i = a_{i,j}\right) = \prod_{i=1}^{n} \Pr\left(x_i = a_{i,j}\right),$$

for all $a_{i,j}$.

Definition: A set S of random variables is *t-way independent* if and only if every t elements of S are mutually independent.

As we will prove later, the set of pseudorandom numbers generated by a 5-way generator is *5-way independent.* This is the key property of this generator that enables the proof that it causes Quicksort to behave well.

We now list the assumptions about Quicksort:

- Without loss of generality, assume that we are sorting a permutation of $\{0, 1, ..., n-1\}$, and that n is prime. (If n is not prime, pick the least prime greater than n, and pad the remaining elements of the array with large numbers. For any n, there is always a prime between n and $2n$.)
- The parameters and seed for the 5-way generator are selected as follows: $p = n$, and a, b, c, d, and e are chosen randomly, uniformly, and independently from $\mathcal{Z}_p$. Note that this is only $O(\log n)$ bits of randomness.

- $A[0 \mathinner{..} n-1]$ is the input permutation, which is read-only, for reasons that will become clear later. $B[0 \mathinner{..} n-1]$ is an auxiliary array in which the sorting is actually done.
- The version of Quicksort is nonrecursive (for simplicity of the proof). Because of this, we need to define what it means to partition on a pivot element.

Definition: To *partition on z* means the following: If Quicksort has partitioned previously on z, then do nothing. Otherwise, suppose $z = B[t]$. Find the greatest $t_1 < t$, and the least $t_2 > t$ such that Quicksort has partitioned previously on $B[t_1]$ and $B[t_2]$. (If no such t_1 exists, let $t_1 = -1$. If no such t_2 exists, let $t_2 = n$.) Permute $B[t_1 + 1 \mathinner{..} t_2 - 1]$ in any way that satisfies the following condition: if z ends at $B[s] = z$, then $B[j] < z$ for all $t_1 < j < s$, and $B[j] > z$ for all $s < j < t_2$. The *cost* of this partition is $t_2 - t_1$.

For this result, it will not matter whether the partition algorithm is stable. Note that the time used by any efficient partition algorithm is proportional to this definition of cost.

Lecture 15

Quicksort with the 5-Way Generator, continued

February 15, 1991
Notes: João Setubal

Theorem 15.1 (Karloff and Raghavan [23]): There is an implementation of Quicksort that, on every permutation of n distinct keys, uses $O(\log n)$ random bits and runs in $O(n \log n)$ expected time.

CONSTRUCTION:

```
algorithm 5-Way Quicksort ($A[0 .. n-1]$)
comment: assumes the elements of $A$ are distinct and $n$ is prime;
begin
    choose $a, b, c, d, e \in \mathcal{Z}_n$, randomly, uniformly, and independently;
    $B \leftarrow A$;
    for $i$ from $0$ to $n-1$ do
        begin
            $X_i \leftarrow (a + bi + ci^2 + di^3 + ei^4) \bmod n$;
            P1: partition $B$ on $A[X_i]$
        end ;
    P2: partition $B$ on each number not previously used as a pivot, in any
      order;
    return $B$
end 5-Way Quicksort.
```

Note that some of the X_i's produced by the generator may previously have been produced and used as pivots. In that case, according to the definition of partition, nothing happens in statement P1. By the pigeonhole principle, in that case some of the keys in A will not be used as pivots in P1, which explains the need for the "cleanup" phase (statement P2).

CORRECTNESS:

Because of P2, when *5-Way Quicksort* terminates it has partitioned on every input key. We still have to show that all these partitions produce a sorted output. By induction on the number i of times the algorithm has partitioned in P1 and P2, it will be shown that every element that has been used as a pivot is in its correct sorted position in B, with all lesser elements at lesser indexed positions in B and all greater elements at greater indexed positions in B. (Note that, by the definition of partition, this condition never changes as a result of subsequent partitions.)

Basis ($i = 0$): Since there have been no partitions, the conditions are vacuously satisfied.

Induction ($i \geq 0$): Suppose that the pivot chosen in the $(i+1)$st partition is z. If the algorithm has partitioned on z before then, by the definition of partition, z is not moved, and by the induction hypothesis the conditions on its position are already satisfied. Otherwise assume that z is in the subarray $B[t_1 + 1 \,..\, t_2 - 1]$, where t_1 and t_2 are the closest indices previously used as pivots, as specified in the definition of partition. Suppose that z ends at index s of B after partitioning on z. Consider first the elements that are in positions with indices less than s in B. By the definition of partition, for all $t_1 < j < s$, $B[j] < z$. By the induction hypothesis, for all $0 \leq j \leq t_1$, $B[j] \leq B[t_1] < z$. Hence, for all $0 \leq j < s$, $B[j] < z$, which is what we wanted to prove for this half of the array. The other half is analogous.

Analysis:

We will need a few lectures to cover this part of the proof. We start with an outline.

1. The expected cost of the ith execution of statement P1 will be shown to be $O(n/i)$, for $i \geq 1$, so the total expected cost of all executions of P1 is

$$\begin{aligned}(n+1) + \sum_{i=1}^{n-1} O\left(\frac{n}{i}\right) &= O(n + n\sum_{i=1}^{n-1}\frac{1}{i}) \\ &= O(n \log n).\end{aligned}$$

   The last line follows because the Harmonic series up to n is bounded by $\ln n + O(1)$ (see Knuth [25, page 74]).

2. The total expected cost of all executions of P2 will be shown to be $O(n)$.

The remainder of the algorithm beyond P1 and P2 also runs in time $O(n \log n)$, so that this is the total running time. (This assumes that each of the $O(n)$ arithmetic operations on $\log n$ bit numbers required to generate the n pseudorandom numbers can be accomplished in $O(\log n)$ time. Whether this is true or not depends on the model of computation. However, note that, as is common in algorithm analysis, there is an implicit assumption that each of the $O(n \log n)$ expected comparisons in the partitioning steps can be done in $O(1)$ time, which is also unrealistic if bit operations are counted. The technically accurate statement is that the expected number of comparisons and arithmetic operations is $O(n \log n)$.)

Next we present a lemma that strengthens both Markov's and Chebyshev's inequalities.

Lemma 15.2: Let k be a positive even integer. Let U be a discrete random variable such that $\mu = E(U)$ and $E((U-\mu)^k)$ both exist. Let $\tau_k > 0$ be the positive real kth root of $E((U-\mu)^k)$. Then, for any $t > 0$,

$$\Pr(|U - \mu| \geq t\tau_k) \leq \frac{1}{t^k}\ .$$

Remark: $E((U-\mu)^k)$ is called the kth *central moment* of U. For the special case $k=2$, this moment is also known as the *variance* of U, and τ_2 is thus its standard deviation. Chebyshev's inequality is the special case $k=2$ of Lemma 15.2. Note also the similarity to Markov's inequality, Lemma 3.1.

Proof:

$$\begin{aligned} \Pr(|U-\mu| \geq t\tau_k) &= \Pr((U-\mu)^k \geq t^k \tau_k^k) && (k \text{ is even}) \\ &\leq \frac{1}{t^k}. \end{aligned}$$

The last line follows from Markov's inequality (Lemma 3.1), since $\tau_k^k = \mathrm{E}((U-\mu)^k)$, again using the fact that k is even so that Markov's inequality is applicable. $\square$

Lecture 16

Quicksort with the 5-Way Generator, continued

February 20, 1991
Notes: Simon Kahan

16.1. How to Choose Among Markov's, Chebyshev's, and Chernoff's Inequalities

Lemma 15.2 provides an upper bound on the probability that a random variable is far from its mean. This bound is often tighter than those provided by Markov's and Chebyshev's inequalities: the bound decreases in proportion to the kth power of the distance from the mean. A drawback is that the lemma requires knowledge of the kth central moment of the variable's distribution, which may not always be readily available. Furthermore, the constant of proportionality in this bound is the kth central moment itself, so that if this moment is large the bound may be weaker than that given by the lower order inequalities.

In contrast, Markov's inequality (Lemma 3.1) depends only on knowledge of the mean, but yields a bound that decreases only in proportion to the distance from the mean; Chebyshev's inequality (the case $k = 2$ of Lemma 15.2) depends on both the mean and variance, and yields a bound that decreases in proportion to the square of the distance from the mean. Because the constants of proportionality in all of these bounds depend upon the particular distribution, we cannot generally predict which inequality, or which k, will produce the most useful bound.

Finally there is Chernoff's inequality (Theorem 2.1), which usually gives the tightest bound. It does so because it applies only to the binomial distribution, so that *all* moments are known. (For a more generally applicable form of Chernoff's inequality, see Raghavan [33, Theorems 1 and 2].)

16.2. t-Way Independence of the t-Way Generator

In Lecture 14 we defined both mutual and t-way independence. An illustration of the difference between these concepts is given by the following example:

Example 16.1: Let X_1 and X_2 be random bits chosen uniformly and independently from the set $\{0, 1\}$. Let $X_3 = X_1 \oplus X_2$. The set $\{X_1, X_2, X_3\}$ of random variables is 2-way independent:

for all $i \neq j$ and all x and y, $\Pr(X_i = x \mid X_j = y) = \Pr(X_i = x) = 1/2$. Yet the same set is not mutually independent: for example,

$$\Pr(X_1 = 0 \wedge X_2 = 0 \wedge X_3 = 0) = \frac{1}{4} \neq \frac{1}{8} = \prod_{i=1}^{3} \Pr(X_i = 0).$$

Definition: The *Vandermonde* matrix $V = V(\alpha_0, \alpha_1, \ldots, \alpha_{t-1})$ is the $t \times t$ matrix whose entries are $V_{ij} = \alpha_{i-1}^{j-1}$, for $1 \leq i, j \leq t$ (where 0^0 is defined to be 1).

Lemma 16.2: For any integral domain D, if $\alpha_0, \alpha_1, \ldots, \alpha_{t-1} \in D$ are distinct, then $V = V(\alpha_0, \alpha_1, \ldots, \alpha_{t-1})$ is invertible.

