EECS572
Randomness and Computation

Fall 2022 All Course Members

December 20, 2022

Abstract

This is an accumulated scribe notes taken by the course members[1] of EECS572, an advanced graduate-level theory course taught at University of Michigan by Mahdi Cheraghchi. Topics include various randomized algorithm, Randomized Complexity, Markov Chains, Random Walks, Expander Graphs, Pseudo-random Generators, Hardness v.s. Randomness.

We'll use Randomized Algorithms [MR95], Computational Complexity: A Modern Approach [AB09], and also Probability and Computing: Randomized Algorithms and Probabilistic Analysis [MU05] as our references.

This course is offered in Fall 2022, and the date on the covering page is the last updated time.

[1]Maintain by course staff via selecting/rearranging the submitted scribe notes. For a complete list of contribution, please see Appendix A.

Contents

Chapter 1

Introduction to Randomness

Lecture 1: Introduction

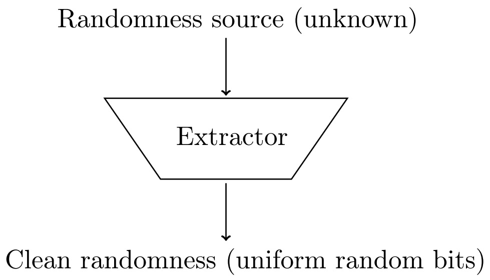
Here's some important links: course information and syllabus, Piazza, and Slack channel. If you have anything, please don't hesitate to email us via eecs572-staff@umich.edu. 30 Aug. 10:30

Lecture 2: Overview of Role of Randomness

We are going to study two things in this course: purifying randomness and simulating randomness. 1 Sep. 10:30

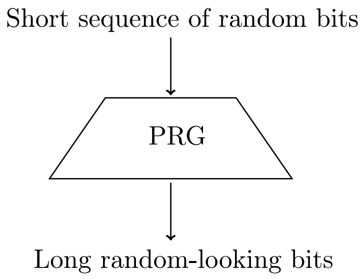
1.1 Purifying Randomness

This has to do with the situation where you have some randomness source that is unknown (think of it as entropy) and want to get clean randomness (uniform random bits) using some magic box. The box is called an extractor. This is useful for cryptography, where high quality randomness is needed.


Randomness source (unknown)
Extractor
Clean randomness (uniform random bits)

1.2 Simulating Randomness

This is something that we may have seen before, like pseudo random generator (PRG). Suppose we have generated a short sequence of random bits, we can then use the PRG to expand it and generate a long sequence of random-looking bits.



Chapter 2

Polynomial Identity Testing

2.1 Polynomial Identity

The general questions we want to ask is the following.

Problem 2.1.1 (Polynomial identity testing)**.** Given $Q(x_1, \dots, x_n)$ which is a polynomial in $x_1, \dots, x_n$, is it identically zero?

Note. For the polynomial Q to be identically zero, all coefficients of monomials in Q need to be zero.

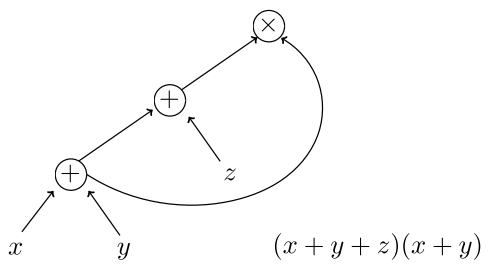
Example (Bad example)**.** In $\mathbb{F} = \{0, 1\}$, $Q(x) = x(x + 1)$ is not identically zero but evaluates to zero everywhere.[a]

[a]The definition of a polynomial being zero or not is debatable in these bad examples.

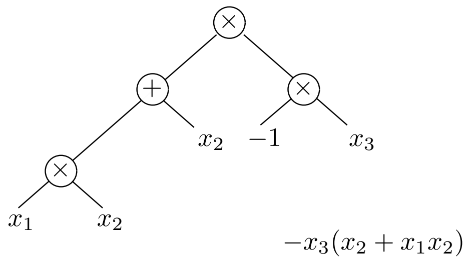
Example (Polynomial Identity)**.** Example of polynomial identity:

$$(a - b)(a + b) = a^2 - b^2.$$

Remark (Polynomial representation)**.** In the above example, we have $Q(a, b) = (a-b)(a+b)-(a^2-b^2)$. The input is a description of Q, which could be encoded as a formula (e.g. represented as a tree), or as an algebraic circuit where the representation could be a directed acyclic graph instead of tree, so that the representation is more compact.



(a) Circuit representation



(b) Tree representation

Figure 2.1: Compact representation of a polynomial

The polynomial could even be provided as a black box. Ideally we could have an algorithm that supports the black box case, which is the strongest case since it also supports trees and circuits as the special cases.

$$\begin{tikzcd} x_1, \cdots, x_n \arrow[r] & \boxed{Q} \arrow[r] & Q(x_1, \cdots, x_n) \end{tikzcd}$$

Figure 2.2: Polynomial as a black box

How do we design an algorithm that given $Q(x_1, \ldots, x_n)$ can determine if $Q(x_1, \ldots, x_n) \equiv 0$? A naive solution is to just expand the polynomial. The problem is that the expansion can increase the number of terms exponentially.

Note (Algebraic structure). In order to break down the polynomial into sum of monomials, we need to know what algebraic structure the polynomial is defined over. Examples of algebraic structures include fields (e.g. real numbers, rationals, complex numbers, finite fields like modulo arithmetic) and rings (e.g. integers).

A great idea is to just plug in random things and see if you always get zero. This is however not as foolproof as the naive solution. It may have what we call **one-sided error**. If ever get non-zero results, then returns NOT ZERO. Else returns ZERO (maybe?).

In order to bound the probability of failing, we have the following.

2.2 Schwartz-Zippel Lemma

Lemma 2.2.1 (Schwartz-Zippel lemma). Let $Q(x_1, \ldots, x_n) \in \mathbb{F}[x_1, \ldots, x_n] \setminus \{0\}$. Let $d = \deg(Q)$ be the total degree of Q.[a] Fix any finite set $S \subseteq \mathbb{F}$, pick $r_1, ..., r_n \in S$ uniformly at random and independently. Then

$$\Pr(Q(r_1, ..., r_n) = 0) \leq \frac{d}{|S|}.$$

[a]i.e. the largest degree of a monomial in Q

Using Lemma 2.2.1, the algorithm picks any set S of large enough size (e.g. $|S| \geq 2d$). Then the probability that the algorithm is incorrect (i.e. getting a one-sided error) is at most $1/2$. Probability of error can be improved by repeating the algorithm or picking a larger S.

Lecture 3: PIT and Perfect Matchings

6 Sep. 10:30

As previously seen. As we saw last lecture, the Schwartz-Zippel lemma can be applied to define an efficient, randomized algorithm for polynomial identity testing. It turns out that the algebraic structure of polynomials is broadly useful and is studied intensely in theoretical computer science under the moniker the *polynomial method*. We now see a proof of the Schwartz-Zippel lemma and an application of polynomial identity testing to perfect matchings in graphs.

We want to determine whether the polynomial $Q(x_1, \ldots, x_n)$ is identically zero. Being able to do this is enough to check polynomial identities by just subtracting the polynomials onto the same side of the equation. Our algorithm from last time amounts to evaluating Q on random inputs. The intuition is that it is unlikely for nonzero polynomials to repeatedly evaluate to zero on random inputs.

Remark. It suffices to choose $|S| = 2d$ for the Schwartz-Zippel lemma to be useful for our algorithm from last lecture.

Before seeing the proof of Lemma 2.2.1, we first need the following.

Lemma 2.2.2. Let $Q(x) \in \mathbb{F}[x]$ for $\mathbb{F}$ a field. Then, Q has at most $\deg(Q)$ distinct roots.

Proof. If $Q(r) = 0$ for some $r \in \mathbb{F}$, then $Q(x) = Q(x) - Q(r)$. If $Q(x) = \sum_{i=0}^{d} c_i x^i$, then

$$Q(x) = Q(x) - Q(r) = \sum_{i=0}^{d} c_i x^i - \sum_{i=0}^{d} c_i r^i = \sum_{i=0}^{d} c_i (x^i - r^i) = (x - r) \sum_{i=1}^{d} c_i \sum_{j=0}^{i-1} x^j r^{i-1-j}.$$

Hence, $(x-r)$ divides $Q(x)$. Using the Euclidean algorithm, we may then write $Q(x) = S(x)(x-r)$. If $r' \neq r$ is also a root of $Q(x)$, then r' must be a root of $S(x)$. This is because $ab = 0$ implies a or b is 0 for $a, b \in \mathbb{F}$. Repeating this process inductively for each root of $Q(x)$ shows that $Q(x)$ can be written as

$$Q(x) = S'(x) \prod_{\rho\colon Q(\rho)=0} (x - \rho)$$

for some $S'(x) \in \mathbb{F}[x]$. Hence, since the degree of the product of two polynomials is the sum of their degrees, $Q(x)$ has at most $\deg(Q)$ distinct roots. ■

Now we can prove the Schartz-Zippel lemma.

Proof of Lemma 2.2.1. We proceed by induction on n. If $n = 1$, then Q is a univariate polynomial, say $Q(x)$. From Lemma 2.2.2, $Q(x)$ has at most $\deg(Q) = d$ roots, hence for r chosen uniformly at random from S,

$$\Pr(Q(r) = 0) \leq \frac{|\{\rho \in \mathbb{F}\colon Q(\rho) = 0\}|}{|S|} \leq \frac{d}{|S|}.$$

Now, assume the result holds for less than n variables. We prove the claim for n variables. Write Q as a polynomial in the variable x_1 with coefficients in $\mathbb{F}[x_2, \ldots, x_n]$. Let k be the degree of Q when written in this form (the highest power of an x_1 with a nonzero coefficient). We can express Q as

$$Q(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{k} Q_i(x_2, \ldots, x_n) x_1^i.$$

We define $Q_i(x_2, \ldots, x_n) \in \mathbb{F}[x_2, \ldots, x_n]$ to be the coefficient of the degree i term of Q when written as an element of the polynomial ring $(\mathbb{F}[x_2, \ldots, x_n])[x_1]$.

Now, note that, since $\deg(Q) = d$ and $\deg(x^k) = k$, $\deg(Q_k) \leq d - k$. Note that $Q_k \neq 0$ since it is the coefficient of the highest degree term of Q when expressed as a polynomial in x_1. Hence, by our inductive hypothesis, for $r_1, \ldots r_n$ chosen independently and uniformly at random,

$$\Pr(Q_k(r_2, \ldots, r_n) = 0) \leq \frac{d-k}{|S|}. \tag{2.1}$$

Next, define $q(x_1) \coloneqq Q(x_1, r_2, \ldots, r_n)$. Note that $q(x_1) \in \mathbb{F}[x_1]$. By our base case, we know that

$$\Pr(q(r_1) = 0 \,|\, Q_k(r_2, \ldots, r_n) \neq 0) \leq \frac{k}{|S|}. \tag{2.2}$$

Observe that our condition that $Q_k(r_2, \ldots, r_n) \neq 0$ implies that $q(x_1) \neq 0$, and this is necessary to apply the inductive hypothesis. Also note that, $q(r_1) = Q(r_1, \ldots, r_n)$, so $q(r_1) = 0$ implies $Q(r_1, \ldots, r_n) = 0$.

Let A be the event that $Q_k(r_2, \ldots, r_n) = 0$. Using the law of total probability, conditioning on A and denoting the complement of A as A^c, we have

$$\begin{aligned}
\Pr(Q(r_1, \ldots, r_n) = 0) &= \Pr(Q(r_1, \ldots, r_n) = 0 \mid A) \Pr(A) + \Pr(Q(r_1, \ldots, r_n) = 0 \mid A^c) \Pr(A^c) \\
&\leq \Pr(Q(r_1, \ldots, r_n) = 0 \mid A) \frac{d-k}{|S|} + \frac{d}{|S|} \Pr(A^c) \\
&\leq 1 \cdot \frac{d-k}{|S|} + \frac{d}{|S|} \cdot 1 \\
&= \frac{d}{|S|}.
\end{aligned}$$

This first inequality follows from Equation 2.1 and Equation 2.2 and the second follows from probabilities being at most 1. The result then follows by induction. ■

2.3 Perfect Matching

A surprising application of polynomial identity testing to perfect matchings in graphs. Let bipartite graph $G = (U, V, E)$ be given, with U and V the set of n left and n right vertices, respectively, and E the set of the edges. For convenience, we label the vertices of U and V as $u_1, \dots, u_n$ and $v_1, \dots, v_n$. We now have the following definitions.

Definition. Given a graph $G = (V, E)$, we have the following.

Definition 2.3.1 (Matching). A *matching* in G is a subset of E without overlapping endpoints.

Definition 2.3.2 (Perfect matching). A matching is called *perfect* if every vertex in G is an endpoint of some edge in the matching.

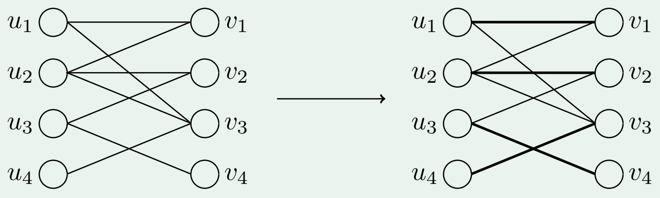


Figure 2.3: A perfect matching in a bipartite graph

In order to apply polynomial identity testing to this problem, we need a polynomial. We will obtain a polynomial by taking the determinant of a special matrix associated with G, which is the so-called Edmonds matrix.

Definition 2.3.3 (Edmonds matrix). The *Edmonds matrix* of a given graph $G = (V, E)$, is defined entry-wise as

$$A_{ij} = \begin{cases} x_{ij}, & (u_i, v_j) \in E; \\ 0, & \text{otherwise.} \end{cases}$$

Example. The Edmonds matrix of the graph given in Figure 2.3 is

$$\begin{pmatrix} x_{11} & 0 & x_{13} & 0 \\ x_{21} & x_{22} & x_{23} & 0 \\ 0 & x_{32} & 0 & x_{34} \\ 0 & 0 & x_{42} & 0 \end{pmatrix}.$$

Remark (Determinant). Formally, the *determinant* of a matrix A is defined as

$$\det(A) := \sum_{\pi \in S_n} \operatorname{sgn}(\pi) A_{1\pi(1)} A_{2\pi(2)} \cdots A_{n\pi(n)},$$

where S_n is the group of permutations of $[n]$ and sgn of a permutation is the parity of the number of transpositions—permutations swapping only two numbers — it can be expressed as a product of -1, with times equal to the number of swapping.

Note. Note that the determinant of the Edmonds matrix is a polynomial in the variables x_{ij}. This is the polynomial we seek.

Each term in the summation of $\det(A)$ involves the product of n matrix entries. If any is zero, the whole term is zero. If A_{ij} is nonzero, then $(u_i, v_j) \in E$. Each $\pi \in S_n$ defines a bijection from $[n]$ to $[n]$ corresponding to a set of potential edges from U to V. If the term in the summation corresponding to π is nonzero, then $(u_i, v_{\pi(i)}) \in E$ for all $i \in [n]$. Since π is bijective, this is a perfect matching. Hence, there is a nonzero term in the summation representing $\det(A)$ if and only if G has a perfect matching. Since each possible nonzero term is an expression in different variables, there cannot be any cancellation amongst the terms. Hence, we have that G has a perfect matching if and only $\det(A) \neq 0$. We may then apply our polynomial identity checking algorithm to determine whether G has a perfect matching.

Remark. There is a polynomial time algorithm for computing determinants. Hence, we have a polynomial time randomized algorithm for concluding whether a bipartite graph has a perfect matching.

Chapter 3

Primality Testing

Lecture 4: Primality Testing Algorithms

3.1 The Prime Numbers

8 Sep. 10:30

Consider the following well-known definition of a prime number.

Definition 3.1.1 (Prime number). A *prime number* is a positive integer $N > 1$ which has no divisors other than 1 and itself.

Our goal is to devise an *efficient* algorithm which answers the so-called primality testing problem.

Problem 3.1.1 (Primality testing). Given a positive integer $N \in \mathbb{Z}^+$, is N prime?

Note (Efficient algorithm). By *efficient algorithm*, we're referring to an algorithm which has polynomial runtime in its input. In this case, the length of the input is the *bit length* of N, and therefore, an efficient algorithm will be one with runtime $O(\log N)$.

3.1.1 Trivial Primality Test

A naive and trivial (but inefficient) algorithm to test for primality is based on Theorem 3.1.1.

Theorem 3.1.1. A positive integer N is prime if and only if $k \nmid N$ for all integers k such that $2 \leq k \leq \sqrt{N}$

Proof. The forward direction is trivial: if N is prime, the only divisors are 1 and N, and since $\sqrt{N} < N$ for all $N > 1$, all integers k for which $2 \leq k \leq \sqrt{N}$ do not divide N.

Contrarily, assume $k \nmid N$ for all integers k such that $2 \leq k \leq \sqrt{N}$, and suppose N is composite. Then N has some divisor m such that $\sqrt{N} < m < N$. There must then also exist an integer $q > 1$ such that $mq = N$. Then $mq = N = \sqrt{N} \cdot \sqrt{N} < m\sqrt{N}$, thus $q < \sqrt{N}$, meaning N has a divisor q such that $2 \leq q \leq \sqrt{N}$. We reach a contradiction and conclude that N must be prime. ■

Using this, we devise the following trivial algorithm for primality testing.

Algorithm 3.1: Trivial Primality Test

```
Data: Integer $N > 1$
Result: True if $N$ is prime, False if not
1 for $k = 2 \dots \lfloor\sqrt{N}\rfloor$ do
2     if $k|N$ then
3         return False
4 return True
```

Remark. Algorithm 3.1 is correct from Theorem 3.1.1, with running time being $O(\sqrt{N}) \gg O(\log N)$, which is not our desired efficiency.

3.2 Fermat's Test

First, recall the following definition of modular congruence.

Definition 3.2.1 (Modular Congruence). If $p \mid (x-y)$, we say $x \equiv y (\bmod p)$, i.e., x and y are *modular congruence.*

Using this, we can introduce Fermat's little theorem, which is needed for the Miller-Rabin algorithm.

Theorem 3.2.1 (Fermat's Little Theorem). If p is prime, $p \nmid a$, then $a^{p-1} \equiv 1 (\mathrm{mod} p)$.

Example. Consider $a = 2$ and $p = 7$, then $2^{7-1} = 2^6 = 64 = (7 \cdot 9 + 1) \equiv 1 (\mathrm{mod} 7)$.

Using this, we can introduce Fermat's test to assess a positive integer's primality.

Algorithm 3.2: Fermat's Test

```
Data: Integer $N > 1$
Result: True if $N$ is prime, False if not
1 $a \leftarrow$ uniform(1, $N$)        // Uniform random integer from $[1, N)$
2
3 if $\gcd(a, N) \neq 1$ then
4     return False
5 if $a^{N-1} \not\equiv 1 (\mathrm{mod} N)$ then
6     return False
7 return True
```

3.2.1 Time Complexity Analysis

We now show that Algorithm 3.2 is efficient. Firstly, retrieving a uniformly random integer can be done in polynomial time (in the bit length of the input). To further show that Algorithm 3.2 runs in polynomial time, we must show that line 3 and line 5 each takes polynomial time.

Note that $\gcd(a, N)$ can be computed efficiently recursively using the Euclidean algorithm.[1] Next, assuming that A, B, and N are all sufficiently large integers, we compute $A^B \bmod N$ via *fast exponentiation* as follows.

Suppose $B = 2^b$, then we have

$$A^2 (\mathrm{mod} N), \quad A^4 \equiv (A^2)^2 (\mathrm{mod} N), \quad A^8 \equiv (A^4)^2 (\mathrm{mod} N).$$

After b steps, we get $A^B \equiv A^{2^b} (\bmod N)$. In general, $B = \sum_{i \in I} 2^i$ (binary expansion), where $i \in I$ if and only if the ith bit of the binary representation of B (starting from the right, counting from 0) is 1. Then,

$$A^B = A^{\sum_{i \in I} 2^i} = \prod_{i \in I} A^{2^i}.$$

Example. $B = 19 = 16 + 2 + 1 = 2^4 + 2^1 + 2^0$, so $A^{19} = A^{2^4} A^{2^1} A^{2^0}$

Now, we have a set of A^n's where each n can be written as $n = 2^i$ for some non-negative integer i. We apply this method for computing A^{2^b}, multiply the results, and take modulo N to get $A^B (\mathrm{mod} N)$.

[1] It can be summarized as $\gcd(a, b) = \gcd(b, a \bmod b)$ assuming $a \geq b$. Since $N > a$, this takes at most $O(\log N)$ time, making it satisfactory for efficiency.

Remark. Fast exponential method is linear in bit length.

Proof. In particular, the runtime is dependent on the bit lengths of B for exponentiation, A for multiplication, and N for modulus. Hence, computing $a^{N-1}(\bmod N)$ on line 5 is polynomial in bit length. ⊛

Combining all arguments, we see that Algorithm 3.2 is a polynomial time algorithm.

3.2.2 Correctness Analysis

If N is prime, then the algorithm will return True, i.e., report N as a prime. If N is composite, the algorithm *might* return False. To see why the latter is merely a possibility, we have the following.

Definition. Let $N > 1$ an integer, we have the following based on Fermat's test.

Definition 3.2.2 (Fermat witness). If $a^{N-1} \not\equiv 1(\bmod N)$, then a is a *Fermat witness.*[a]

[a]Since it witnesses the non-primality of N.

Definition 3.2.3 (Fermat liar). If $a^{N-1} \equiv 1(\bmod N)$, then a is a *Fermat liar.*

Example. Let $N = 221 = 13 \times 17$. $a = 38$ is a Fermat liar, while $a = 24$ is a Fermat witness.

Proof. Since $a^{N-1} = 38^{220} \equiv 1(\bmod 221)$, so $a = 38$ is a Fermat liar for N being composite.

On the contrary, repeating with $a = 24$ gives us $a^{N-1} = 24^{220} \not\equiv 1(\bmod 221)$, making 24 a Fermat witness for N being composite.

Note. We must note that while 24 tells us that 221 is composite, it doesn't tell us *why*, e.g. we still don't know the factors of 221 (unless we figure out via gcd).

⊛

Since Fermat witness exist, Algorithm 3.2 is not guaranteed to return correctly if N is composite. The crucial question is the following.

Problem 3.2.1. How frequent the witness appears?

Lemma 3.2.1. Let $\mathbb{Z}_N^* := \{1 \le a < N : \gcd(a, N) = 1\}$ and $F_N := \{a \in \mathbb{Z}_N^* : a^{N-1} \equiv 1(\bmod N)\}$, we have $|F_N| \mid |\mathbb{Z}_N^*|$.

Proof. The proof requires a bit of group theory. Essentially, it can be shown that $\mathbb{Z}_N^*$ is a group under multiplication, and F_N is a subgroup, and the sizes of subgroups must divide the size of the original group from Lagrange's theorem. ■

Corollary 3.2.1. Either $F_N = \mathbb{Z}_N^*$, or $|F_N| \le \frac{1}{2} |\mathbb{Z}_N^*|$

To summarize, if N is prime, Algorithm 3.2 returns True always; if N is composite **and** there is a Fermat witness, then by Corollary 3.2.1, Algorithm 3.2 returns False with probability at least 1/2.

What if *everything* is a Fermat liar? In this case, Algorithm 3.2 cannot predict that composite number to truly be composite. In deed, the so-called Carmichael numbers do not have Fermat witness, in which case we have $F_N = \mathbb{Z}_N^*$.

Definition 3.2.4 (Carmichael number). A *Carmichael number* is a composite integer N for which every a is a Fermat liar.

Example. $561 = 3 \times 11 \times 17$, $1105 = 5 \times 13 \times 17$, $1729 = 7 \times 13 \times 19$[a] are all Carmichael numbers.

[a] This is a taxicab number!

Remark. There are infinitely many Carmichael numbers, but we have some control on the number of Carmichael numbers in a given interval. Let $C(x)$ denotes the number of Carmichael numbers in $\{1, \ldots, x\}$, then we know

$$\sqrt[3]{x} < C(x) < \frac{x}{\exp\left(\frac{k \cdot \log x \cdot \log\log\log x}{\log\log x}\right)}.$$

Furthermore, the upper-bound is conjectured to be tight with k being a constant.

3.3 Miller-Rabin Algorithm

Algorithm 3.2 is *good enough* for some random input N as the chance of being in $C(x)$ is quite low. But we can actually do better since **we can recognize Carmichael numbers** with a more sophisticated algorithm called Miller-Rabin algorithm.

Before we see the algorithm, we first see the following proposition.

Proposition 3.3.1. Let p be an odd prime, write $p - 1 = q \cdot 2^k$, $k \geq 0$, q odd. Let $a > 0$ be any integer not divisible by p (so $a^{2^k \cdot q} \equiv 1(\bmod p)$). Then either:

(a) $a^q \equiv 1(\bmod p)$, or

(b) one of $a^q, a^{2q}, a^{4q}, \ldots, a^{2^{k-1}q} \equiv -1(\bmod p)$.

Note. In other words, at some point in the exponentiation, we must start with 1 or reach -1 (and hence reach 1 immediately after due to squaring) before we compute $a^{2^k \cdot q}(\bmod p)$.

But indeed, even with this method, a witness exists.

Remark (Miller-Rabin witness)**.** Given odd N, call a such that $\gcd(a, N) = 1$, a *Miller-Rabin witness* for N if the conditions of proposition are false, i.e., for $N - 1 = 2^k \cdot q$,

(a) $a^q \not\equiv 1(\bmod N)$, and

(b) $a^{2^i q} \not\equiv -1(\bmod N) \quad \forall i = 0, 1, \ldots, k-1$

Example. $N = 561$ (a Carmichael number) is a Miller-Rabin witness, i.e., it fails to satisfy Proposition 3.3.1, hence it's not a prime.

Proof. Since $N - 1 = 560 = 2^4 \cdot 35$, $a = 2, q = 35$.

$$\begin{aligned}
2^{35} &\equiv 263 \not\equiv 1 \quad (\bmod 561)\\
2^{70} &\equiv 166 \not\equiv -1(\bmod 561)\\
2^{140} &\equiv 67 \quad \not\equiv -1(\bmod 561)\\
2^{280} &\equiv 1 \quad \not\equiv -1(\bmod 561)\\
2^{560} &\equiv 1 \quad \not\equiv -1(\bmod 561)
\end{aligned}$$

We transition to 1 without reaching -1, also, $a^q = a^{35} \not\equiv 1$, hence Proposition 3.3.1 is not satisfied (both conditions are false). ⊛

Lecture 5: The Probabilistic Method

Let's first proof Proposition 3.3.1. 13 Sep. 10:30

Proof of Proposition 3.3.1. By Fermat's little theorem, $a^{2^k} q \equiv 1 \pmod p$. List $a^q, a^{2q} \dots, a^{2^{k-1}} q$. If $a^q \equiv 1 \pmod p$ we are done. Otherwise, there is some b such that $b \not\equiv 1 \pmod p$ but $b^2 \equiv 1 \pmod p$, i.e. $b \not\equiv 1 \pmod p$ is a square root of $1 \pmod p$. Since p is prime, it is a fact (which we won't prove but follows from $(Z/Z_p)^\times$ being cyclic) that any such b must be -1. ■

Now, we see the formal algorithm which utilize Proposition 3.3.1.

Algorithm 3.3: Miller-Rabin Test

```
Data: Integer $N > 1$
Result: True if $N$ is prime, False if not
$a \leftarrow$ uniform(2, $N$)            // Uniform random integer from $[2, N)$

if $\gcd(a, N) \neq 1$ then
    return False
if $a$ is a Miller-Rabin witness then
    return False
return True
```

We see that If N is prime, then Algorithm 3.3 is always correct guaranteed by Proposition 3.3.1; if not, it is correct with probability at least $3/4$.

3.3.1 Deterministic Primality Test

Now, the problem is that can we make this deterministic, or a more fancy term, *derandomize* it? Assuming generalized Riemann hypothesis, then enumerating $a = 2, \dots, O(\log^2 N)$ will suffice to find a witness.

Remark (Agrawal-Kayal-Saxema Test)**.** The *Agrawal-Kayal-Saxema (AKS) Test* is a fully deterministic algorithm based on Freshman's dream: $N \geq 2$, $\gcd(N, a) = 1$, then N is prime iff

$$(x + a)^N \equiv x^N + a^N \pmod N$$

as a polynomial identity in x. AKS derandomizes all other randomized algorithms we saw for PIT in this special case.

Chapter 4

The Probabilistic Method

A nice quote about combinatorial is the following:

If a combinatorial object is large enough, there's no way to avoid structure.

Hence, we can exploit some kind of structure by a probabilistic argument, which we call it the *probabilistic method*. THis is introduced independently by Claude Shannon and Paul Erdős circa 1947. The goal is simple, we want to show a combinatorial object with some (interesting) properties exists. The general strategy is the following.

(a) Construct the object randomly.

(b) Show that it has the desired properties with probability > 0.

Let's first see some examples to illustrate it.

4.1 Ramsey Theory

We start from a toy example.

Claim. For any 6 guests at a party, either 3 are mutually friends or 3 are all strangers.

Remark. This is not true for 5 guests; consider the graph C_5.

This suggests we look into the following quantity.

Definition 4.1.1 (Ramsey number)**.** The k^{th} *Ramsey number* R_k is the smallest $n \in \mathbb{N}$ such that any 2-coloring of the edges of the complete graph K_n with n vertices has a monochromatic k-clique.

Equivalently, any graph on R_k vertices has either a clique or independent set of size k. We know something about R_k.

Example. We know that $R_3 = 6$, $R_4 = 18$, $43 \leq R_5 \leq 48$, $102 \leq R_6 \leq 165$ and $205 \leq R_7 \leq 540$. For general k, we have

$$k 2^{k/2} \leq R_k \leq \frac{4^k}{\sqrt{k}}.$$

Typically, upper bounds are proven with inductive arguments. Lower bounds can be proven if there is a graph with neither a k-clique nor a k-independent set. We can use random graphs to show existence, which utilizes the idea of probabilistic method.

Theorem 4.1.1 (Ramsey theorem)**.** For every $k \geq 3$, $R_k > 2^{k/2}$.

Proof. Let G be a graph on n vertices, where we independently choose to include an edge between any two vertices with probability $\frac{1}{2}$. Pick a set $S \subset V(G)$ of size k. The probability that S induces a clique is $2^{-\binom{k}{2}}$, and the probability that S induces an independent set is also $2^{-\binom{k}{2}}$. If we'll call these *bad events* B, then we have

$$\Pr(G \text{ contains size } k \text{ clique or independent set}) \leq (\#\text{ possible } S) \cdot \Pr(B) = \binom{n}{k} \frac{2}{2^{\binom{k}{2}}} \leq \frac{n^k}{k!} 2^{1-\binom{k}{2}},$$

where the first inequality is from union bound. If we choose $n = 2^{k/2}$, then

$$\Pr(G \text{ contains size } k \text{ clique or independent set}) \leq \frac{1}{k!} 2^{\frac{k^2}{2}+1-\frac{k^2-2}{2}} = \frac{1}{k!} 2^{k+2/2} < 1$$

for $k \geq 3$. Therefore, $\Pr(G$ contains size k clique or independent set$) > 0$, i.e., there is a graph with $n = 2^{k/2}$ vertices that avoids both cliques and independent sets of size k. ■

Lecture 6: The Probabilistic Method

As previously seen. Sometimes we want to show something exists. 15 Sep. 10:30

4.2 Max-Cut

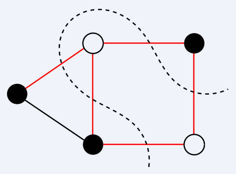
Let's introduce some definitions.