Proof: Let $\vec{a} = (a_0, a_1, \ldots, a_{t-1})^T \in D^t$ be any column vector satisfying $V\vec{a} = 0$. Let $a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{t-1}x^{t-1}$. Let V_i denote the ith row of V. For each $i \in \{0, 1, \ldots, t-1\}$,

$$a(\alpha_i) = a_0 + a_1\alpha_i + \cdots + a_{t-1}\alpha_i^{t-1} = V_i\vec{a} = 0.$$

Since $a(x)$ has t distinct zeros α_i, yet $a(x)$ has degree at most $t-1$, $a(x)$ must be the zero polynomial [29, IV.3.3, Theorem 2]. Summarizing this argument, $V\vec{a} = 0$ implies $\vec{a} = 0$. Hence V is invertible. $\square$

Lemma 16.3: Let F be finite field with n elements. Let t be any integer with $2 \leq t \leq n$. Choose $a_0, a_1, \ldots, a_{t-1} \in F$ randomly, uniformly, and independently. For $i \in F$, let

$$X_i = \sum_{r=0}^{t-1} a_r i^r.$$

Then, for all $i \in F$,

1. X_i is uniformly distributed in F.
2. $\{X_i \mid i \in F\}$ is t-way independent.
3. Given that $X_i = z$, X_j remains uniformly distributed in F, for all $j \neq i$.
4. Given that $X_i = z$, $\{X_j \mid j \in F \text{ and } j \neq i\}$ is $(t-1)$-way independent.

Proof: Let $i_0, i_1, \ldots, i_{t-1}$ be any t distinct elements of F. Let V denote the Vandermonde matrix $V(i_0, i_1, \ldots, i_{t-1})$, $\vec{a} = (a_0, a_1, \ldots, a_{t-1})^T$, and $\vec{y} = (y_0, y_1, \ldots, y_{t-1})^T$, where each y_j is an arbitrary element of F.

1. To show that X_i is uniformly distributed in F, it suffices to show that, for all $y \in F$, $\Pr(X_i = y) = 1/n$.

$$\begin{aligned}\Pr(X_i = y) &= \Pr\left(\sum_{r=0}^{t-1} a_r i^r = y\right) \\ &= \Pr\left(a_0 = y - \sum_{r=1}^{t-1} a_r i^r\right) \\ &= \frac{1}{n},\end{aligned}$$

since a_0 is uniformly distributed in F.

2.

$$\begin{aligned}
\Pr\left(\bigwedge_{j=0}^{t-1}(X_{i_j} = y_j)\right) &= \Pr\left(\bigwedge_{j=0}^{t-1}\left(\sum_{r=0}^{t-1} a_r i_j^r = y_j\right)\right) \\
&= \Pr(V\vec{a} = \vec{y}) \\
&= \Pr(\vec{a} = V^{-1}\vec{y}) && \text{(Lemma 16.2)} \\
&= \frac{1}{n^t} && (\vec{a} \text{ is uniform in } F^t) \\
&= \prod_{j=0}^{t-1} \Pr(X_{i_j} = y_j). && \text{(From part (1))}
\end{aligned}$$

3. For any $y \in F$,

$$\begin{aligned}
\Pr(X_j = y \mid X_i = z) &= \Pr(X_j = y) && \text{(From part (2))} \\
&= \frac{1}{n} && \text{(From part (1))}
\end{aligned}$$

4. Suppose $i, i_0, i_1, \ldots, i_{t-2}$ are t distinct elements of F.

$$\begin{aligned}
\Pr\left(\bigwedge_{j=0}^{t-2} X_{i_j} = y_j \mid X_i = z\right) &= \frac{\Pr\left(\left(\bigwedge_{j=0}^{t-2} X_{i_j} = y_j\right) \wedge (X_i = z)\right)}{\Pr(X_i = z)} \\
&= \frac{\left(\prod_{j=0}^{t-2} \Pr(X_{i_j} = y_j)\right) \Pr(X_i = z)}{\Pr(X_i = z)} && \text{(From part (2))} \\
&= \prod_{j=0}^{t-2} \Pr(X_{i_j} = y_j) \\
&= \prod_{j=0}^{t-2} \Pr(X_{i_j} = y_j \mid X_i = z) && \text{(From part (2))}
\end{aligned}$$

□

Lecture 17

Quicksort with the 5-Way Generator, continued

February 22, 1991
Notes: Denise Draper

We are now in a position to prove the main technical lemma:

Lemma 17.1: Let $X_0, X_1, \ldots, X_{n-1}$ be a pseudorandom sequence output by the 5-way generator. Let S be an arbitrary nonempty subset of $\mathcal{Z}_n$ and let $r = |S|$. Fix any $z \in \mathcal{Z}_n$ and $i \in \{1, 2, \ldots, n-1\}$. Then

$$\Pr\left(\bigwedge_{j=0}^{i-1} (A[X_j] \notin S) \mid A[X_i] = z\right) \leq \frac{4n^2}{i^2 r^2}. \tag{17.1}$$

(Note that if the X_j were truly random, uniform, and independent, the probability above would be $((n-r)/n)^i$, which is generally much smaller than the bound in this lemma. This is the price we pay for pseudorandomness.)

Proof: When $ir \leq n$, the lemma is vacuously true, since the right hand side of inequality (17.1) is then at least 4. Therefore, assume $ir > n$.

For $j \in \{0, 1, \ldots, i-1\}$, let

$$Y_j = \begin{cases} 1 & \text{if } A[X_j] \in S \\ 0 & \text{otherwise} \end{cases}.$$

Let $p = \mathrm{E}(Y_j \mid A[X_i] = z) = \Pr(A[X_j] \in S \mid A[X_i] = z) = r/n$ since, by Lemma 16.3, part (3), X_j is uniformly distributed and independent of X_i.

Let $U = \sum_{j=0}^{i-1} Y_j$. (Thus, the left hand side of inequality (17.1) can be expressed as $\Pr(U = 0 \mid A[X_i] = z)$.) Let

$$\begin{aligned} \mu &= \mathrm{E}(U \mid A[X_i] = z) \\ &= \mathrm{E}\left(\sum_{j=0}^{i-1} Y_j \mid A[X_i] = z\right) \\ &= \sum_{j=0}^{i-1} \mathrm{E}(Y_j \mid A[X_i] = z) \\ &= ip. \end{aligned}$$

Let τ be the positive real fourth root of the fourth central moment $\mathrm{E}\left((U-\mu)^4 \mid A[X_i]=z\right)$. Then we have the following:

$$\begin{aligned}
&\Pr\left(U=0 \mid A[X_i]=z\right) \\
&\quad\le \Pr\left(|U-\mu| \ge \mu \ \mid A[X_i]=z\right) && \text{(Note 1)}\\
&\quad= \Pr\left(|U-\mu| \ge \frac{\mu}{\tau}\cdot\tau \ \mid A[X_i]=z\right) \\
&\quad\le \frac{\tau^4}{\mu^4} && \text{(Lemma 15.2)}\\
&\quad= \frac{1}{\mu^4}\mathrm{E}\left((U-\mu)^4 \mid A[X_i]=z\right) \\
&\quad= \frac{1}{\mu^4}\mathrm{E}\left(\left(\sum_{j=0}^{i-1}(Y_j-p)\right)^4 \mid A[X_i]=z\right) \\
&\quad= \frac{1}{\mu^4}\left[\sum_{j=0}^{i-1}\mathrm{E}\left((Y_j-p)^4 \mid A[X_i]=z\right)\right. \\
&\qquad\qquad \left. + \binom{4}{2}\sum_{0\le j<k<i}\mathrm{E}\left((Y_j-p)^2(Y_k-p)^2 \mid A[X_i]=z\right)\right] && \text{(Note 2)}\\
&\quad= \frac{1}{i^4p^4}\left[i\left(p(1-p)^4+(1-p)(-p)^4\right)\right. \\
&\qquad\qquad \left. + 6\binom{i}{2}\left(p(1-p)^2+(1-p)(-p)^2\right)^2\right] && \text{(Note 3)}\\
&\quad= \frac{ip(1-p)\left((1-p)^3+p^3\right)+6\binom{i}{2}p^2(1-p)^2}{i^4p^4} \\
&\quad\le \frac{ip+3i^2p^2}{i^4p^4} && \text{(Note 4)}\\
&\quad< \frac{4i^2p^2}{i^4p^4} && \text{(Note 5)}\\
&\quad= \frac{4}{i^2p^2} \\
&\quad= \frac{4n^2}{i^2r^2}.
\end{aligned}$$

Note 1. Since U is nonnegative, $|U-\mu| \ge \mu$ is satisfied under two conditions: when $U=0$ and when $U \ge 2\mu$. Thus this probability is no less than the probability of $U=0$ alone.

NOTE 2. The expansion of $\left(\sum_j (Y_j - p)\right)^4$ will yield terms of the form $((Y_a - p)(Y_b - p)(Y_c - p)(Y_d - p))$. If $a \neq b$, then Y_a is independent of Y_b, since the Y_j are functions of the X_j, which are 4-way independent, by Lemma 16.3, part (4). Because of this independence, if there is one subscript, say a, in the term above that is different from the other three, then the expectation of the product can be written as a product of expectations:

$$\begin{aligned}
&\mathrm{E}\left((Y_a - p)(Y_b - p)(Y_c - p)(Y_d - p) \mid A[X_i] = z\right) \\
&\quad = \mathrm{E}\left((Y_a - p) \mid A[X_i] = z\right) \cdot \mathrm{E}\left((Y_b - p)(Y_c - p)(Y_d - p) \mid A[X_i] = z\right) \\
&\quad = 0 \cdot \mathrm{E}\left((Y_b - p)(Y_c - p)(Y_d - p) \mid A[X_i] = z\right) \\
&\quad = 0.
\end{aligned}$$

Thus the only terms that are nonzero are those in which each subscript is present at least twice — exactly the terms appearing in this formula.

NOTE 3. Since $j \neq k$ in the second term, we can again replace the expectation of the product with the product of the expectations: $\mathrm{E}((Y_j - p)^2 (Y_k - p)^2) = \mathrm{E}((Y_j - p)^2) \cdot \mathrm{E}((Y_k - p)^2) = \mathrm{E}((Y_j - p)^2)^2$. Then the expectations are replaced with their values and summed over all terms.

NOTE 4. Several quantities are replaced by greater or equal values: $(1-p)^3 + p^3$ and $1 - p$ are replaced by 1, and $\binom{i}{2}$ is replaced by $\frac{i^2}{2}$.

NOTE 5. By supposition $ir > n$, which implies that $ip > 1$, which in turn implies that $i^2 p^2 > ip$.
□

Lecture 18

Quicksort with the 5-Way Generator, continued

February 22, 1991
Notes: Tracy Kimbrel

18.1. Cost of the ith Partition in P1

Definition: Let Q_i be the cost of the ith iteration of step P1 of *5-Way Quicksort*. For any $i \in \{1, 2, \ldots, n-1\}$ and $y \in \mathcal{Z}_n$, let

$$\begin{aligned} R_{i,y} &= \min(\{A[X_j] - y \mid 0 \le j < i \;\&\; A[X_j] \ge y\} \cup \{n - y\}), \text{ and} \\ L_{i,y} &= \min(\{y - A[X_j] \mid 0 \le j < i \;\&\; A[X_j] \le y\} \cup \{y + 1\}). \end{aligned}$$

(Just prior to the ith pivot, $R_{i,y}$ is the distance from y to the closest previous pivot to the right of y or, if none, to the right end of A; $L_{i,y}$ is the analogous distance to the left.)