Definition 4.2.1 (Cut). Given a graph $G = (V, E)$, a *cut* is any set S such that $\varnothing \neq S \subseteq V$.

Remark (Cut size). The cut size is measured as how many edges are going between S and $V \setminus S$. Formally, consider the edge set $E(S, V \setminus S) = E(S, \overline{S}) = \{(i, j) \in E : i \in S, j \in \overline{S}\}$, then the size of the cut is the size of the edge set $|E(S, \overline{S})|$.

Problem 4.2.1 (Max-cut). Given a graph $G = (V, E)$, the *max-cut* problem is to find the maximum cut $S \subseteq V$, i.e., the maximum $|E(S, \overline{S})|$.

Example. Given that $S = \{\text{black nodes}\}$, $V \setminus S = \{\text{white nodes}\}$, the cut size of S is 5.



Note (Min-cut problem). Find the minimum cut of minimum size. Equivalent to max-flow problem, and easier than max-cut problem.

Remark. This is an NP-hard problem! This is showed by Karps famous 21 NP-Hard Problem list.

4.2.1 Approximation and Hardness

Since it is a NP-Hard problem, what could be our best approach? The answer is to consider approximation algorithm. Approximation algorithm is expected to find a cut that is within a factor of α of the max-cut in polynomial time.

Remark (Approximation on max-cut). There are some result on max-cut using approximation algorithm, with some hardness shown below.

- The best possible α is 0.878 from Goemans-Williamson [GW95].
- Hardness: $\alpha = \frac{16}{17} \approx 0.941$ is NP-Hard [Hås01].
- Assume the unique games conjecture, $\alpha = 0.878$ is the best to hope for [Kho+07].

4.2.2 Random Max-Cut

Using the probabilistic method, for each vertex v in the graph, include v in S with probability 1/2 independently. We now see that this indeed is a 1/2-approximation algorithm. Take any edge $e = (u, v)$, then we have

$$\Pr(e \text{ is a cut}) = \Pr\big((u \in S, v \in \overline{S}) \cup (u \in \overline{S}, v \in S)\big) = \frac{1}{2}.$$

Define

$$X_e = \begin{cases} 1, & \text{if } e \text{ is a cut edge;} \\ 0, & \text{otherwise,} \end{cases}$$

and we know $\mathbb{E}[X_e] = 1/2$ from the probability above. Hence, by denote $X = \sum_{e \in E} X_e$, i.e., the number of cut edge w.r.t. the cut S, then we have

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{e \in E} X_e\right] = \sum_{e \in E} \mathbb{E}\left[X_e\right] = \frac{1}{2} \cdot |E|,$$

meaning that cut S is within an approximation factor of 2 for max-cut.[1]

Algorithm 4.1: Random Max-Cut

Data: A graph $G = (V, E)$, T
Result: A 1/2-approximation cut S

```
1 $S^* \leftarrow \varnothing$
2 for $t = 1, \ldots, T$ do
3     $S \leftarrow \varnothing$
4     for $v_i \in V$ do
5         if uniform(0, 1)$< 1/2$ then                 // sample $v$ in $S$ w.p. half
6             $S \leftarrow S \cup \{v_i\}$
7     if $|S^*| < |S|$ then                             // keep the largest cut
8         $S^* \leftarrow S$
9 return $S^*$
```

4.2.3 Analysis

Remark (Success probability)**.** However, this does not guarantee that we can have a cut with size of $\frac{1}{2}|E|$, because this what we have shown is just in expectation, and running a few times of this algorithm gives us a good chance to succeed.

We can make sure the algorithm outputs the cut of size $(1 - \epsilon) \cdot \frac{1}{2} \cdot |E|$ for some ϵ after a few trials using Markov inequality.

Theorem 4.2.1 (Markov inequality)**.** Given positive random variable $y \geq 0$ such that $\mathbb{E}[y] \leq t$, then $\forall \alpha > 1$,

$$\Pr(y \leq \alpha \cdot t) \leq \frac{1}{\alpha}.$$

Proof. If not, then

$$\mathbb{E}[y] = \Pr(y < \alpha \cdot t) \cdot \mathbb{E}[y \mid y < \alpha \cdot t] + \Pr(y \geq \alpha \cdot t) \cdot \mathbb{E}[y \mid y \geq \alpha \cdot t].$$

And we know $\mathbb{E}[y \mid y < \alpha \cdot t] \geq 0$, $\Pr(y \geq \alpha \cdot t) > 1/\alpha$ because of our assumption, and $\mathbb{E}[y \mid y \geq$

[1]This is because the maximum possible max-cut is just bounded above by $|E|$.

$\alpha \cdot t] \geq \alpha \cdot t$, so we can get $\mathbb{E}[y] > t$. Thus, we got a contradiction, proving the result. ■

We apply Markov inequality on the number of edges **not** in the cut-edge set, we have $\mathbb{E}[y] = \frac{1}{2} \cdot |E|$, then

$$\Pr\left(y \geq (1+\epsilon) \cdot \frac{|E|}{2}\right) \leq \frac{1}{1+\epsilon} = 1 - \epsilon \pm o(\epsilon^2),$$

which is the probability of failure (not getting a 2-approximation solution). But notice that this is just one trial, if we to this for T times, we have

$$\Pr(\text{Fail after } T \text{ independent trials}) \approx (1-\epsilon)^T,$$

then $T = O(\frac{1}{\epsilon})$ brings this down to less than $\frac{1}{100}$.

Remark (Forcing success)**.** If we really want to have $|S| \geq |E|/2$, we can set $\epsilon = n^{-2}$, then the probability of deviating from $\epsilon \cdot |E|/2 < 1$ becomes 0, i.e., we must get a 2-approximation max-cut.

4.3 Max-Independent Set

Definition 4.3.1 (Independent set)**.** Given a graph $G = (V, E)$, the set $S \subseteq V$ is *independent* if there are no edges between nodes in S.

Now, we're interested in the following problem.

Problem 4.3.1 (Maximum independent set)**.** Given a graph $G = (V, E)$, find the size of the largest independent set?

Remark (Hardness)**.** Problem 4.3.1 is a NP-Hard problem [Kar72], and it's also NP-hard to approximate within factor

$$\frac{1}{n^{1-\epsilon}}$$

for all $\epsilon > 0$. The known achievable approximation ratio is $1/n$ (or $2/n$).

4.3.1 Independent Set on Regular Graph

Suppose graph G is d-regular, it is possible to find S such that

$$|S| = \frac{|V|}{d+1} \geq \frac{\mathsf{OPT}}{d+1}$$

where OPT stands for optimal (largest) independent set size. We see the following.

Theorem 4.3.1. There's a randomized algorithm for Problem 4.3.1 achieves $\frac{1}{d+1}$-approximation ratio given the graph is d-regular.

Proof. Again, in the spirit of probabilistic method, we show that G contains a set $S \subseteq V$ such that S is an independent set and

$$|S| \geq \sum_{v \in V} \frac{1}{1 + \deg(V)}.$$

Assign a random weight $w_v \in [0, 1]$ uniformly and independent, and define $v \in V$ is a local minimum if $w_v < w_u$ for all u adjacent to v.

Note. The set of local minimum forms an independent set. Also,

$$\Pr(v \text{ is local minimum}) = \frac{1}{1+\deg(V)}$$

because of symmetry and independence.

As before, we dance an indicator variable X_v for all $v \in V$ such that

$$X_v = \begin{cases} 1, & \text{if } v \text{ is a local minimum;} \\ 0, & \text{otherwise,} \end{cases}$$

we can then show $\mathbb{E}[X_v] = \frac{1}{1+\deg(V)}$, $|S| = \sum_v X_v$, so

$$\mathbb{E}[|S|] = \sum_{v\in V} \mathbb{E}\left[X_v\right] = \sum_{v\in V} \frac{1}{1+\deg(V)} = \frac{|V|}{1+d} \geq \frac{\mathsf{OPT}}{d+1}.$$

■

Lecture 7: Max-3SAT and Derandomized

4.4 Max-3SAT

20 Sep. 10:30

We first look at the so-called MAX-3SAT problem and using derandomization by conditional expectations to create deterministic algorithms out of randomized algorithms.

4.4.1 Boolean Satisfiability Problem

Let's start with a definition.

Definition 4.4.1 (Conjunctive normal form)**.** A *conjunctive normal form* (CNF) formula is a conjunction φ of one or more boolean clauses on $x_1, x_2, \ldots, x_n$ with boolean valued $\{0, 1\}$. Explicitly, φ is in CNF if

$$\varphi(x_1, x_2, \ldots, x_n) = \text{clause}_1 \land \text{clause}_2 \land \text{clause}_3 \land \ldots \land \text{clause}_k$$

where each clause is an or of literals, with a literal being some x_i or its negation $\neg x_i$.

Note (Disjunctive normal form)**.** For every conjunctive normal form, there is an equivalent way to write it in the so-called *disjunctive normal form.*

Definition 4.4.2 (k-CNF)**.** A *k-CNF formula* is a CNF formula in which each clause has exactly k literals from distinct variables.

Example (3-CNF)**.** A 3-CNF formula can be like

$$\varphi = (x_1 \lor \neg x_2 \lor x_4) \land (\neg x_3 \lor x_4 \lor x_5) \land (\neg x_1 \lor \neg x_5 \lor \neg x_6).$$

Now, the boolean satisfiability problem asks the following question: given a k-CNF formula φ, does an assignment exist such that φ is evaluated as true? Formally, we have Problem 4.4.1.

Problem 4.4.1 (k-SAT)**.** Given a k-CNF formula φ, the *k-SAT* problem asks whether φ is satisfiable.

4.4.2 Random MAX-3SAT

Instead of looking at a general k, we consider a simple but also hard enough case when $k = 3$. Specifically, we ask the following question.

Problem 4.4.2 (MAX-3SAT)**.** Given a 3-CNF formula φ and ℓ, the *MAX-3SAT* problem asks is there an assignment of variables such that it satisfies at least ℓ clauses?

Remark (Hardness)**.** Both Problem 4.4.1 and Problem 4.4.2 are proven to be NP-complete, while the former is proven by the Cook-Levin theorem.

Note (Variation)**.** The *search version* of Problem 4.4.2 is defined as follows. Given a 3-CNF formula φ, find an assignment which satisfies as many clauses as possible. We define OPT to be the number of clauses satisfied by an optimum assignment.

Following the definition of OPT, the *approximation version* of Problem 4.4.2 is defined as follows. Given a 3-CNF formula φ and an approximate factor α, find an assignment that satisfies at least $\alpha \cdot \mathsf{OPT}$ clauses.

By the PCP theorem, it has been shown that α cannot be arbitrarily close to 1 for the MAX-3SAT. In other words, $\exists \delta < 1$ such that $\forall \alpha \geq \delta$, the max 3SAT is NP-Hard.

A surprising result is that by randomly assigning x_i, we achieve the best we can do in expectation.

Lemma 4.4.1. For all 3-CNF φ, there exists an assignment that satisfies at least 7/8 of clauses.

Proof. Each clause is satisfied by all but exactly 1 assignment, the one where all the literals in the clause evaluate to false. Imagine we have a uniformly random assignment of variables, and let X_i be a random variable such that

$$X_i = \begin{cases} 1, & \text{if } i^{st} \text{ clause is satisfied;} \\ 0, & \text{otherwise.} \end{cases}$$

Since each clause has 8 different possibilities (2^3), and there is only one situation where the clause is not satisfied, each clause has a probability of 7/8 of being satisfied. Therefore:

$$\Pr(X_i = 1) = \frac{7}{8} = \mathbb{E}[X_i].$$

Let X be a random variable that corresponds to the number of satisfied clauses, i.e., $X = \sum_{i=1}^{n} X_i$. By the linearity of expectation, we have

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}(X_i) = \frac{7}{8} \cdot m \geq \frac{7}{8}\mathsf{OPT}$$

This shows that if we had an algorithm that randomly picks assignments of variables and checks to see how many clauses are satisfied, this would be a randomized algorithm that achieves $\alpha = \frac{7}{8}$. If we were then to repeat the algorithm a polynomial number of times, we could show that there is a good chance to find such an assignment using Markov's inequality. ■

Algorithmically, we have the following.

Algorithm 4.2: Deterministic MAX-3SAT

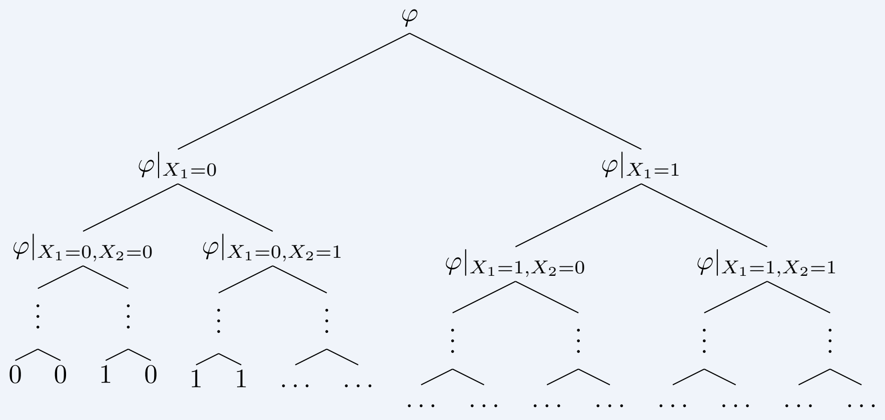
```
Data: A 3-CNF $\varphi(x_1, \dots, x_n)$
Result: A 7/8-approximation assignment $\{x_i\}_{i=1}^n$ (in expectation)
1 for $i = 1, \dots, n$ do
2     $x_i \leftarrow$ uniform($\{0, 1\}$)                  // random assignments
3 return $\{x_i\}_{i=1}^n$
```

Remark (Beyond $\alpha = 7/8$)**.** Håstad [Hås01] showed that in fact it is not possible to do better than $\alpha = \frac{7}{8}$. In other words, $\forall \epsilon > 0$, obtaining $\alpha = \frac{7}{8} + \epsilon$ is NP-Hard.

4.4.3 Derandomization via Conditional Expectations

This method can be applied to a variety of results in the probabilistic method. The high level idea is to fix variables one after another and evaluate how the expectation changes as the variables are chosen.

Example (Expectation Tree of MAX-3SAT). Consider a 3-CNF formula φ and fix a value for X_1. Then, we would be able to compute a new expectation for the number of satisfiable clauses given the assignment of just X_1. By doing this, we can build a tree with φ at the root, and each choice of X_1 would be a child and iterate through all different possible assignments. Each of the leaf nodes would then correspond to a complete evaluation of φ.



For a node ψ in the tree, and denote X_ψ as the number of clauses satisfied in ψ given a random assignment (of unassigned variables). Let ψ be some node at level i, and let $\psi_L = \psi|_{X_{i+1}=0}$ and $\psi_R = \psi|_{X_{i+1}=1}$. Then for each node, we can split the expectation into separate events for the left and right children

$$\mathbb{E}[X_\psi] = \Pr(x_{i+1}=0)\cdot\mathbb{E}[X_{\psi_L}] + \Pr(x_{i+1}=1)\cdot\mathbb{E}[X_{\psi_R}] = \frac{1}{2}(\mathbb{E}[X_L]+\mathbb{E}[X_R]).$$

In other words, the expectation at a given node is equal to the average of the expectation of the two children, which means that either $\mathbb{E}[X_L]$ or $\mathbb{E}[X_R] \geq \mathbb{E}[X_\psi]$. Now, note that

(a) $\mathbb{E}[X_\psi]$ can be computed exactly in linear time by iterating over each clause and computing the individual expectation (similar to the randomized version).

(b) The expectation of the root node, $\mathbb{E}[X_\psi] \geq \frac{7}{8}\cdot m$ (from analysis of randomized algorithm).

This means we can define our algorithm as follows: go down the tree from the root while satisfying $\mathbb{E}[X_\psi] \geq \frac{7}{8}\cdot m$ throughout, until we get to a leaf, which is the desired assignment. We can do this because we can always choose between the two children of a node and one of the children is guaranteed to have an expectation greater than or equal to $\frac{7}{8}\cdot m$ as we have shown.

Algorithm 4.3: Deterministic MAX-3SAT

```
Data: A 3-CNF $\varphi(x_1,\dots,x_n)$
Result: A 7/8-approximation assignment $\{x_i\}_{i=1}^n$
$Y = X_\varphi$                                  // set a random variable
for $i = 1,\dots,n$ do
    if $\mathbb{E}[Y \mid x_i = 0] \geq 7/8$ then
        $x_i = 0$
        $Y \leftarrow \mathbb{E}[Y \mid x_i = 0]$
    else
        $x_i = 1$
        $Y \leftarrow \mathbb{E}[Y \mid x_i = 1]$
return $\{x_i\}_{i=1}^n$
```

Note. There may be some steps where either choice of variable assignment maintains a large enough expectation. It doesn't matter which is chosen, obviously.

4.5 Lovász Local Lemma

The Lovász local lemma is a useful tool for showing existence when the probabilistic method isn't straightforward. Before we see it formally, we first see why the naive approach doesn't work. The most naive way we bound probability events are the so-called union bound.

Lemma 4.5.1 (Union bound). Let E_i, $i = 1, 2, \ldots, n$ be events in a probability space. Then

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \Pr(E_i).$$

Note. This extends to infinite sum, but needs some formal measure theory notion.

Intuition. It relies on *low dependence* between *bad events*. Suppose we have n bad events $E_1, \ldots, E_n$, we're interested in showing that with probability greater than 0, none of the bad events occur. If we don't know anything about E_i, we can apply the union bound, which gives us

$$\Pr\left(\bigcup_{i=1}^{n} E_i\right) \leq \sum_{i=1}^{n} \Pr(E_i) \leq p \cdot n,$$

if $\forall i, \Pr(E_i) \leq p$. This is only nontrivial if $p < 1/n$. In general this is the easiest to do, but it is often wasteful.

Example. Let's flip 10 coins, and the *bad event* is that some coin lands tails. We can easily calculate that the probability that no bad events happen is $1 - 2^{-10} < 1$. However, using union bound, we get:

$$\Pr(E_i) = \frac{1}{2} \Rightarrow \Pr(\text{bad event}) \leq \sum_{i=1}^{10} \Pr(E_i) = 5$$

This result is not incorrect, but it is not useful to us at all.

Lecture 8: Lovász Local Lemma

4.5.1 Independence

22 Sep. 10:30

To explain the Lovász local lemma, we need to explicitly define independence first. We can start with the case involving only two events A and B.

Definition 4.5.1 (Mutual independence). Given events A, B in a probability space, we say A and B are *mutually independent* if $\Pr(A \cap B) = \Pr(A)\Pr(B)$.

How do we generalize independence to multiple events? In other words, we want to define when will A be *independent* to the series of events, $B_1, B_2, \ldots, B_k$. Suppose we just extended Definition 4.5.1 to fit more than two events. In other words, we can say $A, (B, C)$ is independent if A, B are independent and A, C are independent. At first glance, this seems to work, however we can find counter examples to this claim.

Example (Counter example). Let $x, y, z \in \{0, 1\}$ where x, y are uniform and mutually independent while $z \coloneqq x \oplus y$.

Proof. We can see that these are pair-wise independent, but they aren't joint independent as z depends on x and y. Therefore, we have to generalize independence by making sure than an event is independent to every other set of events. ⊛

Definition 4.5.2 (Independence). Given events A and B_i for $i \in \mathcal{I}$ in a probability space, we say event A is *independent* of $\{B_i\}_{i\in\mathcal{I}}$ if

$$\Pr(A \mid \text{any intersection of } B_i) = \Pr(A).$$

More formally, for all $\mathcal{J} \subseteq \mathcal{I}$ with $\mathcal{J} \neq \varnothing$,

$$\Pr\left(A \cap \left(\bigcap\nolimits_{j\in\mathcal{J}} B_j\right)\right) = \Pr(A)\Pr\left(\bigcap\nolimits_{j\in\mathcal{J}} B_j\right).$$

Note (Index set). The only reason we introduce index set $\mathcal{I}$ instead of just saying B_i for $i = 1, \dots, k$ is because Definition 4.5.2 actually extends to countable many intersections, i.e., $|\mathcal{I}|$ can be ∞.

4.5.2 Dependency Graph

Turns out that to represent independence (or dependence, equivalently) between events, a gravarphical form is useful, which we call is a dependency graph.

Definition 4.5.3 (Dependency graph). A *dependency graph* $G = (V, E)$ is a graph such that for every event $\mathcal{E}$ in the sample space, we have a node $v_{\mathcal{E}} \in V$ corresponds to which; while edges $e = (v_{\mathcal{E}_1}, v_{\mathcal{E}_2})$ between nodes $\mathcal{E}_1$ and $\mathcal{E}_2$ represent a dependence between events $\mathcal{E}_1$ and $\mathcal{E}_2$.

Write Definition 4.5.3 more compactly, a dependency graph for $\mathcal{E}_1, \dots, \mathcal{E}_n$ is any graph $G = (V, E)$ where $V = [n] = [1, \dots, n]$ and $\forall i, \mathcal{E}_i$ is independent of the set of events it's not connected to, i.e., $\mathcal{E}_i \perp\!\!\!\perp \{\mathcal{E}_i \mid (i, j) \notin E\}$.

4.5.3 Lovász Local Lemma

The strongest form of Lovász local lemma is the so-called *asymmetric Lovász local lemma.*

Lemma 4.5.2 (Lovász local lemma). Given a dependency graph $G = (V, E)$ for events $\mathcal{E}_1, \dots, \mathcal{E}_n$ in a probability space. Suppose there exist $x_i \in [0, 1]$ for all i, such that

$$\Pr(\mathcal{E}_i) \leq x_i \prod_{(i,j)\in E} (1 - x_j).$$

Then, we have

$$\Pr\left(\bigcap_{i=1}^{n} \overline{\mathcal{E}}_i\right) \geq \prod_{i=1}^{n} (1 - x_i).$$

Proof. Observe that it suffices to show for all $S \subseteq [n]$, for all $i \notin S$, $\Pr\left(\mathcal{E}_i \mid \bigcap_{j\in S} \overline{\mathcal{E}}_i\right) \leq x_i$ since

$$\begin{aligned}
\Pr\left(\bigcap\nolimits_i \overline{\mathcal{E}}_i\right) &= \Pr(\overline{\mathcal{E}}_1) \cdot \Pr\left(\bigcap\nolimits_{i=2}^{n} \overline{\mathcal{E}}_i \mid \overline{\mathcal{E}}_1\right) \\
&= \Pr(\overline{\mathcal{E}}_1) \cdot \Pr(\overline{\mathcal{E}}_2 \mid \overline{\mathcal{E}}_1) \cdot \Pr\left(\bigcap\nolimits_{i=3}^{n} \overline{\mathcal{E}}_i \mid \overline{\mathcal{E}}_1 \cap \overline{\mathcal{E}}_2\right) \\
&= (1 - \Pr(\overline{\mathcal{E}}_1)) \cdot (1 - \Pr(\mathcal{E}_2 \mid \overline{\mathcal{E}}_1)) \dots \left(1 - \Pr\left(\mathcal{E}_n \mid \bigcap\nolimits_{i=1}^{n-1} \overline{\mathcal{E}}_i\right)\right) \\
&\geq (1 - x_1)(1 - x_2) \dots (1 - x_n) = \prod\nolimits_{i=1}^{n} (1 - x_i)
\end{aligned}$$

if we can prove $\Pr\left(\mathcal{E}_i \mid \bigcap_{j\in S}\overline{\mathcal{E}}_i\right) \le x_i$, which is what we want. And indeed, we can prove this by induction. Let $k = |S|$, now we do induct on k.

For the base case, if $k = 0$, from the assumption, we have $\Pr(\mathcal{E}_i) \le x_i$ for all i since we are assuming $\Pr(\mathcal{E}_i) \le x_i \prod_{(i,j)\in E}(1-x_j)$ for all i, and $x_j \in [0,1]$ for all j, so we're multiplying terms which are less than 1, proving the base case.

Now suppose $k := |S| > 0$, and let $S = S_1 \sqcup S_2$ where S_1 contains neighbors of i, i.e., $S_1 = \{j \in S \colon (i,j) \in E\}$, and $S_2 = S \setminus S_1$. Note that if $S_1 = \varnothing$, then $\Pr\left(\mathcal{E}_i \mid \cap_{j\in S}\overline{\mathcal{E}}_j\right) = \Pr(\mathcal{E}_i) \le x_i$, so we may assume that $S_1 \neq \varnothing$. Observe that

$$\Pr\left(\mathcal{E}_i \mid \bigcap_{j\in S}\overline{\mathcal{E}}_j\right) = \frac{\Pr\left(\mathcal{E}_i \cap \left(\bigcap_{j\in S}\overline{\mathcal{E}}_j\right)\right)}{\Pr\left(\bigcap_{j\in S}\overline{\mathcal{E}}_j\right)} = \frac{\Pr\left(\mathcal{E}_i \cap \left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j\right) \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)}{\Pr\left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)}.$$

We can split the numerator and denominator up. For numerator, we have

$$\Pr\left(\mathcal{E}_i \cap \left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j\right) \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right) \le \Pr\left(\mathcal{E}_j \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right) = \Pr(\mathcal{E}_i) \le x_i \prod_{(i,j)\in E}(1-x_j),$$

where the last inequality comes from the assumption. As for the denominator, denote $S_1 := \{j_1, \ldots, j_r\} \neq \varnothing$, we have

$$\begin{aligned}\Pr\left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right) &= \left(1 - \Pr\left(\mathcal{E}_{j_1} \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)\right)\left(1 - \Pr\left(\mathcal{E}_{j_2} \mid \overline{\mathcal{E}}_{j_1}\bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)\right) \\ &\qquad \ldots\left(1 - \Pr\left(\mathcal{E}_{j_r} \mid \overline{\mathcal{E}}_{j_1}\cap\ldots\cap\overline{\mathcal{E}}_{j_{r-1}} \cap \left(\bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)\right)\right) \\ &\ge (1-x_{j_1})\ldots(1-x_{j_r}) \\ &\ge \prod_{(i,j)\in E}(1-x_j).\end{aligned}$$

Combining these, we have

$$\Pr\left(\mathcal{E}_i \mid \bigcap_{j\in S}\overline{\mathcal{E}}_j\right) = \frac{\Pr\left(\mathcal{E}_i \cap \left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j\right) \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)}{\Pr\left(\bigcap_{j\in S_1}\overline{\mathcal{E}}_j \mid \bigcap_{m\in S_2}\overline{\mathcal{E}}_m\right)} \le \frac{x_i\prod_{(i,j)\in E}(1-x_j)}{\prod_{(i,j)\in E}(1-x_j)} = x_i,$$

which proves the inductive step, hence the result follows. ■

Using Lemma 4.5.2, we can prove Corollary 4.5.1, i.e., the symmetric version of Lemma 4.5.2.

Corollary 4.5.1 (Symmetric Lovász Local Lemma). Suppose $\mathcal{E}_i, \ldots, \mathcal{E}_n$ are events in a probability space such that $\Pr(\mathcal{E}_i) \le p$ for all i. If there exists a dependency graph for events $\mathcal{E}_1, \ldots, \mathcal{E}_n$ of degree at most d, and also, $e \cdot p \cdot (d+1) \le 1$,[a] then we have

$$\Pr\left(\bigcup_{i=1}^n \mathcal{E}_i\right) < 1,$$

in other words, $\Pr\left(\bigcap_{i=1}^n \overline{\mathcal{E}}_i\right) > 0$.

[a] e here is the Euler number, which is the base of the natural logarithm, i.e., $\log_e x = \ln x$.

Remark. This version only works if these conditions are fulfilled, and clearly Lemma 4.5.2 is a stronger statement. But nevertheless, Corollary 4.5.1 is still useful whenever it's applicable since its form is much easier.

One application of the Lovász local lemma is in k-SAT, specifically, we'll use Corollary 4.5.1.

Theorem 4.5.1. There is a satisfying assignment for a k-SAT instance if each of its variables appears in at most $2^k/ek$ clauses.

Proof. Suppose that a k-CNF φ has k literals in every clause $C_1, \ldots, C_m$ in which over $x_1, \ldots, x_n$, and each variable appears in less than c clauses.

Take a uniform random assignment of $x_1, \ldots, x_n$ and let the bad event $\mathcal{E}_i$ be the event that clause number i is not satisfied. Then we have

$$\Pr(\mathcal{E}_i) = 2^{-k} =: p.$$

We now use this to bound the probability that every clause is satisfied, i.e., we want to show

$$\Pr\left(\bigcap_{i=1}^{m} \overline{\mathcal{E}}_i\right) > 0$$

Construct a dependency graph $G = (V, E)$ by creating a vertex $v_i \in V$ for every clause C_i, and connecting each clause C_i to clauses C_j if they share a variable with. From the assumption, we know that the degree of G is $\deg(G) < k \cdot c$, equivalently, $\deg(G) \leq k \cdot (c-1)$. Then, from Corollary 4.5.1, if we want

$$\Pr\left(\bigcap_{i} \overline{\mathcal{E}}_i\right) > 0,$$

we need $e \cdot (2^{-k} \cdot (k(c-1)+1)) \leq 1$. Solving this equation for c, we get

$$c \leq \frac{2^k}{e \cdot k}.$$

This proves the following. ∎

Remark. The probability that a random assignment satisfies a k-SAT instance with the above property might be exponentially small. Nevertheless, it exists.

Lecture 9: Moser's Algorithmic Lovász Local Lemma

4.6 Moser's Algorithm

27 Sep. 10:30

We can expand on Lovász's Local Lemma, by going in to detail about an algorithmic version of the proof. This approach was first detailed in a paper by Moser [Mos08]. The analysis that we observe with this algorithmic approach is deemed very *weird*, perhaps one of the top 10 weirdest things we all might encounter, but we'll see why this makes sense.

4.6.1 Motivation

Moser starts his proof by giving an algorithm for k-SAT, which is chosen as it is a general enough application of LLL. For this, we have a k-CNF formula φ with clauses $C_1, \ldots, C_m$ and n variables $x_1, \ldots, x_n$. We denote σ as a uniform random assignment of $x_1, \ldots, x_n$. We want to show that there is a non-zero chance that this assignment σ is going to be a satisfying assignment provided that the number of occurrences is bounded. From Theorem 4.5.1, the bound we needed was $2^k/ek$; however, Moser's proof is able to generate a bound of $2^k/c$ for a constant c that doesn't depend on k.

4.6.2 Algorithm Design

The basic idea of Moser's Algorithm is the following: while $\exists$ clause C_i that is violated by the assignment σ, fix it with the subroutine `Fix()` that takes in the formula φ, the current assignment σ, and i, which represents the index of the violated clause that has k literals (though, if there is no clause that's violated, we are done). For the clause that is unsatisfied, we refresh, or adjust, the variables within that clause. In other words, we adjust the assignment σ', which is σ with variables in C_i re-sampled uniformly random.

We then denote Γ as the set of clauses with shared variables with C_i (including itself). These are the only two things that can be affected by the refresh. Now, while $\exists C_j \in \Gamma$ that is violated by σ' (in some fixed order), we run the subroutine `Fix()`, this time passing φ, σ' and j. We return σ' when this terminates.

In all, we have Algorithm 4.4.

Algorithm 4.4: Moser's Algorithm [Mos08]

```
Data: A k-CNF $\varphi(x_1, \ldots, x_n)$ with $m$ clauses $C_1, \ldots, C_m$
Result: A satisfying assignment $\sigma$
$\sigma \leftarrow$uniform(($x_1, \ldots, x_n$), {0, 1})
while $\exists C_i$ violated by $\sigma$ do
    $\sigma \leftarrow$Fix($\varphi$, $\sigma$, $i$)

Fix($\varphi$, $\sigma$, $i$):
    $\sigma' \leftarrow \sigma$ with variables in $C_i$ re-sampled uniformly at random
    $\Gamma \leftarrow$ set of clauses with shared variables with $C_i$[a]
    while $\exists C_j \in \Gamma$ violated by $\sigma'$ do
        $\sigma' \leftarrow$Fix($\varphi$, $\sigma'$, $j$)
```

[a]Including itself

Problem (Question from classmate)**.** Can this run infinitely?

Answer. Yes, this is entirely possible. Algorithm 4.4 seems like it will not work, as it can fall into an infinite loop. ⊛

Problem (Question from classmate)**.** Do we draw the loops in the dependency graph?

Answer. You can, but it does not have an impact on the proof. This is because it only cares about things you're not connected to. ⊛

Problem (Question from classmate)**.** *While there's a clause that's violated*, if the clause is fixed, is there a chance I'll have to fix it again?

Answer. `Fix()` doesn't let go until a SAT assignment is achieved, i.e., when it returns, it has fixed everything.

Because of this, the top-while can only run `Fix()` as many times as there are clauses, m. Again, worst-case is m times running the top-while. ⊛

4.6.3 Analysis

Let's assume that the algorithm does not get stuck in an infinite loop. By construction, this algorithm is correct because it doesn't let go until everything is satisfied.

Problem (Question from classmate)**.** How is this different than just re-sampling the whole assignment until we find a SAT?

Answer. If we keep re-sampling until SAT, that can take time depending on how many satisfying assignments the formula has. This can be one in practice, so it can easily take exponential time. ⊛

As for time complexity, we need to think about whether this is a poly-time algorithm. Moreover, does this even terminate? We have Theorem 4.6.1, which is an interesting theorem as the original paper does not argue that a satisfying assignment is found, it just argues that `Fix()` terminates in $\mathsf{poly}(n, m)$ time.

Theorem 4.6.1. Suppose each clause shares variables with less than d other clauses where $d < 2^{k-O(1)}$.[a] Then, *all calls* to `Fix()` terminate in $\mathsf{poly}(n, m)$ time.

[a]Where $O(1)$ represents some constant (like 10, 5, or 3).

Proof. We'll consider the *information flow* of this Algorithm 4.4, which is from information theory. An important concept that the proof looks at is the **Log** of execution of an algorithm.

> **Remark** (Log). With recursive algorithms, we have the notion of a stack (the execution Log), where what happens with the stack represents bookkeeping in a sense. We argue that whatever is on the stack has to remain finite, and therefore, the algorithm must terminate.

We can think of the execution Log (or stack of recursive calls) as an ordered list of all C_i being considered by the algorithm, as well as the end of recursive calls, whereupon reaching the end there is nothing else that is violated, and we move up. From here, we need to argue that the stack does not grow very much, which means that the tree cannot have an incredibly large depth. If this is the case, this is an indication of termination. Let H be the minimum number of bits required to encode the execution Log at point where the t^{th} clause is being reassigned (that is, at *time* t in the recursion). We need to upper bound and lower bound this quantity.

Upper Bound We claim that $H \leq \log m + t(k - O(1))$, where m represents the number of clauses and t represents time, and $k - O(1)$ represents $\log |\Gamma|$, the neighborhood of a clause. This is because we can encode H by describing the index of the initial clause C_i, and for each recursive call, we observe that either:

(a) the element of Γ (neighborhood) picked ($k - O(1)$ bits)

(b) end of recursion ($O(1)$ bits, could be as small as 1)

Lower Bound We claim that $H \geq k \cdot t - n$, where n is the number of variables. We can also think of the claim as $H + n \geq k \cdot t$. Let $R \in \{0,1\}^{k \cdot t}$ encode the random values of the variables that are re-sampled for violated clauses, in the order that they appear in the recursion.

There are then two things to observe:

(a) R requires at least $k \cdot t$ bits to describe (it's a uniform random string)

(b) R can be recovered from the knowledge of the Log up to time t and the state of σ of the initial assignment picked by the algorithm for $x_1, \ldots, x_n$. In other words, for every clause that is being reconsidered, which fresh values have I picked.

> **Remark.** Notice that each clause is the **OR** of k literals, so it can be violated in *exactly* one case, i.e., every literal is evaluated as false (consider $\neg x_i$ as one literal).

If we know σ and the execution Log, then we know the fresh values assigned to the last clause at time t. Therefore, we know the assignment at time $t-1$, and we keep backtracking from here.

Based on the upper and lower bounds that we have on H, we can model as follows:

$$k \cdot t - n \leq H \leq \log m + t(k - O(1)) \Rightarrow t \leq O(\log m + n),$$

i.e., `Fix()` can't have more than $(\log m) + n$ recursive calls and, therefore, has to terminate in polynomial time as desired. ■

Corollary 4.6.1. Algorithm 4.4 is a polynomial time algorithm

Proof. We know that the total running time is m times the running time of `Fix()`, which runs in polynomial time as guaranteed by Theorem 4.6.1, so Algorithm 4.4 runs in polynomial time as desired. ■

Chapter 5

Randomized Complexity

Lecture 10: Randomized Complexity

29 Sep. 10:30

5.1 Computational Complexity

5.1.1 Motivation

Let's review some basic notion of computational complexity. In brief, we care about how much resource we need when doing a particular computational task, and computational complexity measure this quantity in different ways.

- Time complexity
- Space complexity
- Communication Complexity
- etc.

And we often interested in asymptotic complexity only. Classical computation models are all deterministic, i.e., Turing machine. Although it can model what we want to measure quite well, but it's not entirely clear how we should model such a deterministic machine with *randomness*. Furthermore, up to this point, we just directly **use** randomness by saying something like *we randomly sample...* Or even a more philosophical question is that, whether the universe has any randomness in it.[1]

5.1.2 Computational Problem

We model a problem by something called language.

Definition 5.1.1 (Language). A *language* L is a subset of $\{0,1\}^*$.

Example (Prime language). PRIMES $:= \{x \mid x$ is the binary representation for a prime number$\}$.

And, our task is that given x and a language L, decide whether $x \in L$ or not, which is the definition of decide. As we have said before, this is sometimes done via Turing machine. We'll not bother write down the formal definition of a Turing machine here, since it's beyond our purpose. Instead, note the following.

Definition 5.1.2 (Decide). A Turing machine $\mathcal{A}$ *decides* a language L if for every input x,

- if $x \in L$, $\mathcal{A}(x) = 1$ and
- if $x \notin L$, $\mathcal{A}(x) = 0$.

[1] Though quantum mechanics seems to guarantee that it does.

Remark (Transition function)**.** A classical Turing machine is equipped with a so-called *transition function* δ, which tells what state of the Turing machine should go to based on the current state and input.

Then, we can simply generalize a classical Turing machine as follows.

Definition 5.1.3 (Nondeterministic Turing machine)**.** A *nondeterministic Turing machine*, or *NTM* for short, is just a TM with multiple transition functions δ_i.

Remark (Computation graph)**.** The main difference between classical TM and NTM can be seen from their *computation graph*: given a fixed input x, NTM can have multiple difference result since it has multiple transition functions δ_i to choose from at each state.

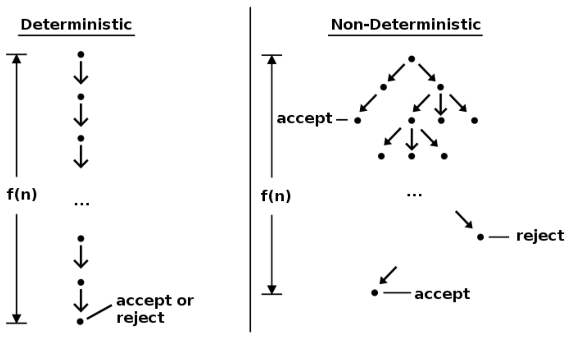


Figure 5.1: Computations graph for TM and NTM.

Remarkably, for an NTM, we say it decides a language if there's at least one accept. And from NTM, we can now simply model randomness as follows.

Definition 5.1.4 (Probabilistic Turing machine)**.** A *probabilistic Turing machine* $\mathcal{A}$ is just a NTM, but in every step in the computation, $\mathcal{A}$ chooses randomly[a] which transition functions to apply.

[a]Equivalent to the machine having a *fair coin*, which, each time it is tossed, comes up **Heads** or **Tails** with equal probability regardless of the past history.

Remark. The difference lies in how we interpret the computations graph: instead of asking whether there *exists* a sequence of choices that makes the TM accept, we ask how large is the *fraction* of choices for which this happens.

5.2 Randomized Complexity Classes

5.2.1 Classical Complexity Classes

Let's first start with some classical complexity classes.

Definition 5.2.1 (Polynomial time)**.** The complexity class P is defined as $L \in \mathsf{P}$ if and only if there is a deterministic polynomial time algorithm $\mathcal{A}$ such that

- if $x \in L$, $\mathcal{A}(x) = 1$ and
- if $x \notin L$, $\mathcal{A}(x) = 0$.

Definition 5.2.2 (Nondeterministic polynomial time)**.** The complexity class NP is defined as $L \in \mathsf{NP}$ if and only if there is a deterministic polynomial time algorithm $\mathcal{A}(x, y)$ such that

- if $x \in L$, $\exists y, |y| = \mathsf{poly}(|x|)$ such that $\mathcal{A}(x, y) = 1$ and
- if $x \notin L$, $\forall y, |y| = \mathsf{poly}(|x|)$ such that $\mathcal{A}(x, y) = 0$.

5.2.2 Randomized Complexity Classes

Now, instead of looking at deterministic TM, we now look at probabilistic TM for defining randomized complexity classes.

Definition 5.2.3 (Randomized polynomial-time)**.** The complexity class RP is defined as $L \in \mathsf{RP}$ if there exists a probabilistic polynomial time algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) \geq 1/2$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 0) = 1$.

Remark. In the definition we can replace $1/2$ by any constant in $(0, 1)$ or even $1/n$, $1/n^2$, any $1/\mathsf{poly}(|x|)$ etc.

Definition 5.2.4 (Co-randomized polynomial-time)**.** The complexity class coRP is defined as $L \in \mathsf{coRP}$ if $\overline{L} \in \mathsf{RP}$, i.e., $\mathsf{coRP} = \left\{\overline{L} \colon L \in \mathsf{RP}\right\}$.

Note. In the definition of RP, if we replace $\geq 1/2$ by > 0, we get NP.

Remark (Monte Carlo algorithm)**.** Algorithms in RP and coRP represent Monte Carlo algorithms.

Example. $L = \{x \mid x \text{ describes a graph with a Hamiltonian cycle}\} \in \mathsf{NP}$.

Definition 5.2.5 (Bounded-error probabilistic polynomial-time)**.** The complexity class BPP is defined as $L \in \mathsf{BPP}$ if there exists a PPT algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) \geq 2/3$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 0) \geq 2/3$.

Note. The $2/3$ can be anything of the form $1/2 + \epsilon$ for constant $\epsilon > 0$ or $\epsilon = \frac{1}{\mathsf{poly}(|x|)}$. Notice that the $\epsilon > 0$ is important since this is the only way one can improve probability of being correct by repetition (and take majority vote).

Definition 5.2.6 (Zero-error probabilistic polynomial-time)**.** The complexity class ZPP is defined as $L \in \mathsf{ZPP}$ if there exists an **expected** PPT algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) = 1$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 1) = 0$.

Equivalently, $L \in \mathsf{ZPP}$ if there's a probabilistic polynomial time algorithm $\mathcal{A}$ such that $\mathcal{A}(x) \in \{\mathsf{accept}, \mathsf{reject}, \mathsf{don't\ know}\}$ and[2]

- if $x \in L$, $\mathcal{A}(x) \in \{\mathsf{accept}, \mathsf{don't\ know}\}$ and
- if $x \notin L$, $\mathcal{A}(x) \in \{\mathsf{reject}, \mathsf{don't\ know}\}$.

[2]We use 1 for accept and 0 for reject implicitly before.

Remark (Las Vegas algorithm)**.** Algorithms for ZPP represent Las Vegas algorithms.

Problem 5.2.1. Show that $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

Definition 5.2.7 (Probabilistic polynomial-time)**.** The complexity class PP is defined as $L \in \mathsf{PP}$ if there exists a PPT algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) > 1/2$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 0) > 1/2$.

Remark. This is similar to BPP, but bounding by $1/2$ makes a huge difference since now $1/2 + \epsilon$ for $\epsilon > 0$ can be arbitrarily close. So if it's inverse-exponentially bounded, then we'll need to run an algorithm exponentially many times in order to show a difference.

Example (Maj-SAT)**.** Given a CNF formula φ, is φ satisfied by a *majority* of the possible assignments? This actually PP-complete [AW21]:

- If φ is 3-CNF, this is in P.
- If φ is 4-CNF, this is NP-complete.

5.2.3 Landscape of Complexity Classes

To summarize, we have the following landscape.

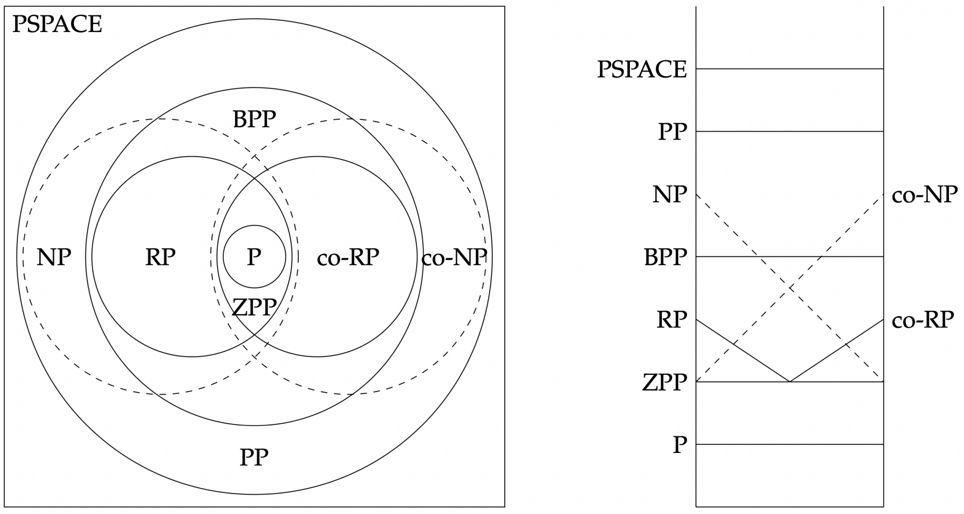


Figure 5.2: Landscape of complexity classes

Unfortunately, Figure 5.2 is not settled down yet, i.e., the relations between complexity classes are not entirely understand yet. We can ask the following questions.

Problem. Is $\mathsf{P} = \mathsf{PSPACE}$?

Answer. This is believed to be false, since otherwise this implies $\mathsf{P} = \mathsf{NP}$. ⊛

Problem. Is $\mathsf{P} = \mathsf{BPP}$?

Answer. Is a big open problem; we think $\mathsf{P} = \mathsf{BPP}$, and we know $\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{BPP}$.

(a) Known to be true under plausible *hardness assumptions*.

(b) For example, Nisan-Widgerson pseudorandom generators assuming we can't always simulate exponential time $\mathsf{DTIME}(2^{O(n)})$ with subexponential space $\mathsf{DTIME}(2^{o(n)})$.

⊛

Lecture 11: PRG and Limited Independence

Following the question last time, we can prove something indeed without assuming hardness assumptions. 4 Oct. 10:30

Lemma 5.2.1. $\mathsf{BPP} \subseteq \mathsf{EXP} = \bigcup_{c>0} \mathsf{DTIME}(2^{n^c})$, i.e. computable in deterministic exponential time.

Proof. We can run the algorithm with all R, and take majority vote, namely to try *every* randomness. Thus, the time it takes is $O(2^{n^c} \cdot n^c)$. ■

Lemma 5.2.1 suggests that we should look exploit some property of $\mathcal{A}$. And indeed, if $\mathcal{A}$ is special, we can make the above more efficient by fooling the algorithm with *fake* randomness as we'll soon see.

5.3 Pseudo-Randomness

Intuitively, in order to perform better, we want to be able to enumerate on shorter random bits and test it against our algorithm to do derandomization. This is done via the so-called pseudo-random generator, where we now discuss.

5.3.1 Pseudo-Random Generator

Before we start, we introduce a widely-used terminology in complexity theory.

Definition 5.3.1 (ϵ-fool)**.** Let s be a random string,[a] a function G of s *ϵ-fools* the algorithm $\mathcal{A}$ for some $\epsilon > 0$ on input x if

$$\Pr_s(\mathcal{A}(x, G(s)) = 1) = \Pr_R(\mathcal{A}(x, R) = 1) \pm \epsilon.$$

[a] Sometimes we call it a *random seed*.

Remark. We see that Definition 5.3.1 only make sense if we consider $|G(s)| \gg |s|$, since otherwise we can just let $G(s) = s$ and let $\epsilon = 0$.

We see that if a function G fools the algorithm $\mathcal{A}$ on every input x, to derandomization, we only need $2^{|s|}\mathsf{poly}(n)$ assuming G runs in $\mathsf{poly}(n)$. This proposes the following.

Definition 5.3.2 (ϵ-pseudo-random generator)**.** An *ϵ-pseudo-random generator* G (against polynomial-time algorithms) for some $\epsilon > 0$ is a function of a random seed s such that for all polynomial-time algorithms $\mathcal{A}$, G ϵ-fools $\mathcal{A}$ on all inputs.

Now, a fundamental problem we should as is that, do PRGs exist? Unfortunately, we don't know! We think yes?

Claim. If PRGs exist, then $\mathsf{P} \neq \mathsf{NP}$.

Proof. Consider the following problem: Given R, is R in the image of G? Firstly, this is a NP problem since an efficient *certificate* exists, i.e., an s such that $G(s) = R$.

On the other hand, if an PRG G exists, then it'll fool every polynomial-time algorithm which try to figure out whether R is produced by G, hence it's not in P, implying $\mathsf{P} \neq \mathsf{NP}$. ⊛

Remark. Assuming unproven assumptions, we can construct PRGs, like Nisan-Wigderson assumption, or in cryptography, the existence of **one-way Functions** (functions that are harder to compute than the inverse).

5.3.2 Limited Independence

We now start to construct some PRG instances with some additional restrictions, in particular, we look into the notion of independence.

Definition. Let T be a finite alphabet of size t and $X_1, \dots, X_n \in T$ be a distribution.

Definition 5.3.3 (Fully independent)**.** The distribution $X_1, \dots, X_n$ is *fully independent* if, for all $b_1 \dots b_n \in T^n$,

$$\Pr(X_1, \dots, X_n = b_1, \dots, b_n) = \frac{1}{t^n}.$$

Definition 5.3.4 (Pairwise independent)**.** The distribution $X_1, \dots, X_n$ is *pairwise independent* if, for all $i \neq j$,

$$\Pr(X_i, X_j = b_i, b_j) = \frac{1}{t^2}.$$

Definition 5.3.5 (k-wise independent)**.** The distribution $X_1, \dots, X_n$ is *k-wise independent* if, for all $i_1, \dots, i_k$ and $b_i \in T$,

$$\Pr(X_{i_1}, \dots, X_{i_k} = b_{i_1}, \dots, b_{i_k}) = \frac{1}{t^k}.$$

With these, we can now construct some problem specific PRGs, i.e, functions which do not fool every polynomial time algorithm, just some instead.

5.3.3 Constructing Pairwise Independent Bits

To construct pairwise independent bits, we do the following.

Algorithm 5.1: Construct Pairwise Independent Bits

```
Data: Uniformly random $k$ bits $b_1, \dots, b_k$
Result: A set $C$ of $2^k - 1$ pairwise independent bits
1 $C \leftarrow \varnothing$
2 for $S \subseteq [k]$ and $S \neq \varnothing$ do                // $2^k - 1$ bits
3     $C_S \leftarrow \bigoplus_{i \in S} b_i$
4     $C \leftarrow C \cup \{C_S\}$
5 return $C$
```

Firstly, the runtime of Algorithm 5.1 is clearly $O(2^k)$, and the correctness of Algorithm 5.1 is ensured by Lemma 5.3.1.

Lemma 5.3.1. C_S in C from Algorithm 5.1 are pairwise independent.

Proof. To see this, take $C_S, C_{S'}$ for $S \neq S' \neq \varnothing$. Then each $C_S, C_{S'}$ is uniform since

$$\Pr(C_S = 1) = \frac{\#\text{ solutions to } \bigoplus_{i \in S} b_i = 1}{2^k} = \frac{2^{k-1}}{2^k} = \frac{1}{2}$$

from linear algebra.[a] So

$$\Pr(C_S = 1, C_{S'} = 0) = \frac{\#\text{ solutions to } \bigoplus_{i\in S} b_i = 1, \bigoplus_{i\in S'} b_i = 0}{2^k} = \frac{2^{k-2}}{2^k} = \frac{1}{4},$$

hence are pairwise independent. ■

[a]Since $\bigoplus_{i\in S} b_i = 1$ is just one linear constraint, hence the solution space is $(k-1)$-dimensional.

Another way to show Lemma 5.3.1 is to observe that

$$C_S \oplus C_{S'} = \bigoplus_{i\in S} b_i \oplus \bigoplus_{j\in S'} b_j = \bigoplus_{i\in S\triangle S'} b_i,$$

which is again uniform.

Remark. If bits $b, b' \in \{0,1\}$ are each uniform and $b \oplus b'$ is uniform, then (b, b') is uniform.

Proof. Let e_{ij} be the event that $(b_1, b_2) = (i, j)$. The uniformity assumptions imply that

$$\Pr(e_{ij} \cup e_{i'j'}) = \frac{1}{2}$$

for each pair of $(i,j) \neq (i',j')$. Let $p := \Pr(e_{00})$. Then, $1/2 - p = \Pr(e_{ij})$ for each $(i,j) \neq (0,0)$. Now, observe that the probability of the union of some other pair of events excluding 00 is $1/2 - p + 1/2 - p$, and which we know is $1/2$ again from uniformity assumption. This implies $p = 1/4$ which in turn implies uniformity of (b_1, b_2), as desired.[a] ⊛

[a]This argument is adapted from Piazza.

All these suggest that for an adversarial algorithm that only checks for pairwise independence, Algorithm 5.1 is sufficient to generate pseudo-randomness in this content.

5.3.4 Derandomize Max-Cut

Indeed, Problem 4.2.1 only looks at pairwise independence, hence we can apply Algorithm 5.1.

As previously seen. We already have Algorithm 4.1 for Problem 4.2.1, which is a 2-approximation algorithm. Specifically, given $G = (V, E)$, we construct $S \subseteq V$ by adding each vertex to S independently with probability 1/2. That is,

(a) $X_1, \ldots, X_n$ (with $n = |V|$) chosen uniformly at random.

(b) $S = \{i \colon X_i = 1\}$.

Then for each edge $(i,j) \in E$, we have $\Pr((i,j) \text{ is cut}) = \Pr(X_i \oplus X_j = 1) = 1/2$. Then

$$\mathbb{E}\left[\left|E(S, \overline{S})\right|\right] = \frac{|E|}{2},$$

where $\left|E(S, \overline{S})\right|$ is the cut size.

However, observe that this the above analysis is still true if $X_1, \ldots, X_n$ are pairwise independent, i.e.,

$$\mathbb{E}_{r_1,\ldots,r_s}\left[\left|E(S, \overline{S})\right|\right] = \frac{|E|}{2}$$

where $r_1, \ldots, r_s \to \boxed{G} \to X_1, \ldots, X_n$ with $s \approx \log n$ and G a pairwise independent PRG. From probabilistic method (over randomness of r_i), there is some $(r_1, \ldots, r_n)$ such that

$$\mathbb{E}_{r_1,\ldots,r_s}\left[\left|E(S, \overline{S})\right|\right] \geq \frac{|E|}{2},$$

so we can search for such a $(r_1, \ldots, r_s)$.

Remark. To enumerate all randomness (of r_i), we only need $2^s = 2^{\log n} = O(n)$ runs, which implies that this is a polynomial-time and deterministic algorithm!

Explicitly, we have the following.

Algorithm 5.2: Deterministic Max-Cut

```
Data: A graph $G = (V, E)$
Result: A 1/2-approximation cut $S$
1 $S^* \leftarrow \varnothing$
2 for $r \subseteq [\log n]$ do                       // Enumerate randomness
3     $X \leftarrow$ PRG($r$)                          // $r = \{r_i\}_{i=1}^{s} \mapsto G(r) = \{X_i\}_{i=1}^{n}$
4     $S \leftarrow \{v_i \in V : X_i = 1\}$
5     if $|S^*| < |S|$ then                            // keep the largest cut
6         $S^* \leftarrow S$
7 return $S^*$
```

We can also do the derandomization of Algorithm 4.1 by conditional expectation, but this one illustrates the power of PRGs, though not the full power of which, only in pairwise independent sense. Also, notice that from our analysis, Algorithm 5.2 is guaranteed to give us a 2-approximation cut.

Remark (Beyond binary field)**.** To further generalize Algorithm 5.1, given an alphabet Q of size q, we can generate pairwise independence from $\log n + \log q$ random bits, where $\log n = k$, i.e., n is the number of variables (or length) of bit string we want to generate.

Proof. Extend this to pairwise independent over $\mathbb{F}_q$:

(a) Pick $a, b \in \mathbb{F}_q$ uniformly at random.

(b) $r_i \leftarrow ai + b, \forall i \in \mathbb{F}_q$.

(c) Output all r_i.

The seed length is $2 \log q$ bits, and the output length is q. We can show that this is pairwise independent. ⊛

Lecture 12: Error Reduction and Concentration Bounds

5.4 Error reduction

6 Oct. 10:30

5.4.1 Serial Repetition

Earlier, we saw that you can derandomize some random algorithms to get a polynomial time algorithm. However, previously the probability of error was too large. What happens if we want to reduce the probability of an error?

Let's focus on RP, where x is input, L is the language, $\mathcal{A}$ is an algorithm, and R is randomness.

As previously seen. Recall that if $\mathcal{A} \in \mathsf{RP}$,

$$\begin{cases} \Pr_R(\mathcal{A}(x, R) = 1) \geq 1/2, & \text{if } x \in L \\ \Pr_R(\mathcal{A}(x, R) = 1) = 0, & \text{if } x \notin L. \end{cases}$$

Previously, we said that to do error reduction, we simply repeat the experiment, which is called serial repetition, which works as follows.

Algorithm 5.3: Error Reduction – Serial Repetition

```
Data: An algorithm $\mathcal{A}$, an input $x$, randomness length $|R|$
Result: The result of $\mathcal{A}(x)$
for $i = 1, \dots, k$ do                         // $O(k \cdot |R|)$ bits of randomness
    $R_i \leftarrow$ uniform($|R|$)               // Sample $R_i$ uniformly at random
for $i = 1, \dots, k$ do
    if $\mathcal{A}(x, R_i) = 1$ then
        return 1
return 0                                         // If all rejects
```

We see that in Algorithm 5.3, we're simply repeating $\mathcal{A}$ k times, and that will increase the $1/2$ to $1 - 2^{-k}$, i.e.,

$$\begin{cases} \Pr(\mathsf{accept}) \geq 1 - (1 - 1/2)^k = 1 - 2^{-k}, & \text{if } x \in L \\ \Pr(\mathsf{accept}) = 0, & \text{if } x \notin L. \end{cases}$$

Remark (Caveat)**.** In Algorithm 5.3, we sample R_i for $i = 1, \dots, k$ randomly, i.e., we need $O(|R| \cdot k)$ bits of randomness!

5.4.2 Chebyshev's Inequality

As one might expect, to do error reduction, Markov inequality is sometimes not strong enough, so we know first look at something stronger, i.e., Chebyshev's inequality. Sometimes, Chebyshev's inequality is also called the second-moment method, since variance is the second-moment.

As previously seen (Variance)**.** The *variance* of X is defined as

$$\mathrm{Var}\left[x\right] = \mathbb{E}\left[(X - \mathbb{E}(X))^2\right] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2.$$

Let's look at the theorem statement.

Theorem 5.4.1 (Chebyshev's inequality)**.** Let X be a random variable with finite expected value and finite non-zero variance. Then for any real number $k > 0$,

$$\Pr(|X - \mathbb{E}[X]| \geq \alpha) \leq \frac{\mathrm{Var}\left[X\right]}{\alpha^2}.$$

Proof. By defining $Y \coloneqq (X - \mathbb{E}[X])^2 \geq 0$ and apply Markov inequality, we have

$$\Pr(|X - \mathbb{E}[X]| \geq \alpha) = \Pr\left(Y \geq \alpha^2\right) \leq \frac{\mathbb{E}\left[Y\right]}{\alpha^2} = \frac{\mathrm{Var}\left[X\right]}{\alpha^2}.$$

∎

Remark. We see that a slightly different from of Theorem 5.4.1 is

$$\Pr(|X - \mathbb{E}[X]| \geq \beta\sigma) \leq \frac{1}{\beta^2}$$

where $\sigma \coloneqq \sqrt{\mathrm{Var}\left[X\right]}$.

Corollary 5.4.1. For $X_1, X_2, \dots, X_n$ in $[0, 1]$ being pairwise independent, and denote $X \coloneqq \sum_{i=1}^{n} X_i/n$, $\mu \coloneqq \mathbb{E}[X]$, we have

$$\Pr(|X - \mu| \geq \epsilon) \leq \frac{1}{n\epsilon^2}.$$

Proof. By applying Theorem 5.4.1, we have

$$\mathrm{Var}[X] = \sum_{i=1}^{n} \frac{\mathrm{Var}[X_i]}{n^2} \leq n \cdot \frac{1}{n^2} = \frac{1}{n}$$

where the linearity of variance is due to pairwise independence.[a] ■

[a]This is a simple exercise to check from definition.

We're now ready to see how we can utilize Theorem 5.4.1 to do error reduction, i.e., instead of simply just repeating the experiment, we can use pairwise independence. Again, we execute $\mathcal{A}$ multiple times and the executions are correlated, but they will be independent. Specifically, any two executions are independent, but across the k iterations, there will be correlation.

5.4.3 2-Point Sampling

Consider a family $\mathcal{H}$ of hash functions from[3] $[2^{k+2}] \to \{0,1\}^{|R|}$ where $|R|$ is the number of random bits needed in each run, and every random function $h \in \mathcal{H}$ has a pairwise independent truth table, i.e., for all i, $h(i)$ is uniform and for all $i \neq j$, $h(i)$ and $h(j)$ are independent.

Remark (Pairwise independent truth table)**.** Here, $\mathcal{H}$ just acts as a family of G we obtain from Algorithm 5.1, where any two slots in the truth table of h are independent, i.e., pairwise independent.

Then we have the following algorithm.

Algorithm 5.4: Error Reduction – 2-Point Sampling

```
Data: An algorithm $\mathcal{A}$, an input $x$, a hash functions family $\mathcal{H}$
Result: The result of $\mathcal{A}(x)$
1 $h \leftarrow$ uniform($\mathcal{H}$)                    // Pick $h \in_R \mathcal{H}$[a]
2 for $i = 1, \ldots, 2^{k+2}$ do
3     if $\mathcal{A}(x, h(i)) = 1$ then
4         return 1
5 return 0                                          // If all rejects
```

[a]The notation $\in_R$ means we pick something uniformly at random.