Notice that, if $A[X_i] = z$, then $Q_i = R_{i,z} + L_{i,z}$. Recall from the outline on page 42 that one of our two subgoals is to show that $\mathrm{E}(Q_i) = O(n/i)$. Toward this end, the next two lemmas bound the expectations of $R_{i,z}$ and $L_{i,z}$.

Lemma 18.1: For any $i \in \{1, 2, \ldots, n-1\}$, any positive integer r, and any $y, z \in \mathcal{Z}_n$,

$$\Pr(R_{i,y} > r \mid A[X_i] = z) \le 4n^2/(i^2 r^2), \text{ and} \tag{18.1}$$

$$\Pr(L_{i,y} > r \mid A[X_i] = z) \le 4n^2/(i^2 r^2). \tag{18.2}$$

Proof: For (18.1), assume $r < n - y$ since otherwise, by the definition of $R_{i,y}$, $\Pr(R_{i,y} > n - y) = 0$, and the statement is vacuously true. Let $S = \{y+1, y+2, \ldots, y+r\}$. Note that $S \subseteq \mathcal{Z}_n$, since $r < n - y$. Then

$$\begin{aligned} \Pr(R_{i,y} > r \mid A[X_i] = z) &= \Pr\left(\bigwedge_{j=0}^{i-1} (A[X_j] \notin S) \mid A[X_i] = z\right) \\ &\le 4n^2/(i^2 r^2), \end{aligned}$$

by Lemma 17.1. The proof of (18.2) is similar. □

Lemma 18.2: For any $i \in \{1, 2, \ldots, n-1\}$ and any $y, z \in \mathcal{Z}_n$,

$$\begin{aligned} \mathrm{E}(R_{i,y} \mid A[X_i] = z) &= O(n/i), \text{ and} && (18.3) \\ \mathrm{E}(L_{i,y} \mid A[X_i] = z) &= O(n/i). && (18.4) \end{aligned}$$

Proof: For (18.3),

$$\begin{aligned} \mathrm{E}(R_{i,y} \mid A[X_i] = z) &= \sum_{s=0}^{n} s \cdot \Pr(R_{i,y} = s \mid A[X_i] = z) \\ &= \sum_{s=0}^{n} \sum_{r=0}^{s-1} \Pr(R_{i,y} = s \mid A[X_i] = z) \\ &= \sum_{r=0}^{n} \sum_{s=r+1}^{n} \Pr(R_{i,y} = s \mid A[X_i] = z) \\ &= \sum_{r=0}^{n} \Pr(R_{i,y} > r \mid A[X_i] = z) \\ &\le 1 + \lceil n/i \rceil + \sum_{r=\lceil n/i \rceil + 1}^{n} \frac{4n^2}{i^2 r^2} && \text{(Lemma 18.1)} \\ &\le 1 + \lceil n/i \rceil + \int_{\lceil n/i \rceil}^{n} \frac{4n^2}{i^2 r^2} dr && \text{(See Figure 18.1)} \\ &= 1 + \lceil n/i \rceil + \left(\frac{-4n^2}{i^2 r} \right) \Big|_{r=\lceil n/i \rceil}^{n} \\ &\le 1 + \lceil n/i \rceil + 4n/i - 4n/i^2 \\ &= O(n/i). \end{aligned}$$

The proof of (18.4) is similar. □

The following lemma completes the first of our two subgoals.

Lemma 18.3: For any $i \in \{1, 2, \ldots, n-1\}$, $\mathrm{E}(Q_i) = O(n/i)$.

Proof:

$$\begin{aligned} \mathrm{E}(Q_i) &= \sum_{z=0}^{n-1} \mathrm{E}(Q_i \mid A[X_i] = z) \cdot \Pr(A[X_i] = z) \\ &= \sum_{z=0}^{n-1} \mathrm{E}(R_{i,z} + L_{i,z} \mid A[X_i] = z) \cdot \Pr(A[X_i] = z) \\ &= O(n/i) \sum_{z=0}^{n-1} \Pr(A[X_i] = z) && \text{(Lemma 18.2)} \end{aligned}$$

$$= O(n/i).$$

□

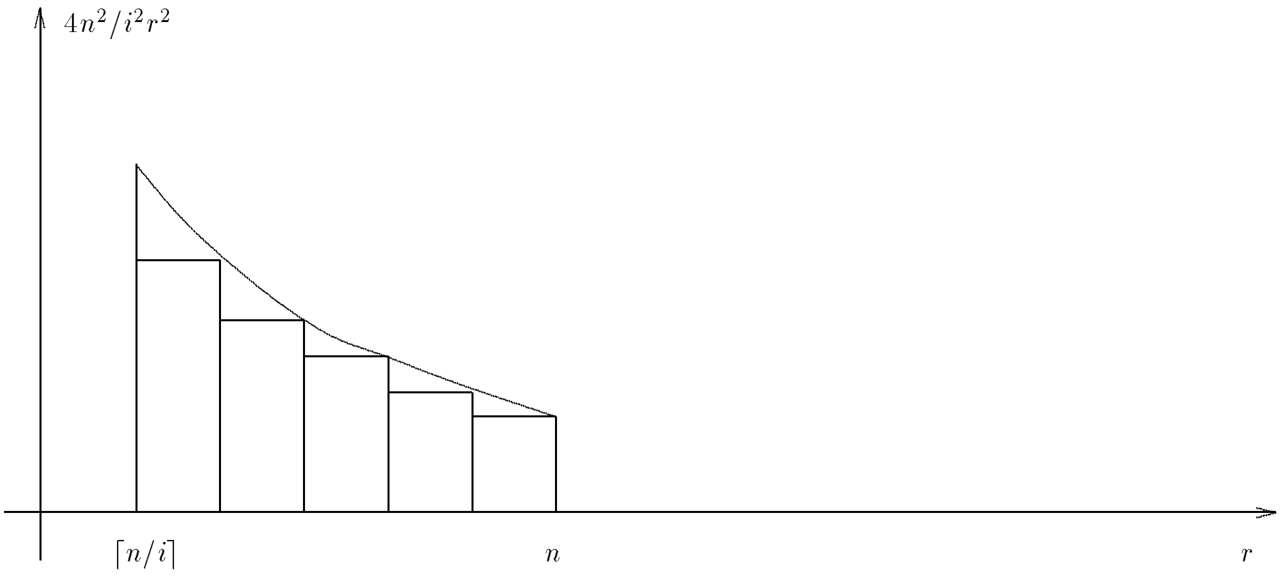

Figure 18.1: Bounding a sum by an integral

18.2. Cost of All Partitions in P2

The second of our two subgoals in the outline on page 42 is achieved in the following lemma.

Lemma 18.4: The total expected cost of all P2 partitions is $O(n)$.

Proof: Let $D_y = L_{n-1,y} + R_{n-1,y}$. (The reason to choose $n-1$ rather than n is so that Lemma 18.2 still applies.) The total cost of all P2 partitions is at most $\sum_{y=0}^{n-1} D_y$.

$$\begin{aligned} \mathrm{E}\left(\sum_{y=0}^{n-1} D_y\right) &= \sum_{y=0}^{n-1} \mathrm{E}(D_y) \\ &= \sum_{y=0}^{n-1} \mathrm{E}(L_{n-1,y} + R_{n-1,y}) \\ &= \sum_{y=0}^{n-1} (\mathrm{E}(L_{n-1,y}) + \mathrm{E}(R_{n-1,y})) \end{aligned}$$

$$\begin{aligned}
&= \sum_{y=0}^{n-1}\sum_{z=0}^{n-1} \left(\mathrm{E}(L_{n-1,y} \mid A[X_{n-1}] = z) \cdot \Pr\left(A[X_{n-1}] = z\right)\right. \\
&\qquad \left. + \mathrm{E}(R_{n-1,y} \mid A[X_{n-1}] = z) \cdot \Pr\left(A[X_{n-1}] = z\right)\right) \\
&= \sum_{y=0}^{n-1}\sum_{z=0}^{n-1} O\left(\frac{n}{n-1}\right) \cdot \Pr\left(A[X_{n-1}] = z\right) && \text{(Lemma 18.2)} \\
&= \sum_{y=0}^{n-1} O(1) \sum_{z=0}^{n-1} \Pr\left(A[X_{n-1}] = z\right) \\
&= \sum_{y=0}^{n-1} O(1) \\
&= O(n).
\end{aligned}$$

□

Notice from the penultimate line of this derivation that $\mathrm{E}(D_y) = O(1)$. That is, any particular subproblem during the cleanup step P2 is expected to have constant size.

Theorem 15.1 follows from Lemmas 18.3 and 18.4 as described in the outline on page 42.

Lecture 19

Space-Bounded Randomness

February 27, 1991
Notes: Tor Jeremiassen

19.1. Pseudorandomness for Space-Bounded Algorithms

The series of lectures on Quicksort with the 5-way pseudorandom generator showed that only a small number of truly random seed bits is needed to obtain $O(n \log n)$ expected running time. In one sense this is not terribly surprising, since we know how to sort in $O(n \log n)$ time without any randomness at all. A more dramatic and general result, due to Nisan [30], shows that there is a pseudorandom generator that can be used to decrease the randomness needed by *any* space-bounded probabilistic algorithm.

More specifically, Nisan showed that, for any R and S, there is a pseudorandom generator G that takes $O(S \log R)$ random seed bits and outputs R pseudorandom bits that "look" truly random to any space S probabilistic algorithm A. That is, if A accepts a language L with 2-sided error, space S, and R random bits, then running A with the pseudorandom generator in place of truly random bits accepts L with only $O(S \log R)$ random bits (and $O(S \log R)$ space).

For instance, if $L \in BPLP$ then L is accepted by a probabilistic algorithm that uses $O(\log^2 n)$ space, polynomial time, and only $O(\log^2 n)$ random bits. In particular, a pseudorandom walk according to this generator solves USTCON with only $O(\log^2 n)$ random bits instead of $\Omega(ne)$.

19.2. Universal Hash Functions

Nisan's generator is built from universal hash functions, which were introduced by Carter and Wegman [9, 10].