The question is, since now we only use fewer random bits, how does the failure probability change? Namely, we want to see if the probability of error has decreased as in serial repetition. Obviously, we know that if $x \notin L$, $\Pr(\mathsf{accept}) = 0$, so we're interested in the error probability when $x \in L$.

Lemma 5.4.1. The failing probability of Algorithm 5.4 is less or equal to 2^{-k}.

Proof. Assume $x \in L$,[a] and define $\sigma_i = \mathcal{A}(x, r_i) \in \{0,1\}$. We are interested in running $\mathcal{A}$ on random choices of randomness r_i. Since $\mathbb{E}[\sigma_i] \geq 1/2$ from the definition of RP, by denoting y to be the number of 0 that we have ever seen in line 3, we have

$$y := \sum_{i=1}^{K} \sigma_i$$

where $K := 2^{k+2}$. Now, the bad event is when $y = 0$, i.e., there are no successes while $x \in L$. Let $\mu := \mathbb{E}[y/K]$, which is $\geq 1/2$,[b] we have $\Pr(\mathsf{reject}) = \Pr(y/K = 0)$. We see that we can use Chebyshev's inequality, specifically, Corollary 5.4.1: let $\epsilon = \mathbb{E}[y/K]$, we have

$$\Pr(\mathsf{reject}) = \Pr\left(\frac{y}{K} = 0\right) \leq \Pr\left(\frac{y}{K} - \mathbb{E}\left[\frac{y}{K}\right] \geq \mathbb{E}\left[\frac{y}{K}\right]\right) \leq \frac{1}{K \cdot \left(\frac{1}{2}\right)^2} = 2^{-(k+2)} \cdot 4 = 2^{-k}$$

[3]We just make the pseudo-random bits a little longer than what we actually need to make our analysis easier.

as desired. ■

[a]Since this is the only error can happen.
[b]Due to the linearity of expectation.

Compare to serial repetition (where we need $k \cdot |R|$ random bits), Algorithm 5.4 utilizes the linear pairwise independent generator to generate $q = 2^{|R|+k+2}$[4] pairwise independent bits using $O(\log q)$ *true* random bits,[5] which is $O(|R|+k)$. As one can see, we went from $k \cdot |R|$ random bits to $k + |R|$ random bits, which is a big improvement. And from Lemma 5.4.1, we see we can achieve the same error reduction as serial repetition, but with only $O(|R|+k)$ bits of randomness instead of $O(|R| \cdot k)$ bits.

Remark (Caveat)**.** Now, we need 2^k iterations. Although the total runtime does not depend on n, but previously we only needed k iterations.

5.5 Chernoff-Hoeffding Bound

What about the concentration when there is full independence? Say we have $X_1, X_2, \dots, X_n$ being fully independent, and we again denote $X = \sum_{i=1}^n X_i$, $\mathbb{E}[X_i] = \mu_i$ and $\mathrm{Var}[X_i] = \sigma_i^2$. We see that

$$\mu = \mathbb{E}[X] = \sum_{i=1}^n \mu_i, \quad \sigma^2 = \mathrm{Var}[X] = \sum_{i=1}^n \sigma_i^2.$$

Example (Binomial distribution)**.** The *binomial distribution* $\mathrm{binom}(n,p)$ has the property that

$$\mu = np \text{ and } \sigma^2 = np(1-p).$$

Proof. We can model this as the event of biased coin flip. Denote the outcome of flipping a coin as follows: head as 1 and tail as 0, i.e., $X_i \in \{0,1\}$ for i^{th} coin flip.

If this coin has probability p being head, then we have $\mu_i = p$ and $\sigma_i^2 = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 = p(1-p)$, hence

$$\mu = np \text{ and } \sigma^2 = np(1-p).$$

⊛

5.5.1 Central Limit Theorem

Remarkably, under the setting of fully independent and identical, i.e., i.i.d. samples, we have a strong concentration behavior on the distribution level.

Theorem 5.5.1 (Central limit theorem)**.** Let $X_1, X_2, \dots, X_n$ being i.i.d., and we denote $X = \sum_{i=1}^n X_i$, $\mu := \mathbb{E}[X]$ and $\sigma^2 := \mathrm{Var}[X]$. Then we have

$$\frac{X-\mu}{\sigma} \to \mathcal{N}(0,1) \text{ as } n \to \infty,$$

where $\mathcal{N}$ is the normal distribution, and $\mathcal{N}(0,1)$ is the standard normal distribution.

Using Theorem 5.5.1, we have the following asymptotic bound.

Corollary 5.5.1. Given i.i.d. samples $X_1, \dots, X_n$ and $X := \sum_{i=1}^n X_i$, $\mu := \mathbb{E}[X]$ and $\sigma^2 := \mathrm{Var}[X]$, for all $\beta > 0$,

$$\Pr(|X-\mu| > \beta\sigma) \to \frac{1}{\sqrt{2\pi}} \int_\beta^\infty e^{\frac{-t^2}{2}} \,\mathrm{d}t \approx \frac{1}{\sqrt{2\pi}} e^{\frac{-\beta^2}{2}}.$$

[4]Now it's should be clear why we use 2^{k+2}: this makes the analysis cleaner and get the same error reduction essentially.
[5]This is from the remark.

Remark (Error function). The function in Corollary 5.5.1 is called the *error function* (erf). This is asymptotic as $n \to \infty$ and is only useful for a *decent* β, i.e., β can't be too small.

5.5.2 Chernoff-Hoeffding bound

The Chernoff-Hoeffding bound gives a sharper estimate (for all n) and all β compare to Corollary 5.5.1.

Theorem 5.5.2 (Chernoff-Hoeffding bound). Given n fully independent samples $X_1, \ldots, X_n \in \{0, 1\}$, let $\mathbb{E}[X_i] = \mu_i$, $X = \sum_{i=1}^n X_i$, $\mu = \mathbb{E}[X] = \sum_{i=1}^n \mu_i$ and $p = \mu/n$, we have

(a) $\Pr(X > \mu + \lambda) \leq \exp\left(-nH_p(p + \frac{\lambda}{n})\right)$ for $0 < \lambda < n - \mu$

(b) $\Pr(X \leq \mu - \lambda) \leq \exp\left(-nH_{1-p}(1 - p + \frac{\lambda}{n})\right)$ for $0 < \lambda < \mu$

where

$$H_p(x) = x \ln\left(\frac{x}{p}\right) + (1 - x) \ln\left(\frac{1 - x}{1 - p}\right).$$

Note (Relative entropy). $H_p(x)$ is sometimes called *relative entropy*. It's with the property $H_p(x) \geq 0$ (and $H_p(x) = 0 \Leftrightarrow x = p$).

Remark. There are lots of variation of Theorem 5.5.2, and this should be the sharpest one can find.

Corollary 5.5.2. Given n fully independent samples $X_1, \ldots, X_n \in \{0, 1\}$, let $\mathbb{E}[X_i] = \mu_i$, $X = \sum_{i=1}^n X_i$, $\mu = \mathbb{E}[X] = \sum_{i=1}^n \mu_i$ and $p = \mu/n$, we have

$$\max\left(\Pr(X \geq \mu + \lambda), \Pr(X \leq \mu - \lambda)\right) \leq \exp\left(-\frac{2\lambda^2}{n}\right).$$

This is a much clean-up version of Theorem 5.5.2, which we see that this is exponentially decay on both upper and lower tails.

Lecture 13: Proof of Chernoff Bounds and its Applications

Let's now prove the Chernoff-Hoeffding bound. 11 Oct. 10:30

Proof of Theorem 5.5.2. We start by proving the upper tail first, i.e.,

$$\Pr(X \geq \mu + \lambda) \leq e^{-nH_p(p + \frac{\lambda}{n})}.$$

Firstly, to simplify the notation, we let $m := \mu + \lambda$. Since e^x is a monotonically increasing function, for all $t > 0$, we have

$$\Pr(X \geq m) = \Pr\left(e^{tX} \geq e^{tm}\right).$$

Notice that $e^{tX} \geq 0$, we can apply Markov's inequality to this expression, and we get

$$\Pr(X \geq m) \leq \frac{\mathbb{E}\left[e^{tX}\right]}{e^{tm}}.$$

Since all X_i are independent, we have

$$\Pr(X \geq m) \leq e^{-tm} \prod_{i=1}^n \mathbb{E}[e^{tX_i}] \Rightarrow \Pr(X \geq m) \leq e^{-tm} \prod_{i=1}^n (\mu_i e^{t \cdot 1} + (1 - \mu_i) e^{t \cdot 0})$$

$$\Rightarrow \Pr(X \geq m) \leq e^{-tm} \prod_{i=1}^n (\mu_i e^t + 1 - \mu_i)$$

We now use the arithmetic geometric mean inequality,[a] which states

$$\frac{1}{n}\sum_{i=1}^{n} a_i \geq \sqrt[n]{\prod_{i=1}^{n} a_i},$$

implying

$$\Pr(X \geq m) \leq e^{-tm}\left(\frac{e^t\left(\sum_i X_i + n - \sum_i X_i\right)}{n}\right)^n \Rightarrow \Pr(X \geq m) \leq e^{-tm}\left(e^t p + 1 - p\right)^n.$$

To optimize for t, we let $f(t) \coloneqq \exp(n\ln(pe^t + 1 - p) - tm)$ and setting $\mathrm{d}f(t)/\mathrm{d}t = 0$, we have

$$e^t = \frac{m(1-p)}{(n-m)p}.$$

Substituting this value for e^t back, we have

$$\begin{aligned}
\Pr(X \geq m) &\leq \exp\left(n\ln\left(\frac{m(1-p)}{n-m} + 1 - p\right) - m\ln\left(\frac{m(1-p)}{n-m}\right)\right)\\
\Rightarrow \Pr(X \geq m) &\leq \exp\left(n\left(\ln\left(\frac{1-p}{1-m/n}\right) - m/n\ln\left(\frac{(1-p)m/n}{(1-m/n)\,p}\right)\right)\right)\\
\Rightarrow \Pr(X \geq m) &\leq \exp\left(n\left(\left(1-\frac{m}{n}\right)\ln\left(\frac{1-p}{1-m/n}\right) + \frac{m}{n}\ln\left(\frac{p}{m/n}\right)\right)\right)\\
\Rightarrow \Pr(X \geq m) &\leq \exp\left(-nH_p\left(p + \frac{\lambda}{n}\right)\right),
\end{aligned}$$

since $m = \mu + \lambda$ and $m/n = p + \lambda/n$, proving the theorem. The proof for the lower tail probability, i.e.,

$$\Pr(X \leq \mu + \lambda) \leq e^{-nH_{1-p}(1-p+\frac{\lambda}{n})},$$

is symmetric by considering $\widetilde{X}_i \coloneqq 1 - X_i$ and proceed it the same way. ■

[a]The AGM inequality can be proved by taking log on both sides of the expression and observing that because of the convexity of log, log(average) > average(log).

While we considered X_i to be independent indicator variables, the inequality holds for any $X_i \in [0,1]$. This is done by showing that if $Y \in [0,1]$, $Z \in \{0,1\}$ and $\mathbb{E}[Y] = \mathbb{E}[Z]$, then for all convex f,

$$\mathbb{E}[f(y)] \leq \mathbb{E}[f(z)].$$

In the above proof, we just consider $f(x) = e^{tx}$.

Remark (Hoeffding's inequality). Even further, Theorem 5.5.2 can be extended to $X_i \in [a_i, b_i]$ by scaling and shifting. Doing so results in what is known as *Hoeffding's inequality*, which is an extremely useful tool. Hoeffding's inequality states that

$$\begin{aligned}
\Pr(X \leq \mu - \lambda) &\leq \exp\left(\frac{-2\lambda^2}{\sum_{i=1}^n |b_i - a_i|}\right)\\
\Pr(X \geq \mu + \lambda) &\leq \exp\left(\frac{-2\lambda^2}{\sum_{i=1}^n |b_i - a_i|}\right),
\end{aligned}$$

and therefore,

$$\Pr(|X - \mu| \geq \lambda) \leq 2\exp\left(\frac{-2\lambda^2}{\sum_{i=1}^n |b_i - a_i|}\right).$$

Similar to Hoeffding's inequality, recall Corollary 5.5.2.

As previously seen. In Corollary 5.5.2, we have

$$\Pr(X \geq \mu + \lambda) \leq \exp\left(\frac{-2\lambda^2}{n}\right);$$
$$\Pr(X \leq \mu + \lambda) \leq \exp\left(\frac{-2\lambda^2}{n}\right)$$

and therefore,

$$\Pr(|X - \lambda| \geq \mu) \leq 2\exp\left(\frac{-2\lambda^2}{n}\right)$$

We now prove Corollary 5.5.2.[6]

Proof of Corollary 5.5.2. Let $z = \frac{\lambda}{n} \Rightarrow 2nz^2 = \frac{2\lambda^z}{n}$. From Theorem 5.5.2, we have

$$\Pr(X \geq \mu + \lambda) \leq \exp\left(-nH_p(p + \frac{\lambda}{n})\right) \Rightarrow \Pr(X \geq \mu + \lambda) \leq \exp(-nH_p(p+z)).$$

Now set

$$f(z) = (p+z)\ln\left(\frac{p+z}{p}\right) + (1-p-z)\ln\left(\frac{1-p-z}{1-p}\right) - 2z^2,$$

observe that $f(0) = 0$, and by taking the derivative, we see that $f(z)$ is a non-decreasing function in z. Therefore, $f(z) \geq 0$ for all z, implying

$$\Pr(X \geq \mu + \lambda) \leq \exp\left(\frac{-2\lambda^2}{n}\right).$$

It's easy to see that the other case can be proved in the same fashion. ■

Finally, we see the last corollary.

Corollary 5.5.3. For all $\beta \in (0,1)$, we have

$$\Pr(X \leq (1-\beta)\mu) \leq \exp(-\mu(\beta + (1-\beta)\ln(1-\beta))) \leq \exp\left(\frac{-\beta^2\mu}{2}\right),$$

and also,

$$\Pr(X \geq (1+\beta)\mu) \leq \exp(-\mu(-\beta + (1+\beta)\ln(1+\beta))) \leq \begin{cases} \exp\left(\frac{-\beta^2\mu}{2+\beta}\right) & \forall\beta > 0; \\ \exp\left(\frac{-\beta^2\mu}{3}\right) & 0 < \beta \leq 1. \end{cases}$$

The proof of this corollary is left as an exercise. Instead, we now see some applications and examples.

Example (Fair coin tosses). Consider a case where we are flipping fair coins, where we interpret a head as 1, tail as 0, and we get a head with probability p, a tail with probability $1-p$.

We see that this is just a series of $\mathrm{ber}(p_i)$ with $p_i = 1/2$, $X_i \in \{0,1\}$, $\mathbb{E}[X_i] = 1/2$ and $\mu = n/2$. From Corollary 5.5.2, let $X \coloneqq \sum_i X_i$, we have

$$\Pr(|X - \mu| \geq \lambda) \leq 2e^{-2\lambda^2/n}$$

The standard deviation for X is $\sigma = \sqrt{np(1-p)} = \sqrt{n}/2$.[a] Now if $\lambda = \beta\sigma$,

$$\Pr(|\#\text{ tail} - n/2| > \beta\sigma) \leq 2e^{-\beta^2/2},$$

[6]This can also be used to prove Hoeffding's inequality.

which is similar to the central limit theorem, but is applicable to all n not just $n \to \infty$.

[a]This is from the independence of X_i, hence we have linearity of variance.

Example (Biased coin tosses). Extending the previous case, if now $p_i = 3/4$, then $\mu = 3n/4$. Then

$$\Pr(\text{less than half of the flips are heads}) = \Pr\left(X \leq \frac{n}{2}\right) = \Pr\left(X \leq \left(1 - \frac{1}{3}\right)\mu\right).$$

From Corollary 5.5.3 and let $\beta = 1/3$,

$$\Pr(X \leq (1 - 1/3)\mu) \leq \exp\left(\frac{-(1/3)^2}{2} \cdot \frac{3n}{4}\right) = e^{-n/24},$$

hence the probability of less than half of the flips are heads decays exponentially w.r.t. n.

We now see some interesting applications of Theorem 5.5.2.

5.5.3 Randomized Routing

We consider the so-called network routing problem, which aims to determine how to best route packets in a network without congestion.

Problem 5.5.1 (Network Routing). Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the *network routing problem* asks a routing which sends packets from every $i \in \mathcal{V}$ to its destination $\pi(i)$ while minimizing the delays.

We see that Problem 5.5.1 is way too general and is difficult without any constraints, so we consider the following set-up.

Definition 5.5.1 (Hypercube network). The *hypercube network* is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{0, 1\}^n$, i.e., each node represents an n-bits string, and there is an edge in $\mathcal{E}$ if two nodes differ from exactly one bit.

Remark. If we visualize a hypercube network, it looks just like a hypercube.

Formally, $(u, v) \in \mathcal{E}$ if $\Delta(u, v) = 1$, where $\Delta(u, v)$ represents the hamming distance between vertices u and v. Given this specific graph, we also consider a particular routing scheme called oblivious routing.

Definition 5.5.2 (Oblivious routing). The *oblivious routing* is a routing scheme such that every packet is routed irrespective of the others, i.e., how i goes to $\pi(i)$ only depends on i and $\pi(i)$, and is fixed in advance.

Now, we can consider Problem 5.5.1 of a hypercube network with oblivious routing scheme. Notice that each edge is duplex and allows traffic in both directions, and each edge acts as a queue, i.e., there is a congestion if there are multiple packets on an edge, since they must wait for all the packets that arrived before to finish processing.

It is proved that for any deterministic and oblivious routing strategy, there exists $\pi(\cdot)$ that requires $\Omega\left(\sqrt{2^n/n}\right)$ steps [KKT90]. The best that we can hope for is $O(n)$, so this adversarial π that requires an exponential number of steps is very unfortunate. Luckily, there is a randomized and oblivious strategy that will terminate in $O(n)$ steps with high probability [VB81].

The basic premise is that we can introduce random detours that help spread out traffic and reduce congestion. The algorithm operates in two phases:

- Phase 1: each i is routed to some $\delta(i)$ that is chosen uniformly at random.
- Phase 2: each i is routed from $\delta(i)$ to $\pi(i)$.

In each phase, we take the natural shortest path which is the bit fixing path. To go from any X to Y, we simply flip each bit of X from left to right where it differs with Y. Specifically, we have Algorithm 5.5.

Algorithm 5.5: Network Routing – Randomized Oblivious Routing on Hypercube

```
    Data: A hypercube network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, destination $\pi(\cdot)$
    Result: The routing $\{P_i\}_{i\in\mathcal{V}}$
1   for $i \in \mathcal{V}$ do
2       $\delta(i) \leftarrow$rand($\mathcal{V}$)                                  // Choose a detour
3       $P_i \leftarrow$ShortestPath($i$, $\delta(i)$)+ShortestPath($\delta(i)$, $\pi(i)$)   // Connect two paths
4   return $\{P_i\}_{i\in\mathcal{V}}$
```

We'll see that each phase takes $O(n)$ with high probability, and hence the entire algorithm takes $O(n)$ with high probability.

Lecture 14: Oblivious Routing and Markov Chains

Toward proving that such a simple Algorithm 5.5 achieves $O(n)$ total delays with high probability, let $D(i)$ be the delay of packet i (when the packet is waiting in the queues), then we see that the total travel time is less than or equal to $n + D(i)$. We want to show that for all $i \in \mathcal{V}$, 13 Oct. 10:30

$$\Pr(D(i) > c \cdot n) \leq e^{-2n}.$$

If this is true, by the union bound on i, we have

$$\Pr(\text{some node takes a delay } > c \cdot n) \leq \frac{2^n}{e^{-2n}} \leq 1.$$

We go from packet i to packet $\delta(i)$ via sequence $P_i = \{e_i, e_2, \ldots, e_k\}$ of edges, i.e.,

$$i \xrightarrow{e_1} \cdot \xrightarrow{e_2} \cdot \xrightarrow{e_3} \ldots \xrightarrow{e_{k-1}} \cdot \xrightarrow{e_k} \delta(i).$$

By using this notation, we see that a (potential) collision happens if $P_i \cap P_j \neq \varnothing$ such that $i \neq j$, and we denote S_i be the set of potential collision nodes, i.e., $S_i := \{j \in \mathcal{V} \colon j \neq i \text{ s.t. } P_i \cap P_j \neq \varnothing\}$.

Note. Observe that once the paths stop colliding, i never meets j again.

Lemma 5.5.1 (Lemma 4.5 [MR95]). $D(i) \leq |S_i|$.

Proof. Argue each unit of delay can be assigned to a distinct element of S_i. ■

Now, define an indicator random variable H_{ij} by

$$H_{ij} = \begin{cases} 1, & \text{if } P_i \cap P_j \neq \varnothing; \\ 0, & \text{otherwise,} \end{cases}$$

we have $\sum_{i\neq j} H_{ij} = |S_i|$. From Lemma 5.5.1, $D(i) \leq |S_i|$, and hence $D(i) \leq \sum_{i\neq j} H_{ij}$. Notice that P_i and P_j are independent random variables, we can apply Chernoff-Hoeffding bounds. To use it, we first need to find the expectation of the previous inequality, which is $\mu := \mathbb{E}\left[\sum_{i\neq j} H_{ij}\right]$. We then have the following.

Lemma 5.5.2. $\mathbb{E}\left[\sum_{j\neq i} H_{ij}\right] \leq n/2$.

Proof. Define $T(e)$ as the number of paths that pass through edge e, and by symmetry of random variables, we know that $\mathbb{E}[T(e)]$ is the same for all edges,[a] so

$$\mathbb{E}[T(e)] = \frac{\mathbb{E}[\text{total lengths of all paths}]}{\#\text{ of directed edges}} = \frac{N \cdot n/2}{N \cdot n} = \frac{2^n \cdot n/2}{2^n \cdot n} = \frac{1}{2}$$

where $N = 2^n$, and we know that the length of the entire path for i is

$$\mathbb{E}[\text{length of path from } i \text{ to } \delta(i)] = \frac{n}{2}$$

since for every i, given a random string $\delta(i)$, in expectation there will be $n/2$ bits differed from i. Because we have established $\mathbb{E}\left[T(e)\right] = 1/2$, and we know $|P_i| \leq n$, for all i,

$$\mu = \mathbb{E}\left[\sum\nolimits_{i\neq j} H_{ij}\right] \leq \mathbb{E}\left[\sum\nolimits_{e\in P_i} T(e)\right] \leq \frac{n}{2},$$

where the middle inequality comes from the fact that we may over-count some collisions, e.g., there's one specific path that passes through e_1 and e_2 in P_i, and in this case, we'll count this twice when summing over $T(e_i)$, but it actually just contribute to the sum of H_{ij} once. But since we want an upper-bound, the result still follows. ■

[a] We can think of this like there are no preferences among edges.

This implies Theorem 5.5.3.

Theorem 5.5.3. The routing induced by Algorithm 5.5 achieves $O(n)$ total delay with probability at least $1 - e^{-2n}$.

Proof. From Lemma 5.5.2, we can apply the Chernoff-Hoeffding bounds. Firstly, recall that $\mu = \mathbb{E}\left[|S(i)|\right] \leq n/2$ and $|S(i)| = \sum_{i\neq j} H_{ij}$, the failure probability is

$$\Pr\left(|S(i)| \geq t\right) \leq \exp\left(\frac{-2(t-\mu)^2}{n}\right) \leq \exp\left(\frac{-2\left(t-n/2\right)^2}{n}\right),$$

we see that if $t = O(n)$, the above probability is less than $\exp(-2n)$ as desired. ■

Chapter 6

Stochastic Process on Graphs

In this section, we focus on stochastic process, which utilizes the power of randomness, and we're going to see how this helps us to design pseudorandom generator.

6.1 Markov Chains

We're going to study an important random model called Markov chain, which helps us understand the so-called stochastic process.

Definition 6.1.1 (Probabilistic finite state machine)**.** Let Ω be the set of states, and a *probabilistic finite state machine* is a finite state machine such that its transition function is a distribution.

We'll not really discuss probabilistic finite state machine, but this is how the concept stochastic process arises in randomized complexity. Now, given a probabilistic finite state machine, consider the sequence of visited states $x_0, x_1, \ldots, x_t \in \Omega$ which is sampled from the sequence of random variables $X_0, \ldots, X_t$, we then have the following definition which characterize a class of such sequences.

Definition 6.1.2 (Markov chain)**.** A *Markov chain* is a sequence of random variables $X_0, \ldots, X_t \in \Omega$ such that

$$\Pr(X_{t+1} = x_{t+1} \mid X_0 = x_0, X_1 = x_1, \ldots, X_t = x_t) = \Pr(X_{t+1} = x_{t+1} \mid X_t = x_t)$$

and $P(x, y) = \Pr(X_{t+1} = y \mid X_t = x)$ for P defined as $P(x, y) = \Pr(x \text{ transitions to } y)$.

Notation (Trnasition matrix)**.** The P defined in Definition 6.1.2 is called a *transition matrix*.

We see that Markov chain requires that given a sequence of random variables, and the next state of this sequence at time only depends on the current state, i.e., it does not depend on any of the states before the current state. Furthermore, the second condition states that the transitions are independent of time.

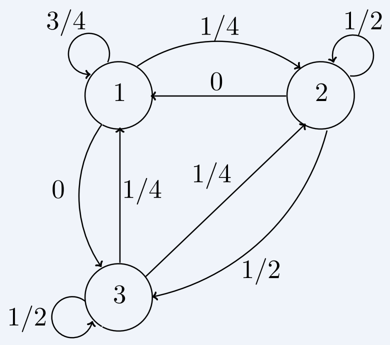
Definition. Let P be a random matrix.

Definition 6.1.3 (Row (column) stochasitc)**.** If all the rows (columns) of P sums to 1, we say P is *row (column) stochastic.*

Definition 6.1.4 (Doubly stochastic)**.** If P is both row and column stochastic, then we say P is *doubly stochastic.*

Notice that the transition matrix P for a Markov chain, the rows must sum to 1, hence P is row stochastic.

Example. The transition matrix for the following Markov chain is doubly stochastic.



Proof. We see that

$$P = \begin{bmatrix} 3/4 & 1/4 & 0 \\ 0 & 1/2 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{bmatrix},$$

which is clearly doubly stochastic. ⊛

6.1.1 Random Walks

Now, a Markov chain can be though of doing a random walk over the set of states, i.e., Ω. Specifically, we can interpret a random walk over Ω as doing a random walk over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. We see that a walk contains

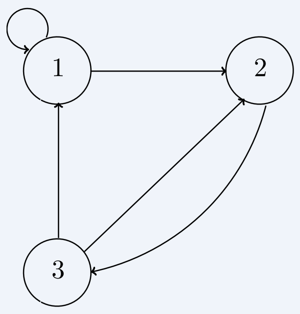
- A sequence of $S_0, S_1, \ldots, S_n$ of nodes where S_0 is the starting vertex.
- At each step i, S_{i+1} is picked uniformly from the neighbors set $\Gamma(S_i)$.

Let $d(i)$ is the out degree of node i, we then see that

$$P(i,j) = \begin{cases} 1/d(i), & \text{if } (i,j) \in \mathcal{E}; \\ 0, & \text{otherwise.} \end{cases}$$

Note. We immediately see that $\sum_j P(i,j) = 1$.

Example. Consider the following random walk.



We see that the corresponding transition matrix P is

$$P = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \end{bmatrix}.$$

A classical question is the following.

Problem. What happens after t time steps? In other words, what's the distribution at time t?

Answer. To see this, we observe that the t-step transition matrix $P^t(x, y)$ is calculated as

$$P^t(x,y) = \begin{cases} P(x,y), & \text{if } t = 1; \\ \sum_z P(x,z)P^{t-1}(x,y), & \text{if } t > 1. \end{cases}$$

Write it in the matrix form, we have

$$P^t = \underbrace{P \times P \times \ldots \times P}_{t \text{ times}}.$$

Now, given an initial distribution $\pi^{(0)} = (\pi_1^{(0)}, \ldots, \pi_n^{(0)})$ where $\pi_i^{(0)}$ is the probability of starting at state i, we see that after one step, this becomes

$$\pi^{(1)} = \pi^{(0)} \times P = \left(\sum\nolimits_z \pi_Z^{(0)} \cdot P(z,1), \sum\nolimits_z \pi_z^{(0)} \cdot P(1,z), \ldots\right).$$

So for t steps, the matrix product becomes $\pi^{(t)} = \pi^{(0)} \times P^t$. ⊛

Lecture 15: Fundamental Theorem of Markov Chains

For random walks over graphs, the matrix P is doubly stochastic if the graph is regular. For an undirected graph, this implies that all the degrees are equal; for a directed graph, this implies that all the in-degrees and out-degrees are equal to a constant d. 20 Oct. 10:30

6.2 Fundamental Theorem of Markov Chains

As previously seen. We introduce the Markov chains last time, and this lecture discusses Markov chains in more detail, discussing the following:

- Properties that Markov chains can possess.
- Stationary distributions.
- The fundamental theorem of Markov chinas.
- Hitting and commute times.

6.2.1 Properties of Markov Chains

Let's start with irreducibility.

Definition 6.2.1 (Irreducible)**.** A Markov chain is *irreducible* if for all $x, y \in \Omega$, there exists a t such that

$$P^t(x,y) > 0.$$

Intuition. Irreducibility essentially means that we can get from any state to any other state (in principle). These irreducible graphs are also called *strongly connected.*

Definition 6.2.2 (Ergodic)**.** A Markov chain is *ergodic* if for there exists a t_0 such that for all $t > t_0$,

$$P^t(x,y) > 0$$

for all $x, y \in \Omega$.

Intuition. Ergodicity essentially means that if we wait for a long enough time, you can get from one node to another.

Remark. The ergodic property is stronger than the irreducible property since it requires all times t after t_0 to have a nonzero probability P^t rather than just one time t. For example, bipartite graphs can be irreducible but not ergodic. A random walk over a bipartite graph would visit each side every other time step, making ergodicity impossible.

Definition 6.2.3 (Aperiodic)**.** A Markov chain is *aperiodic* if for all $x \in \Omega$,

$$\gcd(\{t : P^t(x,x) > 0\}) = 1.$$

Intuition. Aperiodicity essentially means that no matter where we start, if we take all lengths of time wherein it is possible to go from node x back to x, there is no common divisor. This prevents the bipartite example discussed in the previous paragraph.

A common way to avoid the issues with periodicity is to add self loops (directed edges from a node back to itself).

Remark. Essentially, ergodicity is irreducibility plus aperiodicity.

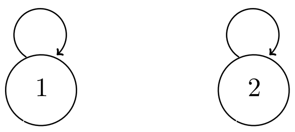
6.2.2 Stationary Distribution

We're interested in the long-term behavior for a Markov chain, specifically, we can characterize this by introducing the stationary distribution.

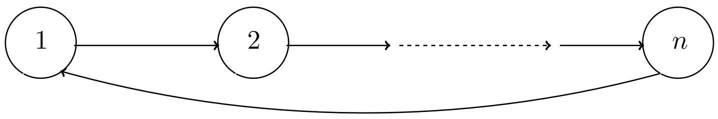
Definition 6.2.4 (Stationary distribution)**.** Given a Markov chain with its transition matrix P, π is *stationary* if $\pi P = \pi$.

Equivalently, If we define π' as $\pi'(y) = \sum_x \pi(x)P(x,y)$, then π is stationary if $\pi = \pi'$. In other words, π is stationary if, after applying P for one time step, we get π.

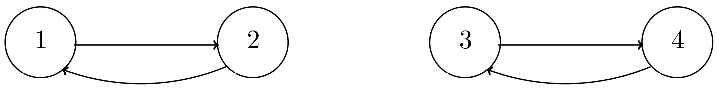
A Markov chain can have more than one stationary distribution. For example, look at the following example of a reducible Markov chain. The two stationary distributions are $(0,1)$ and $(1,0)$.



A periodic Markov chain can have zero, one, or more stationary distribution. For example, the following loop-shaped graph has only one stationary distribution, the uniform distribution.



Another example of a graph with multiple stationary distributions, $(1/2, 1/2, 0, 0)$ and $(0, 0, 1/2, 1/2)$. This graph is both reducible and periodic.



These intuition and observations leads to the following fundamental theorem for Markov chains.

Theorem 6.2.1 (Fundamental theorem of Markov chains)**.** Any finite, irreducible, aperiodic Markov chain satisfies the following:

- It is ergodic.
- There exists a unique stationary distribution π
- Let $h_{ij} = \mathbb{E}[\#$ of steps to go from state i to $j]$. Then for all i, $h_{ii} = 1/\pi(i)$.[a]
- Let $f_{ij} = \Pr($starting from i, at some point get to $j)$. Then for all i, $f_{ii} = 1$.[b]
- $\lim_{t\to\infty} N(i,t)/t = \pi(i)$, where $N(i,t)$ is the number of times state i is reached in t time steps.[c]

[a]This is true for any starting vertex.
[b]This is also true for any starting vertex.
[c]This is true regardless of the starting distribution.

Remark (Useful facts)**.** As we mentioned before, if P is doubly stochastic, then π is uniform. For graph random walks, if the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is not regular, then the stationary distribution is proportional to the degree, i.e.,

$$\pi = \left(\frac{\deg(v_1)}{2\left|\mathcal{E}\right|}, \frac{\deg(v_2)}{2\left|\mathcal{E}\right|}, \dots, \frac{\deg(v_n)}{2\left|\mathcal{E}\right|} \right).$$

Proof. The intuition is that if a vertex is a bottleneck (meaning it has a low degree), it is not visited frequently. This makes its element in the stationary distribution and its expected number of visits $N(i,t)$ relatively small. This also makes h_{ii}, the expected number of steps for a random walk to return to i, relatively large. ⊛

6.3 Hitting, Commute, and Cover Times

6.3.1 Hitting and Commute Times

In the fundamental theorem of Markov chains, h_{ij} defined there is actually quite important, hence we give it a name.

Definition 6.3.1 (Hitting time)**.** The *hitting time* h_{ij} is defined as the expected time to get to state j starting from state i, i.e.,

$$h_{ij} = \mathbb{E}\left[\text{time to get to } j \text{ starting form } i\right].$$

Remark. From the fundamental theorem of Markov chains,

$$h_{ii} = \frac{1}{\pi(i)} \Rightarrow h_{ii} = \frac{2\left|\mathcal{E}\right|}{\deg(i)}.$$

Another interesting quantity we're interested in is the so-called commute time.

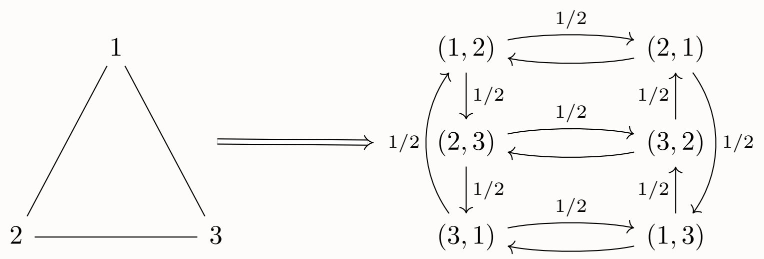
Definition 6.3.2 (Commute time)**.** The *commute time* C_{ij} is defined as the expected time to transition from state i to state j, and then return to state i, i.e.,

$$C_{ij} = \mathbb{E}\left[\text{time to go from } i \text{ to } j \text{ and back to } i\right].$$

Remark. We see that $C_{ij} = h_{ij} + h_{ji}$ form the linearity of expectation.

Theorem 6.3.1. If $(u,v) \in \mathcal{E}$, then $C_{uv} \leq 2\left|\mathcal{E}\right|$.

Proof. Consider a line graph, that is, take a graph and replace each edge by two directed edges where the vertices of the line graph are these directed edges. For example, the following is the line graph of the triangle.



Assume this line graph is ergodic. If it is not, add self loops. Given the original transition matrix P, we now have a new transition matrix Q where

$$Q_{(u,v)(v,w)} = P_{vw} = \frac{1}{\deg(v)},$$

we see that this transition matrix is doubly stochastic since for all $(v, w) \in \mathcal{E}$,

$$\sum_{(x,y)\in\mathcal{E}} Q_{(x,y)(v,w)} = \sum_{u\in\Gamma(v)} Q_{(u,v)(v,w)} = \deg(v) \cdot \frac{1}{\deg(v)} = 1,$$

implying that the stationary distribution of Q is uniform over $2\,|\mathcal{E}|$ edges. Now consider two events, A and B, defined as

- A: number of steps required to go from u to v and back to u (from v) via the edge $v \to u$.
- B: the random walk enters u in the beginning via the edge $v \to u$.

Claim. $C_{uv} = h_{uv} + h_{vu} \leq \mathbb{E}[A]$.

Proof. This is because A rule out lots of *efficient paths*: no matter how close you get to u from doing the second random walk from v, you need to go back to v in order to go to u again. ⊛

Finally, observe that $\mathbb{E}[A]$ conditioning on B does not change the expectation since the transition probabilities of a Markov chain are only dependent upon the current state, hence we have

$$C_{uv} \leq \mathbb{E}[A] = \mathbb{E}[A \mid B] = h_{(u,v)(v,u)} = \frac{1}{\pi(u,v)} = 2\,|\mathcal{E}|\,,$$

proving the result.[a] ■

[a]Recall that $h_{(u,v)(v,u)}$ is defined in the Markov chain Q.

Lecture 16: Cover Times and its Applications

25 Oct. 10:30

As previously seen. Facts about finite Markov chains:

- Every finite Markov chain has a stationary distribution.
- Every irreducible finite Markov chain has a unique stationary distribution.

6.3.2 Cover Times

Finally, we look at the so-called cover time.

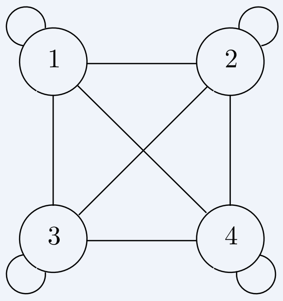
Definition 6.3.3 (Cover time). The *cover time* $\mathbb{C}_u(\mathcal{G})$ *of* u is the expected time it takes to start from $u \in \mathcal{V}$ and visit all other vertices, i.e.,

$$\mathbb{C}_u(\mathcal{G}) = \mathbb{E}[\# \text{ of steps to reach all nodes in } \mathcal{G} \text{ on walk starting from } u].$$

Also, the *cover time* $\mathbb{C}(\mathcal{G})$ *of* $\mathcal{G}$ is defined as $\max_u \mathbb{C}_u(\mathcal{G})$.

This is an interesting problem, as demonstrated by the following three examples.

Example. K_n^* is a complete graph over n vertices with self loops.

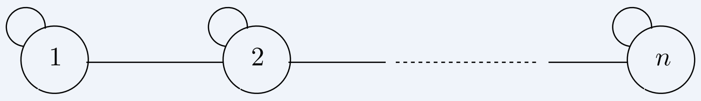


Proof. We see that

- For a random walk on this graph, we visit any node in the graph with equal probability.
- The cover time of this graph is $\mathbb{C}(K_n^*) = \Theta(n \log n)$.
- For further information, see the coupon collector problem.

⊛

Example. L_n^* is a line with self loops.

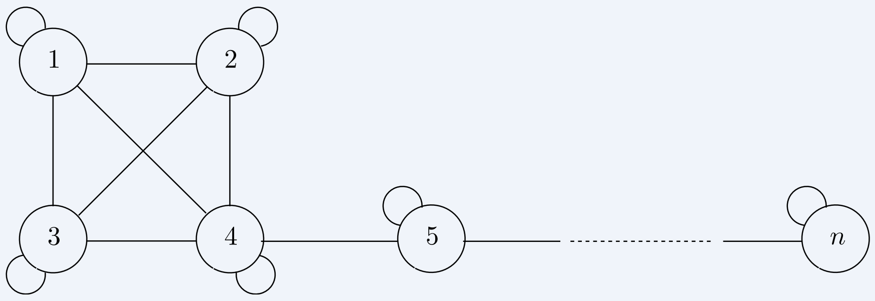


Proof. We see that

- A random walk on this graph is an example of Brownian Motion. Basically, it takes a while to visit every node because the walk can step forward and backward frequently.
- The cover time of this graph is $\mathbb{C}(L_n^*) = \Theta(n^2)$.

⊛

Example. A lollipop graph with self loops.



Proof. We see that

- This graph occurs when you connect the graphs from the previous two examples.
- This is the worst case scenario. If you start the walk in the complete portion, it is very difficult to get to the end of the line.
- The cover time of this graph is $\mathbb{C}(\text{Lollipop}) = \Theta(n^3)$.

⊛

Theorem 6.3.2. For any connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the cover time is never more than cubic, i.e.,

$$\mathbb{C}(\mathcal{G}) \leq 2\,|\mathcal{E}|\,(|\mathcal{V}| - 1) = O(|\mathcal{V}|\,|\mathcal{E}|) = O(|\mathcal{V}|^3).$$

Proof. Let T be a spanning tree of $\mathcal{G}$, let $n = |\mathcal{V}|$, $m = |\mathcal{E}|$. Take an Eulerian tour on the directed version of T (where each edge is replaced by two directed edges), then we have

$$v_0 \to v_1 \to \dots \to v_{2n-2} = v_0.$$

Start a walk w from v_0 that visits $v_1 \to v_{2n-3}$. This visits all vertices, and the cover time starting from v_0 is

$$\mathbb{C}_{v_0}(\mathcal{G}) \leq \sum_{j=0}^{2n-3} h_{(v_j, v_{j+1})} = \sum_{(u,v)\in T} C_{uv},$$

where the first inequality follows from summing up the hitting time from each vertex to next in tour, and the equality follows from that this is an upper bound on time to cover the whole graph. Now, we apply Theorem 6.3.1, i.e., $C_{uv} \leq 2m$ (for $(u, v) \in \mathcal{E}$), which implies

$$\mathbb{C}_{v_0}(\mathcal{G}) \leq |T| \times 2m = 2(n-1)m$$

for an arbitrary v_0, proving the result. ■

Turns out that there's a well-known application which solves the USTCON with efficient space requirement. Hence, to better characterize it, we introduce two more complexity classes.

Definition 6.3.4 (Logarithmic-space)**.** The complexity class L is defined as $L \in \mathsf{L}$ iff there is a deterministic polynomial time, logarithmic space algorithm $\mathcal{A}$ such that

- if $x \in L$, $\mathcal{A}(x) = 1$ and
- if $x \notin L$, $\mathcal{A}(x) = 0$.

Same as RP, we have the so-called RL.

Definition 6.3.5 (Randomized logarithmic-space)**.** The complexity class RL is defined as $L \in \mathsf{RL}$ if there exists a probabilistic polynomial time, logarithmic space algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) \geq 1/2$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 1) = 1$.

Now, we look at the problem formally.

Problem 6.3.1 (USTCON)**.** Given an undirected graph $\mathcal{G}$ and vertices $s, t \in \mathcal{V}$, *USTCON (undirected s-t connectivity)* asks whether there is a path from s to t.

We know that this is easy enough to solve in linear time using BFS, DFS, Dijkstra, etc. But these algorithms also take linear space in $|\mathcal{E}|$, and by using what we have discussed, we can do better.

Algorithm 6.1: USTCON – Random Walks

```
Data: An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $s, t \in \mathcal{V}$
Result: True if $s$ and $t$ are connected, False otherwise
$x_0 \leftarrow s$
for $i = 1, \dots, T$ do
    $x_i \leftarrow$ RandWalk($x_{i-1}$)                // One-step random walk from $x$
    if $x_i = t$ then
        return True
return False
```

Theorem 6.3.3. USTCON is in RL.

Proof. We see that USTCON can be solved by Algorithm 6.1 with one-side error using $\log n$ space since all we need to store is where we are in the graph, which takes $\log(n)$ space for n vertices. Therefore, this algorithm requires $O(\log n)$ working memory. Now, we're interested in how large should T be to reach a good enough error probability? We've learned that $T = O(n^3)$ (the cover time) suffices. The expected hitting time is then $h_{st} = O(n^3)$, so by Markov inequality, if t is reachable, we have

$$\Pr(\text{fail}) = \Pr(t \text{ isn't reached in } T \text{ steps}) \leq \frac{h_{st}}{T} \leq \frac{1}{2}$$

for $T = 2h_{st} = O(n^3)$. ■

Remark. Actually, USTCON is in L, meaning there is a deterministic algorithm that takes logarithmic space to solve this problem, by derandomizing the random walk. This is done by converting any graph to the highly-connected version of itself, by turning it into an expander graph [Rei08].

Lecture 17: Expander Graphs

6.4 Expander Graphs

27 Oct. 10:30

Expander graphs can mimic a complete graph with only a few edges, i.e., they have the high-connectivity of a complete graph with only a constant degree!

Remark. This allows them to have small degree; in particular, the degree of an expander graph can be 3, which is not possible for most complete graphs.

The most interesting property for an expander graph is probably the following.

(a) Complete graphs have around n^2 edges, and a random walk quickly converges to the uniform distribution.

(b) Expander graphs have only a linear number of edges due to constant degrees, but a random walk also converges to the uniform distribution quickly.

Expander graphs also possess other properties of complete graphs, such as the fact that the distance between any two vertices (its diameter) is small. And before we introduce the definition formally, we first see the motivation.

Note (Regular expander)**.** While expander graphs are always directed, they are not always regular. However, regular expander graphs are much easier to work with notation-wise, so we will only consider regular expander graphs in this course.

For a more thorough and detailed discussion, see professor Thatchaphol Saranurak's graph course on fast graph algorithm.

6.4.1 Spectral Expander Graphs

Consider an undirected d-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and its associated normalized adjacency matrix A.[1] This makes A symmetric and doubly stochastic, so A defines the Markov chain of random walks over $\mathcal{G}$.

Our main focus in this section is about random walk on such a graph. If we take this random walk starting from a distribution $p = (p_1, \dots, p_n)^\top$ on $\mathcal{V}$ and taking ℓ steps, we get the distribution $A^\ell \cdot p$.[2]

Remark. A uniform distribution is a stationary distribution.

Proof. Since the graph is d-regular, $\mathbb{1} = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \dots & \frac{1}{n} \end{bmatrix}^\top$ is a stationary distribution since $\mathbb{1} = A \cdot \mathbb{1}$ because A is doubly stochastic. ⊛

We see that $\mathbb{1}$ is the eigenvector with eigenvalue 1 for A, which suggests that we look into its eigenvalues and eigenvectors of A. Firstly, since A is symmetric, by spectral theorem, all eigenvalues are real numbers, and eigenvectors form an orthogonal basis for $\mathbb{R}^n$.

Sorting the eigenvalues respect to their absolute value, i.e., $|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$, and observe the following.

Claim. $\lambda_1 = 1$.

Proof. Observe that any symmetric stochastic matrix has eigenvalues in $[-1, 1]$, and hence $|\lambda_1| \leq 1$. With the fact that we already found one eigenvalue with value 1, $\lambda_1 = 1$ with eigenvector $\mathbb{1}$. ⊛

Now, since A is symmetric, the absolute values of eigenvalues are the same as the singular values, hence the second-largest eigenvalue of $\mathcal{G}$, denoted by $\lambda(\mathcal{G})$, is

$$\lambda(\mathcal{G}) \coloneqq |\lambda_2| = \max_{\substack{v \in \mathbb{1}^\perp \\ v \neq 0}} \frac{\|A \cdot v\|_2}{\|v\|_2} = \max_{\substack{v \in \mathbb{1}^\perp \\ \|v\|_2 = 1}} \|A \cdot v\|_2,$$

where $\mathbb{1}^\perp = \{v \in \mathbb{R}^n \colon \langle v, \mathbb{1} \rangle = 0\}$.

Notation (p-norm)**.** $\|\cdot\|_p$ is the *p-norm* defined by $\|v\|_p^p = \sum_i |v_i|^p$ for $v \in \mathbb{R}^n$.

Definition 6.4.1 (Spectral gap)**.** Given a graph $\mathcal{G}$, the *spectral gap of* $\mathcal{G}$ is defined as $1 - \lambda(\mathcal{G})$, which is the difference between the first and second eigenvalues.

As you might already notice, the upshot of having small $\lambda(\mathcal{G})$ is because that this leads to expander graphs. This definition is purely algebraic since we only look at the spectral gap, hence we call such a graph an spectral expander graph.

Definition 6.4.2 (Sepctral expander)**.** A d-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$ is an *spectral (n, d, λ)-expander* if $\lambda(\mathcal{G}) \leq \lambda$.

Remark (Maximum spectral gap)**.** The spectral gap is maximized when $\lambda(\mathcal{G}) = 0$, which occurs when $\mathcal{G}$ is complete with self-loops.

Proof. In this case, the adjacency matrix is defined as

$$A = \frac{1}{n} \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}.$$

⊛

[1] We normalize A by dividing it by d, the regular degree of $\mathcal{G}$.

[2] Notice that instead of applying A from the right as πA as before, we can apply A from the left since now A is symmetric.

Remark (Minimum spectral gap)**.** The spectral gap is minimized when $1 - \lambda(\mathcal{G}) = 0$, which occurs when $\mathcal{G}$ is disconnected or bipartite.

Proof. Specifically, if $\mathcal{G}$ is bipartite (with equal parts), eigenvalues come in $\pm$ pairs (for each $\lambda, -\lambda$ is also an eigenvalue). And since $\lambda_1 = 1$, we know that $|\lambda_2| = 1$ since there exists another $\lambda_i = -1$. ⊛

In general, we want to maximize the spectral gap because the larger the gap is, the faster we converge to the stationary distribution, which is another motivation to have an expander graph: since larger the gap $(1 - \lambda(\mathcal{G}))$ is, smaller the $\lambda(\mathcal{G})$ is.

Intuition. This can be seen from the fact that smaller $\lambda(\mathcal{G})$, the more connected the graph is, the extreme case being the complete graph.

Recall that we're interested in regular graph, hence we ask the following question.

Problem. If $\mathcal{G}$ is d-regular, what's the smallest possible $\lambda(\mathcal{G})$?

Answer. Turns out that in this case, we have

$$\lambda(\mathcal{G}) \geq \frac{2\sqrt{d-1}}{d}(1 - o(1)) \to \frac{2\sqrt{d-1}}{d}$$

for $n \to \infty$. This is known as the Alon-Boppana bound [Nil91]. ⊛

Definition 6.4.3 (Ramanujan graph)**.** A family of graphs $\mathcal{G}$ is *Ramanujan* if as $n \to \infty$,

$$\lambda(\mathcal{G}) = \frac{2\sqrt{d-1}}{d}.$$

Definition 6.4.4 (Lubotzky-Phillips-Sarnak graph [LPS88])**.** The *Lubotzky-Phillips-Sarnak graph* is a fully explicit construction of Ramanujan graph for $d = p+1$ with p being prime and $p \equiv 1 \pmod 4$.

Remark. The construction is based on some deep number theoretical statements called Ramanujan-Petersson conjecture. This was extended to p being power of prime by Morgenstern [Mor94].

Furthermore, the existence of Ramanujan graphs for all degrees d was proved recently [MSS13],[a] and the polynomial construction of which was given by Cohen [Coh16].[b]

[a]A caveat is that this is only for bipartite graphs, and their spectral gaps are defined using λ_3.

[b]But the construction is polynomial in graph size and thus is non-fully explicit, since a fully explicit construction requires the map from (v, i) to the i^{th} neighbor of v to be computable in polynomial time in input length $|(v, i)|$.

In particular, as we noticed before, a random walk converges to the uniform distribution quickly when $\lambda(\mathcal{G})$ is small, as guaranteed by Theorem 6.4.1.

Theorem 6.4.1. For all initial distribution p, $\left\|A^\ell p - \mathbb{1}\right\|_2 \leq \lambda^\ell$.

Proof. Recall that

- $\forall v \in \mathbb{1}^\perp : \|Av\|_2 \leq \lambda \|v\|$ (from the definition of λ)
- $A \cdot \mathbb{1} = \mathbb{1}$
- A keeps $\mathbb{1}$ invariant, i.e., for all $v \in \mathbb{1}^\perp$, $Av \in \mathbb{1}^\perp$ since

$$\langle \mathbb{1}, Av \rangle = \langle A\mathbb{1}, v \rangle = \langle \mathbb{1}, v \rangle = 0.$$

Now, observe that $\lambda(A^\ell) \le \lambda^\ell$ since when $\ell = 2, v \in \mathbb{1}^\perp$, we have

$$\left\|A^2 v\right\|_2 \le \lambda \cdot \|Av\|_2 \le \lambda^2 \|v\|_2 .$$

Write $p = \alpha \cdot \mathbb{1} + p'$ where $p' \perp \mathbb{1}$. Then, because p is a probability distribution, we have

$$\langle p, \mathbb{1}\rangle = \frac{1}{n} \Rightarrow \alpha = \frac{\langle p, \mathbb{1}\rangle}{\langle \mathbb{1}, \mathbb{1}\rangle} = \frac{1/n}{1/n} = 1$$

and since $p' \perp \mathbb{1}$, we know that $p = \mathbb{1} + p'$, hence

$$A^\ell p = A^\ell \langle 1 + p'\rangle = \mathbb{1} + A^\ell p'.$$

By orthogonality, we use the Pythagorean theorem, which implies

$$\|p\|_2^2 = \|\mathbb{1}\|_2^2 + \|p'\|_2^2 \Rightarrow \|p'\|_2 \le \|p\|_2$$

Now, observe that for any vector, $\|p\|_2 \le \|p\|_1$, therefore, $\|p'\|_2 \le \|p\|_2 \le \|p\|_1 = 1$, implying that

$$\|A^\ell p - \mathbb{1}\|_2 = \|A^\ell (p-1)\|_2 \le \lambda^\ell \|p'\|_2 \le \lambda^\ell .$$

■

Using this, we can say that for $\lambda < 1$ (by a constant), a random walk converges to the uniform distribution exponentially fast. In particular, if $\mathcal{G}$ is regular and connected with self-loops (or more generally, not bipartite), then $\lambda(\mathcal{G}) \le 1 - 1/12n^2$ [AB09, Lemma 21.4]. This implies for any graph, as long as it's connected and not bipartite, a random walk of polynomial length, the distribution is going to converge to uniform distribution exponentially fast.

Lecture 18: Combinatorial Expander, Probability Amplification

We are going to take the combinatorial view of expander graphs to understand where the name comes from and how the graphs expand. 1 Nov. 10:30

6.4.2 Combinatorial Expander Graphs

Given a subset S of a d-regular graph, the neighborhood $\Gamma(S)$ of S can be either the vertices or the edges that connect S, and turns out that these two situations are both interesting enough to consider for the notion of expansion. For now, let's consider the edge case.

Definition 6.4.5 (Edge expander)**.** A d-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$ is a (n, d, p)-*edge expander* if for all $S \subseteq \mathcal{V}$ with $|S| \le n/2$, the fraction $|S|/|\Gamma(S)| \ge p$ for $\Gamma(S) \subseteq \mathcal{E}$.

Note. The reason we require $|S| \le n/2$ in Definition 6.4.5 is $\Gamma(S) = \Gamma(\mathcal{V} \setminus S)$, and we only want to consider the smaller side.

Indeed, one may observe that there is no reason to require an edge expander to be a d-regular graph. But if it is d-regular, then a (n, d, ρ)-expander requires that $|E(S, \overline{S})| \ge \rho \cdot d \cdot |S|$, where $\overline{S} = V \backslash S$ and ρ is some constant (say $\frac{1}{10}, \frac{1}{100}$, or even 10^{-10}).

Problem. Show that these exist a (n, d, ρ)-expander with $d = 7$, $\rho = \frac{1}{1000}$ or something similar.

Answer. Try to construct d-regular graphs randomly with the number of vertices going to infinity and then use probabilistic method with union bound and the probability of bad events. Turns out the largest ρ to hope for is (approaching) $1/2$ (degree grows with $1/2 - \rho$). ⊛

Here is a very bad example.

Example (Bad example)**.** The 2D grid with intersections as vertices is not a good expander.

Proof. Look at a t by t set S of vertices inside a 2D grid, there are $O(t)$ edges out of S, and $O(t^2)$ edges connected to S. This implies that if this were an expander we would want a constant fraction ρ of edges ($O(t^2\rho)$ edges) going out, but as we increase t, the fraction of edges going out decreases to 0. ⊛

We actually have a relation between spectral expanders and edge expanders.

Theorem 6.4.2. If $\mathcal{G}$ is a (n, d, λ)-spectral expander, then $\mathcal{G}$ is a $(n, d, \frac{1-\lambda}{2})$-edge expander.

Proof. See [AB09, §21.2.3, §21.2.4]. ■

We see that to have a good edge expander, we need smaller λ! The converse is generally not true since a bipartite graph can be a good edge expander while $\lambda(\mathcal{G}) = 1$. Any way, we still have the following.

Theorem 6.4.3. If $\mathcal{G}$ is a (n, d, ρ)-edge expander with self loops, then it's a $(n, d, 1-\epsilon)$-spectral expander where $\epsilon = \min(\rho^2/2, 2/d)$.

Note. If $\mathcal{G}$ doesn't have self loops and λ_2 is the second eigenvalue (without taking absolute value), $\lambda_2 \leq 1 - \rho^2/2$.

6.4.3 Error Reduction

An interesting and important application of expanders is to do error reduction for RP. Recall that we have the following methods to amplify the success probability in *yes case* to $1 - 2^{-k}$ in RP, assuming the algorithm takes m-bits random string at each execution, by

- Serial repetition: This requires $m \cdot k$ random bits with running the algorithm k times.
- 2-point sampling: This requires $m + O(k)$ random bits with running the algorithm 2^k times.

Now, we can actually achieve this with expanders and take advantages on both cases, i.e., using only $m + O(k)$ random bits with running the algorithm k times, we achieve $1 - 2^{-k}$ success probability! To do this, take a $(M := 2^m, d, 1/10)$- expander graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that is available to us with some constant d and the construction (or description) is explicit. Then, the algorithm runs as follows.

Algorithm 6.2: Error Reduction – Expander Walk

Data: An $(M := 2^m, d, 1/10)$-spectral expander graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, algorithm $\mathcal{A}$ for $L \in$ RP which takes m bits of randomness, input x for $\mathcal{A}$, k

Result: True if $x \in L$, False otherwise

```
1 $v_0$ ←rand($\mathcal{V}$)                         // $m$ random bits
2 for $i = 1, \dots, k$ do
3     if $\mathcal{A}(x, v_0)$ then
4         return True
5     $v_i$ ←rand($v_{i-1}$)                      // One-step random walk from $v$
6 return False
```

We see that Algorithm 6.2 is a one-sided error algorithm because if x is not in L, we are never going to see an execution that accepts x. Furthermore,

- If $x \notin L$, then Algorithm 6.2 rejects w.p. 1.
- If $x \in L$, we want to say $\Pr(\text{reject}) \leq 2^{\Omega(k)}$ by only using $m + O(k)$ random bits.

Actually, what we want is guaranteed by the following.

Theorem 6.4.4. If $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a $(M := 2^m, d, \lambda)$-sepctral expander graph such that $\lambda \leq 1/10$ and the number of *bad events*[a] B in $\mathcal{V}$ is less than $M/100$, with the length k random walk being

$v_0, v_1, \ldots, v_k$, $\Pr(v_0, \ldots, v_k \in B) \leq 2^{-\Omega(k)}$.

[a]This means the random bits which lead to the algorithm $\mathcal{A}$ fails.

Proof. Let's create a diagonal matrix N which models the bad events set B such that

$$N_{r,r} = \begin{cases} 1, & \text{if } r \in B; \\ 0, & \text{otherwise.} \end{cases}$$

Then, given any distribution $q \in \mathbb{R}^M$, applying N on q will take the q conditioned on landing in B, i.e.,

$$\|Nq\|_1 = \Pr_{r \leftarrow q}(r \in B).$$

With this interpretation, we see that

$$\begin{aligned} \|NAq\|_1 &= \Pr_{r \leftarrow q}(\text{start at } r \text{ take 1 steps, land in } B) \\ \|(NA)^2 q\|_1 &= \Pr_{r \leftarrow q}(\text{start at } r \text{ take 2 steps, land in } B \text{ both times}) \\ &\vdots \\ \|(NA)^\ell q\|_1 &= \Pr_{r \leftarrow q}(\text{start at } r \text{ take } \ell \text{ steps, land in } B \text{ every time}) \end{aligned}$$

With Lemma 6.4.1 (we're going to prove this next time), consider the initial uniform distribution as p_0, we see that

$$\Pr(v_1, \ldots, v_k \in B) = \|(NA)^k p_0\|_1 = \sqrt{M}\|(NA)^k p_0\|_2 \leq 5^{-k}\|p_0\|_2 = \sqrt{M}\frac{1}{5^k}\frac{1}{\sqrt{M}} = 1/5^k,$$

where we use the Cauchy-Schwartz inequality to show that for all $v \in \mathbb{R}^n$, $\|v\|_1 \leq \sqrt{n}\|v_2\|_2$. This shows that the failure probability is exponentially small in k, as desired. ■

Lecture 19: Shamir's Secret Sharing

We now prove the omitted lemma. 1 Nov. 10:30

Lemma 6.4.1. For any distribution q, $\|NAq\|_2 \leq \|q\|_2/5$ for N defined in Theorem 6.4.4.

Proof. Consider the spectral decomposition whose eigenvectors are $v_1, \ldots, v_M$ and eigenvalues are $\lambda_1, \ldots, \lambda_M$ such that $\|v_i\|_2 = 1$, $v_1 = \frac{1}{\sqrt{M}}\mathbb{1}$ with $\lambda_1 = 1$. From linear algebra, $\{v_i\}_{i=1}^M$ forms a basis, hence for any distribution q, we write

$$q =: \sum_{i=1}^M \alpha_i v_i$$

with $\|q\|_2^2 = \sum_{i=1}^M \alpha_i$. We see that

$$|NAq|_2 = \left\| \sum_{i=1}^M NA; a_i v_i \right\|_2 = \left\| \sum_{i=1}^M \alpha_i \lambda_i N v_i \right\|_2 \leq \|\alpha_i \lambda_i N v_1\|_2 + \left\| \sum_{i=2}^M \alpha_i \lambda_i N v_i \right\|_2,$$

where the last step follows from the triangle inequality. Since $\lambda_1 = 1$ and $|B| \leq M/100$,

$$\|\alpha_1 \lambda_1 N v_1\|_2 = |\alpha_1| \sqrt{\sum_{i \in B} \frac{1}{M}} = |\alpha_1| \sqrt{|B|/M} = \frac{|\alpha_1|}{10} \leq \frac{\|q\|_2}{10}.$$

Now, since for all $w \in \mathbb{R}^M$, we have $\|Nw\|_2 = \sqrt{\sum_{i\in B} w_i^2} \leq \|w\|_2$, with $\lambda_i \leq 1/10$,

$$\left\|\sum_{i=2}^{M} \alpha_i \lambda_i N v_i\right\|_2 \leq \left\|\sum_{i=2}^{M} \alpha_i \lambda_i v_i\right\|_2 = \sqrt{\sum_{i=2}^{M} (\alpha_i v_i)^2} \leq \sqrt{\sum_{i=2}^{M} \frac{\alpha_i^2}{100}} \leq \frac{\|q\|_2}{10}.$$

Combining both inequalities, we have

$$\|NAq\|_2 \leq \|\alpha_1 \lambda_1 N v_1\|_2 + \left\|\sum_{i=2}^{M} \alpha_i \lambda_i N v_i\right\|_2 \leq 2\frac{\|q\|_2}{10} = \frac{\|q\|_2}{5}.$$

■

So we have "best of both world" probability amplification for RP by using Algorithm 6.2. Can we do similarly for BPP? In this case, one difficulty is that we may find errors on two sides (i.e., both false negatives and false positives).