Notation: $\Pr_{x \in X}(...)$ and $\mathrm{E}_{x \in X}(...)$ denote probability and expectation, respectively, when x is chosen randomly and uniformly from X. This notation will be useful, as the expressions that arise will often involve more variables than just x, and there would otherwise be confusion about which random variable is being quantified.

Definition: Let E and F be finite sets with $|E| > 1$. A set H of functions $h : E \to F$ is called a *universal family of hash functions* if and only if, for any two distinct $x_1, x_2 \in E$, and any $y_1, y_2 \in F$,

$$\Pr_{h \in H}(h(x_1) = y_1 \wedge h(x_2) = y_2) = \frac{1}{|F|^2} \, .$$

Example 19.1: Let $E = F$ be a field and let $H = \{h_{a,b} \mid a, b \in F\}$, where $h_{a,b}(x) = ax + b$. H is a universal family of hash functions, as follows. Lemma 16.3 says that if a and b are chosen randomly, uniformly, and independently from F, then $ax + b$ is uniformly distributed in F for any $x \in F$, and $\{ax + b \mid x \in F\}$ is pairwise (i.e., 2-way) independent. Using these two facts,

$$\begin{aligned}
\Pr_{h_{a,b} \in H}(h_{a,b}(x_1) = y_1 \wedge h_{a,b}(x_2) = y_2) &= \Pr_{a,b \in F}(ax_1 + b = y_1 \wedge ax_2 + b = y_2) \\
&= \Pr_{a,b \in F}(ax_1 + b = y_1) \Pr_{a,b \in F}(ax_2 + b = y_2) \\
&= \frac{1}{|F|^2} \; .
\end{aligned}$$

Lecture 20

Properties of Universal Hash Functions

March 1, 1991
Notes: Donald Chinn

Next we prove two properties of universal families of hash functions needed for our goal of constructing Nisan's pseudorandom generator. The first says that a randomly chosen hash function distributes any input from the domain uniformly over the codomain.

Lemma 20.1: If H is a universal family of hash functions $h : E \to F$, and $x_1 \in E$ and $y_1 \in F$, then

$$\Pr_{h \in H}(h(x_1) = y_1) = \frac{1}{|F|} .$$

Proof: Let $x_2 \in E - \{x_1\}$.

$$\begin{aligned}
\Pr_{h\in H}(h(x_1) = y_1) &= \Pr_{h\in H}\left(\bigvee_{y_2 \in F} (h(x_1) = y_1 \;\&\; h(x_2) = y_2)\right) \\
&= \sum_{y_2 \in F} \Pr_{h \in H}(h(x_1) = y_1 \;\&\; h(x_2) = y_2) \\
&\qquad \text{(the disjuncts are mutually exclusive events)} \\
&= \sum_{y_2 \in F} \frac{1}{|F|^2} \qquad \text{(definition of universal hash function)} \\
&= \frac{1}{|F|} .
\end{aligned}$$

□

Definition: Let $h : E \to F, A \subseteq E, B \subseteq F$, and $\epsilon > 0$. We say h is *ϵ-random on (A, B)* if and only if

$$\left| \Pr_{x \in E}(x \in A \;\&\; h(x) \in B) - p(A)p(B) \right| \le \epsilon,$$

where $p(A) = |A|/|E|$ and $p(B) = |B|/|F|$.

The idea behind this definition is that, if we choose $x \in E$ and $y \in F$ randomly, uniformly, and independently, then $\Pr(x \in A \;\&\; y \in B) = p(A)p(B)$. Thus, if ϵ is small and h is ϵ-random on (A, B), then if we choose only x randomly and uniformly, $h(x)$ "looks like" a random, independent element of F, at least with respect to membership in B. One can see a foreshadowing of pseudorandomness in this definition and the following theorem: given a random seed x, $h(x)$ can be used in place of a random y.

Theorem 20.2 (Nisan [30]): Let H be a universal family of hash functions $h : E \to F$. Let $A \subseteq E$, $B \subseteq F$, and $\epsilon > 0$. Then

$$\Pr_{h \in H}(h \text{ is not } \epsilon\text{-random on } (A, B)) \leq \frac{p(A)p(B)(1 - p(B))}{\epsilon^2 |E|}.$$

Proof: Let M be a $|E| \times |H|$ matrix such that

$$M_{x,h} = \begin{cases} 1 & \text{if } h(x) \in B \\ 0 & \text{otherwise} \end{cases} .$$

Since H is universal, for any $x \in E$,

$$\begin{aligned} \mathop{\mathrm{E}}_{h \in H}(M_{x,h}) &= \Pr_{h \in H}(h(x) \in B) \\ &= |B| \cdot \frac{1}{|F|} \qquad \text{(Lemma 20.1)} \\ &= p(B) \ . \qquad (20.1) \end{aligned}$$

In words, this says that the average value in any row of M is $p(B)$.

Now let $U = \mathrm{E}_{x \in A}(M_{x,h})$. U is the average value, over rows in A, of the column of M labeled h. We will calculate the variance (that is, the second central moment) of U as a function of the random variable h. (Note that x is bound in the definition of U, so that U is not a function of the random variable x.)

Before calculating the variance of U, we must calculate its mean:

$$\begin{aligned} \mathop{\mathrm{E}}_{h \in H}(U) &= \mathop{\mathrm{E}}_{h \in H}\left(\mathop{\mathrm{E}}_{x \in A}(M_{x,h})\right) \\ &= \mathop{\mathrm{E}}_{x \in A}\left(\mathop{\mathrm{E}}_{h \in H}(M_{x,h})\right) \\ &= \mathop{\mathrm{E}}_{x \in A}(p(B)) \qquad \text{(Equation 20.1)} \\ &= p(B) \ . \end{aligned}$$

The variance of U is

$$\begin{aligned} &\mathop{\mathrm{E}}_{h \in H}((U - p(B))^2) \\ &\quad = \mathop{\mathrm{E}}_{h \in H}\left(\left(\mathop{\mathrm{E}}_{x \in A}(M_{x,h}) - p(B)\right)^2\right) \end{aligned}$$

$$\begin{aligned}
&= \mathop{\mathrm{E}}_{h\in H}\left(\left(\mathop{\mathrm{E}}_{x_1\in A}(M_{x_1,h}) - p(B)\right)\left(\mathop{\mathrm{E}}_{x_2\in A}(M_{x_2,h}) - p(B)\right)\right) \\
&= \mathop{\mathrm{E}}_{h\in H}\left(\mathop{\mathrm{E}}_{x_1\in A}(M_{x_1,h} - p(B))\mathop{\mathrm{E}}_{x_2\in A}(M_{x_2,h} - p(B))\right) \\
&= \mathop{\mathrm{E}}_{h\in H}\left(\mathop{\mathrm{E}}_{x_1,x_2\in A}((M_{x_1,h} - p(B))(M_{x_2,h} - p(B)))\right) \\
&= \mathop{\mathrm{E}}_{x_1,x_2\in A}\left(\mathop{\mathrm{E}}_{h\in H}((M_{x_1,h} - p(B))(M_{x_2,h} - p(B)))\right) \\
&= \mathop{\mathrm{E}}_{x_1,x_2\in A}\left(\mathop{\mathrm{E}}_{h\in H}((M_{x_1,h} - p(B))(M_{x_2,h} - p(B)))\;\middle|\; x_1 \neq x_2\right)\mathop{\mathrm{Pr}}_{x_1,x_2\in A}(x_1 \neq x_2) \\
&\quad + \mathop{\mathrm{E}}_{x_1,x_2\in A}\left(\mathop{\mathrm{E}}_{h\in H}((M_{x_1,h} - p(B))(M_{x_2,h} - p(B)))\;\middle|\; x_1 = x_2\right)\mathop{\mathrm{Pr}}_{x_1,x_2\in A}(x_1 = x_2) \\
&= \mathop{\mathrm{E}}_{x_1,x_2\in A}\left(\mathop{\mathrm{E}}_{h\in H}(M_{x_1,h} - p(B))\mathop{\mathrm{E}}_{h\in H}(M_{x_2,h} - p(B))\;\middle|\; x_1 \neq x_2\right)\mathop{\mathrm{Pr}}_{x_1,x_2\in A}(x_1 \neq x_2) \\
&\quad + \mathop{\mathrm{E}}_{x_1\in A}\left(\mathop{\mathrm{E}}_{h\in H}((M_{x_1,h} - p(B))^2)\right)\cdot\frac{1}{|A|} \\
&\qquad\qquad (h(x_1) \text{ and } h(x_2) \text{ are independent if } x_1 \neq x_2) \\
&= 0 + \frac{p(B)(1-p(B))^2 + (1-p(B))(-p(B))^2}{|A|} \qquad \text{(Equation 20.1)} \\
&= \frac{p(B)(1-p(B))}{|A|}.
\end{aligned}$$

The proof will be completed next lecture.

Lecture 21

Universal Hash Functions and Nisan's Generator

March 4, 1991
Notes: Elizabeth Walkup

21.1. Conclusion of the Proof of Theorem 20.2

Recall the following quantities:

- $U = \mathrm{E}_{x \in A}(M_{x,h}) = \Pr_{x \in A}(h(x) \in B)$ is a function of the random variable h.
- The mean of U is $p(B)$.
- The variance of U is $p(B)(1 - p(B))/|A|$.

Let τ denote the positive real square root of the variance of U. Then the proof of Theorem 20.2 is completed as follows:

$$\begin{aligned}
&\Pr_{h \in H}(h \text{ is not } \epsilon\text{-random on } (A,B)) \\
&\quad = \Pr_{h \in H}\left(\left|\Pr_{x \in E}(x \in A \ \&\ h(x) \in B) - p(A)p(B)\right| > \epsilon\right) \\
&\quad = \Pr_{h \in H}\left(\left|\Pr_{x \in E}(x \in A) \cdot \Pr_{x \in E}(h(x) \in B \mid x \in A) - p(A)p(B)\right| > \epsilon\right) \\
&\quad = \Pr_{h \in H}\left(\left|p(A) \cdot \Pr_{x \in A}(h(x) \in B) - p(A)p(B)\right| > \epsilon\right) \\
&\quad = \Pr_{h \in H}\left(|U - p(B)| > \frac{\epsilon}{p(A)}\right) \\
&\quad = \Pr_{h \in H}\left(|U - p(B)| > \frac{\epsilon}{p(A)\,\tau} \cdot \tau\right) \\
&\quad \leq \frac{(p(A))^2 \tau^2}{\epsilon^2} \qquad \text{(Lemma 15.2)} \\
&\quad = \frac{(p(A))^2 p(B)\,(1 - p(B))}{\epsilon^2 |A|}
\end{aligned}$$

$$= \frac{p(A)p(B)(1-p(B))}{\epsilon^2|E|}.$$

□

21.2. Nisan's Generator

Definition: Let t be a positive integer, and let H be a universal family of hash functions $h:\{0,1\}^t \to \{0,1\}^t$. For every $k \in \mathcal{N}$, define the pseudorandom generator $G_k : \{0,1\}^t \times H^k \to \{0,1\}^{t\cdot 2^k}$ recursively as follows:

$$\begin{aligned} G_0(x) &= x\ ,\ \text{and} \\ G_k(x,h_1,h_2,\ldots,h_k) &= G_{k-1}(x,h_1,h_2,\ldots,h_{k-1}) \circ G_{k-1}(h_k(x),h_1,h_2,\ldots,h_{k-1}) \end{aligned}$$

for all positive integers k, where $\circ$ denotes concatenation.