As previously seen. Looking back, our strategy was to take many samples and use a majority vote of those samples. The exact way we analyze this majority vote strategy was to use a Chernoff-Hoeffding bound to show with high probability the majority vote is right provided we have enough samples.

The strategy generalizes from RP to BPP, but the analysis uses expander-Chernoff bounds, which states the following.

Theorem 6.4.5 (Expander-Chernoff bound)**.** Let $\mathcal{G}$ be a (n, d, λ)-spectral expander, and fix $B \subseteq [n]$ such that $|B| = \beta n$. Given a random walk $v_1, v_2, \ldots, v_k \in [n]$ in $\mathcal{G}$, define indicator random variables $X_i = 1$ if $v_i \in B$ such that $\mathbb{E}[X_i] = |B|/n$, and define $X = \sum_i X_i$, so $\mathbb{E}[X] = \beta k$. Then we have that

$$\Pr(|X - \beta k| > \delta k) \leq 2\exp\left(-\frac{(1-\lambda)\delta^2 k}{4}\right).$$

Remark. We see that The Chernoff bounds hold even if the randomness is drawn from a random walk on an expander graph, which may induce dependencies in the randomness!

In other words, the probability of the majority nodes encountered in our walk being bad randomness (those which would cause our BPP algorithm to return an incorrect result) is exponentially low. One intuition for why the expander-Chernoff bound hold is because after $\log n$ steps, the distribution of the walk on the vertices is uniform. Notably, when λ is a constant, this is asymptotically the same as our standard Chernoff-Hoeffding bound we have seen before - even though our graph can be of low degree.[3]

[3] Check here to see the proof and more awesome facts.

Chapter 7

Information Theoretic Cryptography

7.1 Threshold Secret Sharing

Suppose you have some secret $s \in \mathbb{F}_q$, and you would like to share this secret among a board of n directors. We cannot trust any individual or small subset of the directors, but we would like that if there are sufficiently many directors working together, they can recover s. This is modeled by the so-called t-out-of-n secret sharing scheme.

Definition 7.1.1 (t-out-of-n secret sharing scheme)**.** Given any secret $S \in \mathbb{F}_q$, the *t-out-of-n secret sharing scheme* satisfies both correctness and security properties given n shares $S_1, \dots, S_n \in \mathbb{F}_q$.

Definition 7.1.2 (Correctness)**.** Any t shares can recover the secret S.

Definition 7.1.3 (Security)**.** Any $t-1$ shares reveal nothing about the secret S.[a]

[a]i.e., any collection of $t-1$ shares takes some distribution invariant of the secret S.

Note. Actually, the t-out-of-n secret sharing schemes are quite applicable in reality: many papers in cryptography take a threshold secret sharing scheme as a black-box.

As a first step, let consider the case that $t = n$, i.e., we want an n-out-of-n secret sharing schemes, which is often called the XOR secret sharing. The basic idea is to try to generate shares such that they sum up to S^*.

Example (XOR secret sharing)**.** The *XOR secret sharing scheme* is described as the following. Given a secret $S^* \in \mathbb{F}_q$, generate $n-1$ shares $S_1, S_2, \dots, S_{n-1}$ independently and uniformly at random, and define the last share $S_n \coloneqq S^* - \sum_{i=1}^{n-1} S_i$. This is a valid n-out-of-n secret sharing scheme.

Proof. We see that this scheme satisfies correctness since the sum of the shares is exactly S^* given $t = n$. As for security, consider fixing any $n-1$ shares and index this set by T, we need to show that for any choice of S^*, the distribution of those $n-1$ variables $\{S_i\}_{i\in T}$ would be the same, and hence observing them reveals nothing. To show this, we compute the probability over the randomness of generating the $n-1$ shares uniformly at random, i.e.,

$$\Pr(S_i = \sigma_i \colon i \in T) = \frac{\#\{(S_1, \dots, S_n) \colon \forall i \in T, S_i = \sigma_i \text{ and } \sum_i S_i = S^*\}}{q^{n-1}} = \frac{1}{q^{n-1}},$$

where the numerator is 1 since we have a unique solution. This shows that $(S_i)_{i\in T}$ is the uniform distribution, hence satisfies the security property. ⊛

7.1.1 Shamir's Secret Sharing Scheme

Shamir uses the powerful property of polynomials to design his secret sharing mechanism. [Sha79], described as follows. Assume that $q > n$ (this is a necessary condition for any t-out-of-n threshold secret sharing scheme).

Definition 7.1.4 (Shamir's secret sharing scheme)**.** Given any secret $S^* \in \mathbb{F}_q$, the *Shamir's secret sharing scheme* is a t-out-of-n threshold secret sharing scheme which generates n shares by evaluating $S_i = p(x_i)$ on distinct nonzero points $x_1, \ldots, x_n \in \mathbb{F}_q \setminus \{0\}$[a] with $p(x)$ being

$$p(x) = S^* + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1},$$

where $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$ are chosen independently and uniformly at random.

[a]This is where we need $q > n$. And note that $S^* = p(0)$, so we must choose the x_i to be nonzero points.

Note. $x_1, x_2, \ldots, x_n$ are common information, i.e., everyone knows.

Intuition. Suppose we have a line (i.e., $\deg(p) = 1$). To uniquely reconstruct the line, we need 2 points on the line (by evaluation). In general, any k points uniquely determine a degree $k-1$ polynomial since $k-1$ points are insufficient to uniquely determine a degree $k-1$ polynomial. This is the core idea which appears in Shamir's t-out-of-n secret sharing scheme.

We now show that Shamir's secret sharing scheme is a valid t-out-of-n threshold secret sharing scheme. First, we show the correctness. Fix t evaluations (x_i, S_i) for $i \in T$. Without loss of generality by relabelling, suppose that $T = \{1, \ldots, t\}$. Then, we have the following system of linear equations to describe the shares.

$$\underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_t & x_t^2 & \ldots & x_t^{t-1} \end{pmatrix}}_{\text{Evaluation points } \boldsymbol{X}} \underbrace{\begin{pmatrix} S^* \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix}}_{\text{Coefficients } \vec{c}} = \underbrace{\begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_t \end{pmatrix}}_{\text{Shares } \vec{S}} .$$

We can invert the square Vandermonde matrix $\boldsymbol{X}$ because x_i are distinct nonzero elements of $\mathbb{F}_q$.[1] Then, we extract the first coordinate of $\vec{c}$ to obtain the secret S^*.

To show security, without loss of generality, suppose we have the value of this polynomial on $t-1$ points $x_1, x_2, \ldots, x_{t-1}$ which means we miss one equation. Hence, there are t unknowns $S^*, a_1, \ldots, a_{t-1}$ and $t-1$ equations. Fixing S^*, there are $t-1$ unknowns and with the $t-1$ equations based on the Vandermonde matrix property, there is a unique solution. This implies that we have a uniform distribution on $S_1, S_2, \cdots, S_{t-1}$, i.e., for any choice of S^* the distribution of shares $S_1, S_2, \ldots, S_n$ is $(t-1)$-wise independent.

Remark. Recall homework 3! This is exactly what we have shown.

7.2 Privacy Amplification

Two users, Alice and Bob, have never met each other, but need to share a secret key (say n bits) between each other. However, an eavesdropper learns t bits of the communication between Alice and Bob. This compromises the privacy of their communication. How can Alice and Bob still agree on a new uniform key in without knowing what specific bits the eavesdropper knows?

Lecture 20: Privacy Amplification via Extractors

8 Nov. 10:30

[1]In other words, its determinant is non-zero.

As previously seen. Recall the privacy amplification setup, where Alice and Bob can communicate using bit strings through a channel, while some bits can be lost. The goal is to make this situation secure again, which is to amplify the privacy and derive the key.

Alice first uses a generator G with randomness input to process the key k into s and Bob will use the inverse function G^{-1} to convert the key k back. At the same time, the Eavesdropper can observe at most t bits of s (arbitrarily chosen). To make sure the Eavesdropper unable to get any information, the processed result s should be t-wise independent.

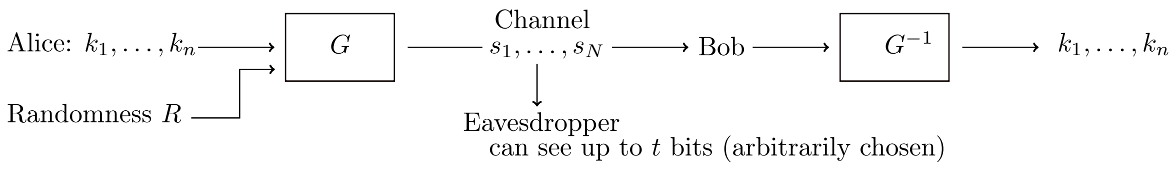


Figure 7.1: The function G outputs t-wise independent $s_1, \ldots, s_N$ for any input $k_1, \ldots, k_n$.

7.2.1 Privacy Amplification via Polynomial Evaluation

To design G, we can use a similar approach as the Shamir's secret sharing scheme.

Definition 7.2.1 (Polynomial evaluation scheme). Given any secret $(k_1, \ldots, k_n) \in \mathbb{F}_q^n$, the *polynomial evaluation scheme* generates N shares by evaluating $S_i = p(x_i)$ on distinct nonzero points $x_1, \ldots, x_N \in \mathbb{F}_q \setminus \{0\}$ with $p(x)$ being

$$p(x) = k_1 + k_2 x + k_3 x^2 + \ldots + k_n x^{n-1} + x_n(r_1 + r_2 x + r_3 x^2 + \ldots r_t x^{t-1}),$$

where $r_1, \ldots, r_t \in \mathbb{F}_q$ are chosen independently and uniformly at random with $n + 1 \leq N \leq q + 1$.

By the same analysis as Shamir, we can see that $s_1, \ldots, s_n$ is t-wise independent for any fixed $k_1, \ldots, k_n$. If $k_1, \ldots, k_n$ is random, then $s_1, \ldots, s_n$ is $(n + t)$-wise independent, which means when $N = n + t$, $s_1, \ldots, s_n$ is jointly uniform.

Note. Polynomial evaluation scheme construction needs q large enough $(q + 1 \geq N)$.

7.2.2 Privacy Amplification via Symbol Fixing Extractors

But can we achieve the same thing if q isn't large enough? We can use the so-called randomness extractor. We first introduce a special kind of randomness extractor for a special kind of random source.

Definition 7.2.2 (k-symbol fixing source). A *k-symbol-fixing source* $X \in \mathbb{F}_q^n$ satisfies X_i is uniformly random and independent if $i \in S$, and X_i is fixed to arbitrary values if $i \notin S$ where $S \subset [n]$ and $|S| \geq k$.

Note. The k of a k-symbol fixing source is also called entropy, and we will soon see the reason.

Definition 7.2.3 (k-symbol fixing extractor). A deterministic function $\text{Ext}(\cdot) \colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ for some $m \leq n$ is a *k-symbol-fixing extractor* over $\mathbb{F}_q$ if for all k-symbol-fixing source X, we have[a]

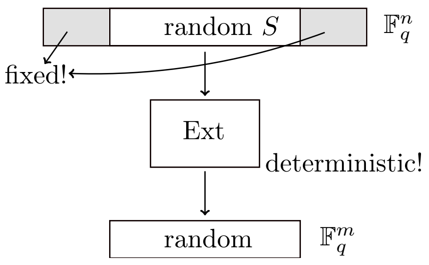
$$\text{Ext}(X) \sim \mathcal{U}_m.$$

[a] $\mathcal{U}_m$ is the uniform distribution over $\mathbb{F}_q^m$.

The figures below show the definition of the function Ext.

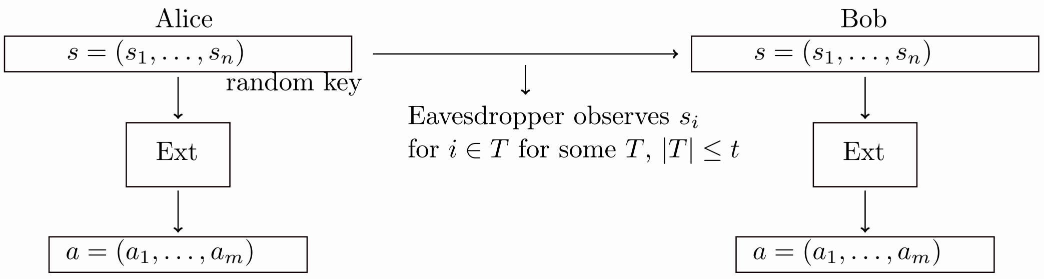
$$\begin{tikzcd} x = (x_1, \ldots, x_n) \in \mathbb{F}_q^n \arrow[r] & \boxed{\text{Ext}} \arrow[r] & y = \text{Ext}(x) \in \mathbb{F}_q^m \\ & & (m \leq n) \end{tikzcd}$$

More intuitively, we have the following.



Problem. How do Alice and Bob use Ext for privacy amplification?

Answer. They can generate a sequence $a = (a_1, \dots, a_m) = \mathrm{Ext}(s_1, \dots, s_n) \in \mathbb{F}_q^m$ as key!



⊛

Claim. Conditioned on the information Eve has, $a = (a_1, \dots, a_m)$ is uniform and independent of Eve's view, i.e., for $|T| \leq t$,

$$\Pr((a_1, \dots, a_m) = (\alpha_1, \dots, \alpha_m) \mid s_i = \sigma_i, i \in T) = q^{-m}.$$

Proof. Conditioned on observation $s_i = \sigma_i$ for $i \in T$, the distribution of $s_1, \dots, s_n$ is a symbol-fixing distribution. Then

$$\mathrm{Ext}(s_1, \dots, s_n \mid s_i = \sigma_1, i \in T) \sim \mathcal{U}_m,$$

which implies $(a_1, \dots, a_m \mid s_i = \sigma_i) \sim \mathcal{U}_m$. ⊛

Ideally, we want $\mathrm{Ext}\colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ where $m \approx k$, where k is the entropy. When q is small, we can't ensure that the whole entropy is extracted, so there will be some *loss*, and we have to allow some *errors*, in the sense that the output is not exactly uniform but close to uniform. And indeed, we can measure discrepancy between two distributions via the so-called statistical distance.

Definition 7.2.4 (Statistical distance). Given two distributions X, Y on some universe Ω, and let $p_i = \Pr(X = i)$ and $q_i = \Pr(Y = i)$. The *statistical distance* $d_{TV}(X, Y)$ between X and Y is defined as

$$d_{TV}(X, Y) = \frac{1}{2} \sum_{i \in \Omega} |p_i - q_i| \in [0, 1].$$

We see that in Definition 6.2.4, d_{TV} is defined as the half of the ℓ_1 norm of the difference vector. Notice that we have an equivalent way to calculate the statistical distance

Remark. The statistical distance between X and Y is equivalently defined as

$$d_{TV}(X, Y) = \max_{S \subseteq \Omega} |\Pr(X \in S) - \Pr(Y \in S)|.$$

Corollary 7.2.1. For all function $D\colon \Omega \to \{0,1\}$,

$$|\Pr(D(X)=1)-\Pr(D(Y)=1)| \leq d_{TV}(X,Y).$$

Now, we are ready to define the randomness extractors with some slackness. Firstly, we define the following.

Notation. If $d_{TV}(X,Y)\leq\epsilon$, then we denote this as $X \approx_\epsilon Y$.

Finally, we have the so-called (k,ϵ)-symbol fixing extractor, defined as follows.

Definition 7.2.5 ((k,ϵ)-symbol fixing extractor)**.** A deterministic function $\mathrm{Ext}(\cdot)\colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ for some $m\leq n$ is a (k,ϵ)-*symbol fixing extractor* over $\mathbb{F}_q$ if for all k-symbol-fixing source X, we have

$$\mathrm{Ext}(X)\approx_\epsilon \mathcal{U}_m.$$

Lecture 21: Constructing Extractors via Expander graphs

10 Nov. 10:30

As previously seen. Our goal now is to construct a (k,ϵ)-symbol fixing extractor for constant q, and indeed, we achieve this via a random walk on expanders.

7.2.3 Symbol Fixing Extractors via Expanders

Let G be a (M,q,λ)-spectral expander with $q\geq 3$, where $M=q^m$ is the number of vertices and q is the degree of the graph. Given $X=(X_1,\dots,X_n)\in[q]^n$, take a walk as described by X (each X_i[2] tells us which neighbor t to pick at the i^{th} step), and output the label of destination.

Intuition. Intuitively,

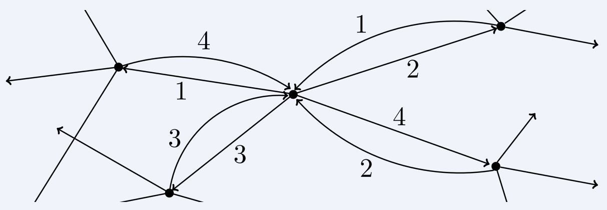
- If X_i is random, we get closer to the stationary distribution (uniform).
- If X_i is fixed, hopefully the step doesn't change the distance to the uniform distribution.

The latter one means that if we have a distribution $p=(p_1,\dots,p_M)$, if we shuffle the probabilities around (since X_i is fixed), this action can be reversed.

This suggests that the walk should be *reversible*, i.e., the action of the walk by a (fixed) X_i is a bijection over the vertices. if the current distribution $p=(p_1,\dots,p_M)$ gets shuffled, we want the result to be $p\mapsto(p_{\sigma^{-1}(1)},\cdots,p_{\sigma^{-1}(M)})$, where σ is the bijection on vertices defined by the step. This is ensured by consistent labeling.

Remark (Consistent labeling)**.** A graph has a *consistent labeling* if no vertex has 2 incoming edges with the same label. The good news is that we know that both Ramanujan graph and Lubotzky-Phillips-Sarnak graph satisfy this property.

Example. Let a vertex v with four neighbors, each of them provides an incoming edge to v. If all the edges have different labels, then we have consistent labeling.



[2] X_i are independent, though some are fixed.

Now, denote π to be the uniform distribution, i.e., $\pi = 1/M \cdot (1, 1, \ldots, 1) \in \mathbb{R}^M$, we're ready to show that such a construction indeed gives us a (k, ϵ)-symbol fixing extractor.

Suppose $\lambda \leq q^{-\alpha}$, $\alpha \in (0, 1/2)$, $M = q^m$ vertices and degree of q. The ultimate goal is to have m as large as possible such that $m \approx k$. If we take a k-step random walk starting p, and ending in p', then $\|p' - \pi\|_2 \leq \lambda^k$ from Theorem 6.4.1. The same is true if only k (out of n) steps are random, the rest are fixed. We now try to bound the statistical distance of one step, but first, recall the following.

Remark. For any distribution $p' \in \mathbb{R}^M$, we have

$$d_{TV}(p', \pi) = \frac{1}{2}\|p' - \pi\|_1 \leq \frac{\sqrt{M}}{2}\|p' - \pi\|_2,$$

where the inequality comes from the Cauchy-Schwartz inequality.

This implies that

$$2\epsilon := 2 \cdot d_{TV}(p', \pi) \leq \sqrt{M} \cdot \lambda^k \leq q^{\frac{m}{2}} \cdot q^{-\alpha k} = 2^{(\frac{m}{2} - \alpha k) \log q}.$$

So, given ϵ, we can get

$$m = 2\alpha k - \frac{2}{\log q} \log\left(\frac{1}{2\epsilon}\right).$$

When q is a constant, $\epsilon = 10^{-10}$, $\alpha \approx \frac{1}{2}$, we get $m \approx k - O(1)$, which extract almost all randomness!

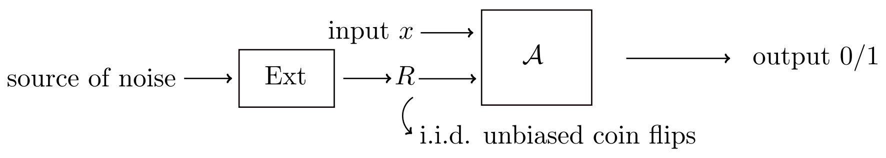
Note. If $q = 2$ and we implement this on a cycle with M nodes since there are no 2-regular expanders. It turns out that

$$\epsilon \approx \frac{\sqrt{M}}{2} \exp\left(-\frac{\pi^2 k}{2M^2}\right),$$

so ϵ is only useful when the number of output bits, $\log M$, is bounded by $\log M < \log k/2$.

7.3 Randomness Extractors

We're now interested in what's beyond the (k, ϵ)-symbol fixing extractor (which can only extract randomness from a symbol fixing source), which can turn any source of randomness into *pure randomness*?[3] If it exists, this extractor function can be applied to extract randomness from source of noise and be used for randomized algorithms!



Example (Generate uniform randomness from biased source). Suppose we have a biased coin X with $\Pr(X = 1) = p$ with $|p - 1/2| > \epsilon$, and we want to build a uniformly random source without knowing the p, by flipping the coin twice.

$$\Pr(X_1 = 1, X_2 = 0) = p(1 - p);$$
$$\Pr(X_1 = 0, X_2 = 1) = (1 - p)p.$$

These two events have the same probability. In this way, we actually *extract* a random distribution from a biased source without knowing what p is.

Unfortunate, this is too ideal.

[3]Ideally, we want the output of i.i.d. unbiased coin flips, i.e., we want to be able to flip coins!

Proposition 7.3.1. A perfect extractor doesn't exist.

Proof. Let $S_0 = \mathrm{Ext}^{-1}(\{0\})$ and $S_1 = \mathrm{Ext}^{-1}(\{1\})$. Suppose $\mathrm{Ext}\colon \{0,1\}^n \to \{0,1\}$, and say $|S_0| \geq 2^{n-1}$. Let x be uniform on S_0, then $\mathrm{Ext}(\{x\}) = 0$ always, which contradicts with the requirement. ■

However, there are two workarounds, but let's first consider how one should measure the quality of some randomness.

Lecture 22: Randomness Extractors

7.3.1 Randomness Measurement

15 Nov. 10:30

Generally, to measure randomness, we use entropy. Shannon entropy is one such example.

Definition 7.3.1 (Shannon entropy). Given a distribution $p(x)\colon X \to \mathbb{R}_+$ on X with $\sum_x p(x) = 1$, the *Shannon entropy* $H(X)$ of $p(x)$ is defined as

$$H(X) \coloneqq \sum_{x\in X} p(x) \log \frac{1}{p(x)}.$$

Remark. Shannon entropy does not ensure a functional extractor.

Proof. Since we're only taking in one sample from our distribution as noise, we can't do such averaging calculations. Furthermore, although there could be approximately n random bits on average, there can still remain the case where individual bits are not random. ⊛

As such, we require something stronger for extractors. Thankfully, the above issue is addressed by the so-called min-entropy when we want to measure the distribution X on n-bit strings.

Definition 7.3.2 (Min-entropy). Given a distribution $p(x)\colon X \to \mathbb{R}_+$ on X with $\sum_x p(x) = 1$, the *min-entropy* $H_\infty(X)$ of $p(x)$ is defined as

$$H_\infty(X) \coloneqq \min_{x\in X} \log\left(\frac{1}{p(x)}\right) = -\log \max_{x\in X} p(x).$$

Example (Flat source). The uniform distribution on a set of size S has $\log|S|$ bits of entropy.

Example. The k-bit fixing source with k random q-ary bits has min-entropy $k\log q$.

Proof. Where $|S| = k$, we get $|S| = q^k$, implying $H_\infty(X) = \log|S| = k\log q$. ⊛

Finally, it's also useful to consider the mixture of distributions since there's an interesting fact regarding the convex combination of distributions.

Definition 7.3.3 (Convex combination). Given a set of distributions $X_1, \ldots, X_N$ the *convex combination* over $\{X_i\}_{i=1}^N$ is given by $\sum_i \lambda_i X_i$ for some $\lambda_1, \ldots, \lambda_N \geq 0$ such that $\sum_i \lambda_i = 1$.

The convex combination is actually a distribution: it can be interpreted as first sampling according to λ_i, then with the i, we sample from the distribution X_i. And interestingly, we have the following fact.

Remark. Any distribution with $H_\infty(x) \geq k$ is a convex combination of flat distribution with min-entropy k.

7.3.2 Seedless and Seeded Extractors

Firstly, from what we have seen, as an analogous to k-symbol fixing source, we can now define the so-called k-source by measuring the randomness via min-entropy.

Definition 7.3.4 (k-source)**.** A *k-source* $X \in \mathbb{F}_q^n$ is a distribution on X such that $H_\infty(X) \geq k$.

Hence, analogous to (k, ϵ)-symbol fixing extractor, we can define the so-called (k, ϵ)-extractor.

Definition 7.3.5 ((k, ϵ)-extractor)**.** A (k, ϵ)*-extractor* over $\mathbb{F}_q$ is a deterministic function $\mathrm{Ext}(\cdot) \colon \mathbb{F}_q^n \to \mathbb{F}_q^m$ for some $m \leq n$ such that for all k-source X, we have

$$\mathrm{Ext}(X) \approx_\epsilon \mathcal{U}_m.$$

Remark. We already know in the unrestricted case, i.e., the (k, ϵ)-extractor can't exist from Proposition 7.3.1.

Hence, our goal now is to see if there is one function (random extractor) that is as general purpose as possible, i.e. extract uniform randomness from any source satisfy entropy requirement.

Note. Specifically, we hope for the following.

(a) We need assumption on $H_\infty(X)$.

(b) We want to have an explicit expression for practically.

(c) Ideally, we want the extractor to be deterministic.

We first look at the so-called seedless extractors, where we have a *structured family of sources* depending on application such as bit / symbol-fixing extractors for privacy amplification.

Example (2-source)**.** The 2-source[a] is that given $X = (X_1, X_2)$, where X_1, X_2 are independent, and each satisfy $H_\infty(X_1) \geq k_1, H_\infty(X_2) \geq k_2$.

$$\begin{tikzcd}
\text{source 1 of noise } (n \text{ bits}) \arrow[rd] & & \\
 & \boxed{\mathrm{Ext}} \arrow[r] & \text{many bits } (\gg d) \\
\text{source 2 of noise } (n \text{ bits}) \arrow[ru] & &
\end{tikzcd}$$

[a] This is different from the k-source.

Some other examples are the following.

Example (Affine source)**.** Let X to be uniform over some unknown k-dimensional subspace of $\mathbb{F}_q^n$.

Example (Samplable source)**.** Let X to be sampled by some efficient but unknown algorithm taking pure random as input.

But since the seedless extractors is too specific and depends on problems, we instead consider a more general extractor called seeded (k, ϵ)-extractor, such functions (extractors) require a small (d-bits) pure independent random seed.

Definition 7.3.6 (Seeded (k, ϵ)-extractor)**.** A *seeded* (k, ϵ)*-extractor* over $\mathbb{F}_q$ is a deterministic function $\mathrm{Ext}(\cdot) \colon \mathbb{F}_q^n \times \mathbb{F}_q^d \to \mathbb{F}_q^m$ for some $m \leq n + d$ such that for all k-source $X \in \mathbb{F}_q^n$ and purely random $Z \in \mathbb{F}_q^d$, we have

$$(Z, \mathrm{Ext}(X, Z)) \approx_\epsilon \mathcal{U}_{d+m}.$$

$$\begin{tikzcd}
\text{source of noise } (n \text{ bits}) \arrow[r] & \boxed{\mathrm{Ext}} \arrow[r] & \text{many bits } (d + m \gg d) \\
 & \text{seed } (d \text{ bits, } d \ll n) \arrow[u] &
\end{tikzcd}$$

Intuition. When the length of the random seed, d, is much less than n (e.g., $d \approx \log n$), even searching all seeds is efficient, and hence we can do derandomization efficiently.

Note (Strong and weak extractor). Notice that if m is not big enough, we can cheat by output seed directly, which leads to the so-called *weak extractor*. This is why we require the algorithm to output $(Z, \mathrm{Ext}(X, Z))$ together (i.e., the extractor's output and the seed). In this case, seeded (k, ϵ)-extractor is a *strong extractor*.

Now, we're going to focus on the case that $\mathbb{F}_q = \mathbb{F}_2 = \{0, 1\}$. And in this case, it has been shown that there is a seeded (k, ϵ)-extractor with

- $m = k - \log \epsilon^{-2} - O(1)$;
- $d = \log \epsilon^{-2} + O(1)$.

On the other hand, proved by Radhakrishuan, Ta-shma [RT00], any seeded (k, ϵ)-extractor satisfies $\epsilon < 1/2$ must have

- $m \leq k - O(1)$;
- $d \geq \log\big((n-k)\epsilon^{-2}\big) - O(1)$.

Anyway, the upshot is that it is actually possible to construct a seeded (k, ϵ)-extractor!

7.3.3 Stimulate Randomized Algorithm

Random extractors with seed can be used to purify the source required by a randomized algorithm $\mathcal{A}(X, R)$ where X is the input, and R is a uniform random seed with length $|R| = \mathsf{poly}(|X|)$. To generate R, we can start from an imperfect random noise R' with enough min-entropy $H_\infty(R') \gg |R|$. And the extractor can *purify* the noise source as $R = \mathrm{Ext}(R', Z)$.

Remark. In the next lecture, we will see that we can achieve more than half probability of correctness by exhaustively searching all random seeds.

Lecture 23: Stimulate Algorithms with Low-Quality Randomness

We now want to simulate a randomized algorithm by using a seeded extractor. 17 Nov. 10:30

As previously seen. A seeded extractor takes an entropy source, few pure random bits as a seed, then turns them into high-quality random bits. To simulate the algorithm, we run through all possible seeds and record the output, then take the majority vote.

To do this, we consider a seeded extractor $\mathrm{Ext}\colon \{0,1\}^N \times \{0,1\}^d \to \{0,1\}^m$ such that

- $m = |R| \leq \mathsf{poly}(n)$ where R is noise, $n = |x|$ for x being the input;
- $N = \mathsf{poly}(n)$, and $d = O(\log n)$;
- the error ϵ of the extractor $\epsilon \leq 1/10$.

Then, the algorithm runs as follows.

Algorithm 7.1: Stimulate Randomized Algorithm

```
Data: Randomized algorithm $\mathcal{A}$, input $x$, extractor $\mathrm{Ext}(\cdot,\cdot)$, noise source $R'$
Result: $\mathcal{A}(x)$
1 for $Z \in \{0,1\}^d$ do                          // polynomially-many $Z$
2     $\mathcal{A}(x, \mathrm{Ext}(R', Z))$
3 return majority vote
```

Clearly, Algorithm 7.1 runs in polynomial time if the original algorithm $\mathcal{A}$ runs in polynomial time. Furthermore, we see that Algorithm 7.1 is actually a BPP algorithm.

Theorem 7.3.1. Algorithm 7.1 is a BPP algorithm.

Proof. To analyze the error probability of Algorithm 7.1, define T to be the set of *good randomness* for $\mathcal{A}$ such that $\mathcal{A}$ produces the correct output with $r \in T$, i.e.,

$$T = \{r \colon \mathcal{A}(x, r) \text{ is correct}\}.$$

In this case, the *bad uniform randomness* occupies $\leq 1/10$ of the entire space, i.e.,

$$\Pr_R(\text{uniform } R \notin T) \leq \frac{1}{10}.$$

But since we don't have uniform randomness, instead, we have $\operatorname{Ext}(R', Z) \approx_\epsilon \mathcal{U}$ with $\epsilon < 1/10$,

$$\Pr_{R',Z}(\operatorname{Ext}(R', z) \notin T) \leq \frac{1}{10} + \epsilon \leq \frac{1}{10} + \frac{1}{10} \leq \frac{1}{5}.$$

Therefore, the simulation is incorrect with probability $\leq 1/5$ with a random source R'. Taking the expectation of our testing routine of $\mathcal{A}$, say $\mathcal{A}'$, we have $\mathbb{E}_{R',z}[\mathcal{A}'(R', z)] \leq 1/5$. Finally, to bound the overall error probability, denote indicator variables B for bad events such that

$$B = \begin{cases} 1, & \text{if } \mathcal{A}'(R', z) \text{ is wrong;} \\ 0, & \text{otherwise.} \end{cases}$$

By summing up the expectations, we have $\mathbb{E}_{R'}[\mathbb{E}_z[B]] \leq 1/5$. Using Markov's inequality to get the probability that the expectation of B is $\geq 1/2$ we have,

$$\Pr_{R'}\left(\mathbb{E}_Z[B] \geq \frac{1}{2}\right) \leq \frac{1/5}{1/2} = \frac{2}{5}.$$

Hence, the majority vote (over Z) is incorrect with probability $\leq 2/5$, meaning that all together, the simulation (with majority vote) is correct with probability $\geq 3/5$. Thus, we have what we need for a BPP algorithm. ■

7.4 Constructing Randomness Extractors

Constructing an optimal extractor is an open problem, but we can get pretty close. Here, we will construct an extractor that extracts almost the whole entropy of the input at the cost of seed length. The key lemma is the so-called leftover hash lemma [ILL89], which shows how to construct extractors from universal hash family.

7.4.1 Constructing Universal Hash Families

Firstly, let's introduce the universal hash family.

Definition 7.4.1 (Universal hash family)**.** Given a family of hash functions $\mathcal{H} = \{h_1, h_2, \ldots, h_D\}$ of size $D = 2^d$ where $h_i \colon \{0,1\}^n \to \{0,1\}^m$ is *universal* is for all $x \neq x'$,

$$\Pr_{i \sim [D]}(h_i(x) = h(x')) \leq \frac{1}{2^m}.$$

Construct a universal hash family is similar to what we did for pairwise independence. Consider $q = 2^n$, and $i \in \mathbb{F}_q$ is taken uniform random, then for $x \in \mathbb{F}_q$,

$$h_i(x) = f(i \times x),$$

where f is just truncating the input to m bits, since we cut out the stuff that we don't really want. Then, for $x \neq x' \in \mathbb{F}_q$,

$$\Pr_i(h_i(x) = h_i(x')) = \Pr_i(f(ix) = f(ix')) = \Pr_i(f\big(i(x - x')\big) = 0^m).$$

We know that $i(x-x')$ is uniform random over $\mathbb{F}_q$ since $x-x' \neq 0$, and that the probability that $i(x-x')$ is 0^m in the first m bits is 2^{-m}, which is exactly the requirement for $\mathcal{H} = \{h_i\}_i$ being a universal hash family.

7.4.2 Constructing Randomness Extractors via Universal Hash Families

Now, we can construct a near optimal extractor based on a universal hash family by considering $\operatorname{Ext}(x, Z) := h_Z(x)$. Interestingly, this is actually a (k, ϵ)-extractor, shown in leftover hash lemma.

Theorem 7.4.1 (Leftover hash lemma)**.** Given a universal hash family $\mathcal{H}$, define $\operatorname{Ext}\colon \{0,1\}^n \times [D] \to \{0,1\}^m$ as $\operatorname{Ext}(x, Z) = h_Z(x)$. Then for all k and ϵ such that $m \leq k - 2\log(1/\epsilon)$, Ext is a (k, ϵ)-extractor.

Proof. Let $x \in \{0,1\}^n$ be uniform over a set S of size $K = 2^k$. We want to show $(\operatorname{Ext}(X, Z), Z) \approx_\epsilon \mathcal{U}$.

Let $p(y, Z)$ be the distribution of $(\operatorname{Ext}(X, Z), Z)$ where $y \in \{0,1\}^m$ and $z \in [D]$. Then the square of the ℓ_2 (Euclidean) distance between p (as a vector) and uniform distribution $\mu := \mathcal{U}$ over $\{0,1\}^m \times [D]$ is

$$\begin{aligned}
\|p-\mu\|_2^2 &= \sum_{y,Z}\left(p(y,Z) - \frac{1}{2^m \cdot D}\right)^2 \\
&= \sum_{y,Z} p^2(y,Z) - 2\cdot\frac{2^{-m}}{D}\underbrace{\sum_{y,Z} p(y,Z)}_{1} + \frac{2^{-m}}{D} \\
&= \sum_{y,Z} p^2(y,Z) - 2\cdot\frac{2^{-m}}{D}\cdot 1 + \frac{2^{-m}}{D} \\
&= \sum_{y,Z} p^2(y,Z) - \frac{2^{-m}}{D}
\end{aligned}$$

The sum of probabilities squared,[a] $\sum_{y,Z} p^2(y, Z)$ is given by

$$\begin{aligned}
\sum_{y,Z} p^2(y,Z) &= \Pr_{Z,Z',x,x'}\left((h_Z(x), Z) = (h_{Z'}(x'), Z')\right) \\
&= \Pr(Z = Z')\Pr(h_Z(x) = h_{Z'}(x') \mid Z = Z') \\
&= \frac{1}{D}\left(\Pr(x = x') + \frac{1}{|S|^2}\sum_{x,x'\in S, x\neq x'} \Pr_Z(h_z(x) = h_z(x'))\right)
\end{aligned}$$

Then, we know that $\Pr(x = x') = 1/K$, and $\Pr_Z(h_Z(x) = h_Z(x')]$ from universal hashing is $\leq 2^{-m}$, we have

$$\sum_{y,Z} p^2(y,Z) \leq \frac{1}{D}\left(\frac{1}{K} + \frac{1}{K^2}\cdot K^2 \cdot 2^{-m}\right) \leq \frac{1}{DK} + \frac{2^{-m}}{D},$$

implying

$$\|p-\mu\|_2^2 \leq \frac{1}{DK} + \frac{2^{-m}}{D} - \frac{2^{-m}}{D} = \frac{1}{DK}.$$

Then, by using Cauchy-Schwartz inequality, we have

$$\|p-\mu\|_1^2 \leq 2^m \cdot D \cdot \|p-\mu\|_2^2 \leq s^m \cdot D \cdot \frac{1}{DK} = \frac{2^m}{K}.$$

Assuming $m \leq k - \log(1/\epsilon^2) = k - 2\log\epsilon^{-1}$, we have $\|p-\mu\|_1^2 \leq \epsilon^2$, i.e., $p \approx_\epsilon \mathcal{U}$. ∎

[a]This is known as the *collision probability*. The whole purpose of hashing functions is to control the collision probability.

It is unfortunate, however, since it needs to run through all the seeds. So the following question arises.

Problem. Is there a more *seed-length-friendly* construction?

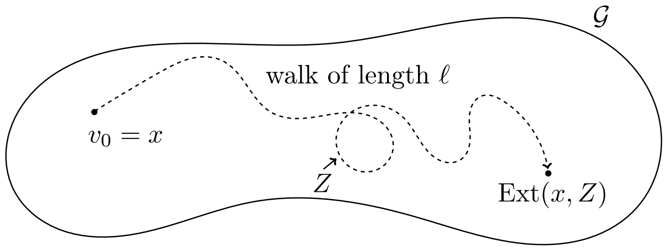
Answer. The answer is yes, and it lies within expander graphs. The expander graph method is what is considered a weak extractor, i.e., the seed is included in the output. The other caveat is that the entropy must be large (pretty close to n). ⊛

Lecture 24: Expander Extractors, PSG: Hardness & Randomness

7.4.3 Constructing Randomness Extractors via Expanders

22 Nov. 10:30

Consider $\mathcal{G}$ is a $(2^n, d, \lambda)$-spectral expander, where 2^n is the number of nodes, d is a constant and $\lambda(\mathcal{G}) \leq 1/2$. Moreover, consider an extractor such that $\operatorname{Ext}\colon \{0,1\}^n \times \{0,1\}^{\ell \cdot \log d} \to \{0,1\}^n$, where the seed length is $\ell \cdot \log(d)$.[4]



Our goal is to show that if X is uniform on a set $S = \{0,1\}^n$ of size 2^k (k entropy), then $\operatorname{Ext}(X, Z) \approx_\epsilon \mathcal{U}_n$ where Z is an independent seed.

Consider a vector $p \in \mathbb{R}^{2^n}$ of probabilities of the initial distribution of v_0 which is uniform on S, and let's say that p' is the final distribution, and finally let $\mu \coloneqq \mathcal{U}_{2^n} = 1/2^{-n} \cdot (1, 1, \dots, 1)$ to be the uniform distribution over vertices.

As previously seen. If we take a step, our distance multiplies by λ as shown in Theorem 6.4.1, i.e.,

$$\lVert p' - \mu \rVert_2 \leq \lambda^\ell \cdot \lVert p - \mu \rVert_2$$

Then, since

- $\lVert p - \mu \rVert_2 \leq \lVert p \rVert_2 + \lVert \mu \rVert_2 \leq 2^{-k/2} + 2^{-n/2} \leq 2^{1-k/2}$;
- $\lambda^\ell \leq 2^{-\ell}$ since $\lambda \leq 1/2$,

we have $\lVert p' - \mu \rVert_2 \leq 2^{-\ell} \cdot 2^{1-k/2}$. Then, by letting the length be

$$\ell \coloneqq \frac{n}{2} - \frac{k}{2} + \log(\epsilon^{-1}) + 1,$$

we have

$$\lVert p' - \mu \rVert_2 \leq 2^{(-\frac{n}{2} + \frac{k}{2} - \log \epsilon^{-1} - 1) + (1 - \frac{k}{2})} = 2^{-\frac{n}{2} + \log \epsilon}$$

In terms of the ℓ_1 norm, using Cauchy-Schwartz inequality, we have

$$\lVert p' - \mu \rVert_1 \leq 2^{\frac{n}{2}} \cdot \lVert p' - \mu \rVert_2 = \epsilon.$$

This means that the seed length ℓ to achieve error ϵ is $\ell = O(n - k + \log \epsilon^{-1})$.

Remark. This is useful when k is large, because if k is small, the head-start isn't useful enough to have ϵ close to zero. Also, this is often used as a subroutine in other larger routines.

[4]This is because our random walk is of length ℓ, and we are choosing one of the d possibilities for each step.

Chapter 8

Pseudo-Random Generators

As previously seen. Earlier, we had talked about PRGs in the context of pairwise and k-wise independence, but now we'll talk about what these objects are, how we can construct them, and if they even exist.

We said that a PRG generates *fake* randomness that is indistinguishable from true randomness to an observer (fool the observer).

Remark. While extractors generate true randomness from noise, PRGs generate noise (fake randomness) from true randomness.

8.1 Existence of PRGs and the Consequences

Let's first formalize (recall) some relevant definitions.

As previously seen. A distinguisher $D\colon \{0,1\}^\ell \to \{0,1\}$ is a function that outputs True, or False, answering whether two distributions are different.

While a PRG $G\colon \{0,1\}^n \to \{0,1\}^\ell$ with $\ell \gg n$ bits is a function that take an n-bit seed, output a long and seemingly random looking bits.

And D is said to be ϵ-*fooled* by a PRG G if

$$\left| \Pr_{y\sim\mathcal{U}_\ell}(D(y)=1) - \Pr_{x\sim\mathcal{U}_n}(D(G(x))=1) \right| \leq \epsilon.$$

In other words, if the difference is up to ϵ between a uniformly random distribution and the distribution generated by the pseudo-random generator G. And generalize Definition 5.3.2 a bit, we can say that if G ϵ-fools every distinguisher D in a class $\mathcal{F}$ of distinguishers, then G is an ϵ-*pseudo-random generator for* $\mathcal{F}$.

Notice the following.

Remark. The distinguisher can't be all-powerful, let's say allowing exponential time.

Proof. No, if they have an infinite amount of time, then they can tell that $\ell \gg n$ is suspect. ⊛

Therefore, we need to limit the distinguisher quite a bit.

Example. k-wise independent source is enough to fool a distinguisher with only k-*local view*.

8.1.1 Existence of PRGs and the Consequences

Practically, the ultimate goal is to have a PRG which fools all polynomial time algorithms (which is defined in the original Definition 5.3.2). However, this is very difficult, since the existence of such a perfect PRG would imply that $\mathsf{P} \neq \mathsf{NP}$ and $\mathsf{P} = \mathsf{BPP}$, both are huge open problems in the complexity theory!

On the other hand, such a perfect PRG is indeed achievable if we assume other fancier hardness assumption beyond $\mathsf{P} = \mathsf{NP}$. Let's first see why the existence of a PRG implies $\mathsf{P} \neq \mathsf{NP}$.

Theorem 8.1.1. Assuming a PRG exists, then $\mathsf{P} \neq \mathsf{NP}$.

Proof. Assume that we have a PRG $G\colon \{0,1\}^n \to \{0,1\}^\ell$, which ϵ-fools every polynomial time algorithm. Then, consider the language L defined as $L := \{G(x) \mid x \in \{0,1\}^n\}$. Observe the following.

- $L \notin P$: Otherwise, there is a polynomial time $\mathcal{A}$ that decides L. Use $\mathcal{A}$ as a distinguisher, we have
$$\left| \Pr_{y\sim\mathcal{U}_\ell}(\mathcal{A}(y) = 1) - \Pr_{x\sim\mathcal{U}_n}(\mathcal{A}(G(x)) = 1) \right| = |2^{n-\ell} - 1| \approx 1,$$
meaning that $\mathcal{G}$ doesn't ϵ-fool $\mathcal{A}$, contradicting to the definition of G.
- $L \in \mathsf{NP}$: If an answer $y \in L$, then by definition, there exists some $x \in \{0,1\}^n$ such that $G(x) = y$. Then, x is a certificate of y, and we can verify it by checking $G(x) = y$ or not.[a]

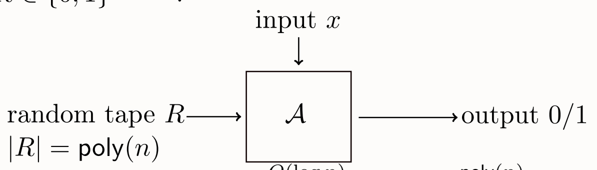
We see that $L \in \mathsf{NP} \setminus \mathsf{P}$, i.e., $\mathsf{P} \neq \mathsf{NP}$. ■

[a]Recall that G is efficient.

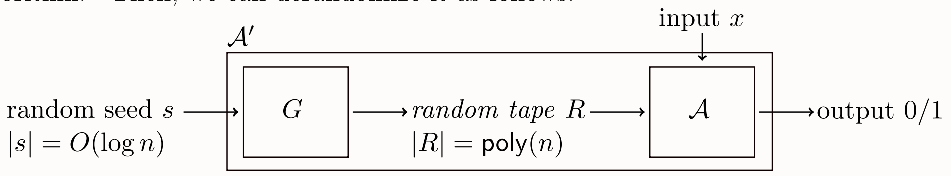
Also, let's see why the existence of PRG implies $\mathsf{P} = \mathsf{BPP}$.

Theorem 8.1.2. Assuming a PRG exists, then $\mathsf{P} = \mathsf{BPP}$.

Proof. Firstly, we know that $\mathsf{P} \subseteq \mathsf{BPP}$, so we only need to show $\mathsf{BPP} \subseteq \mathsf{P}$. Fix and language $L \in \mathsf{BPP}$, and consider a corresponding randomized algorithm $\mathcal{A}(x, R)$ which takes input $x \in \{0,1\}^n$ and a random tape $R \in \{0,1\}^{\mathsf{poly}(n)}$.



Assume that we have a PRG $G\colon \{0,1\}^{O(\log n)} \to \{0,1\}^{\mathsf{poly}(n)}$, which ϵ-fools every polynomial time algorithm.[a] Then, we can derandomize it as follows.



If we fix x, and denote the correct output as $f(x)$, and the seed is small enough, we can enumerate all possibilities and take a majority vote. This is the case since $|s| = O(\log n)$, and the probability over the choice of the seed s is
$$\Pr_s(\mathcal{A}'(x,s) = \mathcal{A}(x, G(s)) = f(x)) = \Pr_R(\mathcal{A}(x,R) = f(x)) \pm \epsilon = \frac{9}{10} \pm \epsilon$$
where since $L \in \mathsf{BPP}$, we may assume $\Pr_R(\mathcal{A}(x,R) = f(x))$ is $9/10$.[b] Finally, by enumerating all the possible seeds and take the majority vote, L can be solved deterministically in polynomial time, i.e., $L \in \mathsf{P}$. This proves that $\mathsf{BPP} \subseteq \mathsf{P}$ as desired. ■

[a]To be more precise, here we assume that for any $c > 0$, there exists a PRG $G\colon \{0,1\}^{O(\log n)} \to \{0,1\}^{\mathsf{poly}(n)}$ that ϵ-fools all polynomial time, deterministic algorithms with worse-case running time n^c.

[b]We choose $9/10$ to allow some slackness ϵ such that we always have $9/10 \pm \epsilon > 2/3$ as required for $\mathcal{A}'$ being a BPP algorithm.

8.1.2 One-Time Pad

PRGs are one of the most common and important objects in cryptography, and we can see the importance of which by examining the one-time pad (OTP) protocol, created by Claude Shannon assuming that PRGs do exist.

Definition 8.1.1 (One-time pad)**.** Suppose Alice wants to send message $m \in \{0,1\}^n$ to Bob while the eavesdropper has access to m. Let $k \in \{0,1\}^n$ be a secret key generated uniformly at random and is shared by Alice and Bob, then the *one-time pad protocol* states that Alice sends $c = m \oplus k$ to Bob, while Bob decodes it as $m = c \oplus k$.

Intuition. The properties of XOR tell us that the distribution of what is communicated will be uniformly random, and a potential attacker won't be able to tell the difference between any messages.

However, this assumes that the key k is uniform random, i.e., we need n bits of true randomness. So the idea is to start from a short, random seed $s \in \{0,1\}^k$ such that $k \ll n$, and blow up the length using a PRG $G\colon \{0,1\}^k \to \{0,1\}^n$ by setting $k \coloneqq G(s)$.

Remark. If an algorithm can distinguish between the OTP encryption of two messages m_1, m_2, it can distinguish between the PRG output and a uniform string.

8.2 Nisan-Wigderson Assumptions

In this section, we're going to study how we can construct such a PRG. The construction is known as the Nisan-Wigderson's pseudo-random generator [NW94]. The underlying idea of the construction is quite simple.

8.2.1 Hardness verses Randomness

Say we are given an output of length ℓ. If this output is uniform, and a distinguisher D scan left to right, everything D saw before does not help with predicting what comes next. If D wants to succeed, it has to predict the bit that comes next based on the information it has from before, as if it can do that, the output will be distinguishable from uniform.

Intuition. So if what comes next is always hard to compute given the prior information, the D will be fooled.

Using the probabilistic method, we show that PRGs can be constructed, where the general idea is that we use the hardness to generate randomness.

Remark (Hardness v.s. Randomness)**.** When something looks really difficult in computing, we can take advantage of the output looking incredibly random.

For concreteness, we'll look at boolean circuits of polynomial size.

8.2.2 Boolean Circuits

In this construction our distinguisher is a boolean circuit of bounded size.

Definition 8.2.1 (Boolean circuit)**.** A *boolean circuit* represents a function from $\{0,1\}^n \to \{0,1\}$ using AND, OR, and NOT gate, with the size being the number of gates.

So the function has n input bits, and binary gates are used on the input to get exactly one output bit at the end. There are some general facts about boolean circuits and boolean functions.

Remark. Any boolean function $f\colon \{0,1\}^n \to \{0,1\}$ on n bits can be computed by a circuit of size $O(2^n \cdot n)$.

Proof. Consider the disjunctive normal form. ⊛

Remark. If $f(x_1, \dots, x_n)$ is computable by a deterministic Turing machine in time $T(n)$, then there is a boolean circuit that computes f in size $\mathsf{poly}(T(n))$.

Proof. See [AB09, §2.3]. ⊛

Effectively, the size of the circuit and the time required to compute are related measures.

8.2.3 Nisan-Wigderson Assumption

The Nisan-Wigderson assumption is that there is no way to encode (compress) time $T(n)$ into circuit size $\ll T(n)$. For concreteness, $T(n) = 2^{O(n)}$.

Notation. If languages in E can be simulated by a circuit of size $2^{O(n)}$, we then write $E = \mathsf{DTIME}(2^{O(n)})$.

Conjecture 8.2.1 (Worst-case Nisan-Wigderson assumption)**.** The *worst-case Nisan-Wigderson assumption* states that there is a language L in $E = \mathsf{DTIME}(2^{\Omega(n)})$ such that any circuit that computes L requires size $s(n) = 2^{\Omega(n)}$.

This means the worst-case Nisan-Wigderson assumption states that

$$E = \mathsf{DTIME}(2^{\Omega(n)}) \not\subseteq \mathsf{SIZE}(2^{\Omega(n)}).$$

Example. Given $(\langle M \rangle, T)$, determine whether $\langle M \rangle$ terminates in T steps.

Same as probabilistic Turing machine, we also have a probabilistic version of boolean circuit behavior.

Definition 8.2.2 (Predict)**.** A boolean circuit C *predicts* a function $f(x)$ with advantage $\epsilon > 0$ if

$$\Pr_{x \sim \mathcal{U}_n} (C(x) = f(x)) \geq \frac{1}{2} + \epsilon.$$

In other words, it's better than a random guess, or the circuit computes $f(x)$ on average. This leads to the following.

Conjecture 8.2.2 (Averagae-case Nisan-Wigderson assumption)**.** The *average-case Nisan-Wigderson assumption* states that there is a language in $E = \mathsf{DTIME}(2^{\Omega(n)})$ such that any circuit that predicts it requires size $s(n) = 2^{\Omega(n)}$.

The main difference between the worst and average case assumptions is that for the average case, we only need correctness for the majority of the inputs, rather than all of them.

Remark. If we have too small of a circuit, it can't even predict the value of the input on even half the input domain.

8.3 Nisan-Wigderson's Pseudo-Random Generators

We're now ready to construct a PRG assuming the Nisan-Wigderson's conjecture we have discussed. It was shown that the average case and worst case assumptions are equivalent. [IW97] Furthermore, we have the following.

Theorem 8.3.1 (Nisan-Wigderson theorem)**.** Assume the average-case assumption, then there exists a pseudo-random generator G against all polynomial sized circuits with seed length $O(\log(n))$ and

$0 < \epsilon < 1/10$.

This directly implies the following.

Corollary 8.3.1. Assume the average-case assumption, then $\mathsf{P} = \mathsf{BPP}$.

Proof. The result follows from Nisan-Wigderson theorem and Theorem 8.1.2. ■

Lecture 25: Proof of Nisan-Wigderson Theorem (One-Bit Stretch)

Now, we prove Nisan-Wigderson theorem assuming the average-case assumption holds. By the assumption, we have a hard function $f\colon \{0,1\}^n \to \{0,1\}$, which is supposed to be hard on average, and any circuit of small size does not correlate well with this function. So the truth tables are different in around half of the positions. 29 Nov. 10:30

8.3.1 The One-Bit Stretch Case

We want to construct a PRG that takes n bits of seed and produces $\ell(n) > n$ pseudo-random bits. Firstly, consider $\ell(n) = n+1$, i.e., we construct a PRG that stretches n bits to $n+1$ bits.

Remark. This is where the name *one-bit stretch* comes from.

To do this, the idea is to simply apply f on the seed $x_1, \dots, x_n$ to get the 1 expanded bit, i.e.,

$$G(x_1, \dots, x_n) = \big(x_1, \dots, x_n, f(x_1, \dots, x_n)\big).$$

To show this is a valid construction, it's suffices to show that from any good distinguisher D, we can construct an algorithm $\mathcal{A}$ for predicting f.

Remark. It's sufficient since given such algorithm $\mathcal{A}$, while f is hard, $\mathcal{A}$ can compute it efficiently, a contradiction. So such a good D can't exist.

This suggests a proof idea for the following theorem.

Theorem 8.3.2. Given a seed with length n and a hard function $f\colon \{0,1\}^n \to \{0,1\}$, $G\colon \{0,1\}^n \to \{0,1\}^{n+1}$ defined as

$$G(x_1, \dots, x_n) = \big(x_1, \dots, x_n, f(x_1, \dots, x_n)\big)$$

is a PRG.

Proof. Let D be a probabilistic polynomial-time Turning machine (PPT) distinguisher such that

$$\left| \Pr_{x}(D(x, f(x)) = 1) - \Pr_{x, b \in \{0,1\}}(D(x, b) = 1) \right| > \epsilon.$$

To use D to compute f non-trivially (better than random guess), we can construct a PPT estimator $\mathcal{A}$ that estimates f with D and runs as follows.

Algorithm 8.1: $\mathcal{A}_b(x)$ which predicts f

```
Data: An input $x$, efficient and good distinguisher $D$
Result: $\mathcal{A}_b(x)$ that predicts $f$
1 $b$ ← rand($\{0,1\}$)                    // flip a random $b \in \{0,1\}$
2 if $D(x, b) = 1$ then                     // $b = f(x)$
3     return $b$
4 else
5     return $\neg b$
```

In other words, $\mathcal{A}_b(x) = b \oplus \neg D(x, b)$ given b is random among $\{0,1\}$. Now, we show that $\mathcal{A}_b(x)$ is indeed predicting $f(x)$, and its prediction is better than a random guess. Observe that it's sufficient to show

$$\Pr_{x,b}(\mathcal{A}_b(x) = f(x)) > \frac{1}{2} + \epsilon$$

since by averaging, this implies that there is a fixed b such that $\Pr_x(\mathcal{A}_b(x) = f(x)) > 1/2 + \epsilon$, i.e., $\mathcal{A}_b(x)$ is predicting $f(x)$ for one fixed b. This is not too hard to see. Firstly, we have

$$\begin{aligned}\Pr_{x,b}(\mathcal{A}_b(x) = f(x)) = \Pr(\mathcal{A}_b(x) = f(x) \mid b = f(x)) \cdot \Pr(b = f(x))\\ + \Pr(\mathcal{A}_b(x) = f(x) \mid b \neq f(x)) \cdot \Pr(b \neq f(x)).\end{aligned}$$

Since $b \in \{0,1\}$, $\Pr(b = f(x)) = \Pr(b \neq f(x)) = 1/2$, hence

$$\begin{aligned}&\Pr_{x,b}(\mathcal{A}_b(x) = f(x))\\ =&\frac{1}{2}\left[\Pr(D(x,b)=1 \mid b = f(x)) + \underbrace{\Pr(D(x,b)=0 \mid b \neq f(x))}_{1-\Pr(D(x,b)=1\mid b\neq f(x))}\right]\\ =&\frac{1}{2} + \underbrace{\Pr(D(x,b)=1 \mid b = f(x))}_{\Pr(D(x,f(x))=1)} - \underbrace{\frac{1}{2}\left[\Pr(D(x,b)=1 \mid b = f(x)) + \Pr(D(x,b)=1 \mid b \neq f(x))\right]}_{\Pr_{x,b}(D(x,b)=1)}\\ =&\frac{1}{2} + \Pr(D(x,f(x))=1) - \Pr_{x,b}(D(x,b)=1)\\ >&\frac{1}{2} + \epsilon,\end{aligned}$$

where the last line follows from the assumption on D.[a] ■

[a]Notice that the last inequality may become $1/2 - \epsilon$. In this case, we simply flip the 1 to 0 in the analysis, then the sign flips again, hence we get back $+\epsilon$.

8.3.2 General Construction

Now, how to get more bits such that $\ell(n) > n+1$? In other words, how to get more pseudo-random bits, one trivial idea is to use a longer seed, e.g.,

$$G(x_1, \dots, x_{2n}) = \big(x_1, \dots, x_n, f(x_1, \dots, x_n), x_{n+1}, \dots, x_{2n}, f(x_{n+1}, \dots, x_{2n})\big),$$

is a PRG with 2 bits of stretch. But this is too wasteful because we are only generating one pseudo-random bit per n true random bits.

Intuition. Allowing some small overlapping in the seed when generating new pseudo-random bits!

This is exactly where the combinatorial design (midterm, problem 1) set system becomes useful, by having many subsets of $[n]$ while all the pairwise intersections remaining small.

As previously seen (Design). Recall that $S_1, \dots, S_m \subseteq U$ is a (ℓ, α)-*design* if for all i, $|S_i| = \ell$ and for all $i \neq j$, $|S_i \cap S_j| \leq \alpha$.

By using the same argument as in the midterm, i.e., probabilistic method, we have the following.

Lemma 8.3.1. For all $\ell > 0$, there exists a collection of subsets $\mathcal{S} = \{S_i\}_{i=1}^m$ of $[t]$ that is a $(\ell, \log m)$-design, there $t = O(\ell/\gamma)$, $m = 2^{\gamma\ell}$. Moreover, this can be constructed in time

$$O(2^t \cdot t \cdot m^2) = O\left(2^{\ell/\gamma + 2\gamma\ell} \cdot \ell/\gamma\right).$$

Proof. The existence follows from probabilistic method, while the time for constructions can be proved via conditional expectations. ■

Lecture 26: Proof of Nisan-Wigderson Theorem

1 Dec. 10:30

Notation (Projection). For $z \in \{0,1\}^t$ and $S \subseteq [t]$, the *projection of* z to the coordinates picked by S is denoted as

$$z|_S \in \{0,1\}^{|S|}.$$

Example. Let $z = (0,0,1,0,1,0)$ and $S = \{1,2,3,5\}$, we have $z|_S = (0,0,1,1)$.

Now, the PRG can be constructed as follows. First, pick a (n,α)-design $S = \{S_1, \dots, S_m\}$ over $[t]$ with a hard function $f\colon \{0,1\}^n \to \{0,1\}$. Then, $G\colon \{0,1\}^t \to \{0,1\}^m$ is defined as

$$G(z) = \left(f(z|_{S_1}), \dots, f(z|_{S_m})\right).$$

As previously seen. As in Theorem 8.3.2, our goal is to estimate $f(x_1, \dots, x_n)$ given $x_1, \dots, x_n$.

Now, we analyze this construction via reduction from distinguisher to predictor for f, in other words, if there is a PPT distinguisher D that have advantage greater than ϵ, then using D we can estimate f.