For example,

$$\begin{aligned} G_1(x,h_1) &= x \circ h_1(x)\ , \\ G_2(x,h_1,h_2) &= x \circ h_1(x) \circ h_2(x) \circ h_1(h_2(x))\ ,\ \text{and} \\ G_3(x,h_1,h_2,h_3) &= x \circ h_1(x) \circ h_2(x) \circ h_1(h_2(x)) \circ h_3(x) \circ h_1(h_3(x)) \circ h_2(h_3(x)) \circ h_1(h_2(h_3(x))). \end{aligned}$$

Note that G_k takes as its random seed a t-bit string and k hash functions, and produces a $t \cdot 2^k$ bit pseudorandom string. In order to be useful, we need to understand how many random seed bits are required to specify one of these hash function.

Example 21.1: As in Example 19.1, take H to be the set of all linear functions on $F = \mathrm{GF}(2^t)$, the finite (Galois) field with 2^t elements. The elements of this field are all polynomials of degree at most $t-1$ with coefficients from $\mathcal{Z}_2$, and the field operations are the usual polynomial operations, but reduced modulo some fixed degree t polynomial. (For more detail see, for example, Lipson [29, Chapter VI].) Thus, every $a \in F$ can be specified by t bits (the t coefficients) in such a way that the field operations are efficiently computable from this representation. Each hash function $h_{a,b} \in H$ in turn can be specified by a pair $(a,b) \in F \times F$, where $h_{a,b}(x) = ax+b$. That is, each of the k hash functions can be represented using $2t$ bits. In summary, G_k requires $t(2k+1)$ random seed bits, and outputs $t \cdot 2^k$ pseudorandom bits.

This is a great expansion if t is not too large compared to k. This suggests choosing t to be a constant, in which case we would obtain exponential expansion of randomness. However, in order for this pseudorandom generator to be able to "fool" any space S bounded probabilistic algorithm A, we will ultimately be forced to choose t to be somewhat greater than S. The reason for this is so that A has insufficient space to retain even a single previously encountered element of F. If this is the case, it will turn out, for example, that the element $h_1(h_2(h_3(x))) \in F$ "looks" completely random and independent to A, even though A has already encountered (but necessarily "forgotten") the 7 elements x, $h_1(x)$, $h_2(x)$, $h_1(h_2(x))$, $h_3(x)$, $h_1(h_3(x))$, and $h_2(h_3(x))$. (Note that, if t were much less than S, then A would have the space to solve these 7 equations in 7 variables —

2 coefficients for each of the 3 hash functions, plus 1 for x — and then predict $h_1(h_2(h_3(x)))$ with complete accuracy, rendering it anything but independent. Thus, the pseudorandom sequence of elements of F output by Nisan's generator is not even 8-way independent. Nevertheless, it "looks" mutually independent to any space-bounded algorithm, as we will prove.)

21.3. Modeling Space-Bounded Probabilistic Algorithms

Let S be the space bound of a probabilistic algorithm A on input w. We will model A on input w as a finite state automaton Q with 2^S states and alphabet $\{0,1\}^t$. Q has a state for every configuration of A, and a transition from state u to state v on symbol $\alpha \in \{0,1\}^t$ if and only if A, on input w in configuration u, when given α as the next t random bits, would be in configuration v after consuming α.

Lecture 22

Automata Under Various Distributions

March 6, 1991
Notes: Simon Kahan

22.1. Modeling Space-Bounded Algorithms (conclusion)

In the finite state model, each state of the automaton corresponds to a configuration of the algorithm. Note that the input to the algorithm is *not* part of this configuration, as the space bound may be quite a bit less than the input length. Indeed, the input is implicit in the structure of the finite state automaton itself. As a result, the model is extremely nonuniform: the finite state automaton may be different not only for every input size, but also for every input.

Because Nisan's proof will rely only on the fact that these finite state automata have 2^S states, his result is actually stronger than previously suggested. If the string of random bits is partitioned into blocks of t consecutive bits, then Nisan's generator can be used for probabilistic algorithms having unlimited space, as long as the space is restricted to S at those points of time at which the first bit of any block of t random bits is about to be consumed.

The basic idea behind Nisan's result is that any information encoded into S bits about previously generated sequences of t pseudorandom bits is insufficient to predict the next t bits generated with any accuracy.

22.2. Behavior of Finite State Automata Under Various Distributions

Definition: For $b \in \mathcal{N}$ let U_b denote the uniform distribution on $\{0,1\}^b$.

Definition: For fixed $h_1, h_2, \ldots, h_k \in H$, let $G_k(*, h_1, h_2, \ldots, h_k)$ denote the distribution of $G_k(x, h_1, h_2, \ldots, h_k) \in \{0,1\}^{t \cdot 2^k}$ induced by choosing x from the distribution U_t.

Definition: Let Q be a finite state automaton with 2^S states and alphabet $\{0,1\}^t$, and let D be a distribution on $\{0,1\}^{tb}$. Then $Q(D)$ denotes the $2^S \times 2^S$ matrix whose (i,j) entry is the probability that, for $y \in \{0,1\}^{tb}$ chosen randomly according to distribution D, Q moves from state i to state j when given the random string y.

Example 22.1: Let $b = 1$ and $D = U_t$. Suppose $\alpha_1, \alpha_2, \ldots, \alpha_d \in \{0,1\}^t$ are the only symbols causing a transition in one step from state i to state j. Then $(Q(D))_{ij} = d/(2^t)$. For $b > 1$,

$Q(U_{tb}) = (Q(U_t))^b$.

Example 22.2: Let $D = G_k(*, h_1, h_2, \ldots, h_k)$. $Q(D)$ is the probability transition matrix for $t \cdot 2^k$ "random" steps of Q, when the randomness comes from $G_k(*, h_1, h_2, \ldots, h_k)$.

We will be interested in comparing $Q(G_k(*, h_1, h_2, \ldots, h_k))$ to $Q(U_{t2^k})$. If these two matrices are similar, then the behavior of Q will be similar under both distributions. We could then conclude that the algorithm underlying Q behaves similarly whether its source of randomness is Nisan's pseudorandom generator or true randomness. We will show that the two matrices are similar by demonstrating that a certain norm of $Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})$ is very small.

Definition: Let $\Re$ denote the real numbers, and let m be a positive integer. For $x \in \Re^m$, define $\|x\| = \sum_{i=1}^m |x_i|$. For an $m \times m$ matrix M, define

$$\|M\| = \max_{x \in \Re^m} \frac{\|Mx\|}{\|x\|}.$$

The following lemma, whose proof is left to the reader, provides some standard properties of this norm that will be needed.

Lemma 22.3: Let M and N be $m \times m$ real matrices. Then

1. $\|M + N\| \leq \|M\| + \|N\|$.
2. $\|MN\| \leq \|M\| \cdot \|N\|$.
3. If M has only nonnegative entries, and the sum of each row of M is 1, then $\|M\| = 1$.
4. If each entry in M has absolute value at most B, then $\|M\| \leq mB$.

Lemma 22.4: Let H be a universal family of hash functions $h : \{0,1\}^t \to \{0,1\}^t$. Let Q be a finite state automaton with 2^S states and alphabet $\{0,1\}^t$. Let $\epsilon > 0$ and $k \in \mathcal{N}$. Then

$$\Pr_{h_1, \ldots, h_k \in H} \Big(\|Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})\| > (2^k - 1)\epsilon \Big) \leq \frac{2^{6S} k}{\epsilon^2 \, 2^t}.$$

Note that when we eventually apply Lemma 22.4 we will want this probability to be much less than one, and so will choose t somewhat greater than S, as suggested earlier.

Proof: The proof is by induction on k.

Basis (k = 0): $G_k(*, h_1, h_2, \ldots, h_k) = G_0(*) = U_t = U_{t2^k}$. Substituting into the statement of the lemma yields $\Pr(0 > 0) \leq 0$, which is true.

The induction step will be covered next lecture.

Lecture 23

Proof of Nisan's Main Lemma

March 8, 1991
Notes: Juan Alemany

Induction ($k > 0$): For any $h_1, h_2, \ldots, h_{k-1} \in H$ and any states i and j of Q, let

$$B_{i,j}^{h_1,\ldots,h_{k-1}} = \{x \mid G_{k-1}(x, h_1, h_2, \ldots, h_{k-1}) \text{ takes } Q \text{ from } i \text{ to } j\}.$$

Note in this definition that the hash functions $h_1, h_2, \ldots, h_{k-1}$ are fixed, and only the t-bit string x varies. In words, $B_{i,j}^{h_1,\ldots,h_{k-1}}$ is the set of such strings x that cause Q to move from state i to state j in $t \cdot 2^{k-1}$ random steps, where the random bits are produced by $G_{k-1}(x, h_1, h_2, \ldots, h_{k-1})$.

Consider the following two events:

E1. $\|Q(G_{k-1}(*, h_1, h_2, \ldots, h_{k-1})) - Q(U_{t2^{k-1}})\| \leq (2^{k-1} - 1)\epsilon$.

E2. For all triples (i, l, j) of states, h_k is $(2^{-2S}\epsilon)$-random on $\left(B_{i,l}^{h_1,\ldots,h_{k-1}}, B_{l,j}^{h_1,\ldots,h_{k-1}}\right)$.

(See page 56 to review the definition of "ϵ-random on (A, B)".)

We now make the following two claims about E1 and E2. The induction step and the lemma will follow from these claims.

Claim 1:

$$\Pr_{h_1,\ldots,h_k \in H}(\neg\text{E1} \vee \neg\text{E2}) \leq \frac{2^{6S}k}{\epsilon^2 2^t}.$$

Claim 2: If E1 and E2 are true, then $\|Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})\| \leq (2^k - 1)\epsilon$.

The two claims imply Lemma 22.4 because

$$\begin{aligned} \Pr_{h_1,\ldots,h_k \in H}\left(\|Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})\| > (2^k - 1)\epsilon\right) &\leq \Pr_{h_1,\ldots,h_k \in H}(\neg\text{E1} \vee \neg\text{E2}) \\ &\leq \frac{2^{6S}k}{\epsilon^2 2^t}. \end{aligned}$$

Proof of Claim 1: We need to bound the probability that at least one of events $\neg$E1 or $\neg$E2 occurs. We examine the separate probabilities that either $\neg$E1 or $\neg$E2 occurs. The combined probability is bounded by the sum of the separate probabilities.

Bounding the probability of ¬E1 is simple since, by the induction hypothesis,

$$\Pr_{h_1,\ldots,h_{k-1}\in H}(\neg\mathrm{E1}) \leq \frac{2^{6S}(k-1)}{\epsilon^2 2^t}.$$

To bound the probability of ¬E2, note that ¬E2 means that there exists a triple (i,l,j) such that h_k is not $(2^{-2S}\epsilon)$-random on $\left(B_{i,l}^{h_1,\ldots,h_{k-1}}, B_{l,j}^{h_1,\ldots,h_{k-1}}\right)$. That is,

$$\begin{aligned}
&\Pr_{h_1,\ldots,h_k\in H}(\neg\mathrm{E2}) \\
&\quad\leq \max_{h_1,\ldots,h_{k-1}\in H} \Pr_{h_k\in H}(\neg\mathrm{E2}) \\
&\quad\leq \max_{h_1,\ldots,h_{k-1}\in H} \sum_{i,l,j} \Pr_{h_k\in H}\left(h_k \text{ is not } (2^{-2S}\epsilon)\text{-random on } \left(B_{i,l}^{h_1,\ldots,h_{k-1}}, B_{l,j}^{h_1,\ldots,h_{k-1}}\right)\right) \\
&\quad\leq \max_{h_1,\ldots,h_{k-1}\in H} \sum_{i,l,j} \frac{2^{4S} p\left(B_{i,l}^{h_1,\ldots,h_{k-1}}\right)}{\epsilon^2 2^t} \qquad \text{(Theorem 20.2)} \\
&\quad= \max_{h_1,\ldots,h_{k-1}\in H} \frac{2^{4S}}{\epsilon^2 2^t} \sum_{i,j}\sum_{l} p\left(B_{i,l}^{h_1,\ldots,h_{k-1}}\right) \\
&\quad= \max_{h_1,\ldots,h_{k-1}\in H} \frac{2^{4S}}{\epsilon^2 2^t} \sum_{i,j} 1 \\
&\quad= \frac{2^{6S}}{\epsilon^2 2^t}.
\end{aligned}$$

Now we combine the separate probabilities of ¬E1 and ¬E2 to obtain Claim 1, as follows:

$$\begin{aligned}
\Pr_{h_1,\ldots,h_k\in H}(\neg\mathrm{E1}\vee\neg\mathrm{E2}) &\leq \Pr_{h_1,\ldots,h_k\in H}(\neg\mathrm{E1}) + \Pr_{h_1,\ldots,h_k\in H}(\neg\mathrm{E2}) \\
&\leq \frac{2^{6S}(k-1)}{\epsilon^2 2^t} + \frac{2^{6S}}{\epsilon^2 2^t} \\
&= \frac{2^{6S}k}{\epsilon^2 2^t}.
\end{aligned}$$

Proof of Claim 2: Assume E1 and E2 hold. Let

$$\begin{aligned}
M &= Q(G_k(*,h_1,h_2,\ldots,h_k))\,, \\
N &= Q(U_{t2^k})\,, \text{ and} \\
L &= Q(G_{k-1}(*,h_1,h_2,\ldots,h_{k-1}))\,.
\end{aligned}$$

We will need two subclaims to establish Claim 2:

Claim 2.1: $\|M - L^2\| \leq \epsilon$.

Claim 2.2: $\|L^2 - N\| \le (2^k - 2)\epsilon$.

Note that Claim 2 follows from Claims 2.1 and 2.2, since

$$\begin{aligned} \|M - N\| &\le \|M - L^2\| + \|L^2 - N\| && \text{(Lemma 22.3 (1))} \\ &\le (2^k - 1)\epsilon \end{aligned}$$

Proof of Claim 2.1: We consider a typical entry of both the matrices M and L^2. If we can bound the difference of these two entries, then we can apply Lemma 22.3 (4) to bound the norm.

First consider a typical entry of M:

$$M_{ij} = \sum_{l=1}^{2^S} \Pr_{x\in\{0,1\}^t}\left(x \in B_{i,l}^{h_1,\dots,h_{k-1}} \wedge h_k(x) \in B_{l,j}^{h_1,\dots,h_{k-1}}\right).$$

Now consider a typical entry of L^2:

$$\begin{aligned} L^2_{ij} &= \sum_{l=1}^{2^S} L_{il}L_{lj} \\ &= \sum_{l=1}^{2^S} \Pr_{x\in\{0,1\}^t}\left(x \in B_{i,l}^{h_1,\dots,h_{k-1}}\right) \Pr_{y\in\{0,1\}^t}\left(y \in B_{l,j}^{h_1,\dots,h_{k-1}}\right) \\ &= \sum_{l=1}^{2^S} p\left(B_{i,l}^{h_1,\dots,h_{k-1}}\right) p\left(B_{l,j}^{h_1,\dots,h_{k-1}}\right). \end{aligned}$$

Thus, for all i and j,

$$\begin{aligned} &\left|M_{ij} - L^2_{ij}\right| \\ &\le \sum_{l=1}^{2^S} \left| \Pr_{x\in\{0,1\}^t}\left(x \in B_{i,l}^{h_1,\dots,h_{k-1}} \wedge h_k(x) \in B_{l,j}^{h_1,\dots,h_{k-1}}\right) - p\left(B_{i,l}^{h_1,\dots,h_{k-1}}\right) p\left(B_{l,j}^{h_1,\dots,h_{k-1}}\right)\right| \\ &\le \sum_{l=1}^{2^S} 2^{-2S}\epsilon && \text{(E2)} \\ &= 2^{-S}\epsilon. \end{aligned}$$

Therefore $\|M - L^2\| \le \epsilon$, by Lemma 22.3 (4).

Lecture 24

Nisan's Lemmas, Concluded

March 11, 1991
Notes: Wendy Thrash

24.1. Conclusion of the Proof of Lemma 22.4

Proof of Claim 2.2: Let $K = Q(U_{t2^{k-1}})$, and note that $N = K^2$. Then

$$\begin{aligned}
\|L^2 - N\| &= \|L^2 - K^2\| \\
&= \|L^2 - KL + KL - K^2\| \\
&= \|(L-K)L + K(L-K)\| \\
&\leq \|L-K\| \cdot \|L\| + \|K\| \cdot \|L-K\| && \text{(Lemma 22.3 (1,2))} \\
&= \|L-K\|(\|L\| + \|K\|) \\
&\leq (2^{k-1} - 1)\epsilon \cdot (\|L\| + \|K\|) && \text{(E1)} \\
&= (2^k - 2)\epsilon\,. && \text{(Lemma 22.3 (3))}
\end{aligned}$$

□

The proof of Nisan's main lemma (Lemma 22.4) is now complete. Let us pause for a moment to glean some intuition from this proof, because it is here that the ingeniousness of Nisan's generator has been revealed.

Our goal is to show that, using Nisan's generator, the probability that the finite state automaton Q accepts a pseudorandom string is close to the probability that it accepts a truly random string. Towards that goal, we have just shown that the behavior of Q under pseudorandomness is close to its behavior under randomness, in the sense that $\|M - N\|$ is small. We accomplished this by showing that both M (representing Q under Nisan's generator) and N (representing Q under randomness) are close to the matrix L^2, which represents Q under Nisan's generator running for half the steps, then running again, independently, for the second half. More precisely, M represents the behavior of Q under

$$G_{k-1}(x, h_1, h_2, \ldots, h_{k-1}) \circ G_{k-1}(h_k(x), h_1, h_2, \ldots, h_{k-1})$$

for random x, and L^2 represents the behavior under

$$G_{k-1}(x, h_1, h_2, \ldots, h_{k-1}) \circ G_{k-1}(y, h_1, h_2, \ldots, h_{k-1})$$

for random and independent x and y: these are close as a result of Theorem 20.2. On the other hand, N represents the behavior of Q under

$$U_{t2^{k-1}} \circ U_{t2^{k-1}},$$

which is close to L^2 by the induction hypothesis.

24.2. Bounding Q's Probability of Accepting

We continue our proof that the probability that Q accepts a pseudorandom string is close to the probability that it accepts a truly random string with two more lemmas. Lemma 24.1 shows how to go from norm to probability of acceptance, provided the hash functions are chosen so as to make the norm small. Lemma 22.4 showed that this is likely to be the case for randomly chosen hash functions, and these two facts are tied together in Lemma 24.2.