Theorem 8.3.3. Assume $\mathcal{S} = \{S_i\}_{i=1}^m$ is a $(n, \log m)$-design over $[t]$. Assume there exists a sufficiently efficient distinguisher $D\colon \{0,1\}^m \to \{0,1\}$ such that

$$\left| \Pr_{R \in \{0,1\}^m}(D(R) = 1) - \Pr_{Z \in \{0,1\}^t}(D(G(Z)) = 1) \right| > \epsilon,$$

then, we can construct a circuit C of size $O(m^2 + |D|)$ such that

$$\left| \Pr_{x \in \{0,1\}^n}(C(x) = f(x)) - \frac{1}{2} \right| \geq \frac{\epsilon}{m}.$$

Proof. Consider a distribution $(y_1, \dots, y_m) \in \{0,1\}^m$ and i.i.d. uniform $(u_1, \dots, u_m) \in \{0,1\}^m$. Then for a distinguisher $D\colon \{0,1\}^m \to \{0,1\}$ satisfies

$$|\Pr(D(y_1, \dots, y_m) = 1) - \Pr(D(u_1, \dots, u_m) = 1)| > \epsilon.$$

Define the hybrid distributions $H_0, \dots, H_m$ over $\{0,1\}^m$ as

$$\begin{aligned} H_0 &= (u_1, u_2, \dots, u_m) \\ H_1 &= (y_1, u_2, \dots, u_m) \\ &\vdots \\ H_m &= (y_1, y_2, \dots, y_m), \end{aligned}$$

i.e., $H_i \coloneqq (y_1, \dots, y_i, u_{i+1}, \dots, u_m)$.

Claim. There exists i such that $|\Pr(D(H_i) = 1) - \Pr(D(H_{i+1}) = 1)| > \epsilon/m$.

Proof. If not, by letting $p_i = \Pr(D(H_i) = 1)$,

$$\begin{aligned} |p_0 - p_m| &\leq |p_0 - p_1 + p_1 - p_2 \dots - p_m| \\ &\leq |p_0 - p_1| + |p_1 - p_2| + \dots + |p_{m-1} - p_m| \\ &\leq m \cdot \frac{\epsilon}{m} \\ &= \epsilon, \end{aligned}$$

which is a contradiction to the definition of D. ⊛

Let $y_1, \dots, y_m$ to be the PRG output, i.e., $y_i = f_i = f(z|_{S_i})$ for random seed z. From the above claim, there exists i such that

$$\left| \Pr_{z,u}(D(f_1, \dots, f_i, u_{i+1}, \dots, u_m)) - \Pr(D(f_1, \dots, f_{i-1}, u_i, \dots, u_m) = 1) \right| > \frac{\epsilon}{m}.$$

Assume first that

$$\Pr_{z,u}(D(f_1, \dots, f_i, u_{i+1}, \dots, u_m)) - \Pr(D(f_1, \dots, f_{i-1}, u_i, \dots, u_m) = 1) > \frac{\epsilon}{m},$$

then u_i being i.i.d. uniform implies that there exist some fixed choice of $u_{i+1}, \dots, u_m$ such that the inequality above still holds.

Note. Up to this point, we have fixed $i, u_{i+1}, \dots, u_m$.

Now, without loss of generality, we assume that for this fixed i, $S_i = \{1, \dots, n\}$ since otherwise, we can relabel the coordinates. Construct $z_1, \dots, z_t$ as

$$z = (z_1, \dots, z_n, \dots, z_t) = (x_1, \dots, x_n, z_{n+1}, \dots, z_t).$$

Recall that

$$\Pr_{x_1, \dots, x_n}(D(f(z|_{S_1}), \dots, f(z|_{S_{i-1}}), u_{i+1}, \dots, u_m) = 1) \\ - \Pr_{x_1, \dots, x_n}(D(f(z|_{S_1}), \dots, f(z|_{S_{i-1}}), u_i, \dots, u_m) = 1) > \frac{\epsilon}{m},$$

so there is also a fixed choice of $z_{n+1}, \dots, z_t$ such that the above holds, and we fix that in the sequence as well.

Note. Up to this point, we have fixed $i, u_{i+1}, \dots, u_m, z_{n+1}, \dots, z_t$.

Intuition. Using the design property, given $x_1, \dots, x_n$, the values of $f(z|_{S_1}), f(z|_{S_{i-1}})$ can each be computed fairly easily with the assumption that $|S_i \cap S_j| \leq \log m$.

Specifically, $f(z|_{S_1})$ only depends on $\log m$ of $x_1, \dots, x_n$ while fixing other inputs. This implies that we can compute $f(z|_{S_i})$ given $x_1, \dots, x_n$ with a circuit of size $O(2^{\log m}) = O(m)$. Altogether, $f(z|_{S_1}), \dots, f(z|_{S_{i-1}})$ can be computed by a circuit of size $O(m^2)$ given $x_1, \dots, x_n$.

Algorithm for computing $f(x_1, \dots, x_n)$ will compute $f_j = f(z|_{S_j})$ for $j = 1, \dots, i-1$; randomly sample $b \in \{0, 1\}$.

Call the distinguisher $D(f_1, \dots, f_{i-1}, b, u_{i+1}, \dots, u_m)$ where $u_{i+1}, \dots, u_m$ are fixed. If the result is 1, indication that $f(x_1, \dots, x_n) = b$ and return b; else return $\neg b$.

By the same analysis as the one-bit case given in Theorem 8.3.2, we have a fixed choice for b that makes this algorithm work. This algorithm achieves an advantage of ϵ/m in estimating $f(x)$ given x. ■

Note. The proof for Theorem 8.3.3 is sometimes called a hybrid argument since the distribution H_i is called the hybrid distribution.

Remark. Theorem 8.3.3 shows that by assuming the average-case Nisan-Wigderson assumption, using a simple reduction arguments, a PRG exists!

By the existence of PRG, Theorem 8.3.1 and Corollary 8.3.1 are proved.

Lecture 27: Trevisan's Extractor and Log-Space Computation

6 Dec. 10:30

8.3.3 Reduction Between Worst-Case and Average-Case

We can show the equivalency between the average-case Nisan-Wigderson assumption and the worst-case Nisan-Wigderson assumption via a reduction. Let's first look at the following fundamental problem coding theory considered.

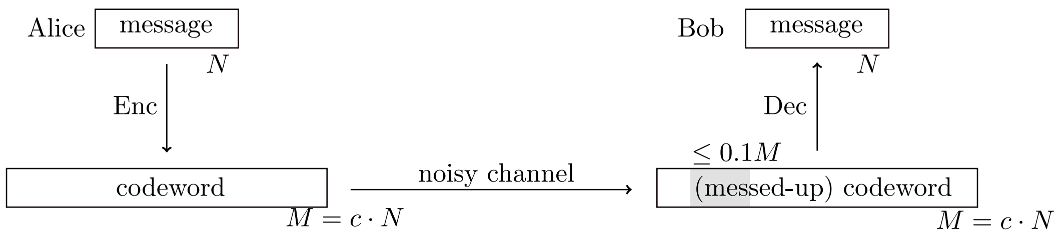


Figure 8.1: Sending message through a noisy channel.

Notation (Error-correcting code)**.** The encoding function $\mathrm{Enc}(\cdot)$ is based on the *error-correcting code.*

After sending the $m = c \cdot n$-bits codeword of an n-bits message through a noisy channel, we can always decode the messed-up codeword exactly to get the original message, providing that the noisy channel only mess-up up to 10% of the codeword.

Note. We need to pay a constant-factor c redundancy in order to make this work.

Then, the idea of hardness amplification is simple. Given a function $f\colon \{0,1\}^n \to \{0,1\}$, i.e., there exists x such that $f(x)$ can't be computed (predicted) by a small circuit. One can simply write out the truth table of f in $N := 2^n$ bits, and encode it with a suitable error-correcting code into a length $M := 2^m$-bits string with $m = \mathsf{poly}(n)$, pretend this encoding is the truth table of a function $F\colon \{0,1\}^m \to \{0,1\}$.

Claim. If f is worst-case hard, then F is average-case hard (with some disclaimers).

Proof. Consider $\mathrm{Enc}(f)$, by changing $\leq 10\%$ of input, we still can recover the exact f. Since there's one input of f is hard, now, we have amplified the number of hard input instances basically. ⊛

Remark (Black box deduction)**.** The above claim doesn't really care about what computation model we're using, i.e., it needs not to be a circuit.

There is something quite important regarding this reduction.

Problem (Locally decodable code)**.** Since the truth table of f and F are enormous, we can't really afford looking through all entries

Answer. We use the so-called *locally decodable code*: given one particular entries of the truth table of f we want, Dec can achieve this by looking at only few entries among the truth table of F. Such a code exists, so it can be done. ⊛

Problem (Listed decodable code)**.** In average-case hardness, instead of predicting 0.9 of the positions, we actually need $1/2 + \epsilon$. Hence, we need to be able to recover from lots of errors.

Answer. Beyond $1/4$, the locally decodable code actually doesn't exist. But we're in luck, since if we relax the requirement a bit, there exists something called *listing decodable code*: we allow the decoder to output a short list of potential answer, i.e., the potential original messages. If the list is short enough, we're fine. ⊛

Note. Beyond 1/2, the channel can simply turn any message into either all 1's or 0's. This is related to the fact that the hardness assumption can't be better than $1/2 + \epsilon$ since there is always an algorithm that can predict a decision problem 50% of the times by random guessing.

The above discussion leads to the Impagliazzo-Wigderson theorem, a generalization the average-case Nisan-Wigderson theorem.

Theorem 8.3.4 (Impagliazzo-Wigderson theorem)**.** Assume the worst-case assumption, then there exists a pseudo-random generator G against all polynomial sized circuits with seed length $O(\log(n))$ and $0 < \epsilon < 1/10$.

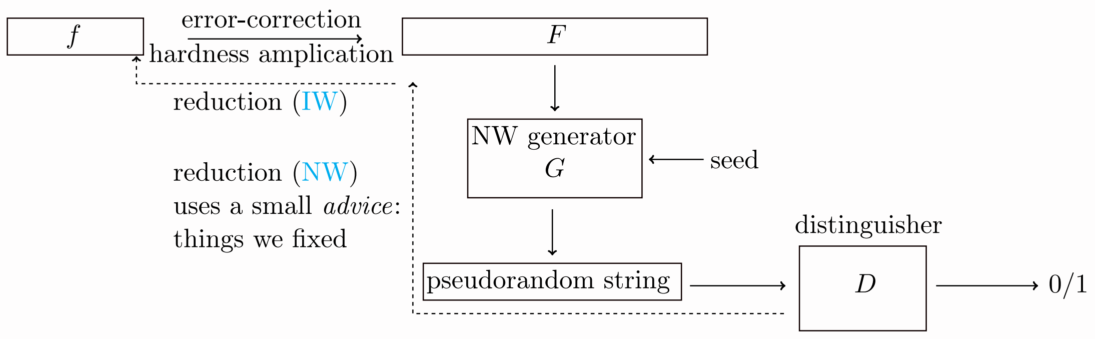
8.3.4 Trevisan's Extractor: The Dual View of the Nisan-Wigderson PSG

Recall that the main reduction in Nisan-Wigderson theorem is based on the fact that the complexity of D is bounded, i.e., it's a non-exponential size circuit.

Now, Trevisan asked the following perhaps, but deep question.

Problem. What will happen if there's no size restriction on D, i.e., it can have exponential size?

Answer. It helps to first look at the overall reduction diagram.



At the first glance, it is just a stupid idea, since we end up predicting f exactly with complexity being exponential, but with exponential size, D can compute any function. However, if we stick on this idea of unbounded D, we first note the following.

Note. f has low entropy.

Proof. The length of the f is much larger than the length of the advice if we do the reduction carefully, which means that it suffices to describe f with a much smaller string, i.e., advice, hence it must have small entropy. ⊛

By letting D is a statistical distinguisher,[a] we observe that if we now make f random, then G is just a seeded extractor! ⊛

[a]This means, let D to be a function such that it maximizes the distance between the pseudorandom string outputted by G and the uniform distribution $\mathcal{U}$.

This interesting finding is due to Trevisan [01], and lots of follow-up works are based on this simple idea. One of the most noticeable results is the following.

Theorem 8.3.5. There exists a (k, ϵ)-extractor Ext: $\{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that $m = k - 2\log\epsilon^{-1} - O(1)$ with $d = O(\log^3(n/\epsilon))$.

Remark. The m is optimal, while the d is not, but still good enough.

8.4 Nisan's Pseudo-Random Generators

Before Nisan and Wigderson working together, Nisan has his own things already, and one of which is about the bounded space pseudo-random generator [Nis90]. Now, we just want a PRG which fools a logarithmic space distinguisher D, i.e., has short memory.

8.4.1 Small Space Complexity Classes

To work with logarithmic space computational model, it helps to formally define the following related notions about the complexity classes compared to P, RP, coRL, and BPL, respectively.

Definition 8.4.1 (Logarithmic space)**.** The complexity class L is defined as $L \in \mathsf{L}$ if and only if there is a deterministic logarithmic space algorithm $\mathcal{A}$ such that

- if $x \in L$, $\mathcal{A}(x) = 1$ and
- if $x \notin L$, $\mathcal{A}(x) = 0$.

Definition 8.4.2 (Randomized logarithmic-space)**.** The complexity class RL is defined as $L \in \mathsf{RL}$ if there exists a probabilistic logarithmic space algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) \geq 1/2$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 0) = 1$.

Definition 8.4.3 (Co-randomized logarithmic-space)**.** The complexity class coRL is defined as $L \in \mathsf{coRL}$ if $\overline{L} \in \mathsf{RL}$, i.e., $\mathsf{coRL} = \left\{\overline{L} \colon L \in \mathsf{RL}\right\}$.

Definition 8.4.4 (Bounded-error probabilistic logarithmic-space)**.** The complexity class BPL is defined as $L \in \mathsf{BPL}$ if there exists a probabilistic logarithmic space algorithm $\mathcal{A}$ such that

- if $x \in L$, $\Pr_R(\mathcal{A}(x, R) = 1) \geq 2/3$ and
- if $x \notin L$, $\Pr_R(\mathcal{A}(x, R) = 0) \geq 2/3$.

Note. To summarize, we have

(a) L: Class of deterministically solvable problems in logarithmic space.

(b) RL: Class of randomized solvable problems in logarithmic space with 1-sided error.

(c) coRL: Class of problems which are complement of RL.

(d) BPL: Class of randomized solvable problems in logarithmic space with 2-sided error.

And still, we assume that these algorithms will terminate.

Problem. Can we have a PRG G against logarithmic-space distinguishers?

Ideally, we want to achieve a seed length in $O(\log n)$ for G, but we don't know how to achieve this, we can only achieve $O(\log^2 n)$. This seems not like a huge deal, but actually it is, at least kind of, due to the following.

Theorem 8.4.1 ([Sav70])**.** There's a way to deterministically simulate a NL algorithm with only $O(\log^2 n)$ space.

Also, for BPL, Savitch shows the following.

Theorem 8.4.2 ([SZ99]). There's a way to deterministically simulate a BPL algorithm with only $O(\log^{1.5} n)$ space.

We see that we already know how to do this in general! But still, if we insist of doing derandomization via PRGs, Nisan showed that there is a PRG again logarithmic-space distinguisher uses $O(\log^2 n)$ seed length [Nis90].

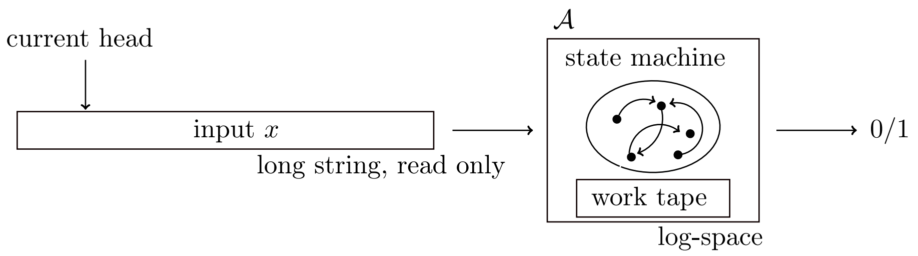
The up-shot is quite simple: we can use extractors to recycle randomness!

Intuition. If an algorithm only has a short-term memory, lots of things can surprise which.

Now, let's first look at one more computational model which makes our analysis easier.

8.4.2 Branching Programs

For fast calculation, we can use the small-size circuist as our computation model; the counterpart to small-space computational model is the so-called branching programs. Say we have a Turing machine, configurations[1] are a way to think about how computation evolves. Hence, a logarithmic space Turing machine should look like the following.



Observe the following.

Remark. A logarithmic machine has $\mathsf{poly}(n)$ possible configurations, where n is the input length.

Proof. To record the head position, we need constant space. The state machine is like source code, which will not grow with size, hence constant again. Then, the only thing which may depend on n is the $O(\log n)$-length working tape, so the possible configuration is $2^{O(1)+O(\log n)} = \mathsf{poly}(n)$. ⊛

Lecture 28: Branching Programs and the Nisan's PRG

The observation of $\mathsf{poly}(n)$ configurations is quite useful, since it directly implies that any logarithmic space Turing machine has to run in polynomial time, since there are just $\mathsf{poly}(n)$ configurations, and you move to another one for each time step. This means: 8 Dec. 10:30

- if it never revisits some configurations: then it must terminate in $\mathsf{poly}(n)$ time step;
- if it revisits a configuration: the Turing machine is deterministic, it'll fall into an infinite loop.

Moreover, we can define the so-called configuration graph for a Turing machine.

Definition 8.4.5 (Configuration graph). The *configuration graph* of a Turing machine is a directed graph which has a node for each configuration, and an edge from node to another if we can go from the first configuration to the next in 1 time step.

This means that a deterministic computation is just a walk over the configuration graph.[2]

[1]The configuration of a Turing machine consists of the current state, where we are on the working tape and the context of which, with where the head is on the input tape (read only).

[2]This is exactly why the *s*-*t* connectivity problem is L-complete, though we will not go deep into this.

Note. For deterministic computation, the out-degree of configuration graph is 1, and for randomized computation, it might not be the case.

Now, we can consider the so-called *branching program*, which is the analogy of circuits, but for space complexity. Firstly, we fix an input, and then the branching program of a randomized logarithmic space algorithm is just a way of looking at the process of the computation, which turns out to have a tree-like structure (neglecting the final merge).

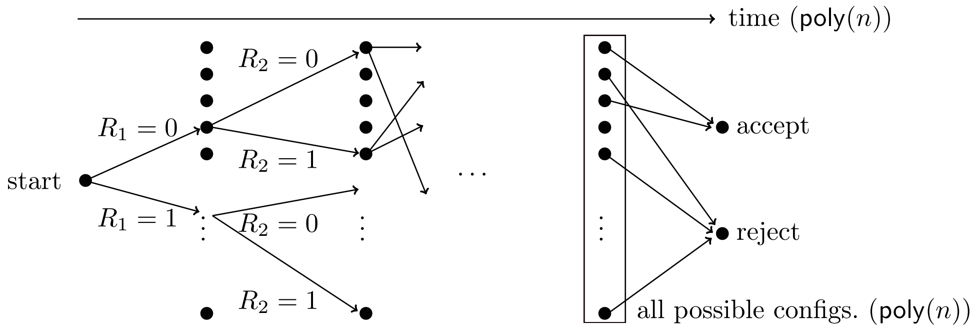


Figure 8.2: Assuming the randomness comes as a string $R = R_1R_2R_3 \cdots R_L$.

Note. The branching program graph is in the size of $\mathsf{poly}(n) \times \mathsf{poly}(n) = \mathsf{poly}(n)$.

8.4.3 Nisan's Pseudo-Random Generator

Now, if this computation takes L step, then we need L uniformly independent coin flips. We want to reduce to only having $\log L$ many random coin flips. As a first step, we ask the following.

Problem. Is it even possible to reduce the number of coin flips needed to $L/2$?

Answer. The main idea is to use an extractor to recycle randomness, i.e., given a random string $R_1R_2 \cdot R_{L/2}$ and a logarithmic length seed Z, we want

$$\mathrm{Ext}((R_1R_2 \cdot R_{L/2}), Z) = R_{L/2+1} \cdot R_L.$$

Since we want a logarithmic space algorithm, so the extractor need to be space efficient too.

Note. The random-walk extractor should be used.

Proof. Performing a random walk can be done in logarithmic space! ⊛

Now, suppose we're at v after $L/2$ steps, and we now try to use Ext to generate the last $L/2$ random bits to complete the walk in the branching program. But since we know that conditioning on the fact that we're at v, which is basically the only thing we know when performing a random walk with logarithm space,

$$H_\infty(\mathrm{Ext}((R_1R_2 \cdot R_{L/2}), Z) \mid \text{we're at } v) \geq L/2 - O(\log n),$$

i.e., the entropy reduces by log(information), with the fact that there are at most $\mathsf{poly}(n)$ possible configuration at each step in the branching program. ⊛

This means reducing the seed-length by half is possible, and the error it causes is neglectable! Now, Nisan's PRG is just the natural idea based on this half-shrink: just recursively apply this idea to both sides, i.e, for the first $L/2$ *coin flips*:

- use $L/4$ *coin flips* to generate $R_1R_2 \cdots R_{L/2}$, and
- use $L/8$ *coin flips* to generate $R_1R_2 \cdots R_{L/4}$,

- etc.

We see that the depth of this recursion is less than $\log L$, hence, the total bits of randomness needed for the random-walk extractor is

$$\log L \times \log n = O(\log n) \times \log n = O(\log^2 n),$$

since at each recursive call, we need $\log n$ bits of true randomness for a random-walk extractor.

Lemma 8.4.1 (Recycling lemma)**.** Let $f\colon \{0,1\}^n \to \{0,1\}^s$ be any function, and $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ be an $(k, \epsilon/2)$-extractor with $k = n - (s+1) - \log \epsilon^{-1}$. Then the statistical distance between $f(x)$ and $\mathcal{U}_m$ is

$$d_{TV}\left(f(x),\mathcal{U}_m), (f(x), \mathrm{Ext}(x,\mathcal{U}_t)\right) < \epsilon$$

where x is uniform over $\{0,1\}^n$.

Proof. The intuition is basically the same: given a random $x \sim \mathcal{U}_n$, $\mathrm{ext}(x, Z)$ will be uniform even conditioned on the knowledge of $f(x)$ where Z is a short length seed. Detail is in [AB09, page 450]. ■

Note. To interpret recycling lemma, $f(x)$ will tell us which node v in $L/2$ layer we end up with, given x to be the first $L/2$ random bits.

By applying recycling lemma, we see that Nisan's PRG only needs a seed with length $O(\log^2 n)$ against any computational model inheres with a branching program with width $\mathsf{poly}(n)$ and time L, with $L = \mathsf{poly}(n)$. And we know that logarithmic space algorithm is one of this kind, so we're done.

Remark. Actually, branching program is much powerful than logarithmic space algorithms, hence it's still a big open problem that whether we can achieve $O(\log n)$ length seed. For example, width 5 branching program is really powerful, and we only know we can achieve around $O(\log n)$ when the width is exactly 3.

Appendix

Appendix A

Acknowledgement

The following is a list of students, organized by the lecture scribe notes they are responsible for.[1]

A.1 Fall 2022

Lecture 1. Pingbang Hu.

Lecture 2. Neha Rama Kumar, Yuzhou Mao, Allanah Matthews.

Lecture 3. Qiyuan Cui, Henry Fleischmann.

Lecture 4. Loris Jautakas, Alec Benjamin Korotney, Alex Li, Allanah Matthews.

Lecture 5. Colin Anthony Finkbeiner, Alec Benjamin Korotney, Lily Wang.

Lecture 6. Serafina Kamp, Zhuocheng Sun, Jiachun Zhang.

Lecture 7. Samuel Benjamin Korman, Feitong Tang, Hongyi Yang.

Lecture 8. Samuel Boardman, Ian Iong Lam, Justin Li, Jiayao Su.

Lecture 9. Zhongqian Duan, Zewen Wu, Tianchen Ye, Charles Beechner Ziegenbein.

Lecture 10. Benjamin Dawson Miller, Lily Wang.

Lecture 11. Benjamin Patrick Kelly, Yucong Lei, Samin Riasat, James Weng.

Lecture 12. Maya Aguas Rao, Dave Yonkers, Lily Wang.

Lecture 13. Kyle Marie Astroth, Anuj Sanjay Tambwekar, Chenming Xing.

Lecture 14. Huanchen Sun, Aditya Vasudevan, David Wang, Dave Yonkers.

Lecture 15. Riya Agarwal, Jacob Hoke Sansom, Mohammad Aamir Sohail.

Lecture 16. Meredith Benson, Allanah Matthews.

Lecture 17. Yuqi Li, Ralph Augustus Worthington, Zhenduo Zhang, Dave Yonkers.

[1]Noticeably, in the main document, the space of the header is limited, so I only list the main scribe note(s) I was referring to when organizing.

Lecture 18. Eric Landgraf, Zhenduo Zhang, Wenfan Jiang, Charles Beechner Ziegenbei.

Lecture 19. Hossein Dabirian, Junliang Huang, Eric Landgraf, Peter Maxwell Ly.

Lecture 20. Yanjun Chen, Wenfan Jiang, Adityasai Koneru, Saumit Kukkadapu, Samin Riasat.

Lecture 21. Zesheng Yu, Lily Wang, Adityasai Koneru.

Lecture 22. Xueshen Liu, Eric Landgraf, Riya Agarwal.

Lecture 23. Thomas Joseph Gregart, Jonathan David Moore, Mingxuan Qu, Dave Yonkers, Riya Agarwal.

Lecture 24. Charles Beechner Ziegenbein, Benjamin Dawson Mille.

Lecture 25. Braden Lucas Crimmins, Keming Ouyang, Jifeng Wang, Riya Agarwal, William Wang, Daiwen Zhang.

Lecture 26. Kellen Arnold Kanarios, Sawan Patel, Jifeng Wang, Weichen Tsai, Zhenduo Zhang, Creighton Glasscock.

Lecture 27. Pingbang Hu.

Lecture 28. Pingbang Hu.

Bibliography

[01] "Extractors and Pseudorandom Generators". In: *J. ACM* 48.4 (July 2001), pp. 860–879. ISSN: 0004-5411. DOI: 10.1145/502090.502099. URL: https://doi.org/10.1145/502090.502099.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN: 9781139477369. URL: https://books.google.com/books?id=nGvI7cOuOOQC.

[AW21] Shyan Akmal and Ryan Williams. *MAJORITY-3SAT (and Related Problems) in Polynomial Time*. 2021. DOI: 10.48550/ARXIV.2107.02748. URL: https://arxiv.org/abs/2107.02748.

[Coh16] Michael B. Cohen. *Ramanujan Graphs in Polynomial Time*. 2016. DOI: 10.48550/ARXIV.1604.03544. URL: https://arxiv.org/abs/1604.03544.

[GW95] Michel X. Goemans and David P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (Nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: https://doi.org/10.1145/227683.227684.

[Hås01] Johan Håstad. "Some Optimal Inapproximability Results". In: *J. ACM* 48.4 (July 2001), pp. 798–859. ISSN: 0004-5411. DOI: 10.1145/502090.502098. URL: https://doi.org/10.1145/502090.502098.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. "Pseudo-Random Generation from One-Way Functions". In: STOC '89. Seattle, Washington, USA: Association for Computing Machinery, 1989, pp. 12–24. ISBN: 0897913078. DOI: 10.1145/73007.73009. URL: https://doi.org/10.1145/73007.73009.

[IW97] Russell Impagliazzo and Avi Wigderson. "P = BPP If E Requires Exponential Circuits: Derandomizing the XOR Lemma". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 220–229. ISBN: 0897918886. DOI: 10.1145/258533.258590. URL: https://doi.org/10.1145/258533.258590.

[Kar72] Richard Karp. "Reducibility Among Combinatorial Problems". In: vol. 40. Jan. 1972, pp. 85–103. ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_8.

[Kho+07] Subhash Khot et al. "Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs?" In: *SIAM Journal on Computing* 37.1 (2007), pp. 319–357. DOI: 10.1137/S0097539705447372. eprint: https://doi.org/10.1137/S0097539705447372. URL: https://doi.org/10.1137/S0097539705447372.

[KKT90] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. "Tight bounds for oblivious routing in the hypercube". In: *Mathematical systems theory* 24 (1990), pp. 223–232.

[LPS88] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. "Ramanujan graphs". In: *Combinatorica* 8 (1988), pp. 261–277.

[Mor94] M. Morgenstern. "Existence and Explicit Constructions of q + 1 Regular Ramanujan Graphs for Every Prime Power q". In: *Journal of Combinatorial Theory, Series B* 62.1 (1994), pp. 44–62. ISSN: 0095-8956. DOI: https://doi.org/10.1006/jctb.1994.1054. URL: https://www.sciencedirect.com/science/article/pii/S0095895684710549.

[Mos08] Robin A. Moser. *A constructive proof of the Lovasz Local Lemma*. 2008. DOI: 10.48550/ARXIV.0810.4812. URL: https://arxiv.org/abs/0810.4812.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. DOI: 10.1017/CBO9780511814075.

[MSS13] Adam Marcus, Daniel A. Spielman, and Nikhil Srivastava. *Interlacing Families I: Bipartite Ramanujan Graphs of All Degrees*. 2013. DOI: 10.48550/ARXIV.1304.4132. URL: https://arxiv.org/abs/1304.4132.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. DOI: 10.1017/CBO9780511813603.

[Nil91] A. Nilli. "On the second eigenvalue of a graph". In: *Discrete Mathematics* 91.2 (1991), pp. 207–210. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(91)90112-F. URL: https://www.sciencedirect.com/science/article/pii/0012365X9190112F.

[Nis90] N. Nisan. "Pseudorandom Generators for Space-Bounded Computations". In: *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*. STOC '90. Baltimore, Maryland, USA: Association for Computing Machinery, 1990, pp. 204–212. ISBN: 0897913612. DOI: 10.1145/100216.100242. URL: https://doi.org/10.1145/100216.100242.

[NW94] Noam Nisan and Avi Wigderson. "Hardness vs randomness". In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(05)80043-1. URL: https://www.sciencedirect.com/science/article/pii/S0022000005800431.

[Rei08] Omer Reingold. "Undirected Connectivity in Log-Space". In: *J. ACM* 55.4 (Sept. 2008). ISSN: 0004-5411. DOI: 10.1145/1391289.1391291. URL: https://doi.org/10.1145/1391289.1391291.

[RT00] Jaikumar Radhakrishnan and Amnon Ta-Shma. "Bounds for Dispersers, Extractors, and Depth-Two Superconcentrators". In: *SIAM Journal on Discrete Mathematics* 13.1 (2000), pp. 2–24. DOI: 10.1137/S0895480197329508. eprint: https://doi.org/10.1137/S0895480197329508. URL: https://doi.org/10.1137/S0895480197329508.

[Sav70] Walter J. Savitch. "Relationships between nondeterministic and deterministic tape complexities". In: *Journal of Computer and System Sciences* 4.2 (1970), pp. 177–192. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(70)80006-X. URL: https://www.sciencedirect.com/science/article/pii/S002200007080006X.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: 10.1145/359168.359176. URL: https://doi.org/10.1145/359168.359176.

[SZ99] Michael Saks and Shiyu Zhou. "$\mathsf{BP}_H\mathsf{SPACE}(H) \subseteq \mathsf{DSPACE}(S^{3/2})$". In: *Journal of Computer and System Sciences* 58.2 (1999), pp. 376–403. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1998.1616. URL: https://www.sciencedirect.com/science/article/pii/S0022000098916166.

[VB81] L. G. Valiant and G. J. Brebner. "Universal Schemes for Parallel Communication". In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. STOC '81. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 1981, pp. 263–277. ISBN: 9781450373920. DOI: 10.1145/800076.802479. URL: https://doi.org/10.1145/800076.802479.