Lemma 24.1: Let H be a universal family of hash functions $h : \{0,1\}^t \to \{0,1\}^t$, and let $h_1, h_2, \ldots, h_k \in H$. Let Q be a finite state automaton with 2^S states and alphabet $\{0,1\}^t$. Let $\delta > 0$. If $h_1, h_2, \ldots, h_k$ satisfy $\|Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})\| \leq \delta$, then

$$\left| \Pr_{x \in \{0,1\}^t}(Q \text{ accepts } G_k(x, h_1, h_2, \ldots, h_k)) - \Pr_{y \in \{0,1\}^{t2^k}}(Q \text{ accepts } y) \right| \leq \delta.$$

Proof: Let $u, a \in \{0,1\}^{2^S}$ be column vectors defined by

$$u_i = \begin{cases} 1 & \text{if } i \text{ is the start state of } Q \\ 0 & \text{otherwise} \end{cases}, \text{ and}$$

$$a_i = \begin{cases} 1 & \text{if } i \text{ is an accepting state of } Q \\ 0 & \text{otherwise} \end{cases}.$$

Let $\langle v, w \rangle$ denote the inner product of real vectors v and w. Then

$$\Pr_{y \in \{0,1\}^{t2^k}}(Q \text{ accepts } y) = \langle Q(U_{t2^k})u, a \rangle$$

and

$$\Pr_{x \in \{0,1\}^t}(Q \text{ accepts } G_k(x, h_1, h_2, \ldots, h_k)) = \langle Q(G_k(*, h_1, h_2, \ldots, h_k))u, a \rangle ,$$

since $Q(\cdot)u$ picks out the ith row of $Q(\cdot)$, where i is the start state, and $\langle Q(\cdot)u, a \rangle$ adds up the probabilities of ending in the various accepting states. Thus,

$$\begin{aligned}
&\left| \Pr_{x \in \{0,1\}^t}(Q \text{ accepts } G_k(x, h_1, h_2, \ldots, h_k)) - \Pr_{y \in \{0,1\}^{t2^k}}(Q \text{ accepts } y) \right| \\
&\quad = |\langle (Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k}))u, a \rangle| \\
&\quad \leq \|(Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k}))u\| && \text{(Definition of vector norm)} \\
&\quad \leq \|Q(G_k(*, h_1, h_2, \ldots, h_k)) - Q(U_{t2^k})\| && \text{(Definition of matrix norm)} \\
&\quad \leq \delta \,. && \text{(Hypothesis)}
\end{aligned}$$

□

Lemma 24.2: Let k, t, and S be positive integers with $k \leq S$ and $t \geq 14S$. Let H be a universal family of hash functions $h : \{0,1\}^t \to \{0,1\}^t$. Let Q be a finite state automaton with 2^S states and alphabet $\{0,1\}^t$. Then

$$\left| \Pr_{\substack{x\in\{0,1\}^t \\ h_1,h_2,\ldots,h_k\in H}} (Q \text{ accepts } G_k(x,h_1,h_2,\ldots,h_k)) - \Pr_{y\in\{0,1\}^{t2^k}} (Q \text{ accepts } y) \right| \leq 2^{-S} .$$

Proof: Let A be the event "Q accepts $G_k(x,h_1,h_2,\ldots,h_k)$", B be the event "$Q$ accepts y", and C be the event "$\|Q(G_k(*,h_1,h_2,\ldots,h_k)) - Q(U_{t2^k})\| > 2^{-2S}$ ". Then (omitting subscripts on probabilities)

$$\begin{aligned}
|\Pr(A) - \Pr(B)| &= |\Pr(A \mid \neg C)(1-\Pr(C)) + \Pr(A \mid C)\Pr(C) - \Pr(B)| \\
&= |\Pr(A \mid \neg C) - \Pr(B) + \Pr(C)(\Pr(A \mid C) - \Pr(A \mid \neg C))| \\
&\leq |\Pr(A \mid \neg C) - \Pr(B)| + \Pr(C)\,|\Pr(A \mid C) - \Pr(A \mid \neg C)| \\
&\leq 2^{-2S} + \Pr(C)\cdot 1 && \text{(Lemma 24.1)} \\
&\leq 2^{-2S} + \frac{2^{6S}\cdot k \cdot 2^{2k}}{(2^{-2S})^2 \cdot 2^t} && \text{(Lemma 22.4, } \epsilon = 2^{-2S}/(2^k-1)\text{)} \\
&= 2^{-2S} + k(2^{10S+2k-t}) \\
&\leq 2^{-2S} + S(2^{-2S}) && (t \geq 14S,\ k \leq S) \\
&= 2^{-2S}(1+S) \\
&\leq 2^{-S} . && (S \geq 1)
\end{aligned}$$

□

Lecture 25

Space-Bounded Randomness, Concluded

March 13, 1991
Notes: Tor Jeremiassen

25.1. Nisan's Pseudorandomness Theorem

Theorem 25.1 (Nisan [30]): If L is accepted by a probabilistic algorithm A with 2-sided (1-sided) error that uses space S, time T, and R random bits, where $2S \leq R = O(S \cdot 2^S)$, then L is accepted by a probabilistic algorithm B with 2-sided (respectively, 1-sided) error that uses space $O(S \log \frac{R}{S}) = O(S^2)$, time $O(ST \log \frac{R}{S}) = O(S^2 T)$, and $O(S \log \frac{R}{S}) = O(S^2)$ random bits. (R, S, and T also need to be "constructible", analogous to the discussion in Corollary 3.2.)

(Note that, unlike the case of deterministic, nondeterministic, or alternating algorithms, the constraint $R = O(S \cdot 2^S)$ is not vacuous: Gill [17] gives natural examples of probabilistic algorithms that run usefully for time double exponential in their space bounds, using that many random bits as well.)

Proof:

Construction: Let $t = 18S$. By increasing S and R by constant factors, we can assume without loss of generality that S is an integral power of 3, and $\frac{R}{18S}$ is an integral power of 2, and that $R \leq 18S \cdot 2^S$. Let $k = \log_2 \frac{R}{18S} \leq S$ (making Lemma 24.2 applicable). Let H be a universal family of hash functions $h : \{0,1\}^t \to \{0,1\}^t$. B simulates A step for step, except that, in place of the R truly random bits that A needs, B uses the string $G_k(x, h_1, h_2, \ldots, h_k)$ as its source of $t \cdot 2^k = 18S \cdot 2^{\log_2(R/(18S))} = R$ random bits, where $x, h_1, h_2, \ldots, h_k$ are chosen randomly, uniformly, and independently, and recorded for use by the generator during the simulation.

Analysis: By Example 21.1 the number of random seed bits that B requires is $t(2k+1) = O(S \log \frac{R}{S})$.

To compute the next t pseudorandom bits, B must apply at most k hash functions to x. Each such application is 1 multiplication and 1 addition in the field GF(2^t). Recall from Example 21.1 that the elements of this field are polynomials of degree at most $t-1$ with coefficients from $\mathcal{Z}_2$. Each such field operation in turn is a polynomial multiplication or addition followed by a division by some fixed irreducible (i.e., unfactorable over $\mathcal{Z}_2$) polynomial $m(x)$ of degree t. By naive algorithms, each of these can be done in $O(t^2)$ time and $O(t)$ space. (Because t is twice an integral power of 3, the polynomial $m(x) = x^t + x^{t/2} + 1$ is irreducible over $\mathcal{Z}_2$ [28, page 146, exercise 3.96]. Shoup [37] gives a more general deterministic polynomial time algorithm for constructing irreducible polynomials, but the space required is proportional to $t^3 = \Theta(S^3)$.)

Since each field operation takes $O(t^2)$ time, each of the k applications of a hash function takes $O(t^2)$ time, since it requires only a constant number of field operations. To produce t pseudorandom bits thus takes time $O(kt^2)$, so simulating all T steps of A takes time $O((T/t)kt^2) = O(Tkt) = O(ST \log \frac{R}{S})$

To record $x, h_1, h_2, \ldots, h_k$ requires $t(2k+1) = O(S \log \frac{R}{S})$ bits of space; field operations require $O(t) = O(S)$ bits of space; to simulate A requires $O(S)$ bits of space; to keep track of the position in the pseudorandom sequence of 2^k blocks requires $O(k) = O(\log \frac{R}{S}) = O(S)$ bits of space. Hence, B runs in space $O(S \log \frac{R}{S})$.

Correctness: Model A as a finite state automaton Q with 2^S states and alphabet $\{0,1\}^t$, as described in Section 21.3.

If A has 2-sided error, then $\Pr(A \text{ accepts } x \mid x \in L) \geq 2/3$ and $\Pr(A \text{ accepts } x \mid x \notin L) \leq 1/3$. By Lemma 24.2, it then follows that

$$\Pr(B \text{ accepts } x \mid x \in L) \geq 2/3 - 2^{-S}$$

and

$$\Pr(B \text{ accepts } x \mid x \notin L) \leq 1/3 + 2^{-S}.$$

The error bound for B can be decreased by applying Corollary 2.2.

If A has 1-sided error, then $\Pr(A \text{ accepts } x \mid x \in L) \geq 1/2$ and $\Pr(A \text{ accepts } x \mid x \notin L) = 0$. By Lemma 24.2,

$$\Pr(B \text{ accepts } x \mid x \in L) \geq 1/2 - 2^{-S}.$$

Since B simulates A,

$$\Pr(B \text{ accepts } x \mid x \notin L) = 0.$$

The error bound for B can be decreased by a second repetition. □

Corollary 25.2: If $L \in \mathit{BPLP}$ (RLP), then L is accepted by a probabilistic algorithm with 2-sided (respectively, 1-sided) error that uses space $O(\log^2 n)$, polynomial time, and only $O(\log^2 n)$ random bits.

In particular, USTCON is so accepted (Theorem 5.2).

Corollary 25.3: If $L \in \mathit{BPLP}$, then L is accepted by a deterministic algorithm that uses space $O(\log^2 n)$.

Proof: Use the space to cycle through all possible seeds, keeping statistics on the number of acceptances. □

Borodin, Cook, and Pippenger [7] and Jung [22] proved a result that is more general than Corollary 25.3.

Open Problem 8: Find a generator for BPLP that uses $o(\log^2 n)$ seed bits. Note that a decrease to $O(\log n)$ seed bits and $O(\log n)$ space would imply that BPLP is the same as deterministic $O(\log n)$ space. In fact, by a result of Ajtai, Komlós, and Szemerédi [3], it suffices to decrease the number of random bits to $O((\log^2 n)/\log\log n)$ and the space to $O(\log n)$.

Lecture 26

Deterministic Amplification

March 15, 1991
Notes: Ed Felten and Dylan McNamee

Suppose a probabilistic algorithm A uses t random bits and has error bound at most $1/3$, but no restriction on space. By Corollary 2.2, running r repetitions of A uses tr random bits, and has error bound $2^{-\Omega(r)}$. Theorem 26.1 below applies Nisan's generator in order to show that this same error bound can be achieved using fewer random bits. (This is known as "deterministic amplification".)

It is surprising that Nisan's result is useful in this context, since A has no *a priori* space restriction. The key to this is a comment from Section 22.1 that you can apply Nisan's technique even if the space is not always limited, provided that the space is restricted at those times when the first bit in each block of t random bits is about to be consumed.

Theorem 26.1 (Nisan [30]): Suppose a probabilistic algorithm A accepts L using t random bits and 2-sided (1-sided) error, but with no restriction on space. If $r = O(t)$, then L can be accepted with 2-sided (respectively, 1-sided) error, error bound $2^{-\Omega(r)}$, and only $O(t \log r)$ random bits.

Proof:

CONSTRUCTION: Assume without loss of generality that t is twice an integral power of 3, and let $S = t/18$. Note that, between repetitions of A, we only need $O(\log r) = O(\log t) = O(\log S)$ space to record the number of acceptances and rejections so far. Model the r repetitions of A as a finite state automaton Q with 2^S states, alphabet $\{0,1\}^t$, and input of length r.

Let $k = \lceil \log_2 r \rceil \leq \log_2 S + O(1) \ll S$. Run the r repetitions of A using $G_k(x, h_1, h_2, \ldots, h_k)$ instead of truly random bits, where $x, h_1, h_2, \ldots, h_k$ are chosen randomly, uniformly, and independently.

ANALYSIS: The number of random seed bits required is $t(2k+1) = O(t \log r)$.

CORRECTNESS: According to Corollary 2.2, this algorithm would have error $2^{-\Omega(r)}$ if we were using truly random bits. By Lemma 24.2, using Nisan's generator introduces an additional error of at most 2^{-S}. So the error probability of the constructed algorithm is at most $2^{-\Omega(r)} + 2^{-S} \leq 2^{-\Omega(r)} + 2^{-\Omega(r)} = 2^{-\Omega(r)}$ because $18S = t = \Omega(r)$. $\square$

Using different techniques, Impagliazzo and Zuckerman [21] achieve this error bound using only $O(t + r)$ random bits.

Appendix A

Assignment #1

Choose as many of the following problems as you care to work on, and take each as far as you can. There are a couple that I don't know how to solve. I don't expect you to tackle all the problems, nor even all the problems that I know how to solve. I will get much more excited about promising partial progress on an open problem than about long solutions to all the routine problems. Keep your answers clear and concise. If you use references, please include citations.

1. Show that $ZPP = RP \cap \text{co}RP$, where co*RP* is the set of languages whose complements are in *RP*.

2. Design, analyze, and prove correct an efficient probabilistic algorithm that, given degree n polynomials $a(x)$, $b(x)$, and $c(x)$, determines whether $a(x)b(x) = c(x)$. What assumptions do you need to make about the coefficient ring? Contrast this with the most efficient deterministic algorithms that you know.

3. We've seen probabilistic identity testing in various guises: Freivalds' algorithm of Section 1.5, Schwartz's algorithm of Lecture 4, and your algorithm in Problem 2. Find other natural settings in which some identity or fact can be verified faster by random testing than the fastest known deterministic verifier. For each one, present the algorithm, its analysis, its correctness, and state the best known deterministic solution.

4. After Solovay and Strassen found a 1-sided error algorithm for compositeness, Adleman and Huang improved it to 0-sided. Similarly, after Aleliunas *et al.* found a 1-sided error algorithm for USTCON, Borodin *et al.* improved it to 0-sided. Can you similarly improve any of the identity tests of Problem 3 to 0-sided error algorithms whose expected time is less than the best known deterministic solutions? The ideas in Problem 1 may prove helpful to you.

5. Improve the constant 2006 in Lemma 4.8.

   (Note added after the course: the constant 89 given in the current version of Lemma 4.8 is an improvement over the original version presented in lecture, which had the constant 2006 in its place. This improvement was a result of the successful solution of this problem by three of the students in the class. It is still a challenging problem to improve that constant to 3, as one student did.)

6. If you had been willing to accept the fact that primality testing is in *ZPP*, and willing to invest the computation time to test for primality, there is a simpler algorithm and proof that establishes Theorem 4.1. Instead of testing $a^k + b^k \equiv c^k \pmod{m}$ for a random positive integer $m = O(k^2 \log^2(a+b+c))$, it suffices to do it for a random *prime* $m = O(k \log(a+b+c))$. Give the simpler algorithm and proof.

(Hint: Use Corollary 4.6.)

7. At a party, n guests check their hats. The hat check person gets them all mixed up, and gives each departing guest a hat chosen randomly and uniformly from those that remain. Fortunately from the point of job security, by this point the guests are too drunk to notice. What is the expected number of guests who go home with their own hat?

8. What is the expected time for a random walk to get from one end of an n-vertex chain to the other? (Hint: There is an elegant proof of a very accurate bound for this problem that uses Theorem 7.2.)

9. Work on any of the open problems from the lecture notes.

Appendix B

Assignment #2

Choose as many of the following problems as you care to work on, and take each as far as you can. There are some that I don't know how to solve. I don't expect you to tackle all the problems. Keep your answers clear and concise. If you use references, please include citations.

1. Investigate the effect of either of the following changes on Karloff and Raghavan's lower bound for Quicksort with linear congruential generators:
   (a) Change Q2 so that Quicksort uses the customary (unstable) partition subprocedure.
   (b) Change L1 – L4 so that the linear congruential generator has a small period.
2. Prove that there are values of m, a, and c with $m \approx n^n$ such that Quicksort with the linear congruential generator having these parameters runs in expected time $O(n \log n)$ on all input permutations.
3. Run some experiments to test the idea given in Open Problem 7 (page 38) that $\lfloor n/4 \rfloor$ successive powers of y will hash quite uniformly, when $n \leq m \leq n^2$, assuming of course that L4 holds.
4. What would the effect be on the analysis of Theorem 15.1 if the generator used were 3-way (i.e., a degree 2 polynomial in i with random coefficients) rather than 5-way? What if it were 7-way?

   Here is one program for solving this problem: Generalize Lemma 17.1 to t-way generators and derive the dependence on t of the right hand side of inequality (17.1). Now carry this dependence on to Lemmas 18.1 – 18.4.
5. Take one of the probabilistic identity-testing algorithms from Problem 3 of Assignment 1 (e.g., Freivalds' matrix multiplication algorithm), and investigate the effect of using some pseudorandom generator (e.g., linear congruential or t-way) in place of true randomness. Can you decrease the amount of randomness without a prohibitive increase in the error bound and running time? Or can you prove that there is an input for which the error bound increases prohibitively?

   (Note added after the course: Kimbrel and Sinha [personal communication] succeeded in proving that Freivalds' algorithm could be made to work with $\log n + O(1)$ random bits. See also J. Naor and M. Naor, "Small-bias probability spaces: efficient constructions and applications", *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, May 1990, 213–223.)

6. Investigate the effect of using Karloff and Raghavan's t-way generator for pseudorandom walks on graphs, where t is not very great. Here is a concrete place to start: Let p be a prime, and $a(x)$ be a degree $t-1$ polynomial whose t coefficients are chosen randomly, uniformly, and independently from the integers modulo p. Investigate the use of $a(i)$ as a generator for a pseudorandom walk on the complete graph with $p+1$ vertices; that is, at the ith step you move to the $a(i)$th adjacency of the current vertex (where adjacencies are numbered in an arbitrary order from $\{0, 1, \ldots, p-1\}$). Can you prove a polynomial upper bound on the expected time to visit each vertex at least once? Instead of the clique, you could try the n vertex cycle, but then you have the slight extra complication of extracting one bit (say, the high order bit) from $a(i)$ in order to determine whether the ith step is clockwise or counterclockwise.

7. Work on any of the open problems from the lecture notes or from Assignment 1.

Bibliography

References

[1] L. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Ann Arbor, MI, Oct. 1978. IEEE.

[2] L. M. Adleman and M.-D. A. Huang. Recognizing primes in random polynomial time. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 462–469, New York, NY, May 1987.

[3] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 132–140, New York, NY, May 1987.

[4] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science*, pages 218–223, San Juan, Puerto Rico, Oct. 1979. IEEE.

[5] E. Bach. Realistic analysis of some randomized algorithms. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 453–461, New York, NY, May 1987.

[6] C. H. Bennett and J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq co - NP^A$ with probability 1. *SIAM Journal on Computing*, 10(1):96–112, 1981.

[7] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58:113–136, 1983.

[8] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, June 1989. See also 18(6): 1283, Dec. 1989.

[9] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.

[10] J. L. Carter and M. N. Wegman. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–277, 1981.

[11] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. In *Proceedings of the Twenty*

First Annual ACM Symposium on Theory of Computing, pages 574–586, Seattle, WA, May 1989.

[12] A. K. Chandra and L. J. Stockmeyer. Alternation. In *17th Annual Symposium on Foundations of Computer Science*, pages 98–108, Houston, TX, Oct. 1976. IEEE. Preliminary Version.

[13] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–509, 1952.

[14] Y. S. Chow and H. Teicher. *Probability Theory.* Springer-Verlag, 1988.

[15] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 1–6, New York, NY, May 1987.

[16] R. Freivalds. Fast probabilistic algorithms. volume 74 of *Lecture Notes in Computer Science*, pages 57–69. Springer-Verlag, 1979.

[17] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, Dec. 1977.

[18] F. Göbel and A. A. Jagers. Random walks on graphs. *Stochastic Processes and their Applications*, 2:311–336, 1974.

[19] J. Hartmanis and J. Simon. On the power of multiplication in random access machines. In *15th Annual Symposium on Switching and Automata Theory*, pages 13–23, 1974.

[20] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, Oct. 1988.

[21] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253, Research Triangle Park, NC, Oct. 1989. IEEE.

[22] H. Jung. Relationships between probabilistic and deterministic tape complexity. In *Mathematical Foundations of Computer Science: Proceedings, 10th Symposium*, volume 118 of *Lecture Notes in Computer Science*, Štrbské Pleso, Czechoslovakia, Aug.-Sept. 1981. Springer-Verlag.

[23] H. J. Karloff and P. Raghavan. Randomized algorithms and pseudorandom numbers. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 310–321, Chicago, IL, May 1988.

[24] R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, Mar. 1987.

[25] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming.* Addison-Wesley, 1969.

[26] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, 1973.

[27] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, second edition, 1981.

[28] R. Lidl and H. Niederreiter. *Finite Fields.* Addison-Wesley, 1983.

[29] J. D. Lipson. *Elements of Algebra and Algebraic Computing.* Addison-Wesley, Reading, MA, 1981.

[30] N. Nisan. Pseudorandom generators for space-bounded computation. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 204–212, Baltimore, MD, May 1990.

[31] N. Pippenger and M. J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26(2):361–381, Apr. 1979.

[32] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12:128–138, 1980.

[33] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. In *27th Annual Symposium on Foundations of Computer Science*, pages 10–18, Toronto, Ontario, Oct. 1986. IEEE.

[34] J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[35] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[36] J. T. Schwartz. Probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, pages 701–717, 1980.

[37] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. In *29th Annual Symposium on Foundations of Computer Science*, pages 283–290, White Plains, NY, Oct. 1988. IEEE.

[38] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, Mar. 1977.

[39] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[40] S. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9:11–12, 1962.

[41] D. J. A. Welsh. Randomised algorithms. *Discrete Applied Mathematics*, 5:133–145, 1983.